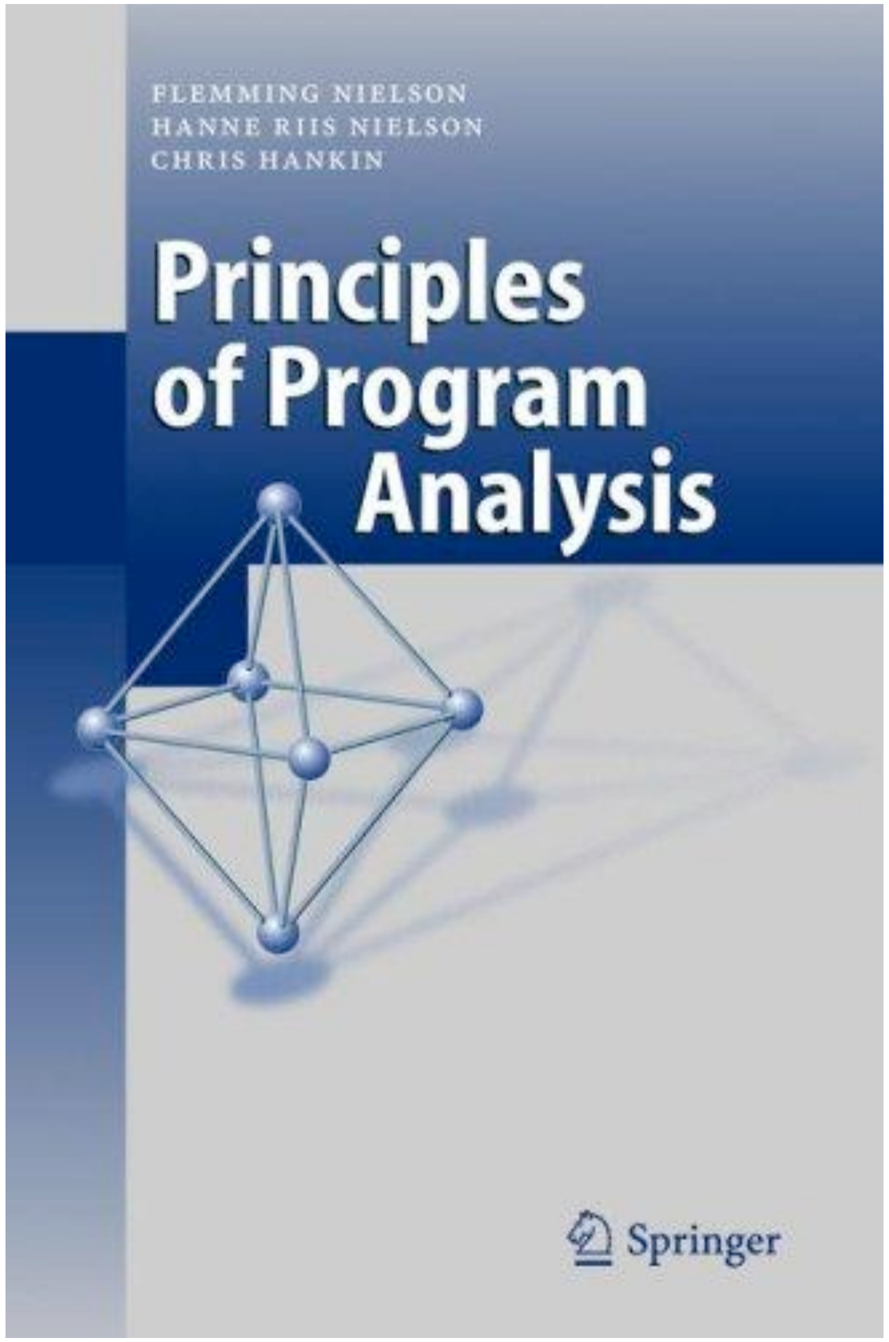
Verification via Abstract Reduction

David Van Horn

& Sam Tobin-Hochstadt



FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN

Principles of Program Analysis

Springer

$$
\begin{array}{lll}
\widehat{\mathsf{C}} \models^{ce}_{\delta} x^{\ell} & \text{iff} & \widehat{\mathsf{C}}(x, ce(x)) \subseteq \widehat{\mathsf{C}}(\ell, \delta) \\
\widehat{\mathsf{C}} \models^{ce}_{\delta} (\lambda x.e)^{\ell} & \text{iff} & \langle \lambda x.e, ce' \rangle \in \widehat{\mathsf{C}}(\ell, \delta) \\
 & & \quad \text{where } ce' = ce \restriction \mathbf{fv}(\lambda x.e) \\
\widehat{\mathsf{C}} \models^{ce}_{\delta} (t^{\ell_1}\ t^{\ell_2})^{\ell} & \text{iff} & \widehat{\mathsf{C}} \models^{ce}_{\delta} t^{\ell_1} \wedge \widehat{\mathsf{C}} \models^{ce}_{\delta} t^{\ell_2} \wedge \\
 & & \forall \langle \lambda x.t^{\ell_0}, ce' \rangle \in \widehat{\mathsf{C}}(\ell_1, \delta) : \\
 & & \quad \widehat{\mathsf{C}}(\ell_2, \delta) \subseteq \widehat{\mathsf{C}}(x, \lceil \delta\ell \rceil_k) \wedge \\
 & & \quad \widehat{\mathsf{C}} \models^{ce'[x \mapsto \lceil \delta\ell \rceil_k]}_{\lceil \delta\ell \rceil_k} t^{\ell_0} \wedge \\
 & & \quad \widehat{\mathsf{C}}(\ell_0, \lceil \delta\ell \rceil_k) \subseteq \widehat{\mathsf{C}}(\ell, \delta)
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{E}[\![x^\ell]\!]_\delta^{ce} & = & \mathsf{C}(\ell,\delta) \leftarrow \mathsf{C}(x, ce(x)) \\
\mathcal{E}[\![(\lambda x.e)^\ell]\!]_\delta^{ce} & = & \mathsf{C}(\ell,\delta) \leftarrow \langle \lambda x.e, ce' \rangle \\
 & & \quad \text{where } ce' = ce \restriction \mathbf{fv}(\lambda x.e) \\
\mathcal{E}[\![(t^{\ell_1} t^{\ell_2})^\ell]\!]_\delta^{ce} & = & \mathcal{E}[\![t^{\ell_1}]\!]_\delta^{ce}; \mathcal{E}[\![t^{\ell_2}]\!]_\delta^{ce}; \\
 & & \text{let } \langle \lambda x.t^{\ell_0}, ce' \rangle = \mathsf{C}(\ell_1, \delta) \text{ in} \\
 & & \quad \mathsf{C}(x, \delta\ell) \leftarrow \mathsf{C}(\ell_2, \delta); \\
 & & \quad \mathcal{E}[\![t^{\ell_0}]\!]_{\delta\ell}^{ce'[x \mapsto \delta\ell]} \\
 & & \quad \mathsf{C}(\ell, \delta) \leftarrow \mathsf{C}(\ell_0, \delta\ell)
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{A}[\![x^{\ell}]\!]^{ce}_{\delta} & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(x, ce(x)) \\
\mathcal{A}[\![(\lambda x.e)^{\ell}]\!]^{ce}_{\delta} & = & \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \{\langle \lambda x.e, ce' \rangle\} \\
 & & \quad \text{where } ce' = ce \restriction \mathbf{fv}(\lambda x.e) \\
\mathcal{A}[\![(t^{\ell_1} t^{\ell_2})^{\ell}]\!]^{ce}_{\delta} & = & \mathcal{A}[\![t^{\ell_1}]\!]^{ce}_{\delta}; \mathcal{A}[\![t^{\ell_2}]\!]^{ce}_{\delta}; \\
 & & \text{foreach } \langle \lambda x.t^{\ell_0}, ce' \rangle \in \widehat{\mathsf{C}}(\ell_1,\delta) : \\
 & & \quad \widehat{\mathsf{C}}(x, \lceil \delta\ell \rceil_k) \leftarrow \widehat{\mathsf{C}}(\ell_2,\delta); \\
 & & \quad \mathcal{A}[\![t^{\ell_0}]\!]^{ce'[x \mapsto \lceil \delta\ell \rceil_k]}_{\lceil \delta\ell \rceil_k}; \\
 & & \quad \widehat{\mathsf{C}}(\ell,\delta) \leftarrow \widehat{\mathsf{C}}(\ell_0, \lceil \delta\ell \rceil_k)
\end{array}
$$

My challenge:

Develop a program analysis for reasoning about:

Space-consumption in a lazy language

State and control in a language with effects

Security in a language with stack inspection

Blame in a language with behavioral contracts

Safe parallelism in a language with futures

My ch

Develop a progr ut:

Spac age

Stat th effects

Secu nspection

Blam al contracts

Safe futures

B-2 - Cody Enterprise - Monday, August 16, 2010

SPORTS/OUTDOORS

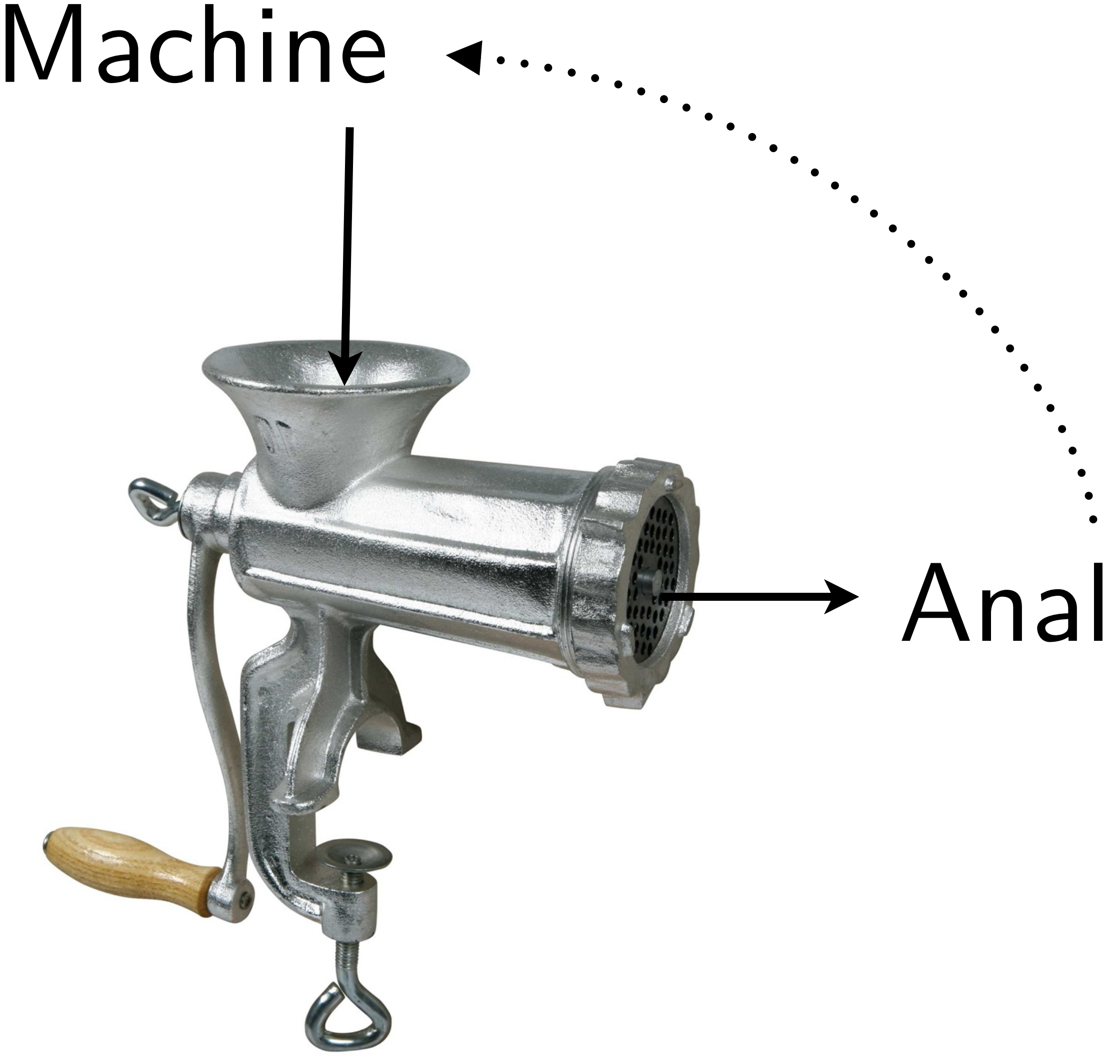
Rough ride

photos by SCOTT SALISBURY

Geoff Marolda of Houston, Texas, is thrown off a bronc as he comes out of the chute during bareback riding recently at the Cody Nite Rodeo.

Abstracting Abstract Machine

Machine



Analysis

Prequel: Syntactic & Functional Correspondence

```
;; Term = Symbol
;;      | (app Term Term)
;;      | (lam Symbol Term)
(struct app (l r))
(struct lam (x e))
```

```
(define (lookup ρ x)
  (match ρ
    [(list-rest (cons y v) ρ)
     (if (eq? y x) v (lookup ρ x))]))

(define (extend ρ x v)
  (cons (cons x v) ρ))
```

The Functional Correspondence

```
;; H.O. Definitional interpreter
(define (eval.0 e)
  ;; term env -> value
  (define (ev e ρ)
    (match e
      [(app l r) ((ev l ρ) (ev r ρ))]
      [(lam x e) (λ (v) (ev e (extend ρ x v)))]
      [x (lookup ρ x)]))

  (ev e '()))
```

```
;; H.O. Definitional interpreter
(define (eval.0 e)
  ;; term env -> value
  (define (ev e ρ)
    (match e
      [(app l r) ((ev l ρ) (ev r ρ))]
      [(lam x e) (λ (v) (ev e (extend ρ x v)))]
      [x (lookup ρ x)]))

  (ev e '()))
```

```
(struct clos (x e ρ))
```

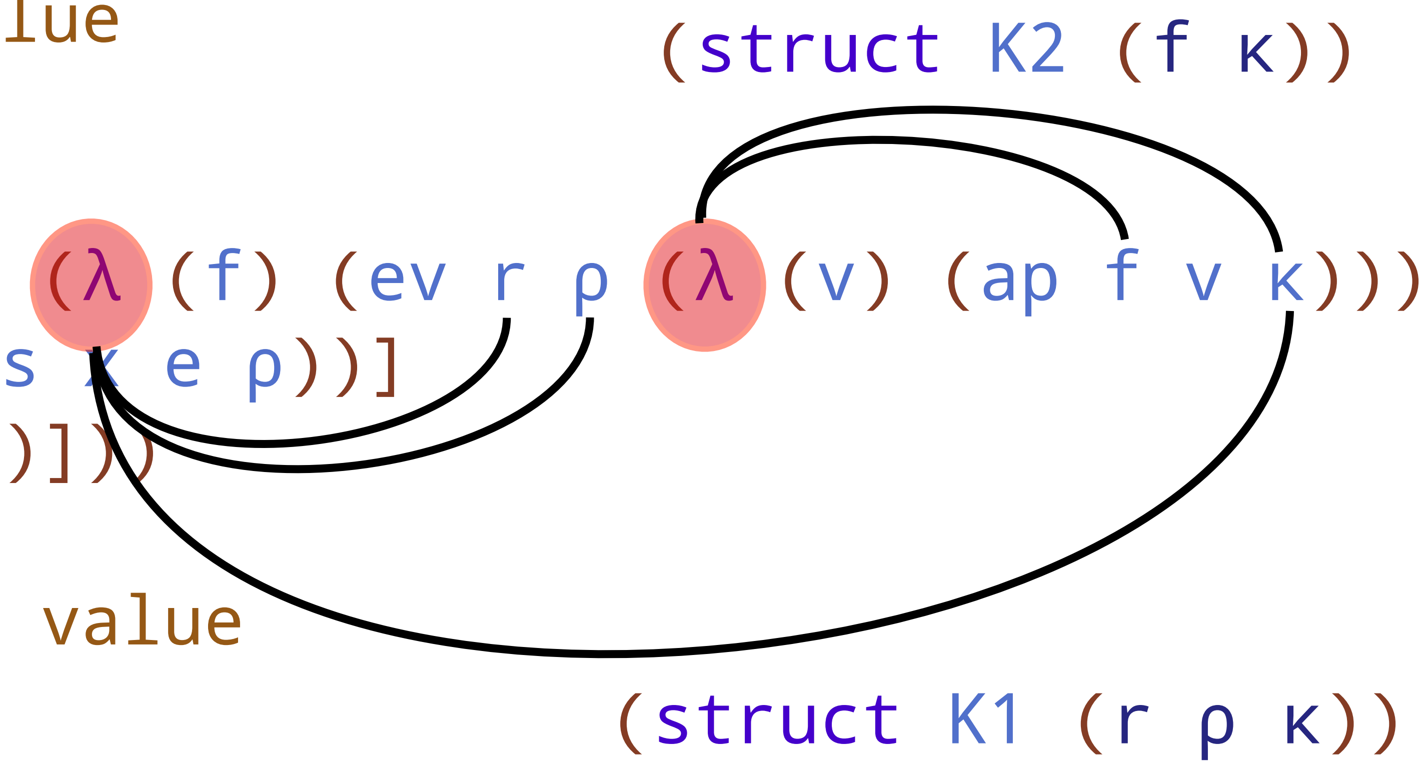
```
;; F.O. Definitional interpreter
;; defunctionalize eval.0
(define (eval.1 e)
  (struct clos (x e ρ))

  ;; term env -> value
  (define (ev e ρ)
    (match e
      [(app l r) (ap (ev l ρ) (ev r ρ))]
      [(lam x e) (clos x e ρ)]
      [x (lookup ρ x)]))

  ;; value value -> value
  (define (ap f v)
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v))]))

  (ev e '()))
```

```
;; H.O. CBV interpreter
;; CPS_V transform eval.1
(define (eval.2 e)
  (struct clos (x e ρ))

  ;; term env cont -> value
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (λ (f) (ev r ρ (λ (v) (ap f v κ)))))]
      [(lam x e) (κ (clos x e ρ))]
      [x (κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (λ (x) x)))
```

```
;; H.O. CBV interpreter
;; CPS_V transform eval.1
(define (eval.2 e)
  (struct clos (x e ρ))

  ;; term env cont -> value
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (λ (f) (ev r ρ (λ (v) (ap f v κ)))))]
      [(lam x e) (κ (clos x e ρ))]
      [x (κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (λ (x) x)))
```

```
(struct K1 (r ρ κ))
```

```
;; H.O. CBV interpreter
;; CPS_V transform eval.1
(define (eval.2 e)
  (struct clos (x e ρ))

  ;; term env cont -> value
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (λ (f) (ev r ρ (λ (v) (ap f v κ)))))]
      [(lam x e) (κ (clos x e ρ))]
      [x (κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (λ (x) x)))
```



(struct K2 (f κ))

(struct K1 (r ρ κ))

```
;; H.O. CBV interpreter
;; CPS_V transform eval.1
(define (eval.2 e)
  (struct clos (x e ρ))

  ;; term env cont -> value                (struct K2 (f κ))
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (λ (f) (ev r ρ (λ (v) (ap f v κ)))))]
      [(lam x e) (κ (clos x e ρ))]
      [x (κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)                      (struct K1 (r ρ κ))
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (λ (x) x)))

           (struct K0 ())
```

```
;; H.O. CBV interpreter
;; CPS_V transform eval.1
(define (eval.2 e)
  (struct clos (x e ρ))

  ;; term env cont -> value                          (struct K2 (f κ))
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (λ (f) (ev r ρ (λ (v) (ap f v κ))))))]
      [(lam x e) (κ (clos x e ρ))]
      [x (κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)                                (struct K1 (r ρ κ))
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (λ (x) x)))

         (struct K0 ())
```

```
;; F.O. CBV interpreter
;; defunctionalize eval.2
(define (eval.3 e)
  (struct clos (x e ρ))
  (struct K0 ())
  (struct K1 (r ρ κ))
  (struct K2 (f κ))

  ;; term env cont -> value
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (K1 r ρ κ))]
      [(lam x e) (co κ (clos x e ρ))]
      [x (co κ (lookup ρ x))]))

  ;; value value cont -> value
  (define (ap f v κ)
    (match f
      [(clos x e ρ)
       (ev e (extend ρ x v) κ)]))

  ;; cont value -> value
  (define (co κ v)
    (match κ
      [(K0) v]
      [(K1 r ρ κ) (ev r ρ (K2 v κ))]
      [(K2 f κ) (ap f v κ)]))

  (ev e '() (K0)))
```

```
;; F.O. CBV interpreter
;; inline ap
(define (eval.4 e)
 (struct clos (x e ρ))
 (struct K0 ())
 (struct K1 (r ρ κ))
 (struct K2 (f κ))

 ;; term env cont -> value
 (define (ev e ρ κ)
  (match e
   [(app l r) (ev l ρ (K1 r ρ κ))]
   [(lam x e) (co κ (clos x e ρ))]
   [x (co κ (lookup ρ x))]))

 ;; cont value -> value
 (define (co κ v)
  (match κ
   [(K0) v]
   [(K1 r ρ κ) (ev r ρ (K2 v κ))]
   [(K2 (clos x e ρ) κ)
    (ev e (extend ρ x v) κ)]))

 (ev e '() (K0)))
```

The Syntactic Correspondence

$$\begin{array}{rcl}
e & ::= & x \mid \lambda x.e \mid (e\ e) \\
c & ::= & (e, \rho) \mid (c\ c) \\
v & ::= & (\lambda x.e, \rho) \\
\mathcal{E} & ::= & [\,] \mid (v\ \mathcal{E}) \mid (\mathcal{E}\ c)
\end{array}$$

$$\begin{array}{rcl}
(x, \rho) & \mathbf{v} & \rho(x) \\
((e_0\ e_1), \rho) & \mathbf{v} & ((e_0, \rho)\ (e_1, \rho)) \\
((\lambda x.e, \rho)\ v) & \mathbf{v} & (e, \rho[x \mapsto v])
\end{array}$$

$$\mathcal{E}[c] \longmapsto_{\mathbf{v}} \mathcal{E}[c'] \text{ iff } c\ \mathbf{v}\ c'$$

$$eval(e) = v \text{ iff } (e, \emptyset) \longmapsto_{\mathbf{v}}^{*} v$$

```
;; Clo = (clo Term Env)
;;     | (app Clo Clo)
(struct clo (e ρ))

;; Ctx = 'hole
;;     | (app Val Ctx)
;;     | (app Ctx Clo)

;; Val = (clo Lam Env)

;; Redex = (app Val Val)
;;       | (clo (app Term Term) Env)
;;       | (clo Symbol Env)
;; redex -> clo
(define (contract r)
  (match r
    [(app (clo (lam x e) ρ) v)
     (clo e (extend ρ x v))]
    [(clo (app e0 e1) ρ)
     (app (clo e0 ρ) (clo e1 ρ))]
    [(clo x ρ)
     (lookup ρ x)]))
```

```
;; clo -> cxt * redex + val
(define (decompose c)
  (match c
    [(clo (lam x e) ρ) c]
    [(app (clo (lam x e) ρ) v)
     (cons 'hole c)]
    [(clo (app e0 e1) ρ)
     (cons 'hole c)]
    [(clo x ρ)
     (cons 'hole c)]
    [(app c0 c1)
     (match c0
       [(clo (lam x e) ρ)
        (match (decompose c1)
          [(cons E r) (cons (app c0 E) r)])]
       [_
        (match (decompose c0)
          [(cons E r) (cons (app E c1) r)])])]))
```

```racket
;; ctx clo -> clo
(define (plug E c)
  (match E
    ['hole c]
    [(app (clo (lam x e) ρ) E)
     (app (clo (lam x e) ρ) (plug E c))]
    [(app E c0)
     (app (plug E c) c0)]))
```

```
;; clo -> clo
(define (reduce c)
  (match (decompose c)
    [(cons E r) (plug E (contract r))]
    [v v]))

;; term -> clo
(define (ev e)
  (define (loop c)
    (match c
      [(clo (lam x e) ρ) c]
      [_ (loop (reduce c))]))
  (loop (clo e '())))
```

BRICS RS-04-26 Danvy & Nielsen: Refocusing in Reduction Semantics

BRICS

Basic Research in Computer Science

Refocusing in Reduction Semantics

Olivier Danvy
Lasse R. Nielsen

BRICS Report Series RS-04-26

ISSN 0909-0878 November 2004

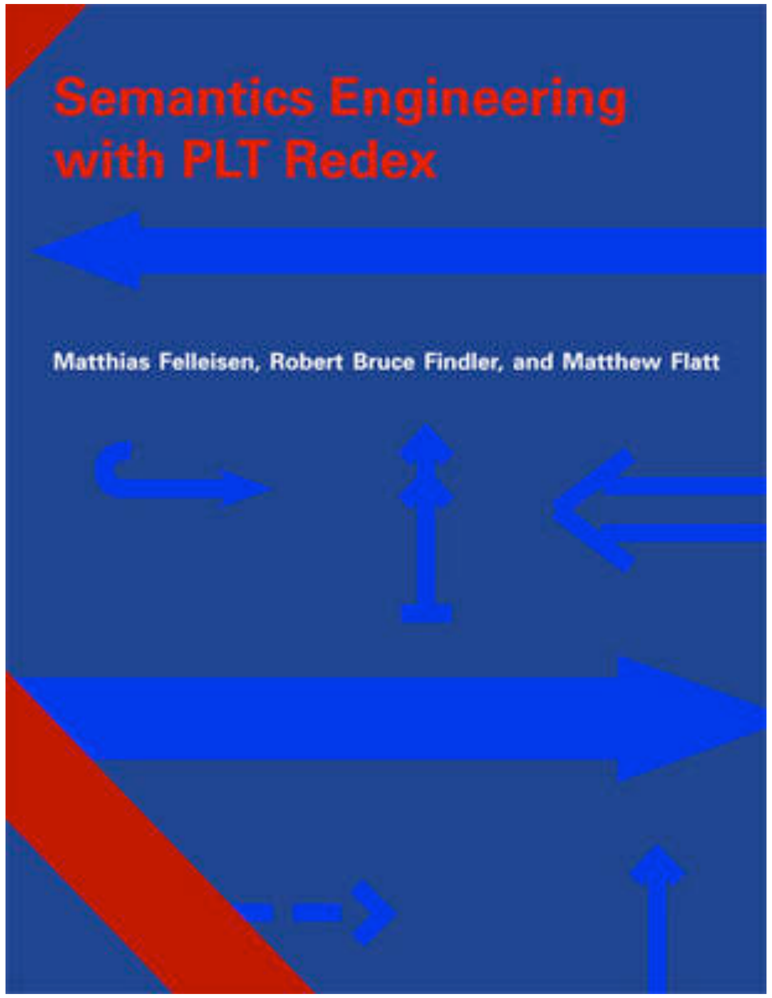
```
;; F.O. CBV interpreter
;; inline ap
(define (eval.4 e)
  (struct clos (x e ρ))
  (struct K0 ())
  (struct K1 (r ρ κ))
  (struct K2 (f κ))

  ;; term env cont -> value
  (define (ev e ρ κ)
    (match e
      [(app l r) (ev l ρ (K1 r ρ κ))]
      [(lam x e) (co κ (clos x e ρ))]
      [x (co κ (lookup ρ x))]))

  ;; cont value -> value
  (define (co κ v)
    (match κ
      [(K0) v]
      [(K1 r ρ κ) (ev r ρ (K2 v κ))]
      [(K2 (clos x e ρ) κ)
       (ev e (extend ρ x v) κ)]))

  (ev e '() (K0)))
```

Abstracting Abstract Machines

```
;; F.O. CBV interpreter
;; store-passing
(define (eval.5 e)
  (define (alloc x σ)
    (cond [(empty? σ) 0]
          [else (add1 (apply max (map car σ)))]))

  (struct clos (x e ρ))

  (struct K0 ())
  (struct K1 (r ρ κ))
  (struct K2 (f κ))

  ;; term env sto cont -> value
  (define (ev e ρ σ κ)
    (match e
      [(app l r) (ev l ρ σ (K1 r ρ κ))]
      [(lam x e) (co κ (clos x e ρ) σ)]
      [x (co κ (lookup σ (lookup ρ x)) σ)]))

  ;; cont value sto -> value
  (define (co κ v σ)
    (match κ
      [(K0) v]
      [(K1 r ρ κ) (ev r ρ σ (K2 v κ))]
      [(K2 (clos x e ρ) κ)
       (define a (alloc x σ))
       (ev e (extend ρ x a) (extend σ a v) κ)]))

  (ev e '() '() (K0)))
```

```
;; F.O. CBV interpreter
;; store-allocated continuations
(define (eval.6 e)
  (define (alloc x σ)
    (cond [(empty? σ) 0]
          [else (add1 (apply max (map car σ)))]))

  (struct clos (x e ρ))
  (struct K0 ())
  (struct K1 (r ρ a))
  (struct K2 (f a))

  ;; term env sto cont -> value
  (define (ev e ρ σ κ)
    (match e
      [(app l r)
       (define a (alloc κ σ))
       (ev l ρ (extend σ a κ) (K1 r ρ a))]
      [(lam x e)
       (co κ (clos x e ρ) σ)]
      [x (co κ (lookup σ (lookup ρ x)) σ)]))

  ;; cont value sto -> value
  (define (co κ v σ)
    (match κ
      [(K0) v]
      [(K1 r ρ a)
       (ev r ρ σ (K2 v a))]
      [(K2 (clos x e ρ) b)
       (define a (alloc x σ))
       (define κ (lookup σ b))
       (ev e (extend ρ x a) (extend σ a v) κ)]))

  (ev e '() '() (K0)))
```

```
(define-language Λ
  [x variable-not-otherwise-mentioned]
  [e x (app e e) (lam x e)]
  [v (clos x e ρ)]
  [ς (ev e ρ σ κ)
     (co κ v σ)
     v]
  [κ (K0)
     (K1 e ρ a)
     (K2 v a)]
  [ρ ((x a) ...)]
  [σ ((a s) ...)]
  [s v κ]
  [(a b) natural])
```



Semantics Engineering with PLT Redex

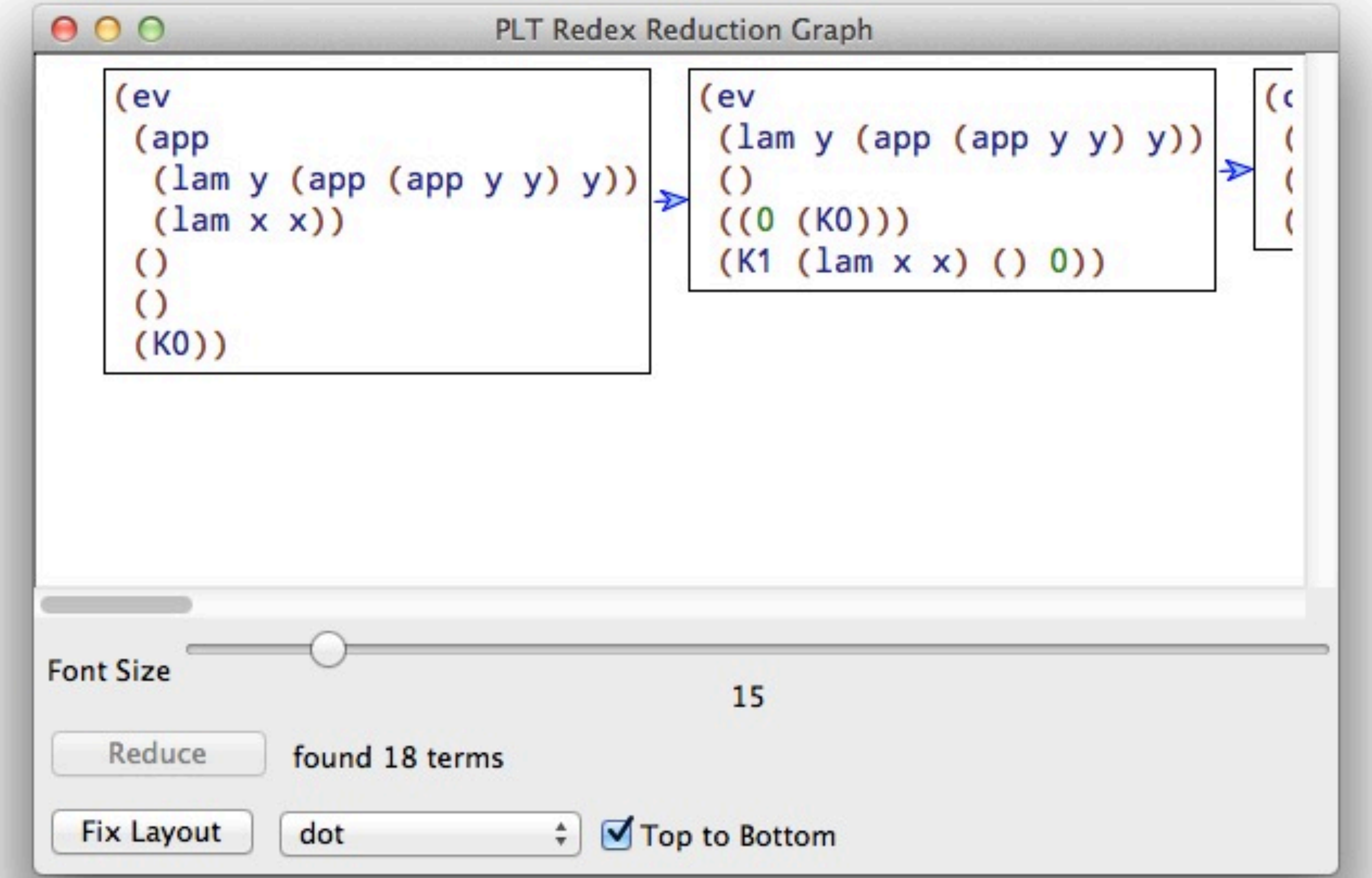
Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt

```
(define-metafunction Λ
  [(lookup ((any_0 any_1) any ...) any_0) any_1]
  [(lookup (any_0 any_1 ...) any_2)
   (lookup (any_1 ...) any_2)])

(define-metafunction Λ
  [(extend (any ...) any_0 any_1)
   ((any_0 any_1) any ...)])
```

```
(define step
  (reduction-relation
   Λ #:domain ς
   (--> (ev (app e_0 e_1) ρ σ κ)
        (ev e_0 ρ (extend σ a κ) (K1 e_1 ρ a))
        (where a (alloc κ σ)))
   (--> (ev (lam x e) ρ σ κ)
        (co κ (clos x e ρ) σ))
   (--> (ev x ρ σ κ)
        (co κ (lookup σ (lookup ρ x)) σ))
   (--> (co (K0) v σ) v)
   (--> (co (K1 e ρ a) v σ)
        (ev e ρ σ (K2 v a)))
   (--> (co (K2 (clos x e ρ) b) v σ)
        (ev e (extend ρ x a) (extend σ a v) κ)
        (where a (alloc x σ))
        (where κ (lookup σ b)))))
```

```
(define-metafunction Λ
  [(alloc any ()) 0]
  [(alloc any ((a any_0) ...))
   ,(add1 (apply max (term (a ...))))])
```

```
(define-metafunction Λ
  [(inj e) (ev e () () (K0))])

(traces step
        (term (inj (app (lam y (app (app y y) y))
                             (lam x x)))))
```



```
(ev
 (app
  (lam y (app (app y y) y))
  (lam x x))
 ()
 ()
 (K0))
```

```
(ev
 (lam y (app (app y y) y))
 ()
 ((0 (K0)))
 (K1 (lam x x) () 0))
```

```
((c
 (
 (
 (
```

Font Size 15

found 18 terms

dot

☑ Top to Bottom

```
(define-language Λ
  [x variable-not-otherwise-mentioned]
  [e x (app e e) (lam x e)]
  [v (clos x e ρ)]
  [ς (ev e ρ σ κ)
     (co κ v σ)
     v]
  [κ (K0)
     (K1 e ρ a)
     (K2 v a)]
  [ρ ((x a) ...)]
  [σ ((a s) ...)]
  [s v κ]
  [(a b) natural])
```

```
(define-language Λ
  [x variable-not-otherwise-mentioned]
  [e x (app e e) (lam x e)]
  [v (clos x e ρ)]
  [ς (ev e ρ σ κ)
     (co κ v σ)
     v]
  [κ (K0)
     (K1 e ρ a)
     (K2 v a)]
  [ρ ((x a) ...)]
  [σ ((a (s ...)) ...)]
  [s v κ]
  [(a b) natural])
```

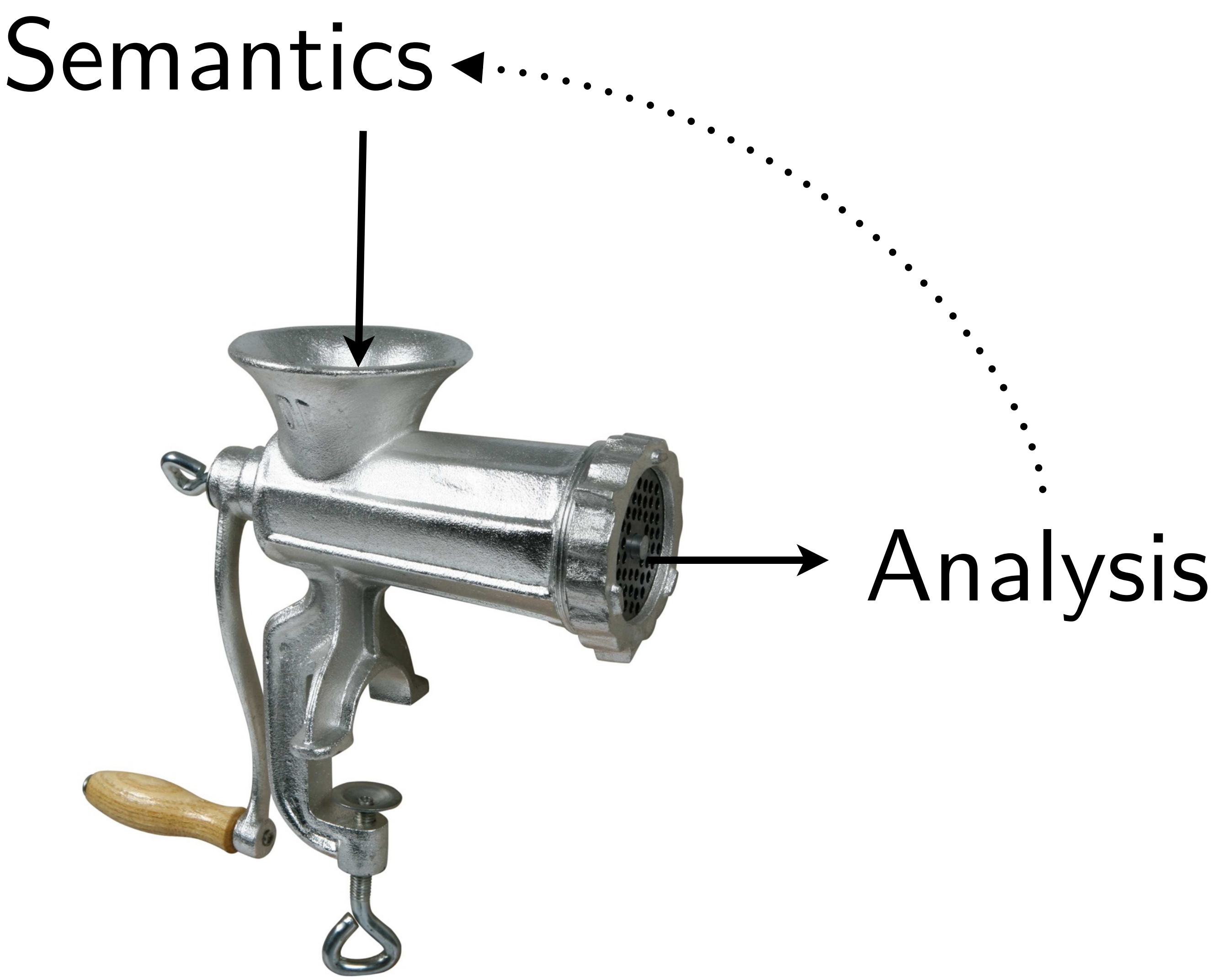
```
(define step
  (reduction-relation
   Λ #:domain ς
   (--> (ev (app e_0 e_1) ρ σ κ)
        (ev e_0 ρ (extend-sto σ a κ) (K1 e_1 ρ a))
        (where a (alloc κ σ)))
   (--> (ev (lam x e) ρ σ κ)
        (co κ (clos x e ρ) σ))
   (--> (ev x ρ σ κ)
        (co κ v σ)
        (where (s_0 ... v s_1 ...)
               (lookup σ (lookup ρ x))))
   (--> (co (K0) v σ) v)
   (--> (co (K1 e ρ a) v σ)
        (ev e ρ σ (K2 v a)))
   (--> (co (K2 (clos x e ρ) b) v σ)
        (ev e (extend ρ x a) (extend-sto σ a v) κ)
        (where a (alloc x σ))
        (where (s_0 ... κ s_1 ...)
               (lookup σ b)))))
```

```
;; No abstraction
(define-metafunction Λ
  [(alloc any ()) 0]
  [(alloc any ((a any_0) ...))
   ,(add1 (apply max (term (a ...))))])
```

```
;; Finite abstraction
(define-metafunction Λ
  [(alloc any σ) 0])
```

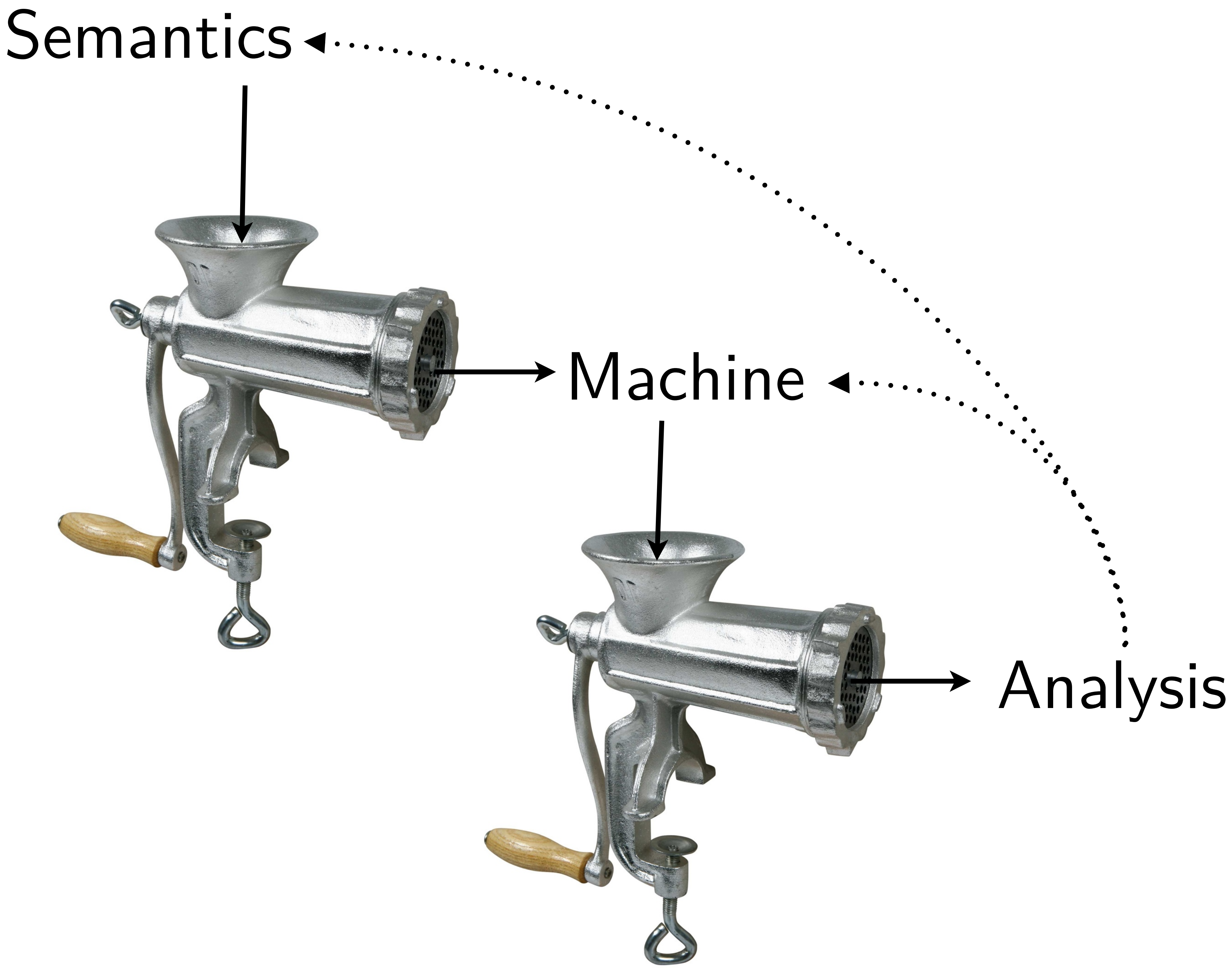
```
;; Pushdown abstraction
(define-metafunction Λ
  [(alloc x σ) 0]
  [(alloc κ ()) 1]
  [(alloc κ ((a any) ...))
   ,(add1 (apply max (term (a ...))))])
```

```
;; Smarter finite abstraction
(define-metafunction Λ
  [(alloc x σ) x]
  [(alloc (K0) σ) K0]
  [(alloc (K1 e ρ a) σ) e]
  [(alloc (K2 (clos x e ρ) a))
   (lam x e)]])
```



Semantics

Analysis



Semantics

Machine

Analysis

The Problem: Modularity matters

- ⋆ Some programs are open (c.f.: the web).
- ⋆ Good components are written in bad languages.
- ⋆ Programs are big; analysis is hard.
- ⋆ Libraries matter.

The black hole approach:

- Shivers, PhD 1991
- Serrano, SAC'95
- Ashley and Dybvig, TOPLAS'98

The componential approach:

- Flanagan, PhD 1997

The type-based approach:

- Banerjee, ICFP'97
- Banerjee and Jensen, MSCS'03
- Lee et al, IPL'02

The contract approach:

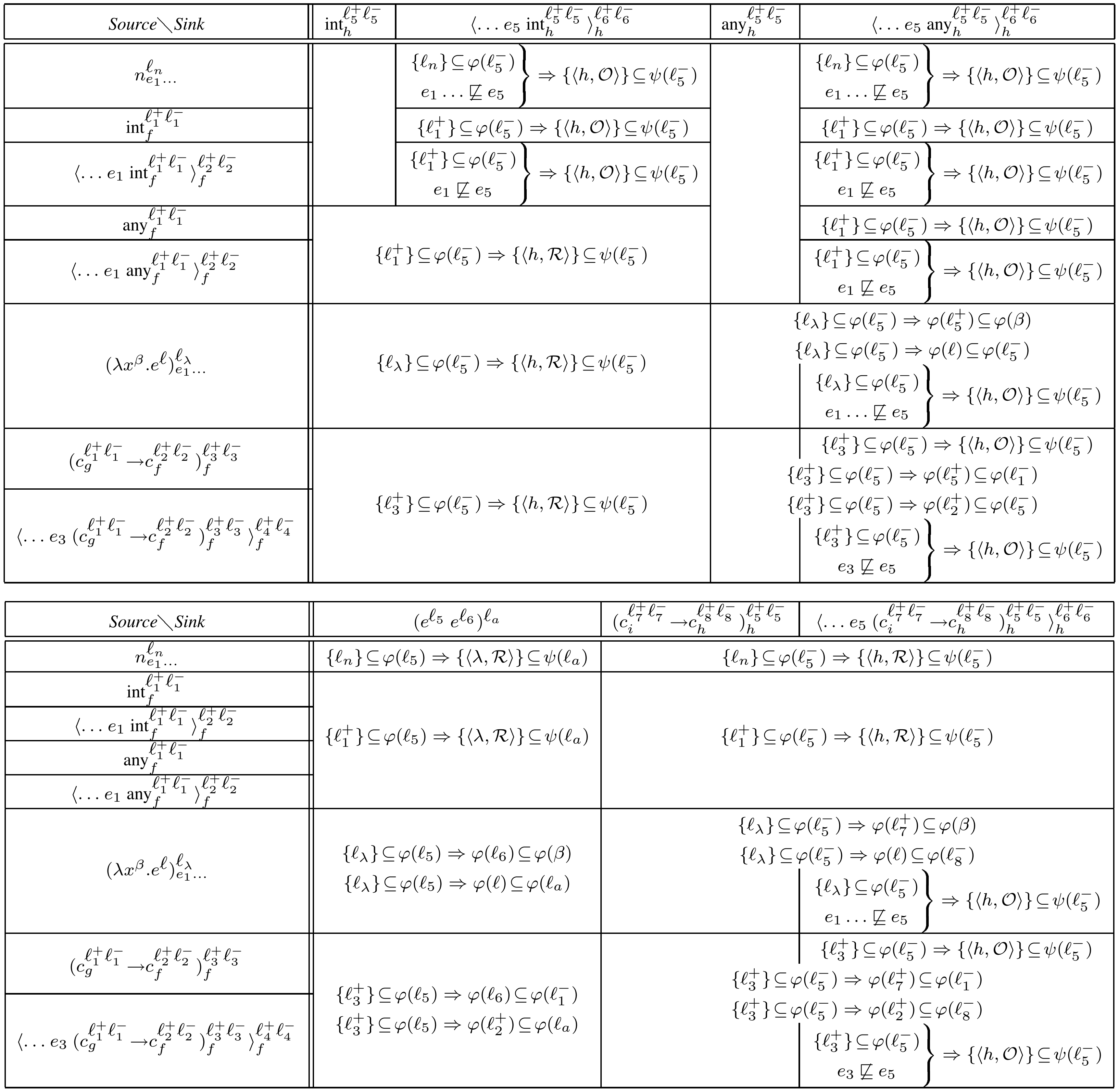
- Meunier, Findler, Felleisen, POPL'06

Analysis

Think hard about modularity

Analysis

Meunier, et al, POPL'06: Modular SBA from Contracts

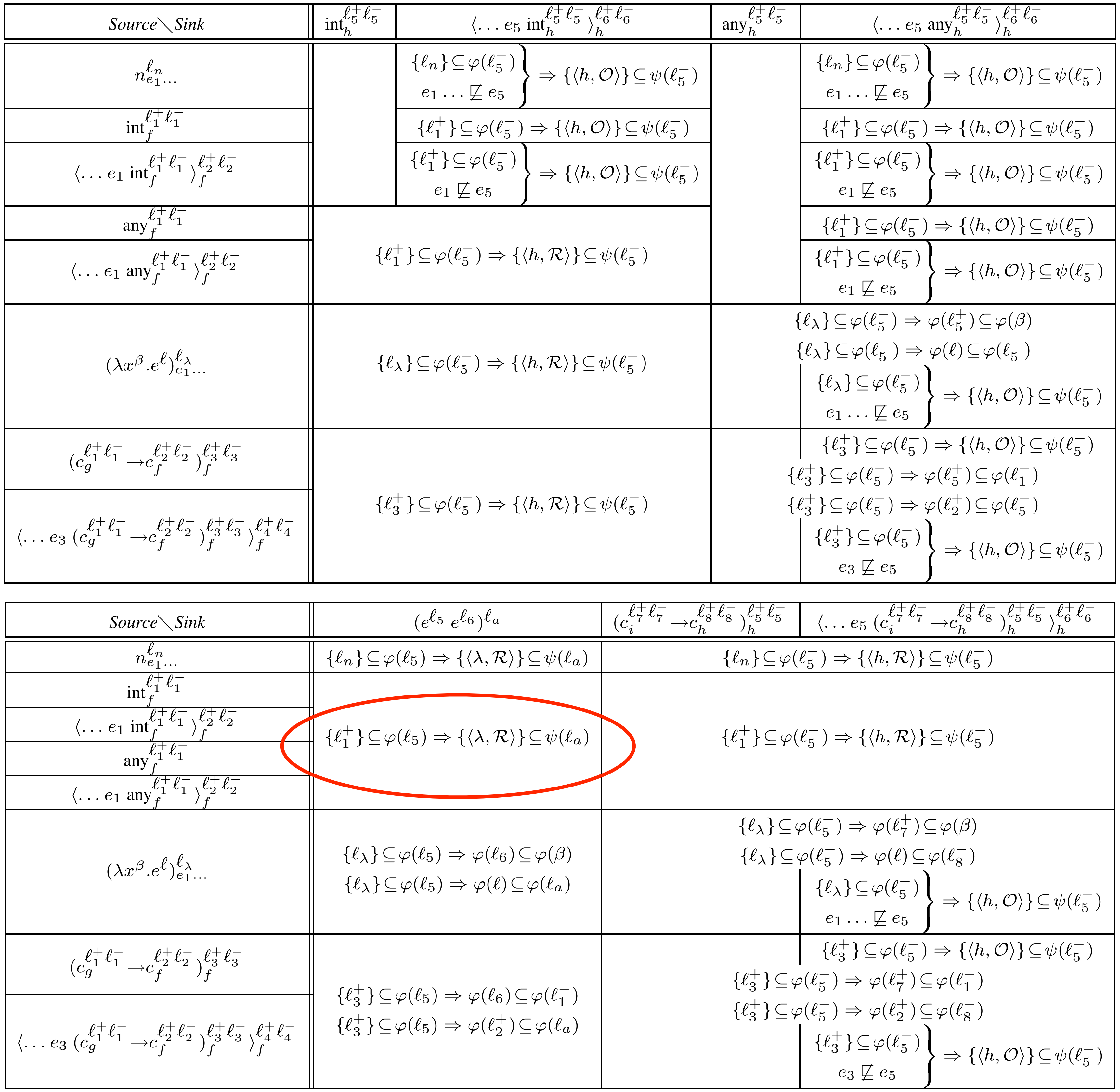


$\textit{Source}\diagdown\textit{Sink}$	$\mathrm{int}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathrm{int}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$	$\mathrm{any}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathrm{any}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$		$\left.\begin{array}{c}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{c}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathrm{int}_f^{\ell_1^+\ell_1^-}$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathrm{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$		$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathrm{any}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$			$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathrm{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$				$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_5^-)$ $\left.\begin{array}{c}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$(c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_5^-)$ $\left.\begin{array}{c}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$				

$\textit{Source}\diagdown\textit{Sink}$	$(e^{\ell_5}\ e^{\ell_6})^{\ell_a}$	$(c_i^{\ell_7^+\ell_7^-}\rightarrow c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ (c_i^{\ell_7^+\ell_7^-}\rightarrow c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$	$\{\ell_n\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_n\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\mathrm{int}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_1\ \mathrm{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$\mathrm{any}_f^{\ell_1^+\ell_1^-}$			
$\langle\ldots e_1\ \mathrm{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_a)$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_8^-)$ $\left.\begin{array}{c}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$(c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_a)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_8^-)$ $\left.\begin{array}{c}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\sqsubseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$			

Table 1. Constraints creation for source-sink pairs.

Meunier, et al, POPL'06: Modular SBA from Contracts



Source$\diagdown$*Sink*	$\mathrm{int}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathrm{int}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$	$\mathrm{any}_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ \mathrm{any}_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$		$\left.\begin{array}{c}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{c}\{\ell_n\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathrm{int}_f^{\ell_1^+\ell_1^-}$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathrm{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$		$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$		$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\mathrm{any}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$			$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$\langle\ldots e_1\ \mathrm{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$				$\left.\begin{array}{c}\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\\ e_1\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_5^-)$ $\left.\begin{array}{c}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$(c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$		$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_5^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_5^-)$ $\left.\begin{array}{c}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$				

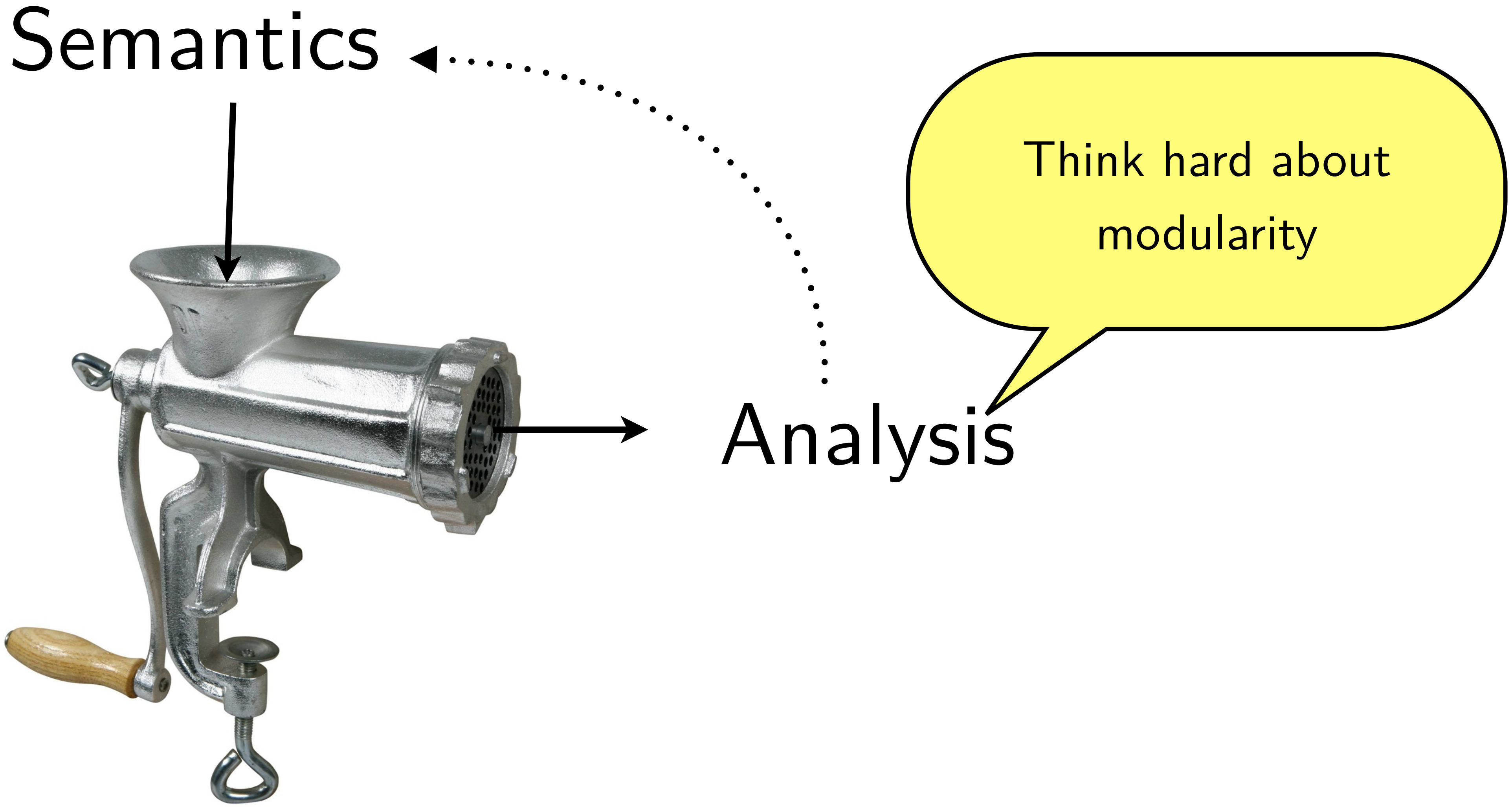
Source$\diagdown$*Sink*	$(e^{\ell_5}\ e^{\ell_6})^{\ell_a}$	$(c_i^{\ell_7^+\ell_7^-}\rightarrow c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}$	$\langle\ldots e_5\ (c_i^{\ell_7^+\ell_7^-}\rightarrow c_h^{\ell_8^+\ell_8^-})_h^{\ell_5^+\ell_5^-}\rangle_h^{\ell_6^+\ell_6^-}$
$n_{e_1\ldots}^{\ell_n}$	$\{\ell_n\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_n\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\mathrm{int}_f^{\ell_1^+\ell_1^-}$	$\{\ell_1^+\}\subseteq\varphi(\ell_5)\Rightarrow\{\langle\lambda,\mathcal{R}\rangle\}\subseteq\psi(\ell_a)$	$\{\ell_1^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{R}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_1\ \mathrm{int}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$\mathrm{any}_f^{\ell_1^+\ell_1^-}$			
$\langle\ldots e_1\ \mathrm{any}_f^{\ell_1^+\ell_1^-}\rangle_f^{\ell_2^+\ell_2^-}$			
$(\lambda x^\beta.e^\ell)_{e_1\ldots}^{\ell_\lambda}$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_a)$	$\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\beta)$ $\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell)\subseteq\varphi(\ell_8^-)$ $\left.\begin{array}{c}\{\ell_\lambda\}\subseteq\varphi(\ell_5^-)\\ e_1\ldots\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$(c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}$	$\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_6)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_a)$	$\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_7^+)\subseteq\varphi(\ell_1^-)$ $\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\Rightarrow\varphi(\ell_2^+)\subseteq\varphi(\ell_8^-)$ $\left.\begin{array}{c}\{\ell_3^+\}\subseteq\varphi(\ell_5^-)\\ e_3\not\subseteq e_5\end{array}\right\}\Rightarrow\{\langle h,\mathcal{O}\rangle\}\subseteq\psi(\ell_5^-)$	
$\langle\ldots e_3\ (c_g^{\ell_1^+\ell_1^-}\rightarrow c_f^{\ell_2^+\ell_2^-})_f^{\ell_3^+\ell_3^-}\rangle_f^{\ell_4^+\ell_4^-}$			

Table 1. Constraints creation for source-sink pairs.

The (General) Solution

Think hard about modularity

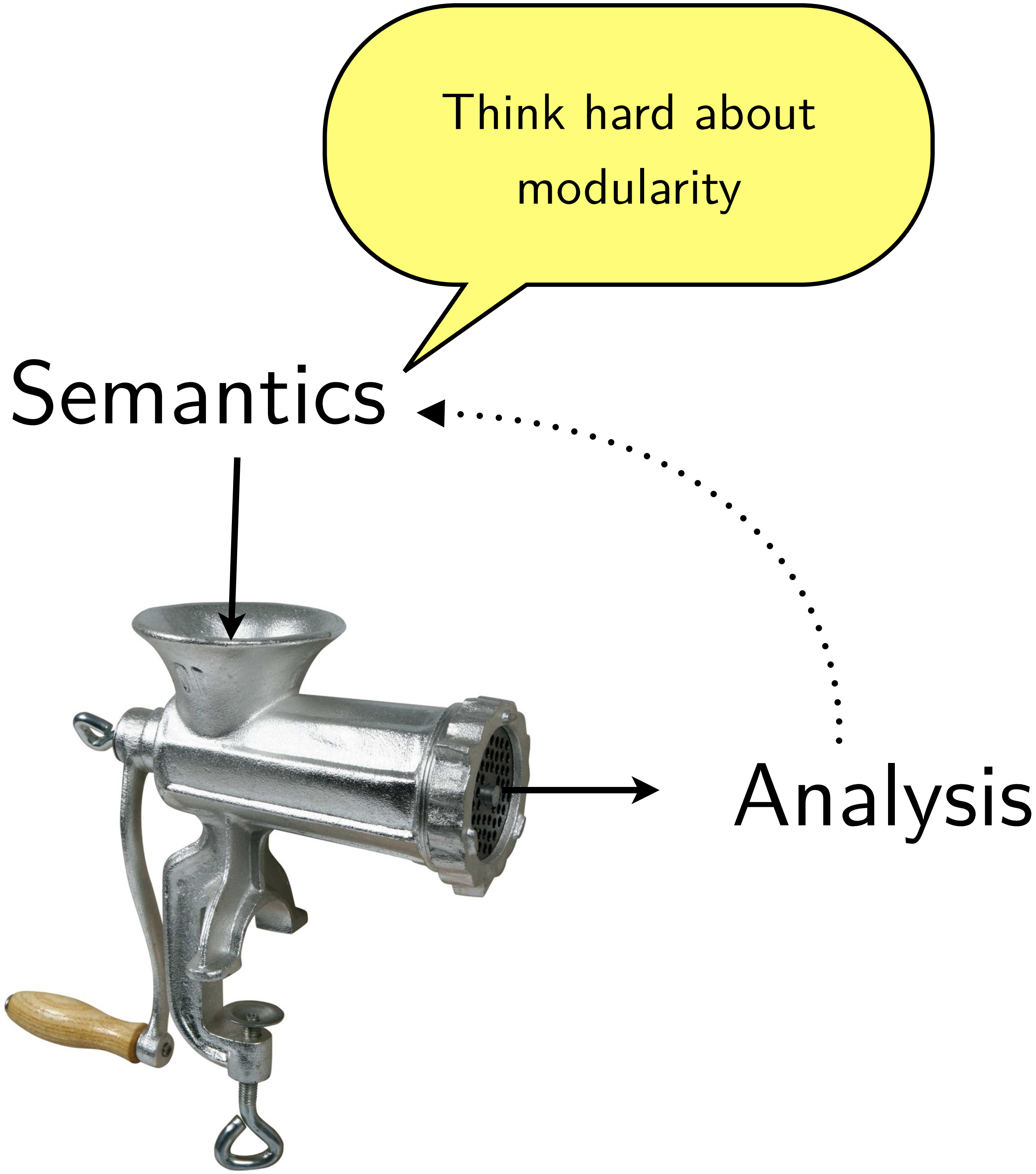
Analysis



Semantics

Analysis

Think hard about modularity



Think hard about modularity

Semantics

Analysis

Semantic Solutions to Program Analysis Problems

Sam Tobin-Hochstadt David Van Horn
PRL, Northeastern University

Abstract

Problems in program analysis can be solved by developing novel program semantics and deriving abstractions conventionally. For over thirty years, higher-order program analysis has been sold as a hard problem. Its solutions have required ingenuity and complex models of approximation. We claim that this difficulty is due to premature focus on *abstraction* and propose a new approach that emphasizes *semantics*. Its simplicity enables new analyses that are beyond the current state of the art.

Current Thoughts, New Ideas

Higher-order program analysis has been an important and recurring topic at PLDI, starting with Shivers' seminal paper [1] and continuing through the present [2]. However, past approaches are limited in the language features they can handle, require intricate formal models that are difficult to develop, verify, and maintain, and do not scale to new questions that we need to answer of programs. We propose a new approach in which interesting *analyses* can be developed by first developing interesting *semantics* and then using known techniques to approximate as a final step.

As an example, Meunier, *et al*. [3] develop a modular program analysis for higher-order behavioral software contracts. Meunier gives an analysis in the form of a large constraint set system and, separately, a dynamic reduction semantics. An important drawback is the dissimilarity between the semantics and the analysis. Both are complicated for the sake of establishing a correspondence, which is accomplished by shoehorning the semantics into an analysis, and tweaking to achieve modularity. Despite these efforts, the soundness theorem does not hold. Worse, the system was then abandoned, as it could not be maintained, extended, or implemented.

In contrast, we have taken the semantics of Meunier's language and *systematically derived* a similar whole-program analysis based on an abstract machine for the language [4]. The machine itself is derived from the semantics through known techniques, making its correctness proof straightforward. This step is purely a semantic refactoring; it has nothing to do with approximation. The machine, however, is in a form that abstracts naturally and transparently [5].

What remains is to make this analysis *modular*, enabling reasoning about programs that are missing some of their components. We solve this problem purely on the semantic side of the equation by extending the dynamic semantics with reductions for programs with missing components. Missing components are regarded as their contracts, which are given reduction rules corresponding to the reductions that may be taken by *any* value satisfying those contracts. As an example, consider the following program fragment consisting of two modules with unknown implementations, keygen and rsa, and a call to rsa to encrypt a string using a key from keygen. Inputs and outputs are annotated with contracts, which are user-defined predicates, i.e. prime?.

```
keygen() : prime? { • }
rsa(k: prime?, s: string?) : string? { • }
rsa(keygen(), "Plain");
```

Under our modular semantics, the program executes as follows:

```
    rsa(keygen(), "Plain");
→   rsa([prime?], "Plain");
→   string?("Plain"); prime?([prime?]); [string?]
→   [string?]
```

The [·] notation denotes a contract treated as a value. Intuitively, it represents the set of all values satisfying the contract. The implementation of keygen is missing, so we cannot know what it returns, but by its specification, it produces a value satisfying prime?, hence it produces [prime?]. To call rsa, we check string? of "Plain" and prime? of [prime?], both of which succeed, so the program produces [string?], an unknown string value. No contracts are violated and thus expensive run-time checks can be eliminated.

To obtain an analysis, this *modular* reduction semantics is run through the same derivation pipeline to reveal a *modular* program analysis. The resulting analysis is easy to verify, extend, and implement, requiring no ingenuity in approximation methods.

The central lesson of this work is that *problems in program analysis can be solved by developing novel program semantics and deriving abstractions conventionally*. Generalizing this observation, we can see that this strategy applies to many analysis problems. Determine the question to be answered, design a semantics that precisely answers this question during evaluation, as in our modular semantics, and finally, use traditional transformations and approximation methods to produce a computable analyzer.

This strategy has several advantages: (1) It is easier to get right. Semantics and analysis correspond closely, making both easier to verify and maintain. (2) Many existing semantics can be repurposed for building analyses of everything from space behavior of lazy languages to security via stack inspection. (3) The PL community has developed a host of intellectual tools for designing and reasoning about semantics which we can re-use for program analysis.

Future Fun

We have taken this approach to leverage dynamic semantics for predicting garbage collection, space consumption, and modularity. There are many exciting opportunities we consider worth pursuing.

1. Using a parallel cost model semantics [6], we can design analyses for predicting space usage of parallel functional programs.
2. Contracts are a form of specification, which we treat as values in a novel semantics. What other kinds of specifications can be treated as values? Giving reductions for values drawn from Hoare-type theory [7] would give rich specifications for effectful components, in turn yielding rich program analyzers.
3. Using a semantics with temporal predicates over program events [8], we can develop higher-order temporal model checkers for history- and stack-based security mechanisms.

These problems seem daunting under current approaches to analysis design, but we conjecture that by taking a semantic approach seriously, solutions will be more easily obtained.

Acknowledgments: We are inspired in part by work with M. Might.

References

[1] O. Shivers. Control flow analysis in Scheme. In *PLDI '88*, pages 164–174.
[2] M. Might, Y. Smaragdakis, and D. Van Horn. Resolving and exploiting the k-CFA paradox. In *PLDI '10*, pages 305–315.
[3] P. Meunier, R. B. Findler, and M. Felleisen. Modular set-based analysis from contracts. In *POPL '06*, pages 218–231.
[4] S. Tobin-Hochstadt and D. Van Horn. Modular analysis via specifications as values. *CoRR*, abs/1103.1362, 2011.
[5] D. Van Horn and M. Might. Abstracting abstract machines. In *ICFP '10*.
[6] D. Spoonhower, G. E. Blelloch, R. Harper, and P. B. Gibbons. Space profiling for parallel functional programs. *JFP*, 20(5-6):417–461, 2010.
[7] A. Nanevski, G. Morrisett, and L. Birkedal. Hoare type theory, polymorphism and separation. *JFP*, 18(5-6):865–911, September 2008.
[8] C. Skalka, S. Smith, and D. Van Horn. Types and trace effects of higher order programs. *JFP*, 18(2):179–249, March 2008.

A (Particular) Instance

Abstract Reduction Semantics for Modular Higher-Order Contract Verification

Sam Tobin-Hochstadt   David Van Horn

Abstract

We contribute a new approach to the modular verification of higher-order programs that leverages behavioral software contracts as a rich source of symbolic values. Our approach is based on the idea of an *abstract* reduction semantics that gives meaning to programs with missing or opaque components. Such components are approximated by their contract and our semantics gives an operational interpretation of contracts-as-values. The result is a executable semantics that soundly approximates all possible instantiations of opaque components, including contract failures. It enables automated reasoning tools that can verify the contract correctness of components for all possible contexts. We show that our approach scales to an expressive language of contracts including arbitrary programs embedded as predicates, dependent function contracts, and recursive contracts. We argue that handling such a feature-rich language of specifications leads to powerful symbolic reasoning that utilizes existing program assertions. Finally, we derive a sound and computable approximation to our semantics that facilitates fully automated contract verification.

1. Behavioral contracts as symbolic values

Whether in the context of dynamically loaded JavaScript programs, low-level native C code, widely-distributed libraries, or simply intractably large code bases, automated reasoning tools must cope with access to only part of the program. To handle missing components, the omitted portions are often assumed to have arbitrary behavior, greatly limiting the precision and effectiveness of the tool. However, programmers who use these components do not make such conservative assumptions. Instead, they attach *specifications* to these components. These specifications increase our ability to reason about programs that are only partially known. But reasoning solely at the level of specification can also make verification and analysis challenging as well as requiring substantial effort to write sufficient specifications.

To tackle these problems, we combine specification-based symbolic reasoning about opaque components with semantics-based concrete reasoning about available components. Our approach to modular program verification is based on computing with *specifications as values*. As specifications, we adopt higher-order behavioral software contracts. Contracts have two crucial advantages for our strategy. First, they provide benefit to programmers outside of verification, since they automatically and dynamically enforce their described invariants. Because of this, modern languages such as C#, Haskell and Racket come with rich contract libraries which programmers already use [9, 15, 17]. Rather than requiring programmers to annotate code with assertions, we leverage the large body of code that already attaches contracts at code boundaries. For example, the Racket standard library features more than 4000 uses of contracts [16]. Second, the meaning of contracts as specifications is neatly captured by their dynamic semantics. As we shall see, we are able to leverage the semantics of contract systems into tools for verification of programs with contracts.

Our plan is as follows: we give a reduction semantics for the core of a higher-order programming language that includes modules and contracts (§4). Next, we take a symbolic execution approach to making our semantics *modular* (§5). This allows us to give non-deterministic behavior to programs in which any number of the component modules are omitted, represented only by their specifications; here given as contracts. We accomplish this by treating contracts as *abstract values*, with the behavior of any of their possible concrete instantiations.

Symbolic execution and refinement calculi have a long history of semantics with abstract elements; contracts as abstract values provides a rich domain of symbols, including precise specifications for abstract higher-order values. These values present new complications to soundness, which we address with a *demonic context*, a universal context for discovering blame for behavioral values (§6).

We note that this semantics is, in itself, a program verifier. The execution of a modular program which runs without contract errors on any path is a verification that the concrete portions of the program never violate their contracts, no matter the instantiation of the omitted portions. This immediately allows us to use contracts for verification in two senses: to verify that programs do not violate their contracts, and verifying rich properties of programs by expressing them as contracts. This technique is surprisingly effective, particularly in systems with many layers, each of which use contracts at their boundaries. For example, the following tail-recursive implementation of insertion sort is verified to live up to its contract, which states that it always produces a sorted list.

As the modular semantics is uncomputable, this verification strategy is necessarily incomplete. To address this, we apply the technique of *abstracting abstract machines* [34] to derive first an abstract machine and then a computable approximation to our semantics directly from our reduction system (§7).

Finally, we turn our semantics into a tool for program verification which is integrated into the Racket [15] toolchain and IDE (§8). Users can click a button and explore the behavior of their program in the presence of opaque modules, either with a non-deterministic and uncomputable semantics, or with a computable approximation.

```
(define-contract list/c
  (rec/c X (or/c empty? (cons/c nat? X))))
(module insert (nat? (and/c list/c (pred sorted?))
                  -> (and/c list/c (pred sorted?)))
  •)
(module insertion-sort
  (list/c (and/c list/c sorted?)
       -> (and/c list/c sorted?))
  (λ (l acc)
     (if (empty? l) acc
         (insertion-sort (cdr l)
                         (insert (car l) acc)))))
(module l list/c •)
(insertion-sort l empty)
```

```
(define/contract dbl
  ((even? . -> . even?) . -> . (even? . -> . even?))
  (λ (f)
    (λ (x)
      (f (f x)))))
```

```
(define/contract dbl
  ((even? . -> . even?) . -> . (even? . -> . even?))
  (λ (f)
    (λ (x)
      (f (f x)))))
```

```
> (dbl (λ (x) 7))
#<procedure>
```

```
(define/contract dbl
  ((even? . -> . even?) . -> . (even? . -> . even?))
  (λ (f)
    (λ (x)
      (f (f x)))))
```

```
> (dbl (λ (x) 7))
#<procedure>
> ((dbl (λ (x) 7)) 8)
contract violation: expected <even?>, given: 7
  contract on dbl from (definition dbl), blaming 'dbl
  contract:
    (-> (-> even? even?) (-> even? even?))
  at: unsaved-editor68934:7.19
>
```

$$
\begin{array}{lcl}
E, F & ::= & x \mid A \mid (E\;E) \mid (\texttt{if}\;E\;E\;E) \mid (o\;E) \mid (C \Leftarrow^{\ell,\ell} E) \\
U, V & ::= & n \mid \texttt{\#t} \mid \texttt{\#f} \mid (\lambda_x x.E) \mid ((C \dashrightarrow C) \Leftarrow^{\ell,\ell} V) \\
C, D & ::= & C \rightarrow C \mid \lfloor \lambda_x x.E \rfloor \\
o & ::= & \texttt{proc?} \mid \texttt{false?} \mid \texttt{add1} \mid \ldots \\
A & ::= & V \mid \texttt{blame}^{\ell}
\end{array}
$$

Basic reductions

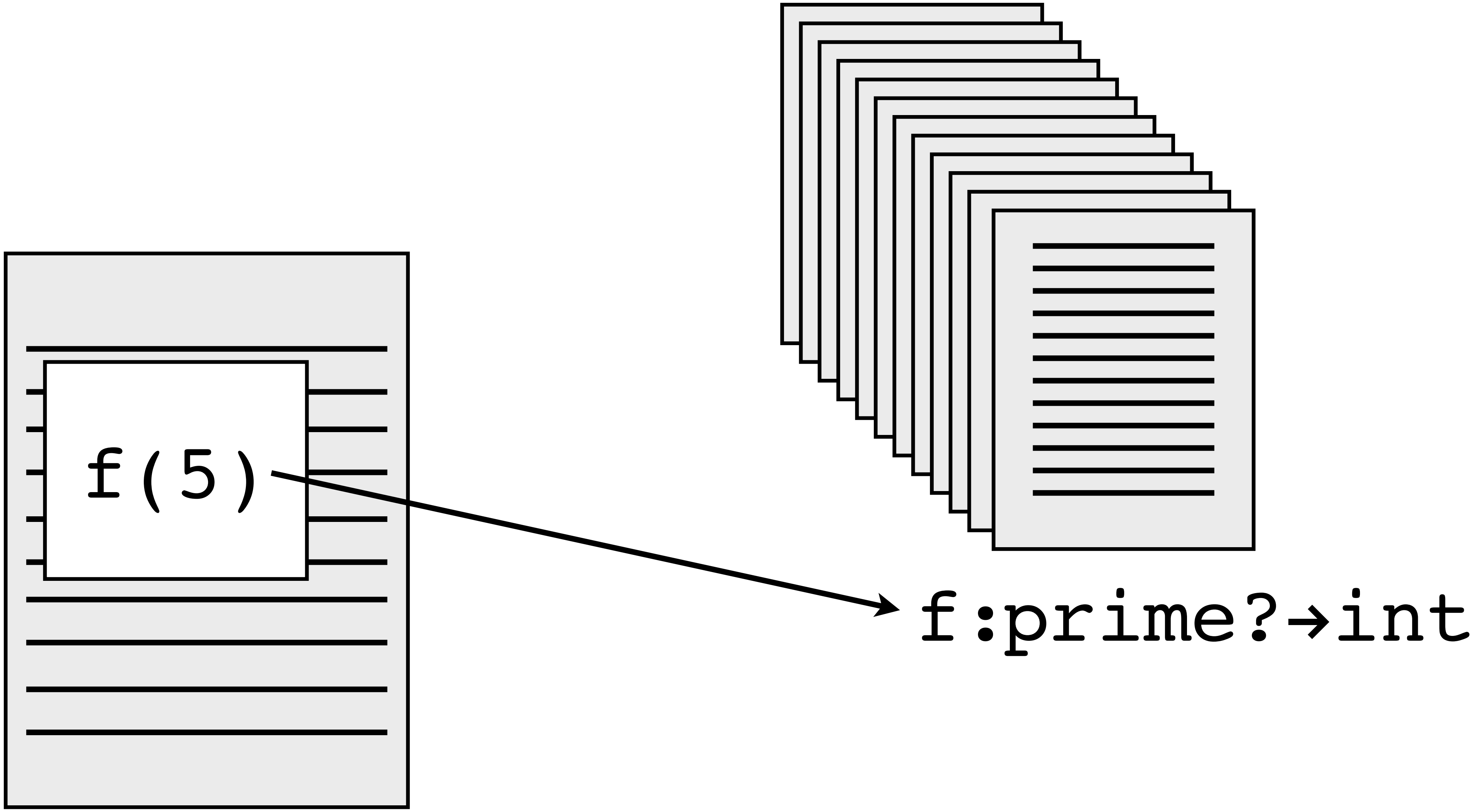
$E \mathrel{\mathbf{c}} F$

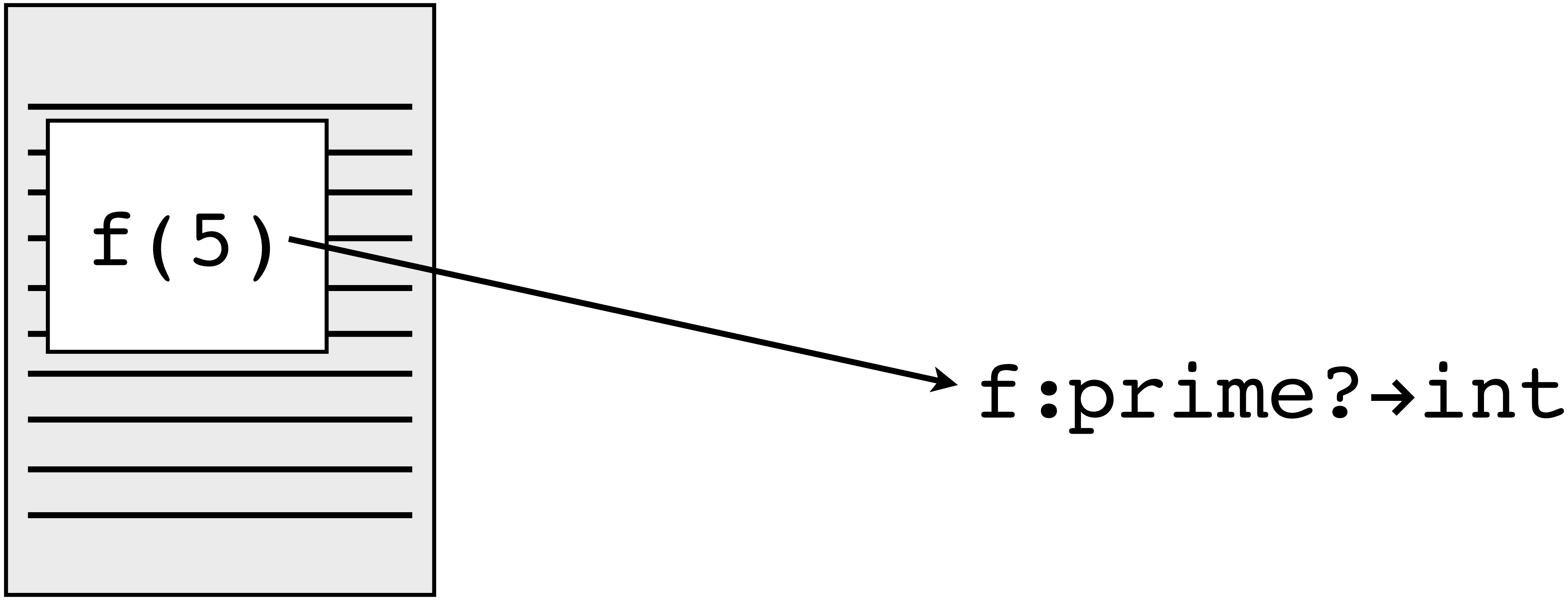
$$
\begin{array}{rcll}
((\lambda_y x.E)\, V) & \mathbf{c} & [(\lambda_y x.E)/y][V/x]E & \\
(o\, V) & \mathbf{c} & A & \text{if } \delta(o, V) \ni A \\
(\texttt{if}\ V\ E\ F) & \mathbf{c} & E & \text{if } \delta(\texttt{false?}, V) \ni \texttt{\#f} \\
(\texttt{if}\ V\ E\ F) & \mathbf{c} & F & \text{if } \delta(\texttt{false?}, V) \ni \texttt{\#t}
\end{array}
$$

Contract checking

$E \; \mathbf{c} \; F$

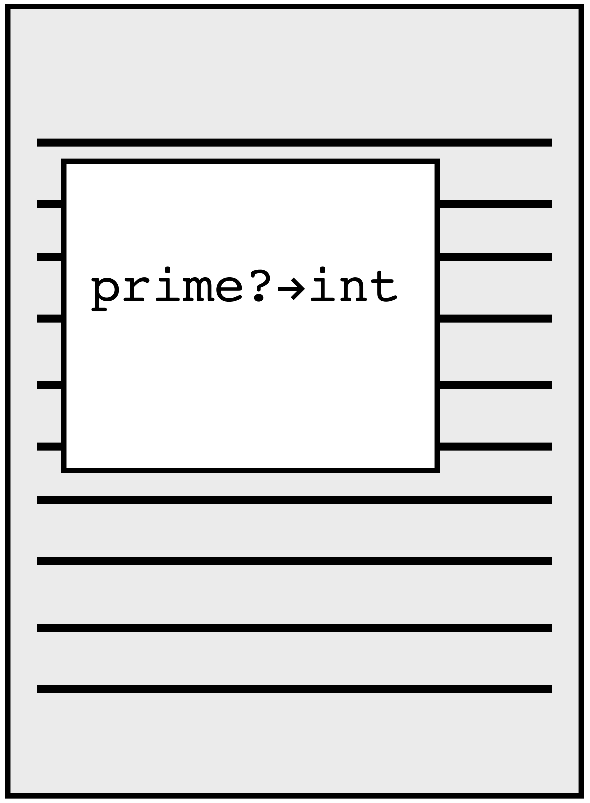
$$\begin{array}{rcl}
(\lfloor E \rfloor \Leftarrow^{\ell,\ell'} V) & \mathbf{c} & (\texttt{if}\ (E\ V)\ V\ \texttt{blame}^{\ell}) \\
(C \rightarrow D \Leftarrow^{\ell,\ell'} V) & \mathbf{c} & ((C \dashrightarrow D) \Leftarrow^{\ell,\ell'} V) \\
 & & \text{if } \delta(\texttt{proc?}, V) \ni \texttt{\#t} \\
(((C \dashrightarrow D) \Leftarrow^{\ell,\ell'} V)\ U) & \mathbf{c} & (D \Leftarrow^{\ell,\ell'} (V\ (C \Leftarrow^{\ell',\ell} U)))
\end{array}$$


f(5)
f:prime?→int

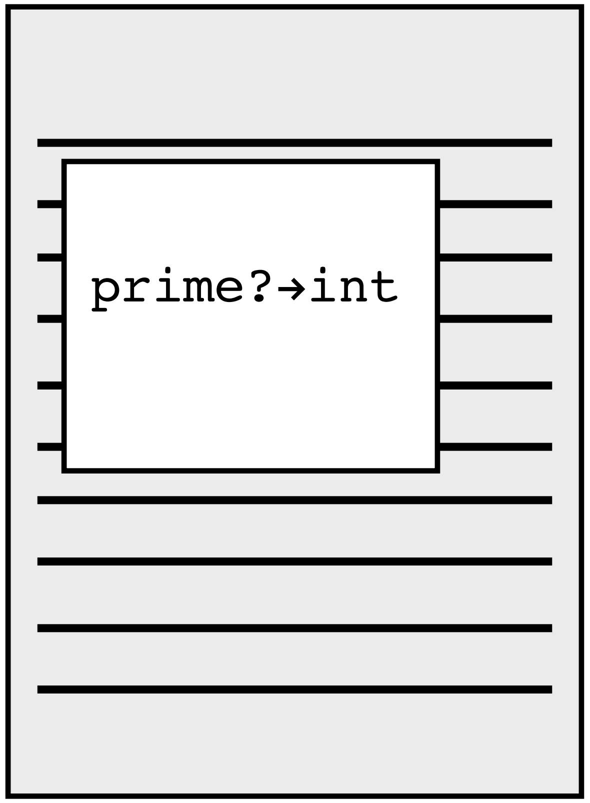


`f(5)`

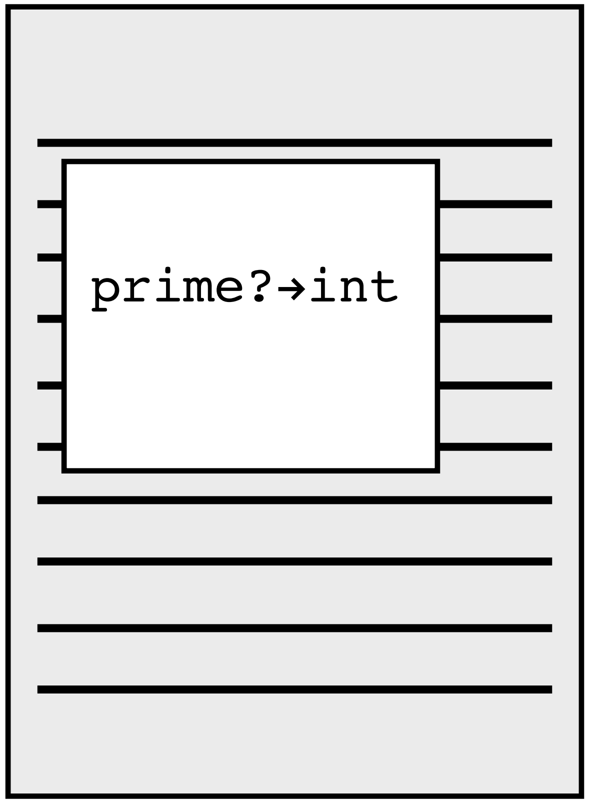
`f:prime?→int`



prime?→int



f → (prime?→int)



$$
\begin{aligned}
\texttt{f} &\to \texttt{(prime?→int)}\\
\texttt{(prime?→int)(5)} &\to^{*} \texttt{int}
\end{aligned}
$$

$$V, U \quad += \quad \bullet \mid V/\mathcal{C}$$

Applying abstract values $E \mathrel{\widehat{\mathbf{c}}} F$

$$
\begin{array}{lcll}
(\widehat{V}\ U) & \widehat{\mathbf{c}} & \bullet/\{D \mid C \rightarrow D \in \mathcal{C}\} & \text{if } \delta(\texttt{proc?}, \widehat{V}) \ni \texttt{\#t} \\
(\widehat{V}\ U) & \widehat{\mathbf{c}} & ((\lambda_y x.(y\ (x\ \bullet)))\ U) & \text{if } \delta(\texttt{proc?}, \widehat{V}) \ni \texttt{\#t}
\end{array}
$$

Contract checking $\qquad E \ \widehat{\mathbf{c}}\ F$

$$
\begin{array}{rcl}
(C \Leftarrow^{\ell,\ell'} V/\mathcal{C}) & \widehat{\mathbf{c}} & V/\mathcal{C} \text{ if } C \in \mathcal{C} \\
(\lfloor E \rfloor \Leftarrow^{\ell,\ell'} V/\mathcal{C}) & \widehat{\mathbf{c}} & (\texttt{if}\ (E\ V/\mathcal{C})\ V'\ \texttt{blame}^{\ell}) \text{ if } \lfloor E \rfloor \notin \mathcal{C} \\
 & & \text{where } V' = V/\mathcal{C} \cup \{\lfloor E \rfloor\} \\
(C \to D \Leftarrow^{\ell,\ell'} V) & \widehat{\mathbf{c}} & ((C \dashrightarrow D) \Leftarrow^{\ell,\ell'} V') \text{ if } \delta(\texttt{proc?}, V) \ni \texttt{\#t} \\
 & & \text{where } V' = V/\{C \to D\}
\end{array}
$$

Scaling up

$$
\begin{array}{lcl}
P, Q & ::= & \boldsymbol{M} E \\
M, N & ::= & (\texttt{module}\ f\ C\ V) \\
E, F & ::= & x \mid f^{\ell} \mid A \mid (E\ E)^{\ell} \mid (\texttt{if}\ E\ E\ E) \mid (o\ \boldsymbol{E})^{\ell} \\
 & \mid & (C \Leftarrow^{f,f}_{f} E) \\
U, V & ::= & n \mid \texttt{#t} \mid \texttt{#f} \mid (\lambda_x x.E) \mid (V, V) \mid \texttt{empty} \\
 & \mid & ((C \dashrightarrow x.C) \Leftarrow^{f,f}_{f} V) \\
C, D & ::= & x \mid C \rightarrow x.C \mid \lfloor \lambda_x x.E \rfloor \mid \langle C,C \rangle \\
 & \mid & C \wedge C \mid C \vee C \mid \mu x.C \\
o & ::= & \texttt{add1} \mid \texttt{car} \mid \texttt{cdr} \mid \texttt{+} \mid \texttt{=} \mid \texttt{cons} \mid o? \\
o? & ::= & \texttt{nat?} \mid \texttt{bool?} \mid \texttt{empty?} \mid \texttt{cons?} \mid \texttt{proc?} \mid \texttt{false?} \\
A & ::= & V \mid \texttt{blame}^{\ell}_{\ell}
\end{array}
$$

Applying abstract values $E\ \widehat{\mathbf{c}}\ F$

$$\begin{array}{rcll} (\bullet/\mathcal{C}\ U) & \widehat{\mathbf{c}} & \bullet/\{[V/x]D \mid C \to x.D \in \mathcal{C}\} & \text{if } \delta(\texttt{proc?}, \bullet/\mathcal{C}) \ni \texttt{\#t} \\ (\widehat{V}\ U) & \widehat{\mathbf{c}} & (\texttt{DEMONIC}\ U) & \text{if } \delta(\texttt{proc?}, \widehat{V}) \ni \texttt{\#t} \end{array}$$

$$\texttt{DEMONIC} = (\lambda_y x.\textsc{amb}(\{(y\ (x\ \bullet)), (y\ (\texttt{car}\ x)), (y\ (\texttt{cdr}\ x))\}))$$

$$\begin{array}{rcl} \textsc{amb}(\{E\}) & = & E \\ \textsc{amb}(\{E\} \uplus \mathcal{E}) & = & (\texttt{if}\ \bullet\ E\ F) \text{ where } F = \textsc{amb}(\mathcal{E}) \end{array}$$

Improving precision via non-determinism $E \mathrel{\widehat{\mathbf{c}}} F$

$$\bullet/\mathcal{C} \mathbin{\dot\cup} \{C_1 \vee C_2\} \quad \widehat{\mathbf{c}} \quad \bullet/\mathcal{C} \cup \{C_i\} \qquad\qquad i \in \{1, 2\}$$

$$\bullet/\mathcal{C} \mathbin{\dot\cup} \{\mu x.C\} \quad \widehat{\mathbf{c}} \quad \bullet/\mathcal{C} \cup \{[\mu x.C/x]C\}$$

Computable approximation

Basic reductions

$$\langle E,\rho,\kappa,\sigma\rangle \longmapsto \langle F,\varrho,\iota,\varsigma\rangle$$

$$
\begin{array}{llll}
\langle (E\ F)^{\ell},\rho,\kappa,\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{ar}^{\ell}(F,\rho,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle (\texttt{if}\ E\ F_1\ F_2),\rho,\kappa,\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{if}(F_1,E_2,\rho,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle (o\ E)^{\ell},\rho,\kappa,\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{op}^{\ell}(o,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle (o\ E\ F)^{\ell},\rho,\kappa,\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{opl}^{\ell}(o,F,\rho,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle x,\rho,\kappa,\sigma\rangle & \longmapsto & \langle V,\varrho,\kappa,\sigma\rangle & \text{if } (V,\varrho)\in\sigma(\rho(x)) \\
\langle V,\rho,\mathsf{ar}^{\ell}(E,\varrho,k),\sigma\rangle & \longmapsto & \langle E,\varrho,\mathsf{fn}^{\ell}(V,\rho,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle V,\rho,\mathsf{fn}^{\ell}((\lambda_y x.E),\varrho,k),\sigma\rangle & \longmapsto & \langle E,\varrho[x\mapsto a,y\mapsto b],\kappa,\sigma[a\mapsto(V,\rho),b\mapsto((\lambda_y x.E),\varrho)]\rangle & \\
\langle V,\rho,\mathsf{fn}^{\ell}(U,\varrho,k),\sigma\rangle & \longmapsto & \langle \texttt{blame}^{\ell}_{\Lambda},\emptyset,\mathsf{mt},\emptyset\rangle & \text{if } \delta(\texttt{proc?},U)\ni\texttt{\#f} \\
\langle V,\rho,\mathsf{if}(E,F,\varrho,k),\sigma\rangle & \longmapsto & \langle E,\varrho,\kappa,\sigma\rangle & \text{if } \delta(\texttt{false?},V)\ni\texttt{\#f} \\
\langle V,\rho,\mathsf{if}(E,F,\varrho,k),\sigma\rangle & \longmapsto & \langle F,\varrho,\kappa,\sigma\rangle & \text{if } \delta(\texttt{false?},V)\ni\texttt{\#t} \\
\langle V,\rho,\mathsf{op}^{\ell}(o,a),\sigma\rangle & \longmapsto & \langle A,\emptyset,\kappa,\sigma\rangle & \text{if } \delta(o^{\ell},V)\ni A \\
\langle ((U,\varrho),(V,\rho)),\emptyset,\mathsf{op}(\texttt{car},a),\sigma\rangle & \longmapsto & \langle U,\varrho,\kappa,\sigma\rangle & \\
\langle ((U,\varrho),(V,\rho)),\emptyset,\mathsf{op}(\texttt{cdr},a),\sigma\rangle & \longmapsto & \langle V,\rho,\kappa,\sigma\rangle & \\
\langle V,\rho,\mathsf{opl}^{\ell}(o,E,\varrho,k),\sigma\rangle & \longmapsto & \langle E,\varrho,\mathsf{opr}^{\ell}(o,V,\rho,k),\sigma\rangle & \\
\langle V,\rho,\mathsf{opr}^{\ell}(\texttt{cons},U,\varrho,a),\sigma\rangle & \longmapsto & \langle ((U,\varrho),(V,\rho)),\emptyset,\kappa,\sigma\rangle & \\
\langle V,\rho,\mathsf{opr}^{\ell}(o,U,\varrho,a),\sigma\rangle & \longmapsto & \langle A,\emptyset,\kappa,\sigma\rangle & \\
\langle \texttt{blame}^{\ell}_{\ell'},\rho,\kappa,\sigma\rangle & \longmapsto & \langle \texttt{blame}^{\ell}_{\ell'},\emptyset,\mathsf{mt},\emptyset\rangle &
\end{array}
$$

Module references

$$
\begin{array}{llll}
\langle f^{f},\rho,\kappa,\sigma\rangle & \longmapsto & \langle V,\emptyset,\kappa,\sigma\rangle & \text{if } (f^{f},V)\in\widehat{\Delta}(\boldsymbol{M}) \\
\langle f^{g},\rho,\kappa,\sigma\rangle & \longmapsto & \langle V,\emptyset,\mathsf{chk}^{f,g}_{h}(C,\emptyset,k),\sigma[k\mapsto\kappa]\rangle & \text{if } (f^{f},(C\Leftarrow^{f,g}_{h}V))\in\widehat{\Delta}(\boldsymbol{M})
\end{array}
$$

Contract checking

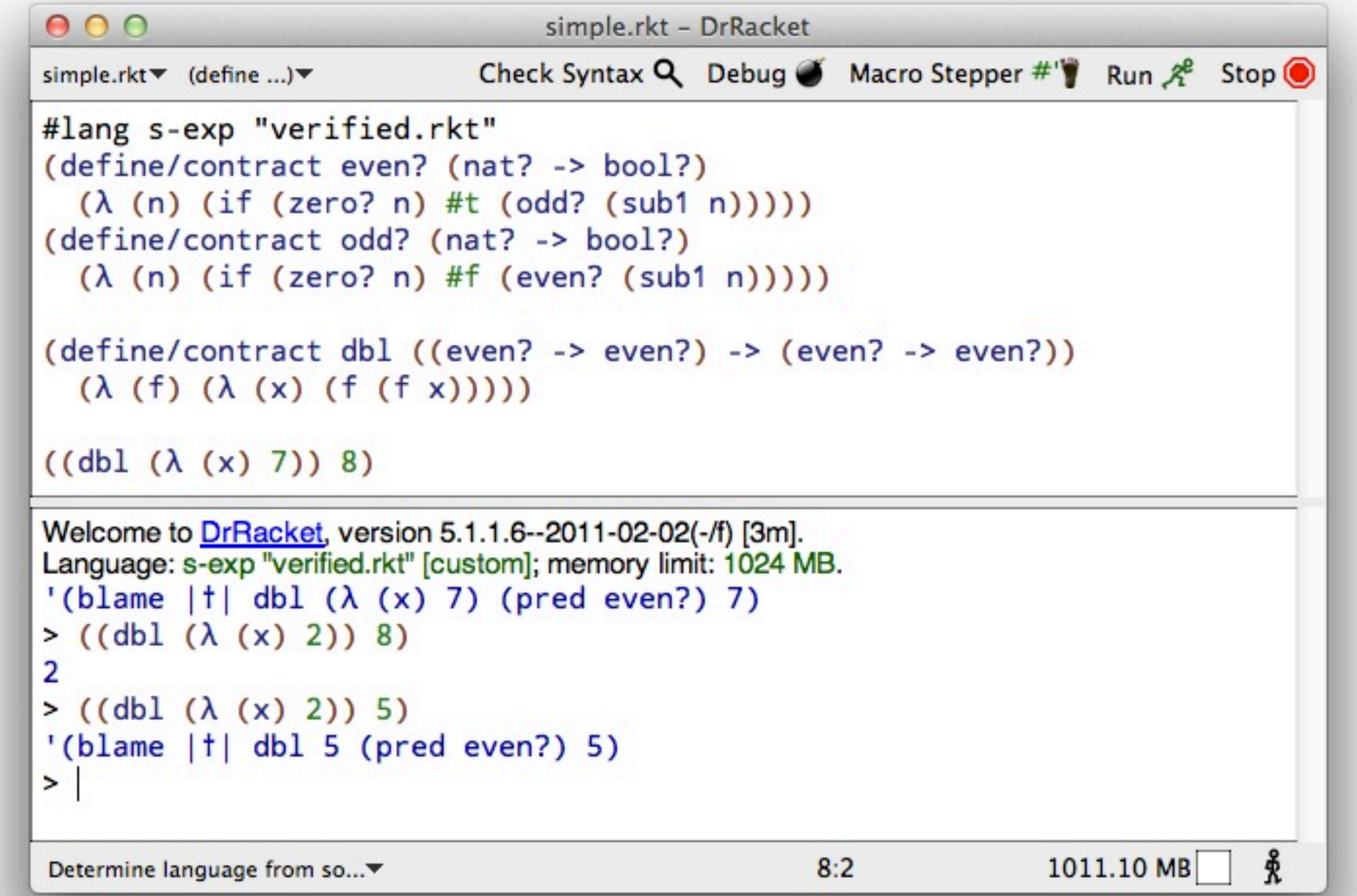
$$
\begin{array}{llll}
\langle (C\Leftarrow^{f,g}_{h}E),\rho,\kappa,\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{chk}^{f,g}_{h}(C,\rho,k),\sigma[k\mapsto\kappa]\rangle & \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(C,\varrho,k),\sigma\rangle & \longmapsto & \langle V,\rho,\mathsf{fn}(U,\varrho,k'),\sigma[k'\mapsto\mathsf{if}(V/\{C\},\texttt{blame}^{f}_{g},\rho,k)]\rangle & \\
 & & \text{where } C \text{ is flat and } U=\textsc{fc}(C,V) & \\
\langle V,\rho,\mathsf{fn}^{\ell}(((C\dashrightarrow x.D)\Leftarrow^{f,g}_{h}a),\varrho,k),\sigma\rangle & \longmapsto & \langle V,\rho,\mathsf{chk}^{g,f}_{h}(C,\varrho,k'),\sigma[k'\mapsto\mathsf{fn}^{\ell}(U,\varrho',k''), & \\
 & & \qquad k''\mapsto\mathsf{chk}^{f,g}_{h}(D,\varrho[x\mapsto b],k), & \\
 & & \qquad b\mapsto(V,\rho)]\rangle & \\
 & & \text{where } (U,\varrho')\in\sigma(a) & \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(C\rightarrow x.D,\varrho,k),\sigma\rangle & \longmapsto & \langle ((C\dashrightarrow x.D)\Leftarrow^{f,g}_{h}a),\varrho,\iota,\sigma[a\mapsto(V,\rho)]\rangle & \text{if } \delta(\texttt{proc?},V)\ni\texttt{\#t} \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(C\rightarrow x.D,\varrho,k),\sigma\rangle & \longmapsto & \langle \texttt{blame}^{f}_{h},\emptyset,\mathsf{mt},\emptyset\rangle & \text{if } \delta(\texttt{proc?},V)\ni\texttt{\#f} \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(C\wedge D,\varrho,k),\sigma\rangle & \longmapsto & \langle V,\rho,\mathsf{chk}^{f,g}_{h}(C,\varrho,i),\sigma[i\mapsto\mathsf{chk}^{f,g}_{h}(D,\varrho,k)]\rangle & \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(C\vee D,\varrho,k),\sigma\rangle & \longmapsto & \langle V,\rho,\mathsf{ar}(U,\varrho,i),\sigma[i\mapsto\mathsf{chk\text{-}or}^{f,g}_{h}(V,\rho,C\vee D,\varrho,k)]\rangle & \\
 & & \text{where } U=\textsc{fc}(C) & \\
\langle V,\rho,\mathsf{chk\text{-}or}^{f,g}_{h}(U,\varrho,C\vee D,\rho',k),\sigma\rangle & \longmapsto & \langle U/\{C\},\varrho,\kappa,\sigma\rangle & \text{if } \delta(\texttt{false?},V)\ni\texttt{\#f} \\
\langle V,\rho,\mathsf{chk\text{-}or}^{f,g}_{h}(U,\varrho,C\vee D,\rho',k),\sigma\rangle & \longmapsto & \langle U,\varrho,\mathsf{chk}^{f,g}_{h}(D,\rho',k),\sigma\rangle & \text{if } \delta(\texttt{false?},V)\ni\texttt{\#t}
\end{array}
$$

Abstract values

$$
\begin{array}{llll}
\langle V,\rho,\mathsf{fn}^{\ell}(\bullet/\mathcal{C},\varrho,k),\sigma\rangle & \longmapsto & \langle E,\rho,\mathsf{begin}(U,\varrho,k),\sigma\rangle & \text{if } \delta(\texttt{proc?},\bullet/\mathcal{C})\ni\texttt{\#t} \\
 & & \text{where } E=\textsc{amb}(\{\texttt{\#t},\textsc{demonic}(\bigwedge\textsc{dom}(\mathcal{C}),V)\}) \text{ and } U=\bullet/\textsc{rng}(\mathcal{C}) & \\
\langle V,\rho,\mathsf{begin}(E,\varrho,k),\sigma\rangle & \longmapsto & \langle E,\varrho,\kappa,\sigma\rangle & \\
\langle \bullet/\mathcal{C}\uplus\{C_1\vee C_2\},\rho,\kappa,\sigma\rangle & \longmapsto & \langle \bullet/\mathcal{C}\cup\{C_i\},\rho,\kappa,\sigma\rangle & \\
\langle \bullet/\mathcal{C}\uplus\{\mu x.C\},\rho,\kappa,\sigma\rangle & \longmapsto & \langle \bullet/\mathcal{C}\uplus\{[\mu x.C/x]C\},\rho,\kappa,\sigma\rangle &
\end{array}
$$

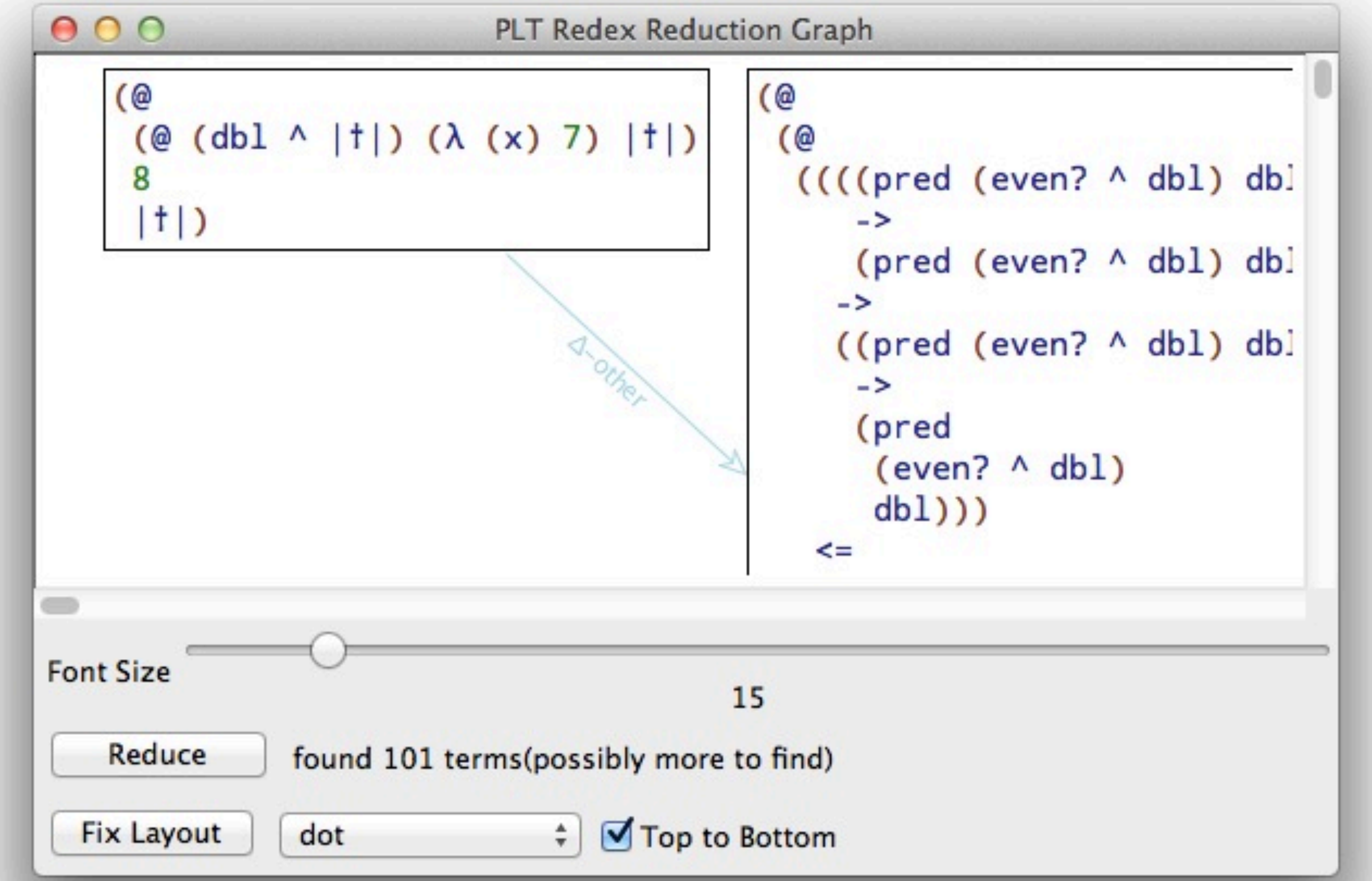
Higher-order pair contract checking

$$
\begin{array}{llll}
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(\langle C,D\rangle,\varrho,k),\sigma\rangle & \longmapsto & \langle \texttt{blame}^{f}_{h},\emptyset,\mathsf{mt},\emptyset\rangle & \text{if } \delta(\texttt{cons?},V)\ni\texttt{\#f} \\
\langle V,\rho,\mathsf{chk}^{f,g}_{h}(\langle C,D\rangle,\varrho,k),\sigma\rangle & \longmapsto & \langle U,\rho,\mathsf{op}(\texttt{car},i),\sigma[i\mapsto\mathsf{chk}^{f,g}_{h}(C,\varrho,k'),k'\mapsto\mathsf{chk\text{-}cons}^{f,g}_{h}(D,\varrho,U,\rho,k)]\rangle & \\
 & & \text{if } \delta(\texttt{cons?},V)\ni\texttt{\#t},\ \text{where } U=V/\{\lfloor\texttt{cons?}\rfloor\} & \\
\langle V,\rho,\mathsf{chk\text{-}cons}^{f,g}_{h}(C,\varrho,U,\rho',k),\sigma\rangle & \longmapsto & \langle U,\rho',\mathsf{op}(\texttt{cdr},i),\sigma[i\mapsto\mathsf{chk}^{f,g}_{h}(C,\varrho,k'),k'\mapsto\mathsf{opr}(\texttt{cons},V,\rho,k)]\rangle &
\end{array}
$$



```
#lang s-exp "verified.rkt"
(define/contract even? (nat? -> bool?)
  (λ (n) (if (zero? n) #t (odd? (sub1 n)))))
(define/contract odd? (nat? -> bool?)
  (λ (n) (if (zero? n) #f (even? (sub1 n)))))

(define/contract dbl ((even? -> even?) -> (even? -> even?))
  (λ (f) (λ (x) (f (f x)))))

((dbl (λ (x) 7)) 8)
```

```
Welcome to DrRacket, version 5.1.1.6--2011-02-02(-/f) [3m].
Language: s-exp "verified.rkt" [custom]; memory limit: 1024 MB.
'(blame |†| dbl (λ (x) 7) (pred even?) 7)
> ((dbl (λ (x) 2)) 8)
2
> ((dbl (λ (x) 2)) 5)
'(blame |†| dbl 5 (pred even?) 5)
> 
```



```
(@
 (@ (dbl ^ |†|) (λ (x) 7) |†|)
 8
 |†|)
```

```
(@
 (@
  ((((pred (even? ^ dbl) dbl
     ->
     (pred (even? ^ dbl) dbl
    ->
    ((pred (even? ^ dbl) dbl
     ->
     (pred
      (even? ^ dbl)
      dbl)))
   <=
```

Font Size 15

found 101 terms(possibly more to find)

dot

☑ Top to Bottom

```
(define-contract list/c
  (rec/c X (or/c empty? (cons/c nat? X))))
```

```
(define/contract sorted? (any? -> bool?) •)

(define/contract insert
  (nat? (and/c list/c sorted?) -> (and/c list/c sorted?))
  •)
```

```
(define/contract insertion-sort
  (list/c (and/c list/c sorted?) -> (and/c list/c sorted?))
  (λ (l acc)
    (if (empty? l)
        acc
        (insertion-sort (rest l)
                        (insert (first l) acc)))))
```

```
(define/contract sort.0
  (list/c -> (and/c list/c sorted?))
  (λ (l)
    (insertion-sort l empty)))
```

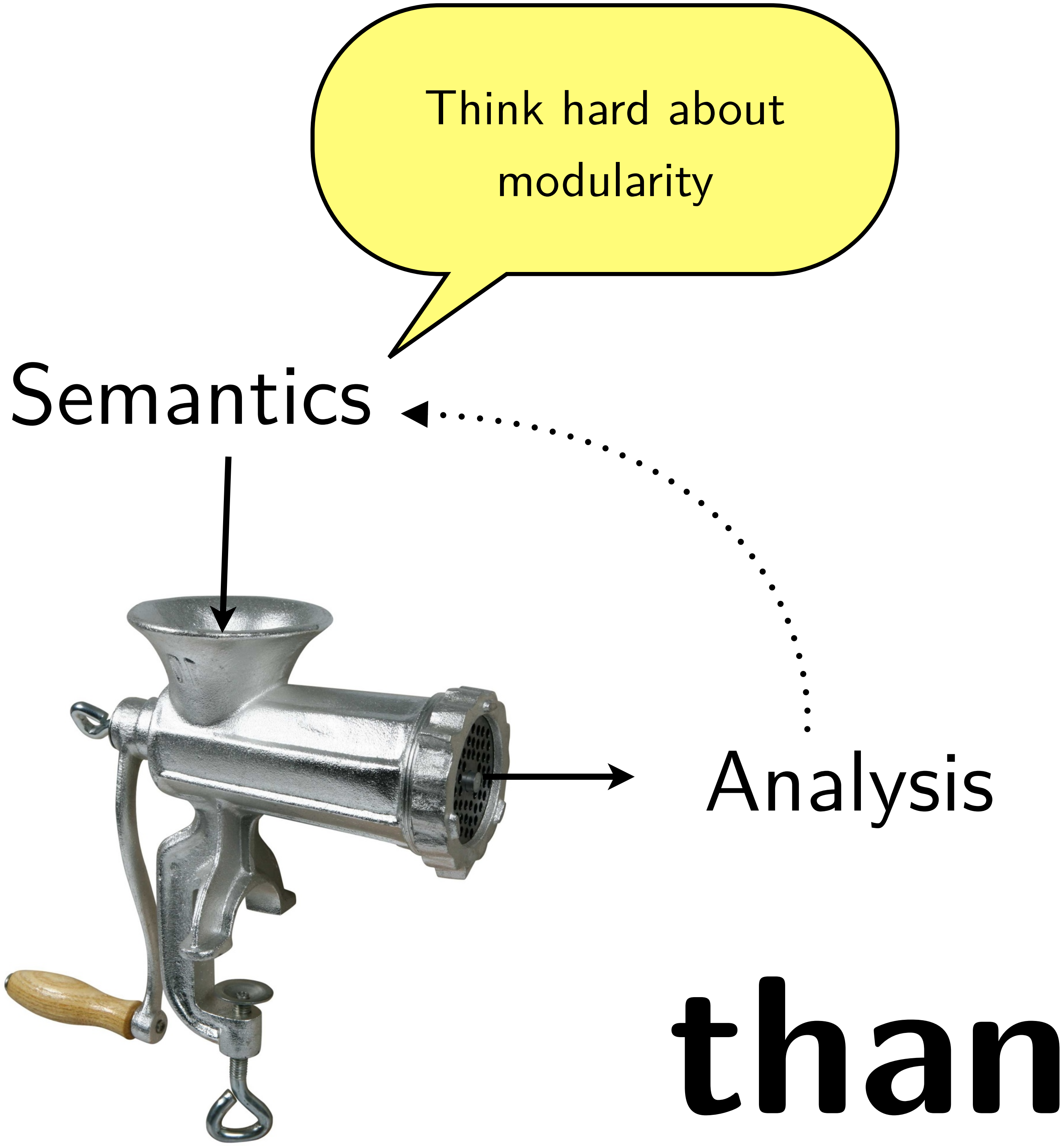
```
(define/contract l list/c •)

> (sort.0 l)
'(● sorted? list/c)
```

```
(define/contract foldl
  any?
  (λ (f b ls)
    (if (empty? ls)
        b
        (foldl f (f (first ls) b) (rest ls)))))


(define/contract sort.1
  (list/c -> (and/c list/c sorted?))
  (λ (l)
    (foldl insert empty l)))


> (sort.1 l)
'(• sorted? list/c)
```

```
(define/contract foldr
  any?
  (λ (f b ls)
    (if (empty? ls)
        b
        (f (first ls) (foldr f b (rest ls))))))

(define/contract sort.2
  (list/c -> (and/c list/c sorted?))
  (λ (l)
    (foldr insert empty l)))
```

```
> (sort.2 l)
'(• sorted? list/c)
```

The message:



Think hard about modularity

Semantics

Analysis

thank you