Scalable Abstractions for Trustworthy Software

David Van Horn

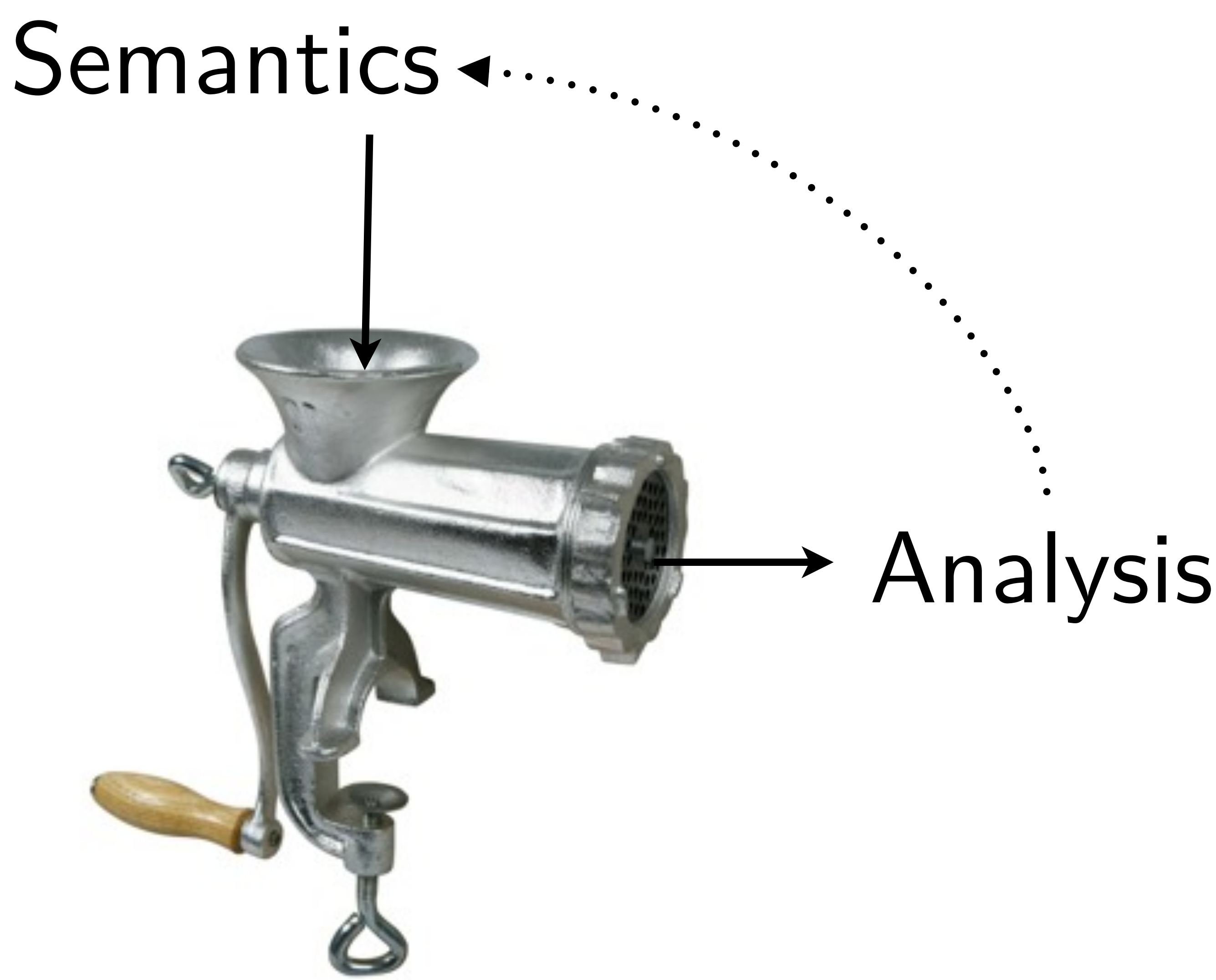
Matthew Might

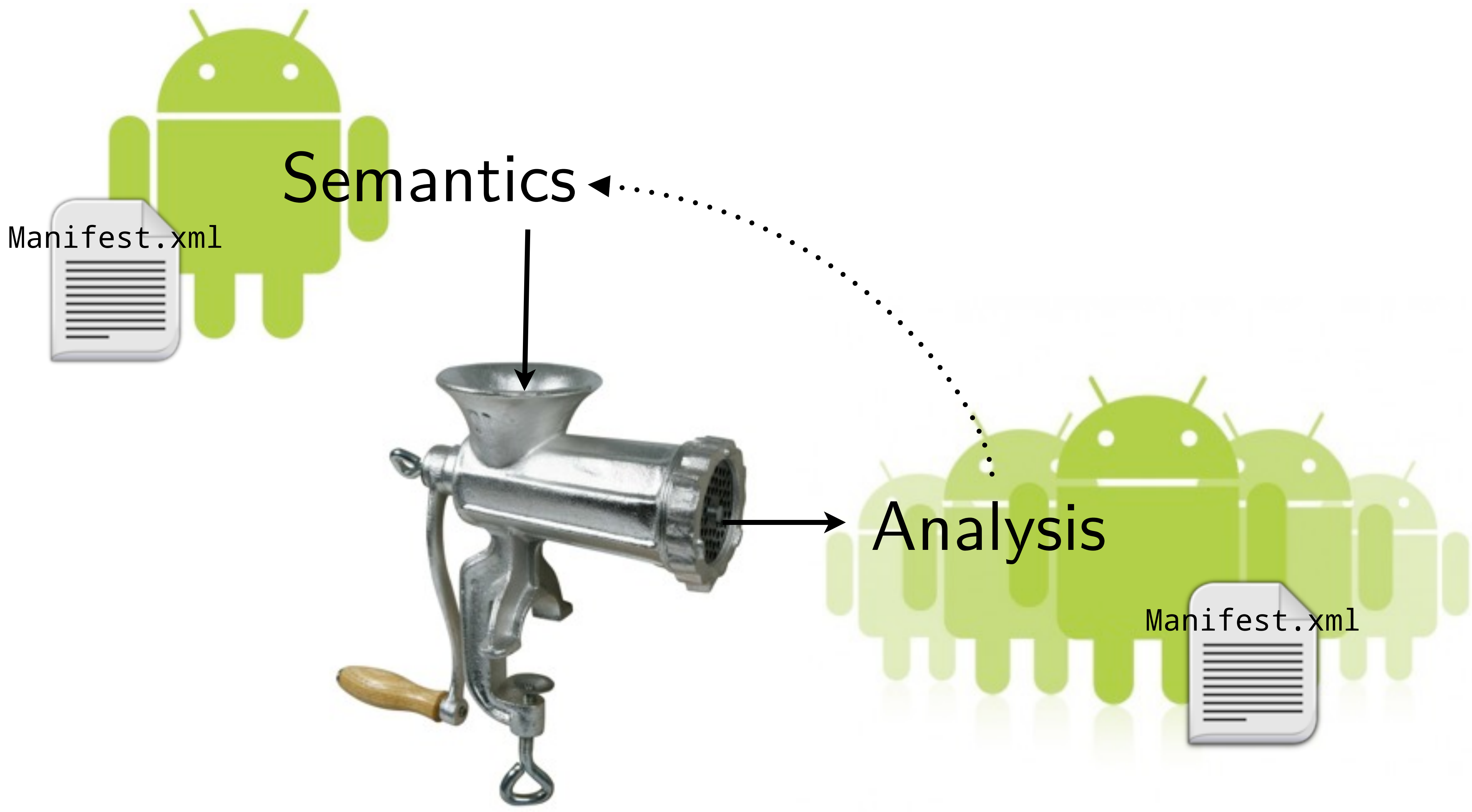
Purpose

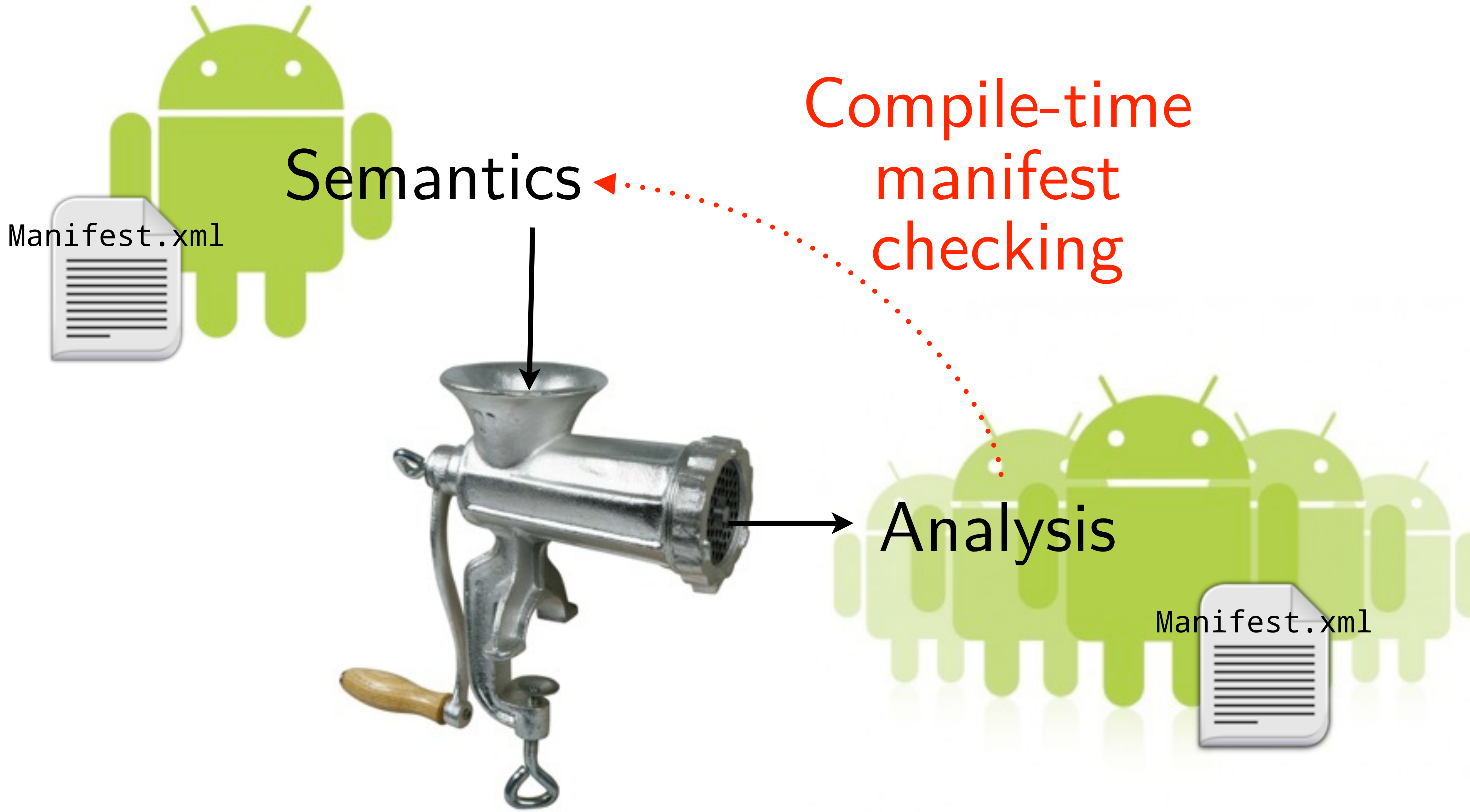
Demonstrate our approach can be *efficient*

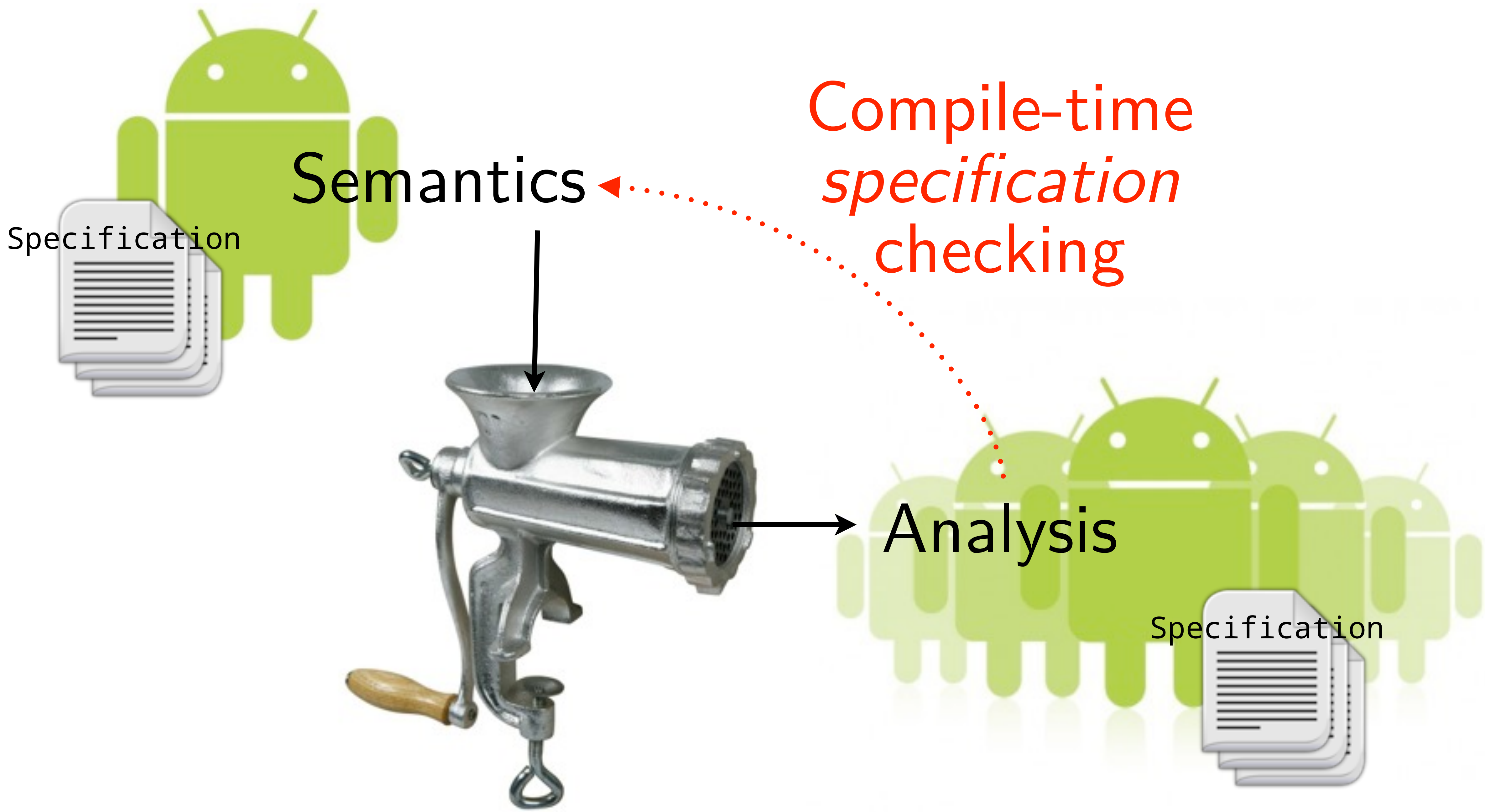
Demonstrate our approach can be *modular*

Sketch how we can verify *rich properties*

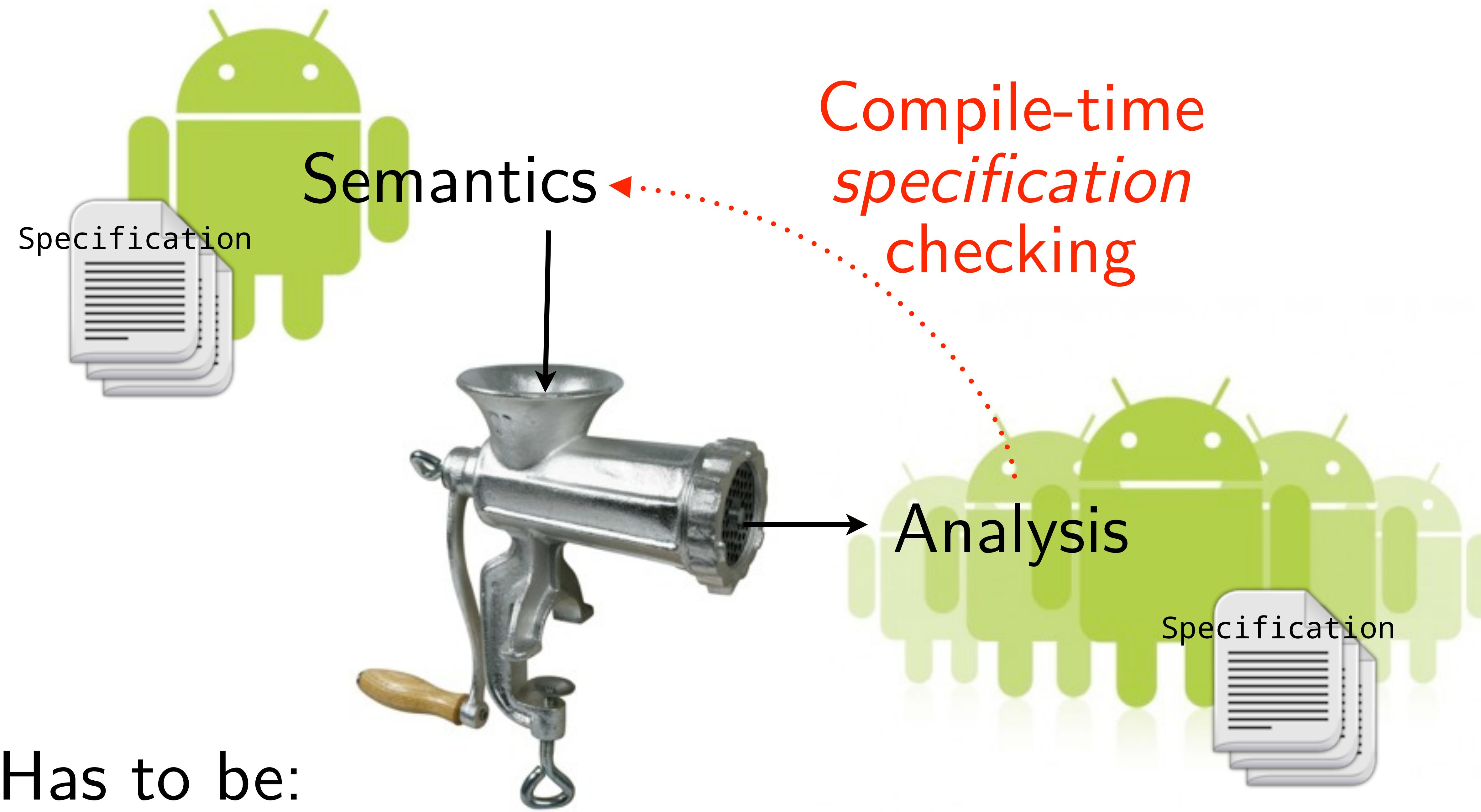

Semantics
Analysis


Semantics
Manifest.xml
Analysis
Manifest.xml


Compile-time
manifest
checking
Semantics
Manifest.xml
Analysis
Manifest.xml



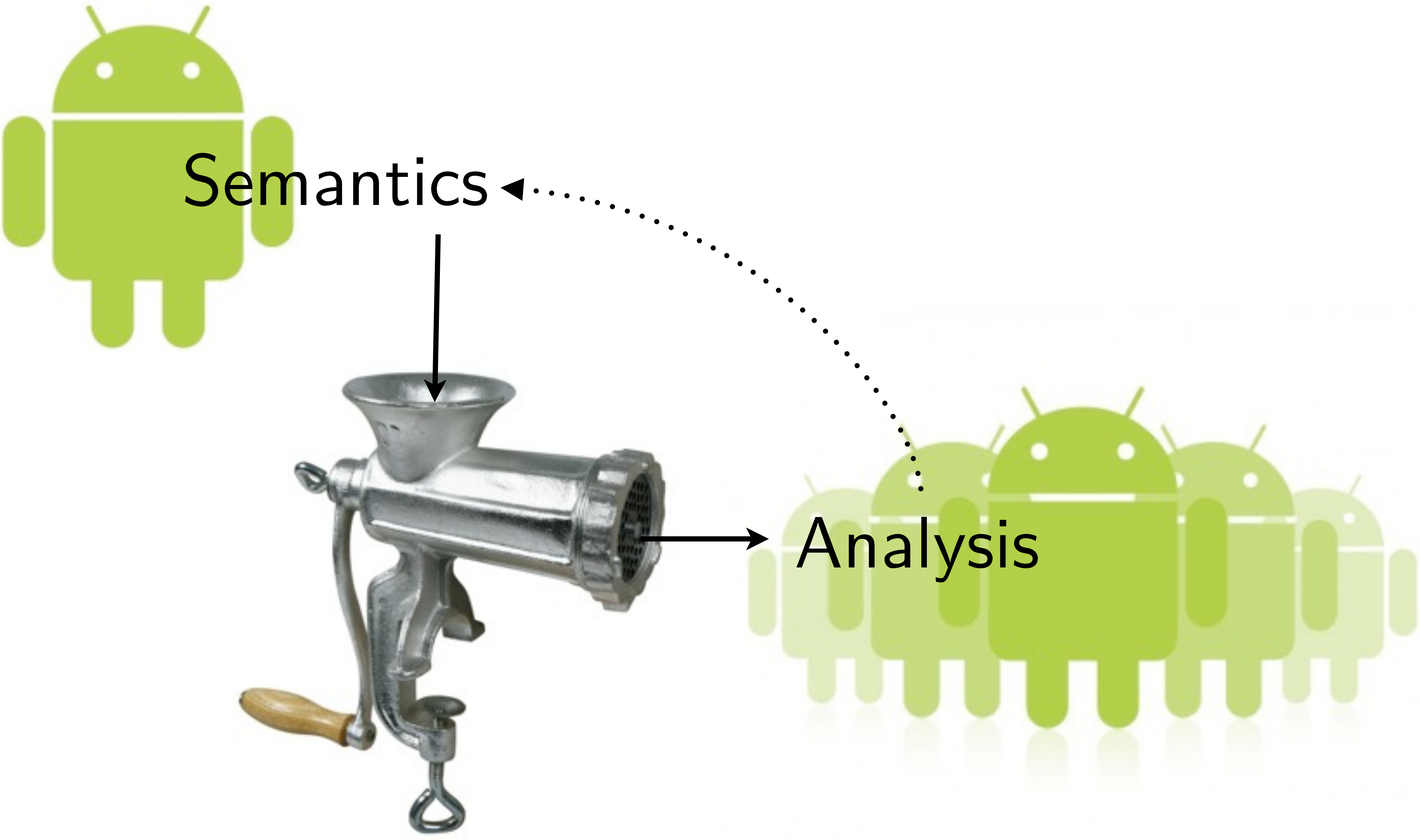
Compile-time
specification
checking
Semantics
Specification
Analysis
Specification

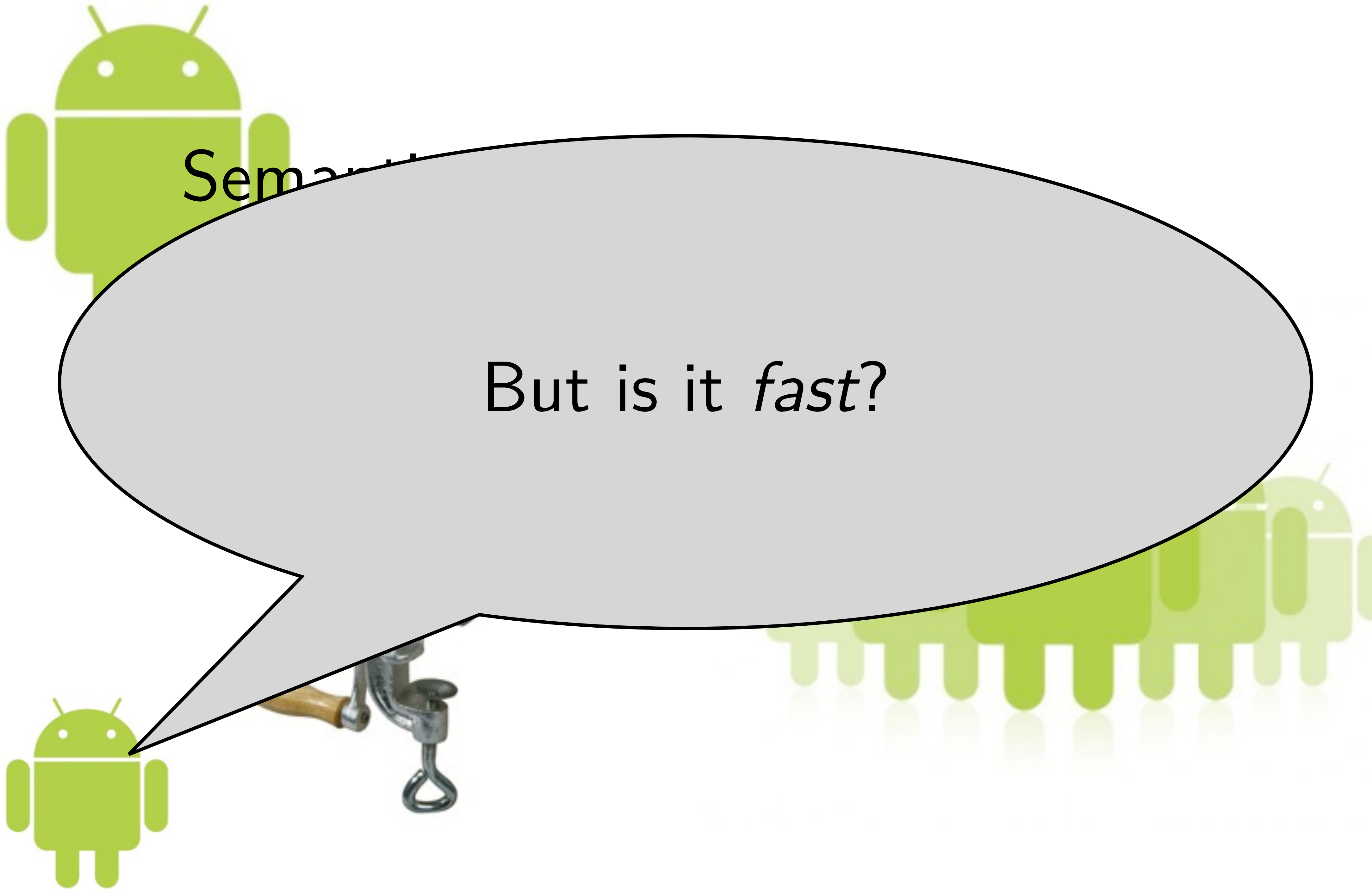


Has to be:

- precise
- fast
- scalable to rich specifications

Efficiency


Semantics
Analysis

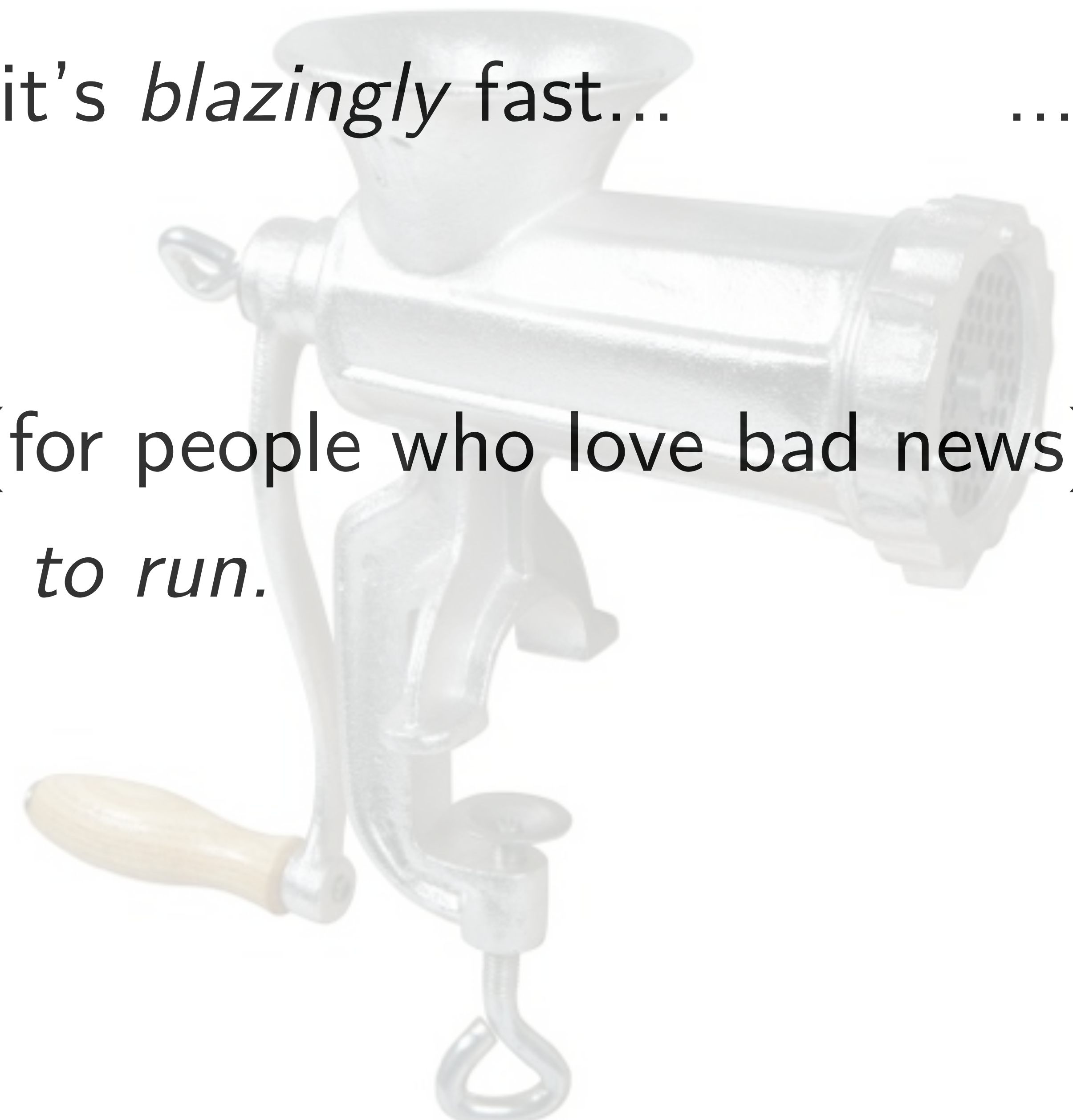

But is it *fast*?

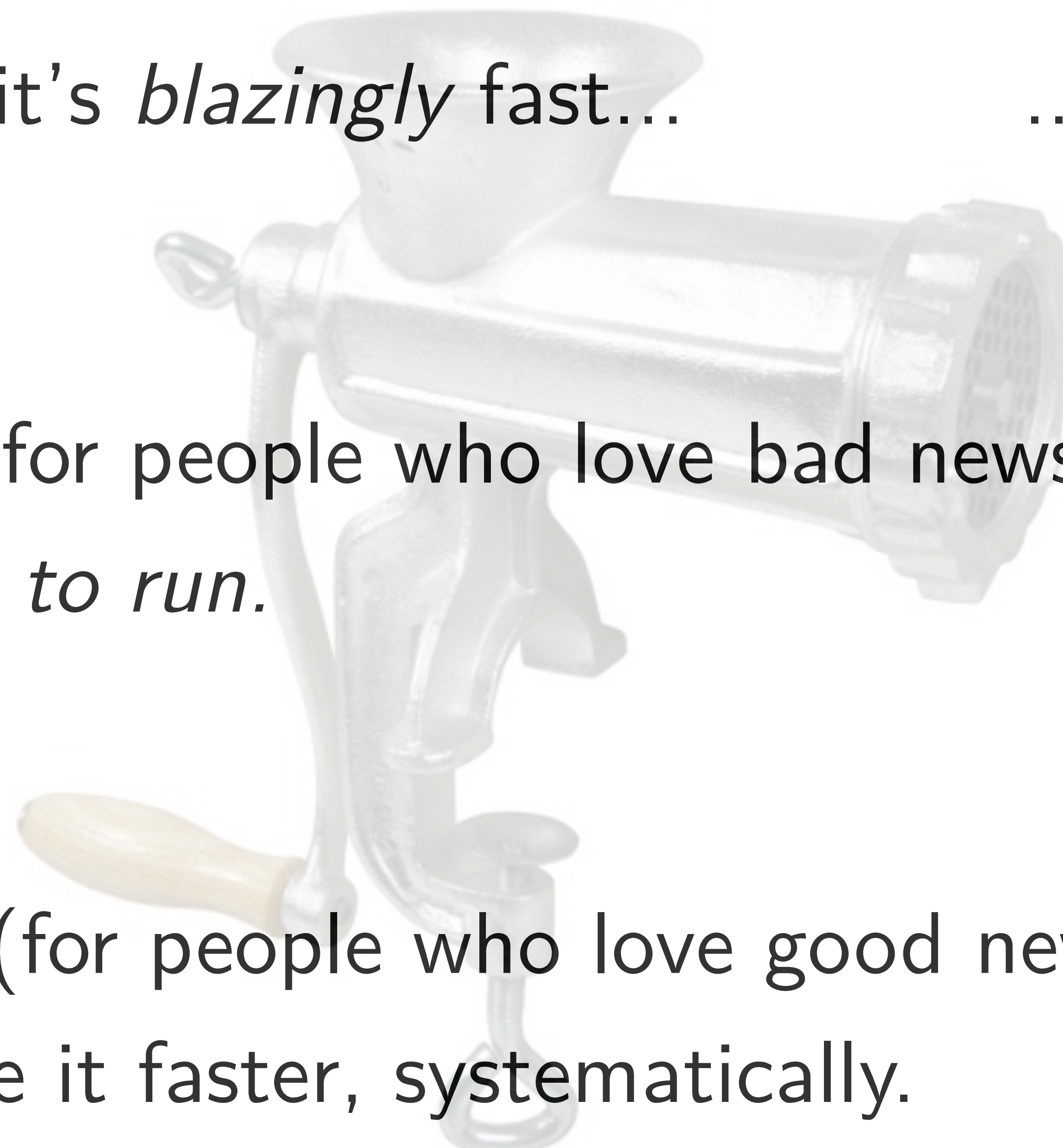
Good news: it's *blazingly* fast...

Good news: it's *blazingly* fast... ...*to implement*.

Good news: it's *blazingly* fast... ...*to implement*.

Good news (for people who love bad news):
it's *dog* slow *to run.*

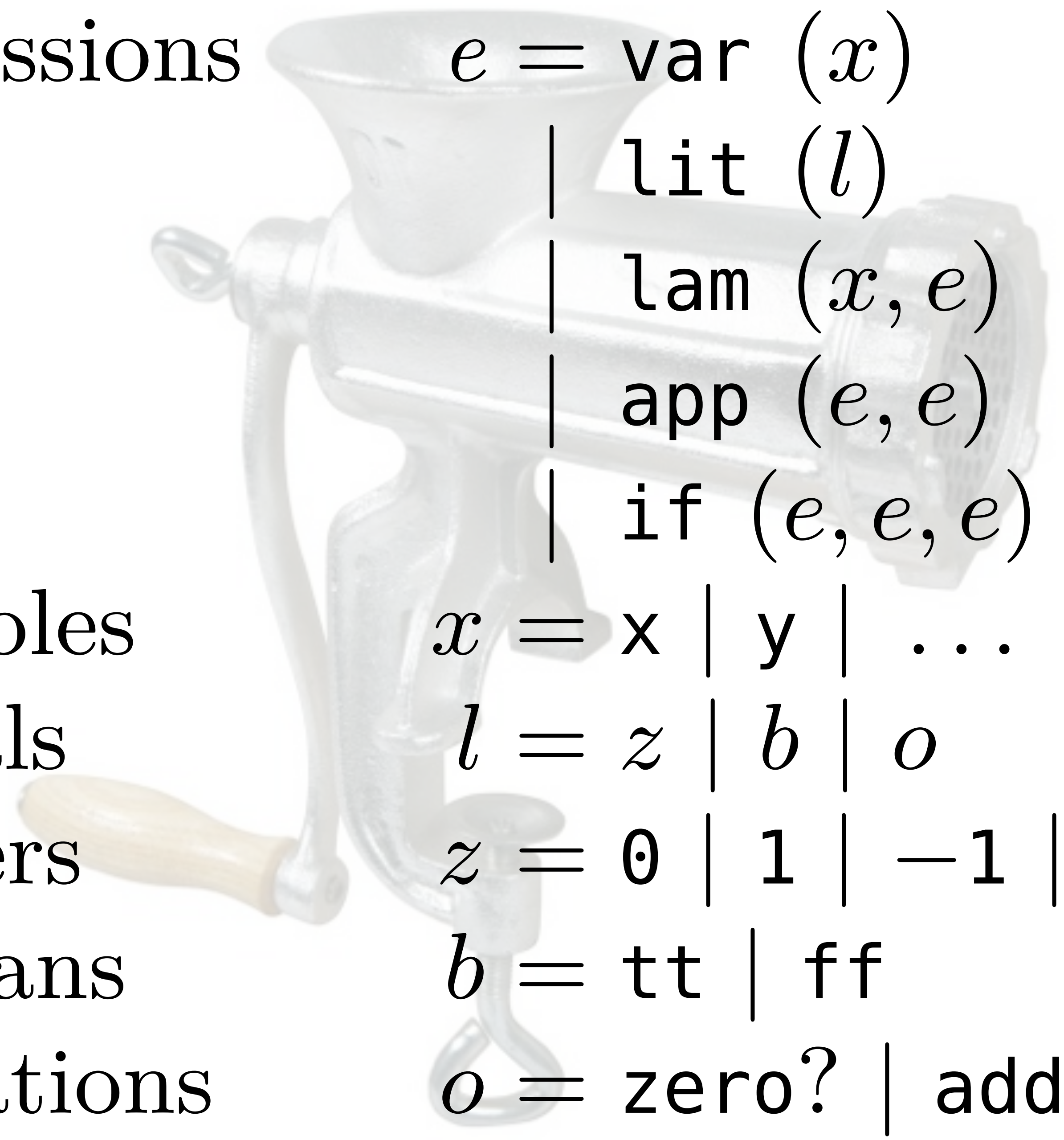




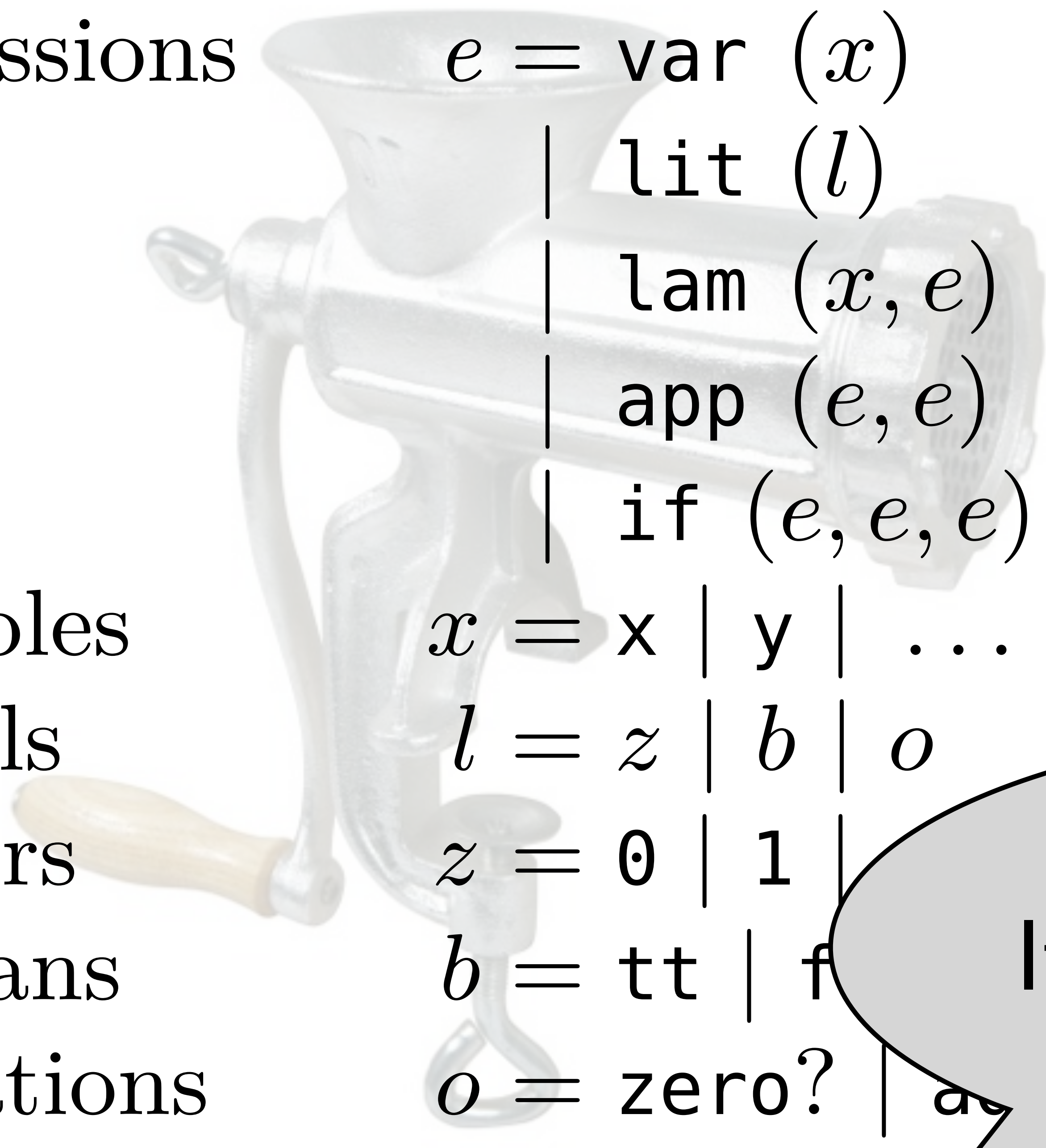
Good news: it's *blazingly* fast... ...*to implement*.

Good news (for people who love bad news):
it's *dog* slow *to run.*

Good news (for people who love good news):
we can make it faster, systematically.

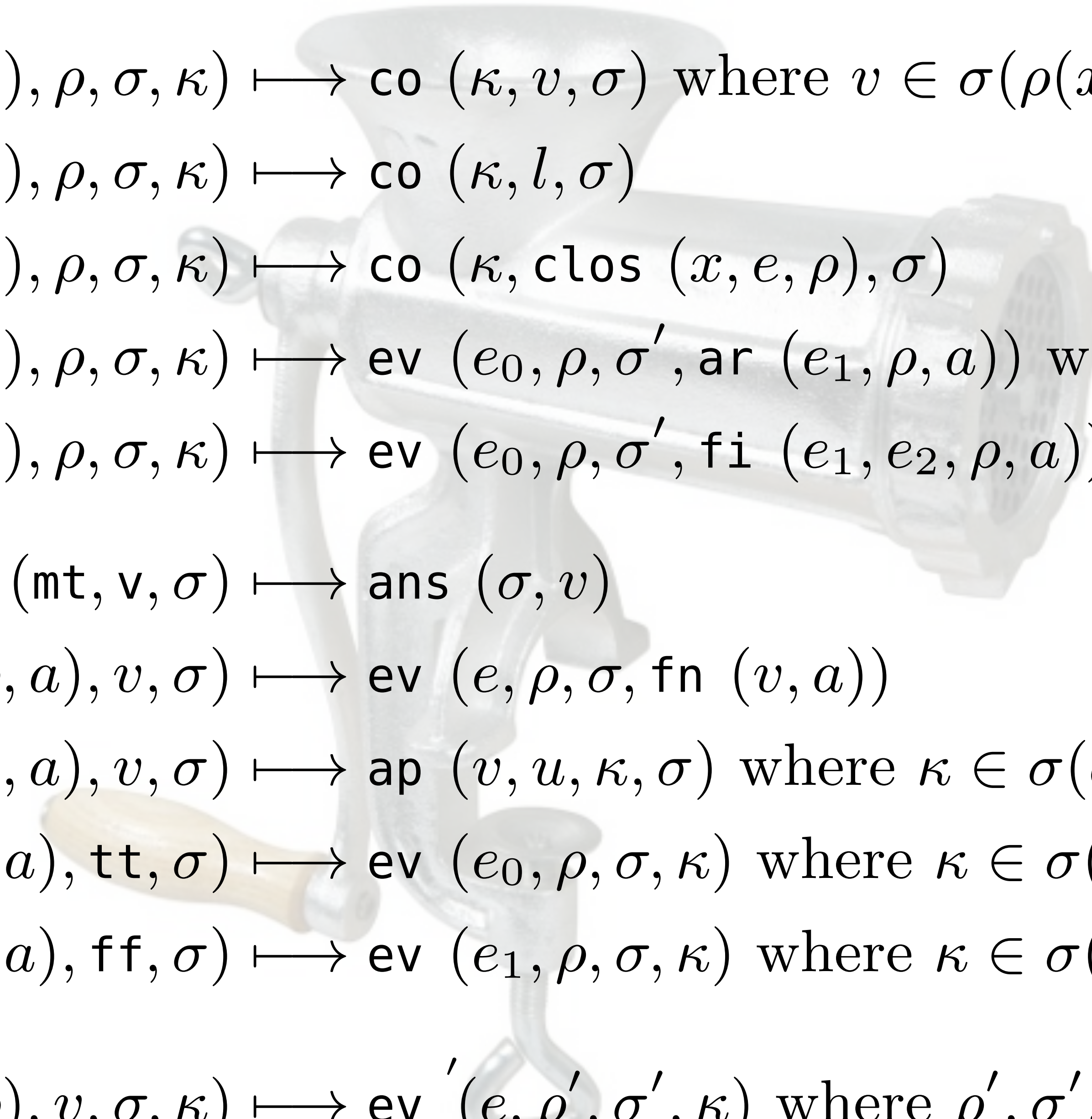


$$
\begin{array}{lrl}
\text{Expressions} & e = & \texttt{var}\ (x) \\
 & \mid & \texttt{lit}\ (l) \\
 & \mid & \texttt{lam}\ (x, e) \\
 & \mid & \texttt{app}\ (e, e) \\
 & \mid & \texttt{if}\ (e, e, e) \\
\text{Variables} & x = & \texttt{x} \mid \texttt{y} \mid \texttt{...} \\
\text{Literals} & l = & z \mid b \mid o \\
\text{Integers} & z = & \texttt{0} \mid \texttt{1} \mid -\texttt{1} \mid \texttt{...} \\
\text{Booleans} & b = & \texttt{tt} \mid \texttt{ff} \\
\text{Operations} & o = & \texttt{zero}? \mid \texttt{add1} \mid \texttt{sub1} \mid \texttt{...}
\end{array}
$$



Expressions	$e = \texttt{var}\ (x)$
	$\mid \texttt{lit}\ (l)$
	$\mid \texttt{lam}\ (x, e)$
	$\mid \texttt{app}\ (e, e)$
	$\mid \texttt{if}\ (e, e, e)$
Variables	$x = \texttt{x} \mid \texttt{y} \mid \ldots$
Literals	$l = z \mid b \mid o$
Integers	$z = \texttt{0} \mid \texttt{1} \mid$
Booleans	$b = \texttt{tt} \mid \texttt{f}$
Operations	$o = \texttt{zero?} \mid \texttt{a}$

It's just Core Java



$$eval(e) = \{\varsigma \mid \texttt{ev}\ (e, \varnothing, \varnothing, \texttt{mt}) \longmapsto\!\!\!\!\twoheadrightarrow \varsigma\} \text{ where}$$

$$\texttt{ev}\ (\texttt{var}\ (x), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x))$$

$$\texttt{ev}\ (\texttt{lit}\ (l), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, l, \sigma)$$

$$\texttt{ev}\ (\texttt{lam}\ (x, e), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, \texttt{clos}\ (x, e, \rho), \sigma)$$

$$\texttt{ev}\ (\texttt{app}\ (e_0, e_1), \rho, \sigma, \kappa) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{ar}\ (e_1, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)$$

$$\texttt{ev}\ (\texttt{if}\ (e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{fi}\ (e_1, e_2, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)$$

$$\texttt{co}\ (\texttt{mt}, \texttt{v}, \sigma) \longmapsto \texttt{ans}\ (\sigma, v)$$

$$\texttt{co}\ (\texttt{ar}\ (e, \rho, a), v, \sigma) \longmapsto \texttt{ev}\ (e, \rho, \sigma, \texttt{fn}\ (v, a))$$

$$\texttt{co}\ (\texttt{fn}\ (u, a), v, \sigma) \longmapsto \texttt{ap}\ (v, u, \kappa, \sigma) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{tt}, \sigma) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{ff}, \sigma) \longmapsto \texttt{ev}\ (e_1, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{ap}\ (\texttt{clos}\ (x, e, \rho), v, \sigma, \kappa) \longmapsto \texttt{ev}\ {}'(e, \rho', \sigma', \kappa) \text{ where } \rho', \sigma', {}' = bind\ (\sigma, x, v)$$

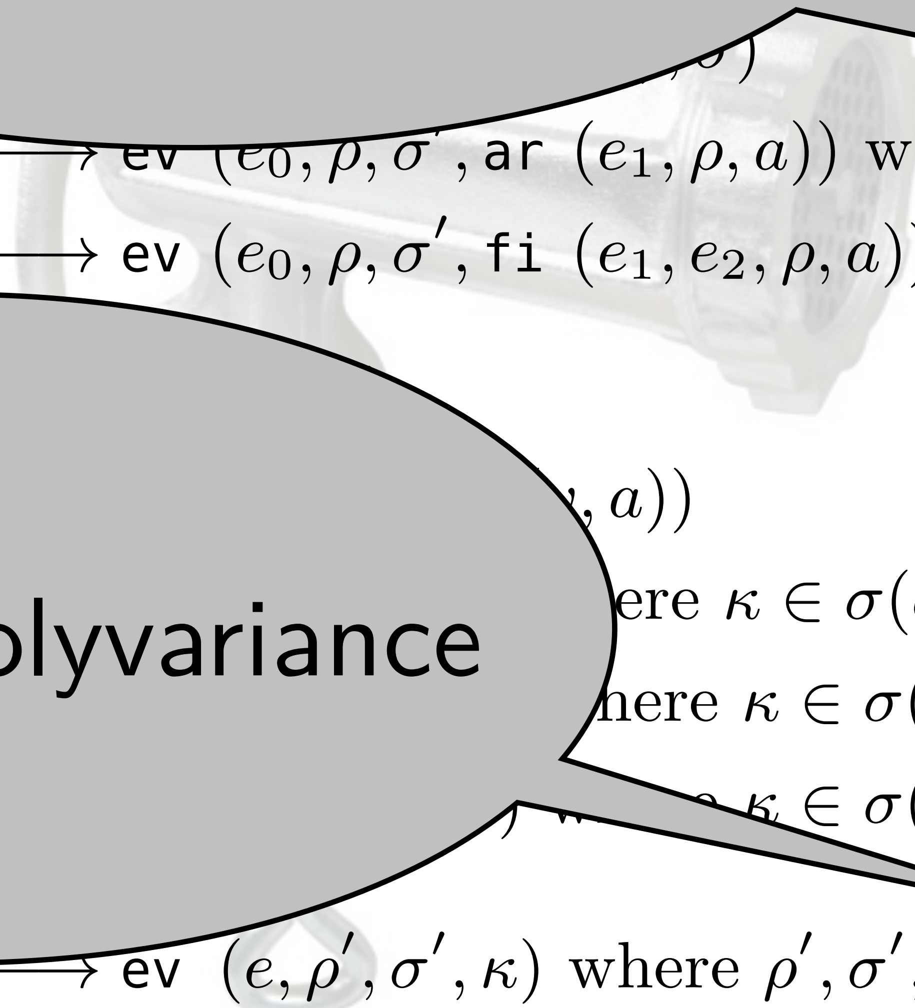
$$\texttt{ap}\ (o, v, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, v', \sigma) \text{ where } \kappa \in \sigma(a) \text{ and } v' \in \Delta(o, v)$$

Arbiter of context sensitivity

$\texttt{ev (lam } (x, \ldots$

$\texttt{ev (app } (e_0, e_1), \rho, \sigma, \kappa) \longmapsto \texttt{ev } (e_0, \rho, \sigma', \texttt{ar } (e_1, \rho, a))$ where $a, \sigma' = push\ (\sigma, \kappa)$

$\texttt{ev (if } (e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \texttt{ev } (e_0, \rho, \sigma', \texttt{fi } (e_1, e_2, \rho, a))$ where $a, \sigma' = push\ (\sigma, \kappa)$



Arbiter of polyvariance

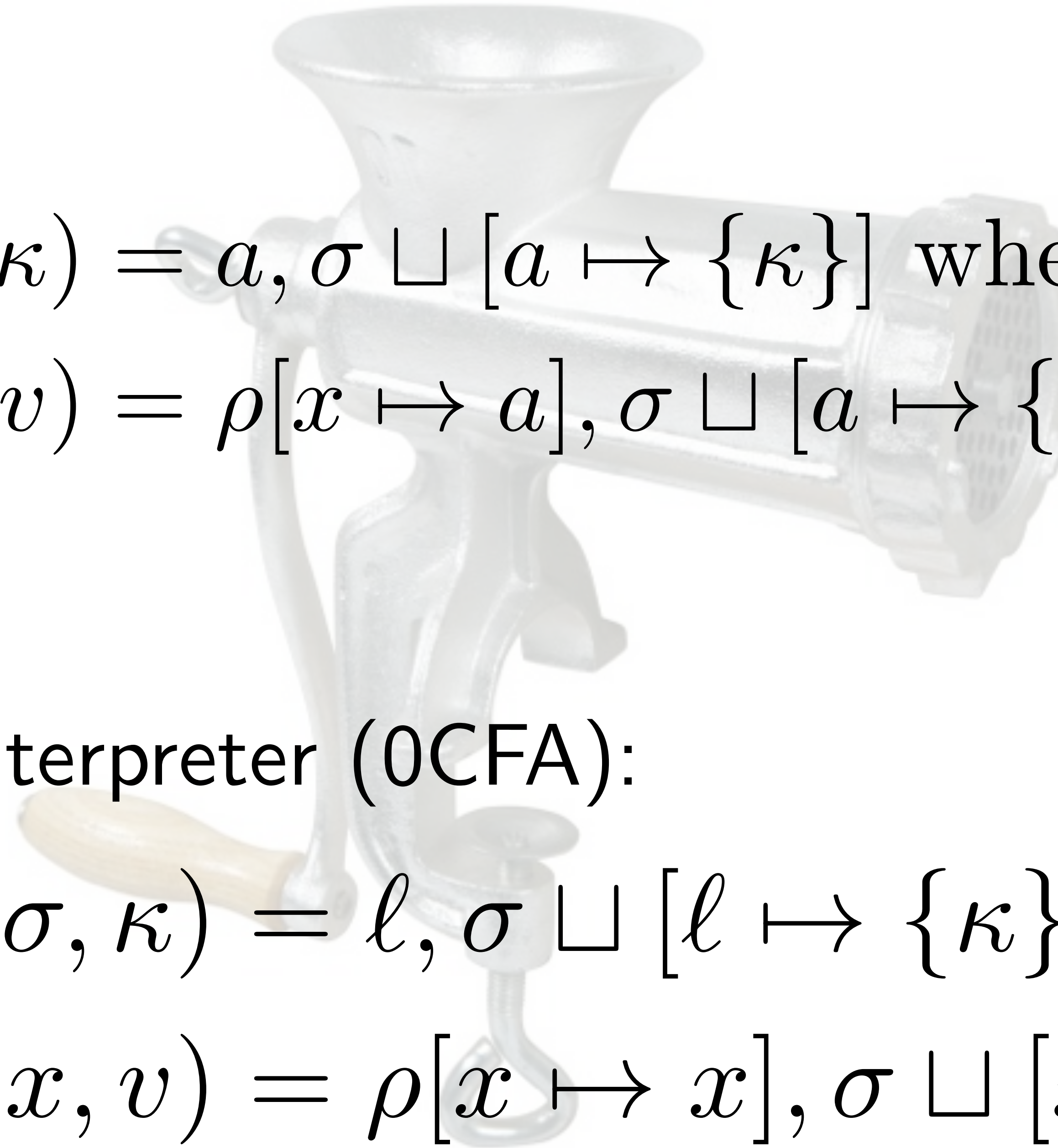
$\ldots$ where $\kappa \in \sigma(a)$

$\texttt{co} \ldots$ where $\kappa \in \sigma(a)$

$\texttt{co (f} \ldots$ where $\kappa \in \sigma(a)$

$\texttt{ap (clos } (x, e, \rho), v, \sigma, \kappa) \longmapsto \texttt{ev } (e, \rho', \sigma', \kappa)$ where $\rho', \sigma', ' = bind\ (\sigma, x, v)$

$\texttt{ap } (o, v, \sigma, \kappa) \longmapsto \texttt{co } (\kappa, v', \sigma)$ where $\kappa \in \sigma(a)$ and $v' \in \Delta(o, v)$



Interpreter:

$$push(\ell, \sigma, \kappa) = a, \sigma \sqcup [a \mapsto \{\kappa\}] \text{ where } a \notin \sigma$$

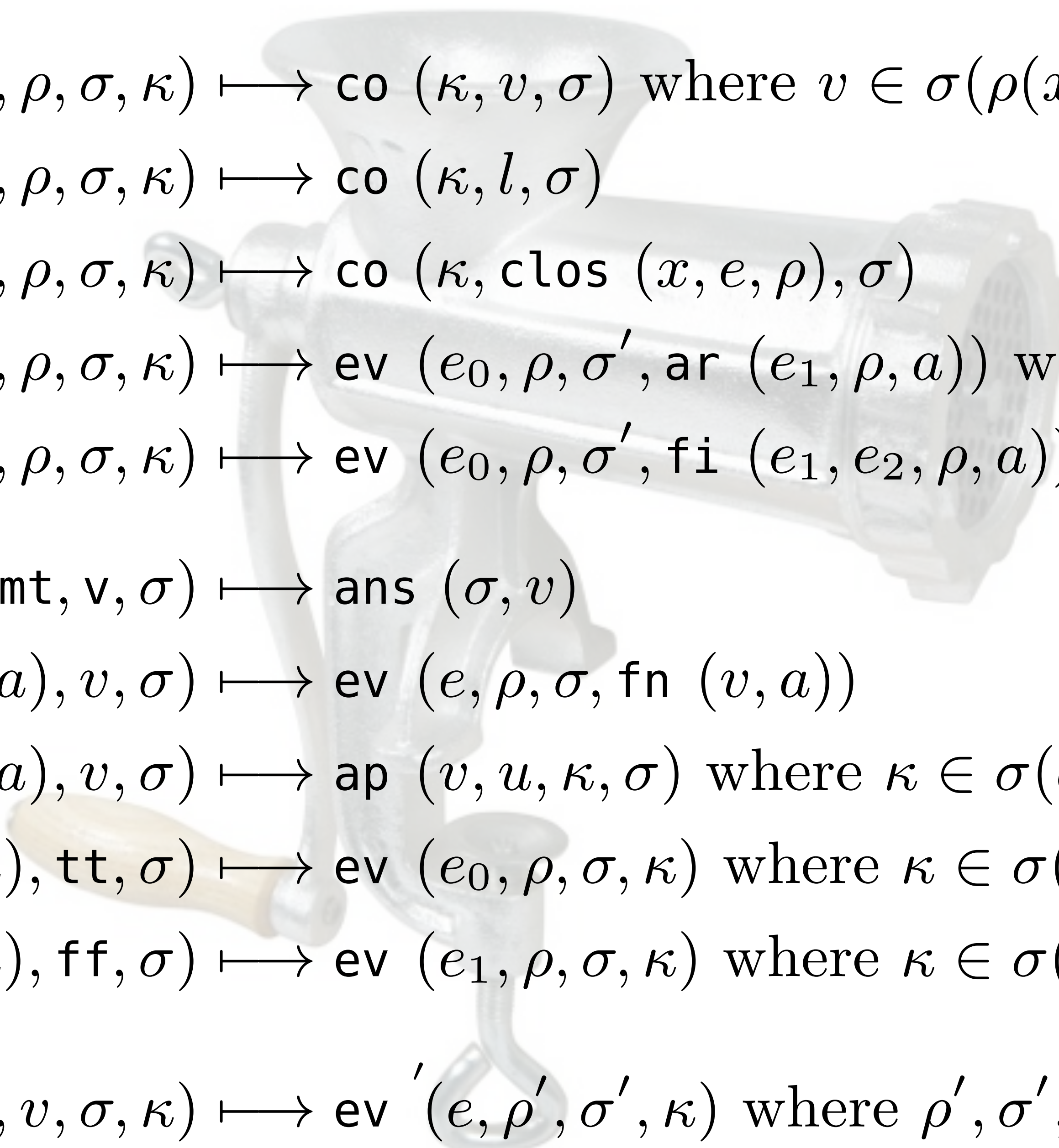
$$bind(\sigma, x, v) = \rho[x \mapsto a], \sigma \sqcup [a \mapsto \{v\}] \text{ where } a \notin \sigma$$

Abstract interpreter (0CFA):

$$push(\ell, \sigma, \kappa) = \ell, \sigma \sqcup [\ell \mapsto \{\kappa\}]$$

$$bind(\sigma, x, v) = \rho[x \mapsto x], \sigma \sqcup [x \mapsto \{v\}]$$

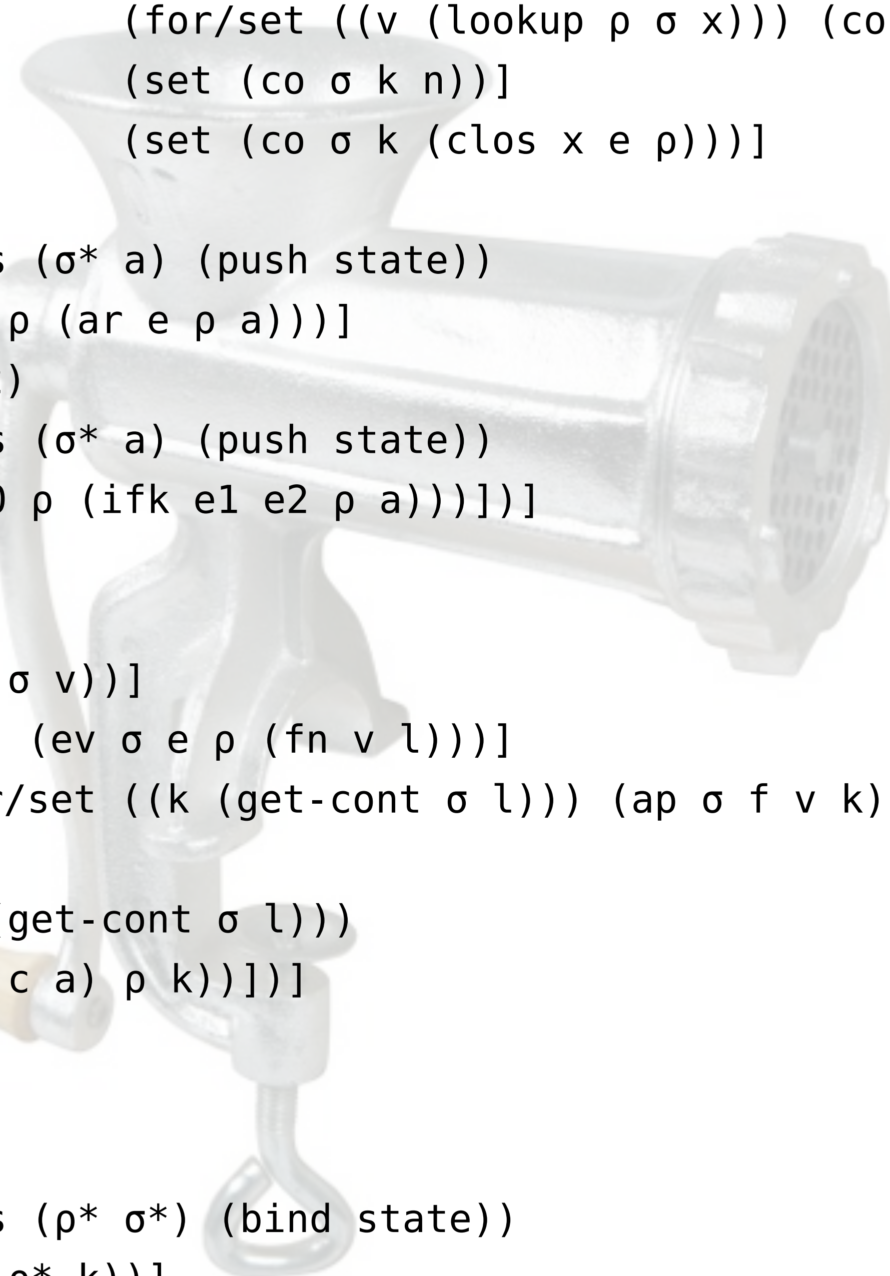
$$eval(e) = \{\varsigma \mid \texttt{ev}\ (e, \varnothing, \varnothing, \texttt{mt}) \longmapsto\!\!\!\!\!\!\!\twoheadrightarrow \varsigma\} \text{ where}$$



$$\begin{aligned}
\texttt{ev}\ (\texttt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x)) \\
\texttt{ev}\ (\texttt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, l, \sigma) \\
\texttt{ev}\ (\texttt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, \texttt{clos}\ (x, e, \rho), \sigma) \\
\texttt{ev}\ (\texttt{app}\ (e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{ar}\ (e_1, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa) \\
\texttt{ev}\ (\texttt{if}\ (e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{fi}\ (e_1, e_2, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)
\end{aligned}$$

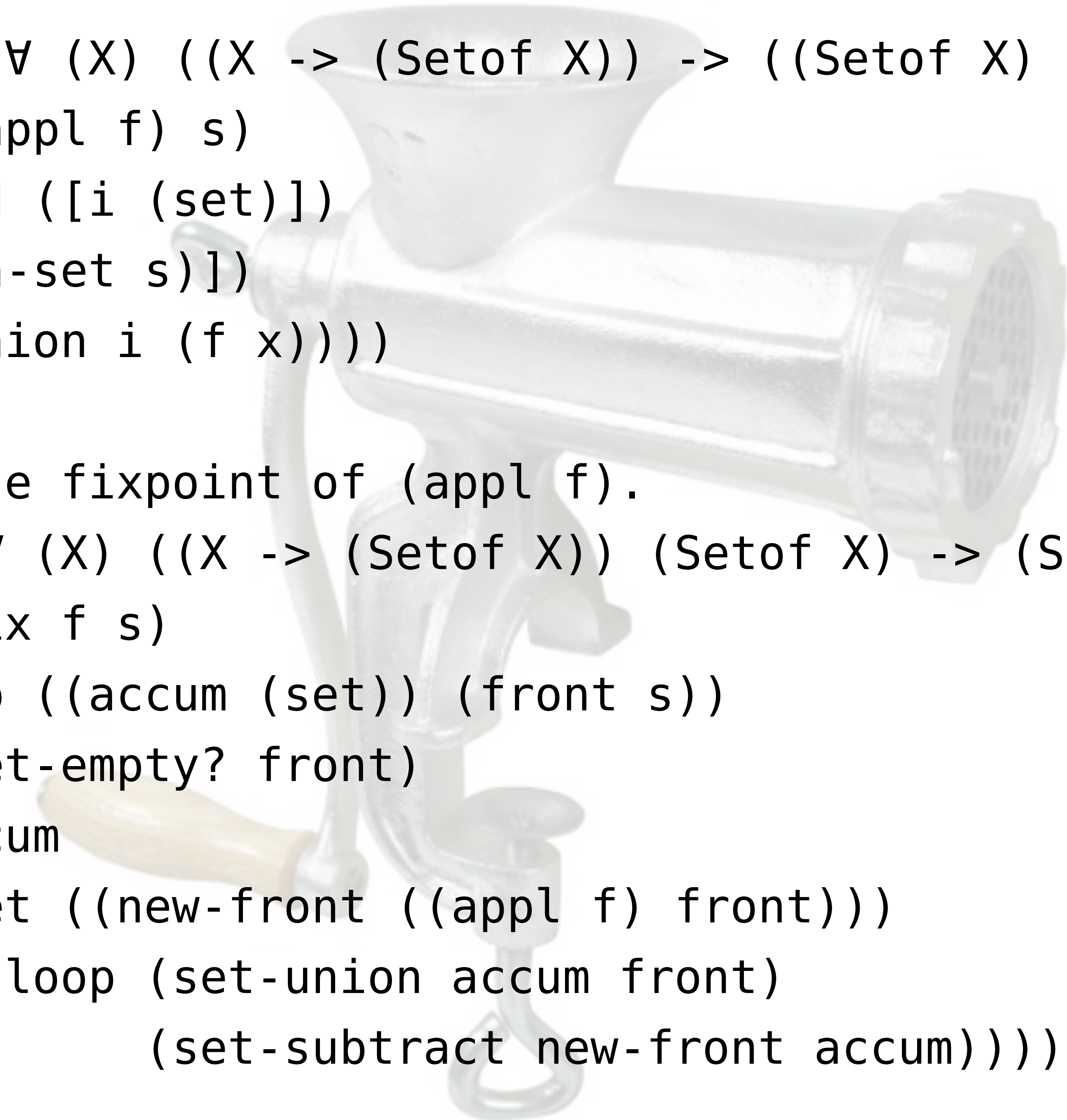
$$\begin{aligned}
\texttt{co}\ (\texttt{mt}, \texttt{v}, \sigma) &\longmapsto \texttt{ans}\ (\sigma, v) \\
\texttt{co}\ (\texttt{ar}\ (e, \rho, a), v, \sigma) &\longmapsto \texttt{ev}\ (e, \rho, \sigma, \texttt{fn}\ (v, a)) \\
\texttt{co}\ (\texttt{fn}\ (u, a), v, \sigma) &\longmapsto \texttt{ap}\ (v, u, \kappa, \sigma) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{tt}, \sigma) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{ff}, \sigma) &\longmapsto \texttt{ev}\ (e_1, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)
\end{aligned}$$

$$\begin{aligned}
\texttt{ap}\ (\texttt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \texttt{ev}\ {}'(e, \rho', \sigma', \kappa) \text{ where } \rho', \sigma', {}' = bind\ (\sigma, x, v) \\
\texttt{ap}\ (o, v, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v', \sigma) \text{ where } \kappa \in \sigma(a) \text{ and } v' \in \Delta(o, v)
\end{aligned}$$

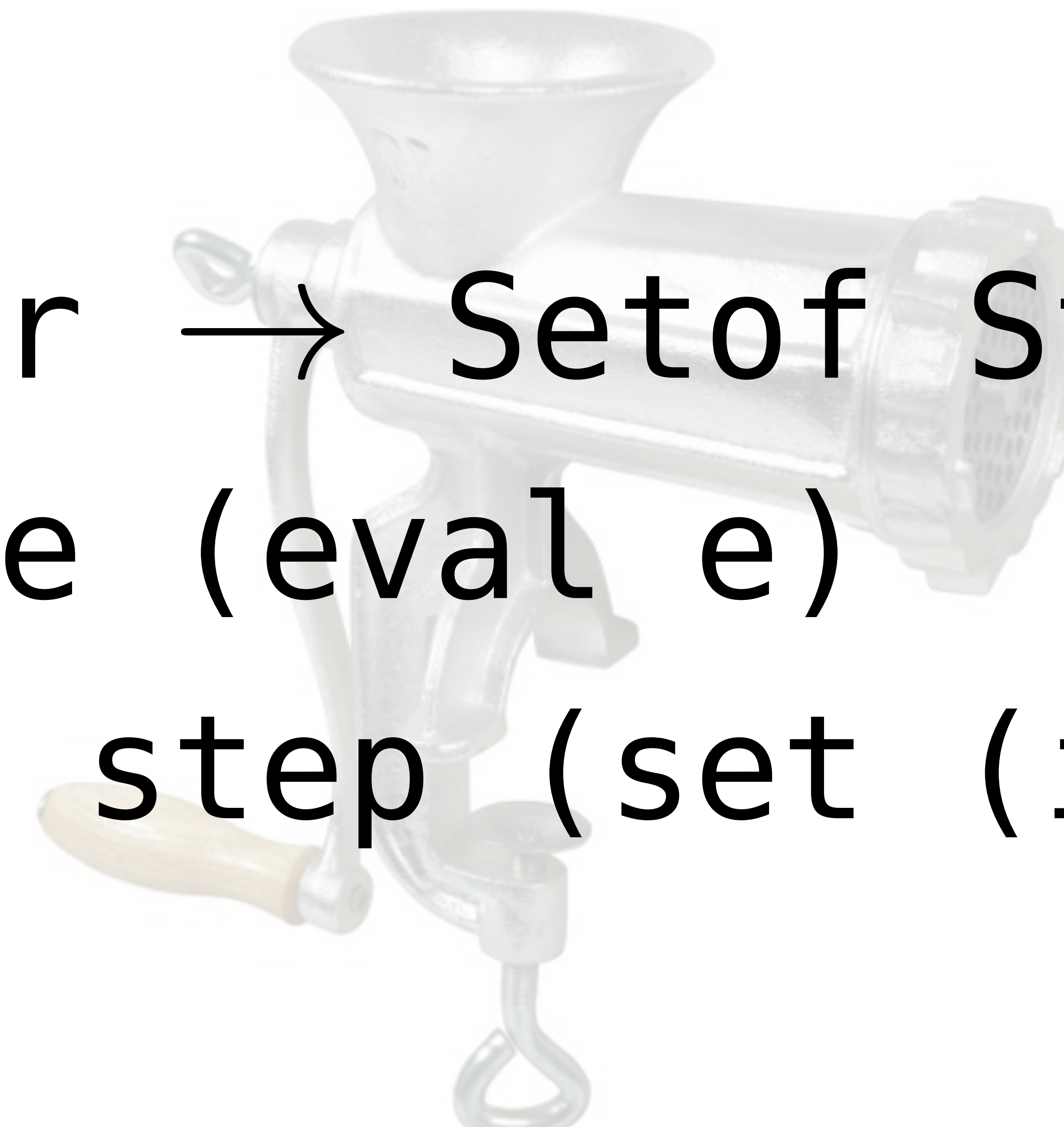


```
;; State → Setof State
(define (step state)
  (match state
    [(ev σ e ρ k)
     (match e
       [(varℓ x)           (for/set ((v (lookup ρ σ x))) (co σ k v))]
       [(litℓ l)           (set (co σ k n))]
       [(lamℓ x e)         (set (co σ k (clos x e ρ)))]
       [(appℓ f e)
        (define-values (σ* a) (push state))
        (set (ev σ* f ρ (ar e ρ a)))]
       [(ifeℓ e0 e1 e2)
        (define-values (σ* a) (push state))
        (set (ev σ* e0 ρ (ifk e1 e2 ρ a)))])]
    [(co σ k v)
     (match k
       ['mt (set (ans σ v))]
       [(arℓ e ρ) (set (ev σ e ρ (fn v l)))]
       [(fnℓ f)   (for/set ((k (get-cont σ l))) (ap σ f v k))]
       [(fiℓ c a ρ)
        (for/set ((k (get-cont σ l)))
          (ev σ (if v c a) ρ k))])]
    [(ap σ fun a k)
     (match fun
       [(clos l x e ρ)
        (define-values (ρ* σ*) (bind state))
        (set (ev σ* e ρ* k))]
       [(? op? o)
        (for*/set ((k (get-cont σ l))
                   (v (Δ o (list v))))
          (co σ k v))
       [_ (set)])])))
```

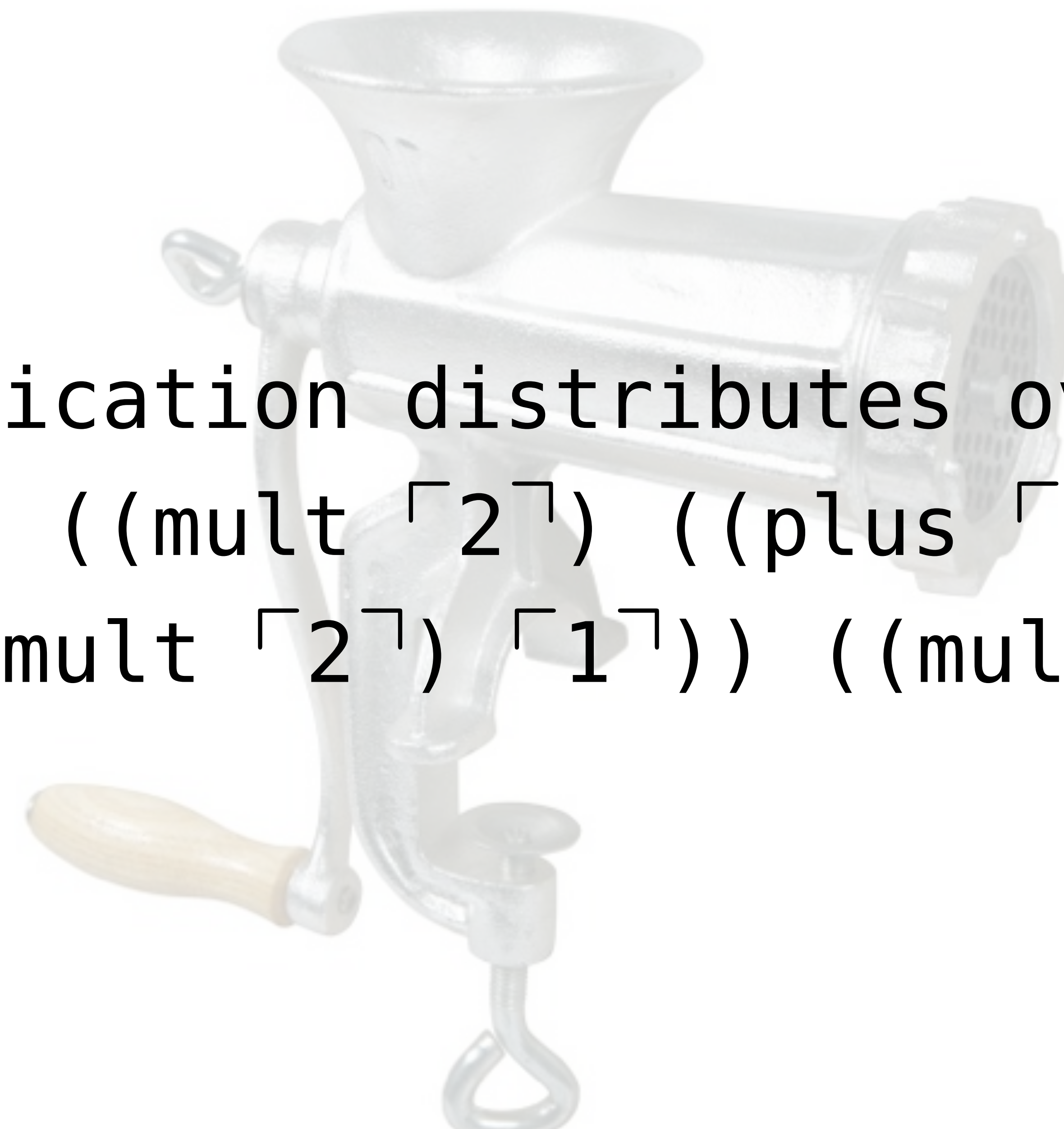
Generic fixpoint calculator



```
;; appl : (∀ (X) ((X -> (Setof X)) -> ((Setof X) -> (Setof X))))
(define ((appl f) s)
  (for/fold ([i (set)])
    ([x (in-set s)])
    (set-union i (f x))))

;; Calculate fixpoint of (appl f).
;; fix : (∀ (X) ((X -> (Setof X)) (Setof X) -> (Setof X)))
(define (fix f s)
  (let loop ((accum (set)) (front s))
    (if (set-empty? front)
        accum
        (let ((new-front ((appl f) front)))
          (loop (set-union accum front)
                (set-subtract new-front accum))))))
```

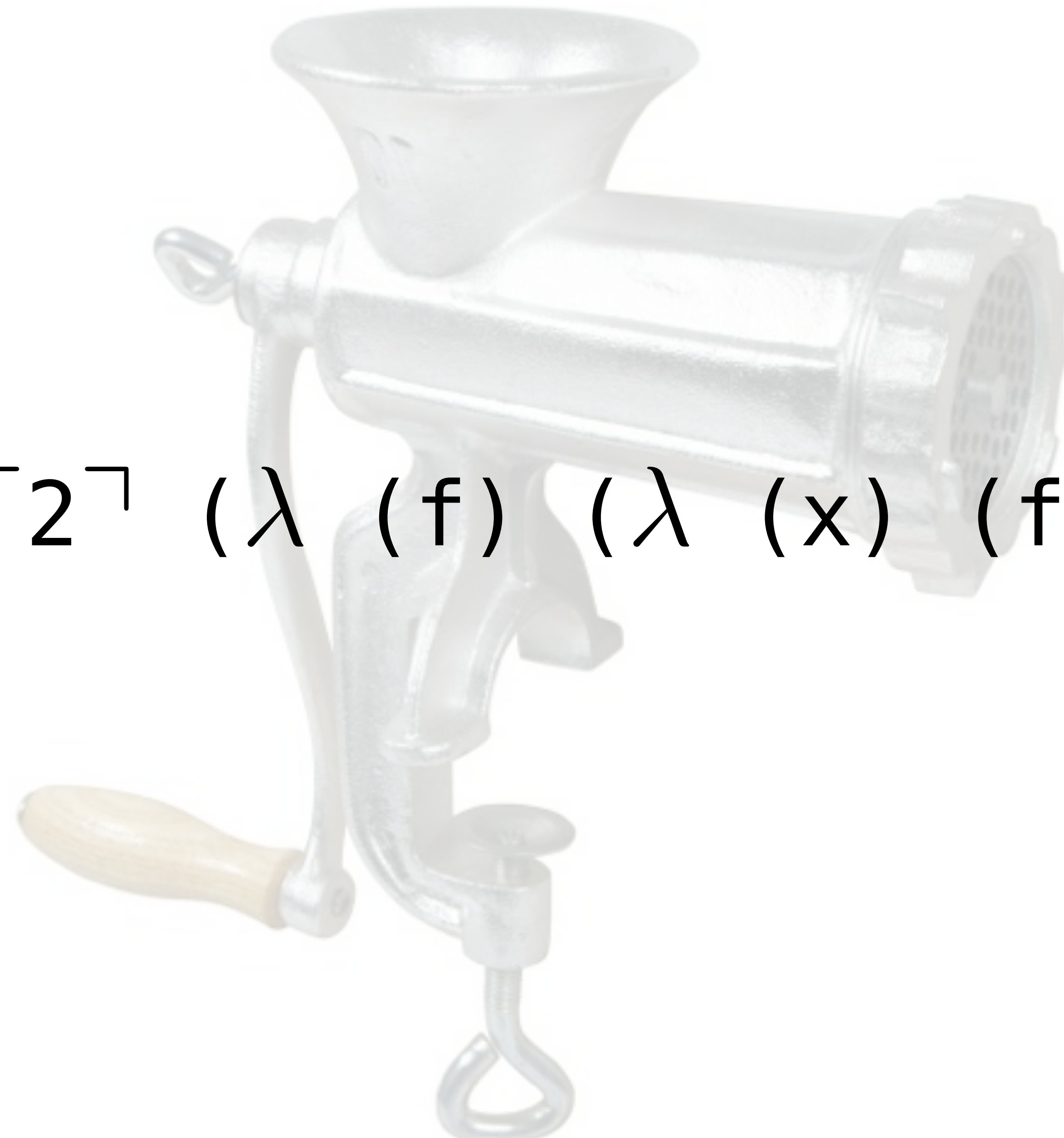


```
;; Expr → Setof State
(define (eval e)
  (fix step (set (inj e))))
```



```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

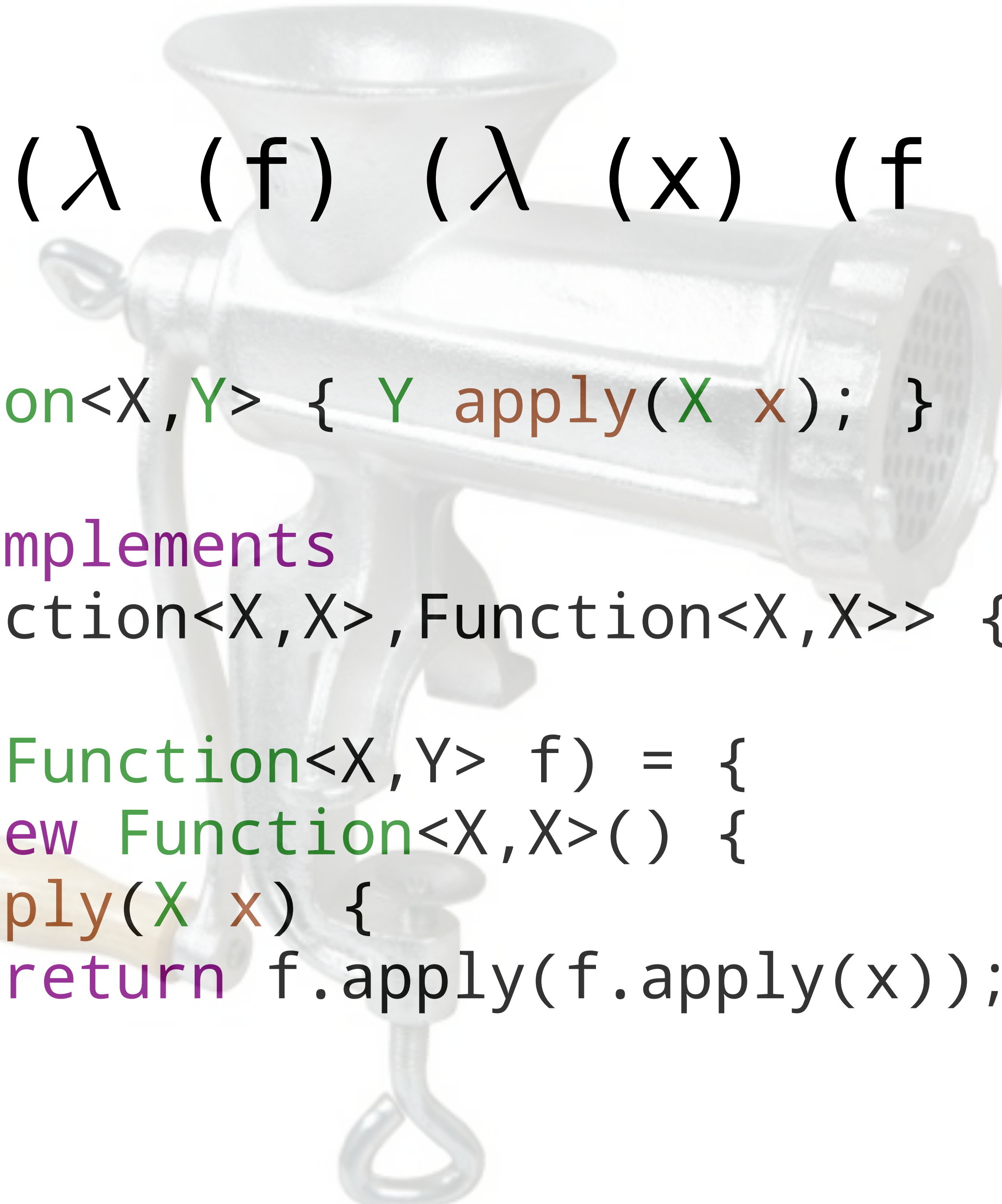
```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```



```
(define ⌜2⌝ (λ (f) (λ (x) (f (f x)))))
```

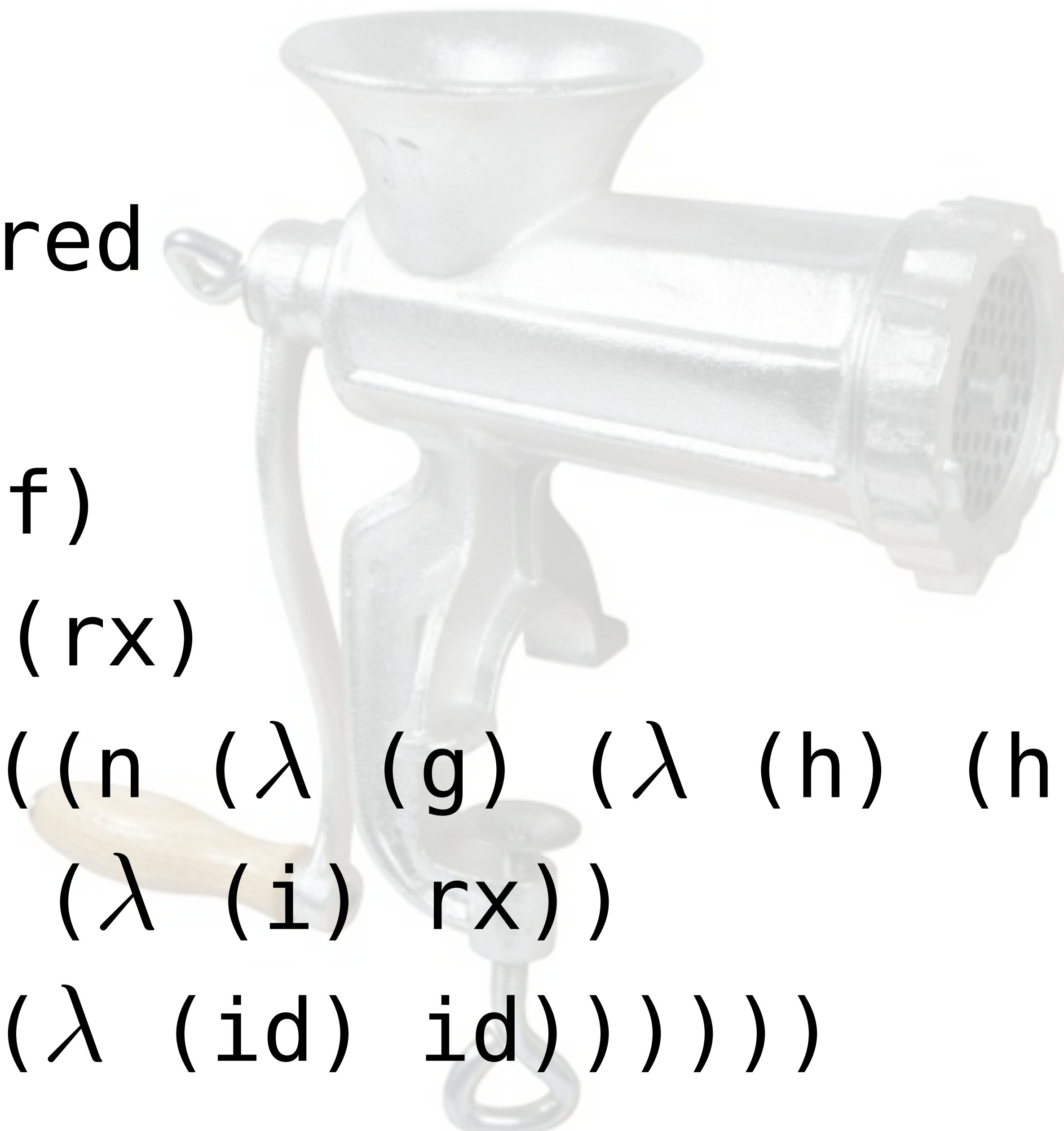
```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

```
(define ⌜2⌝ (λ (f) (λ (x) (f (f x)))))
```

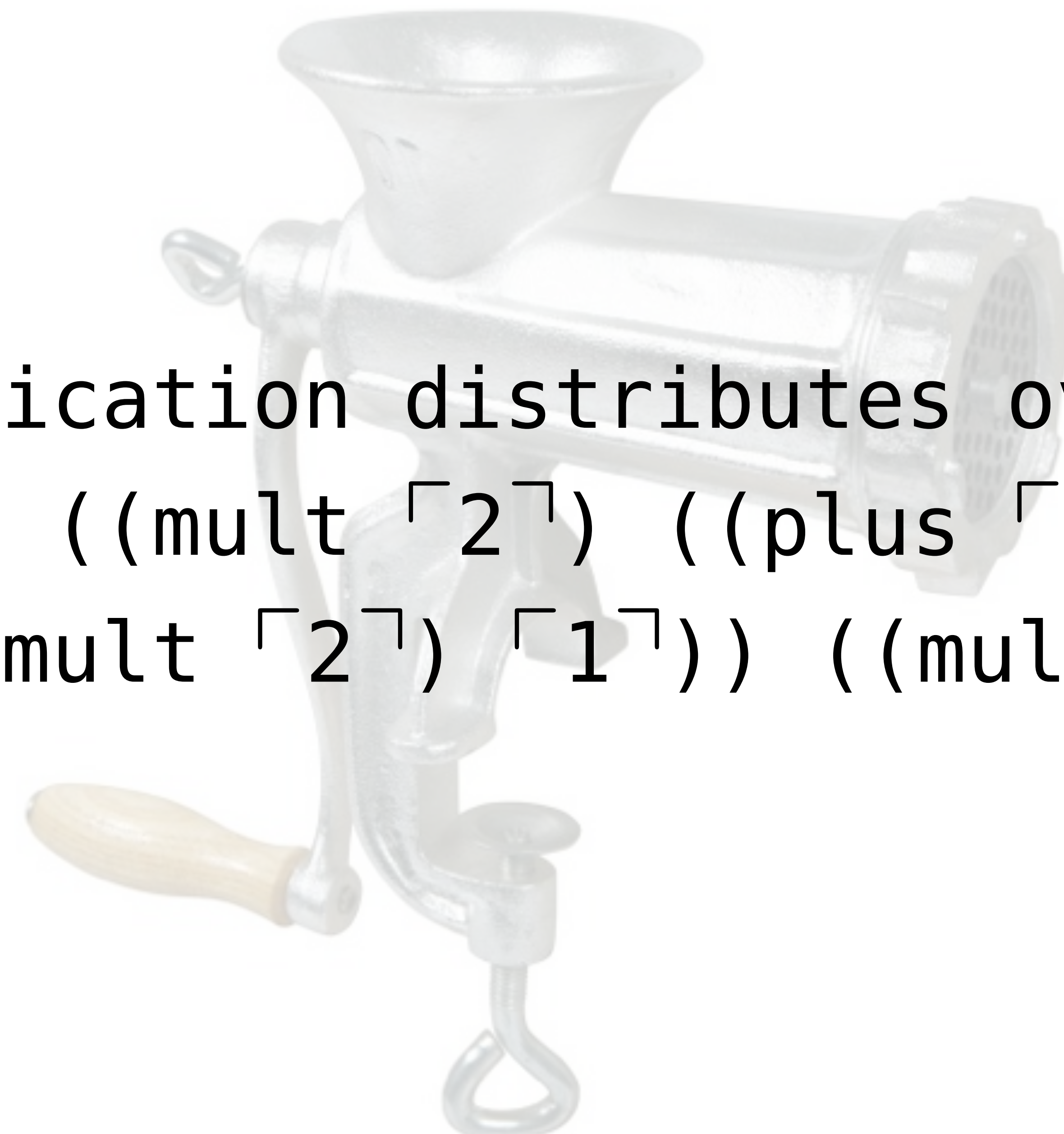


```
interface Function<X,Y> { Y apply(X x); }

class Two<X,X> implements
    Function<Function<X,X>,Function<X,X>> {

    apply(final Function<X,Y> f) = {
        return new Function<X,X>() {
            X apply(X x) {
                return f.apply(f.apply(x));
            };
        }
    }
}

// (λ (f) (λ (x) (f (f x))))
```

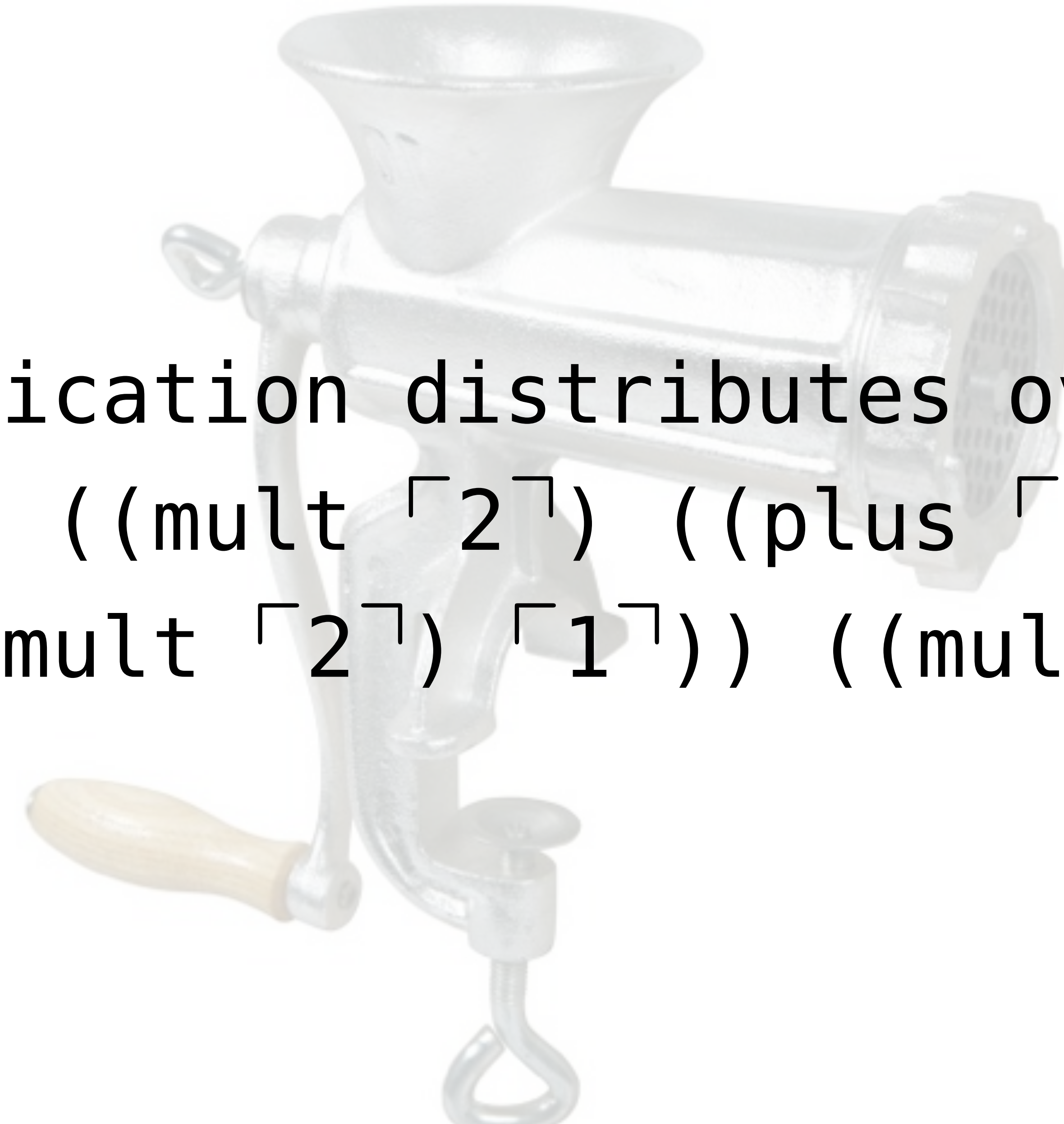
```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝)))))
```



```
(define pred
  (λ (n)
    (λ (rf)
      (λ (rx)
        (((n (λ (g) (λ (h) (h (g rf)))))
          (λ (i) rx))
         (λ (id) id))))))
```

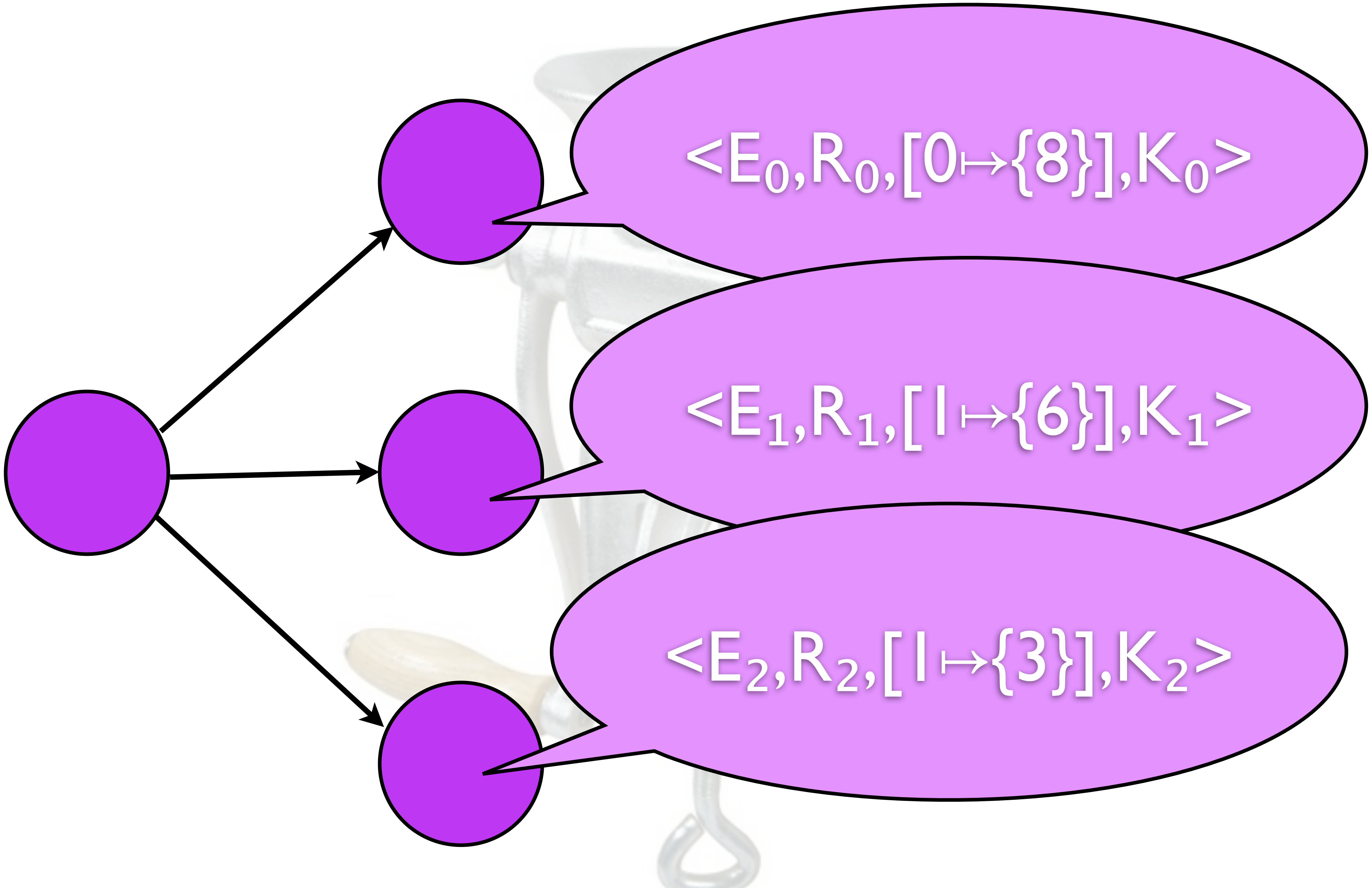


```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

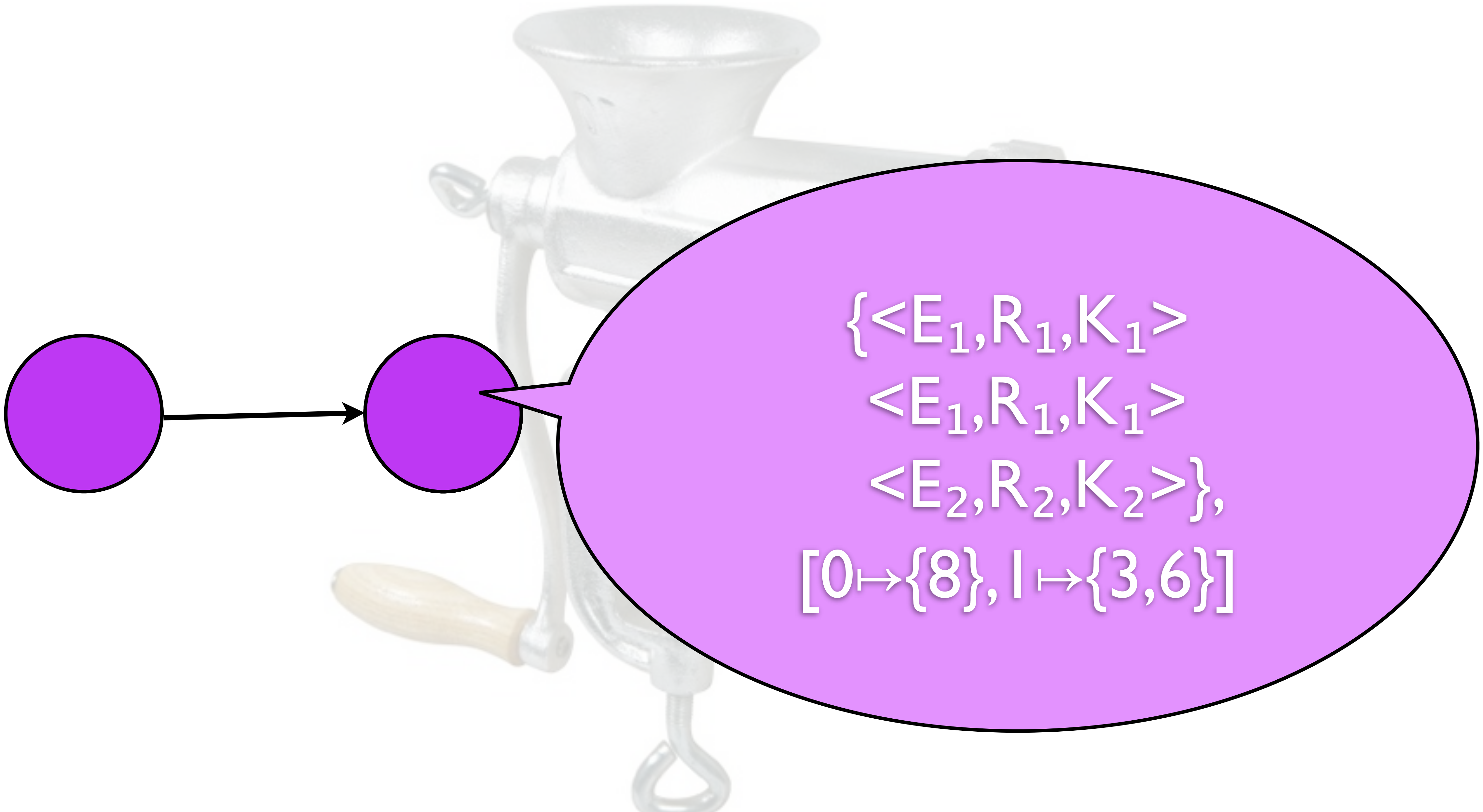


```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

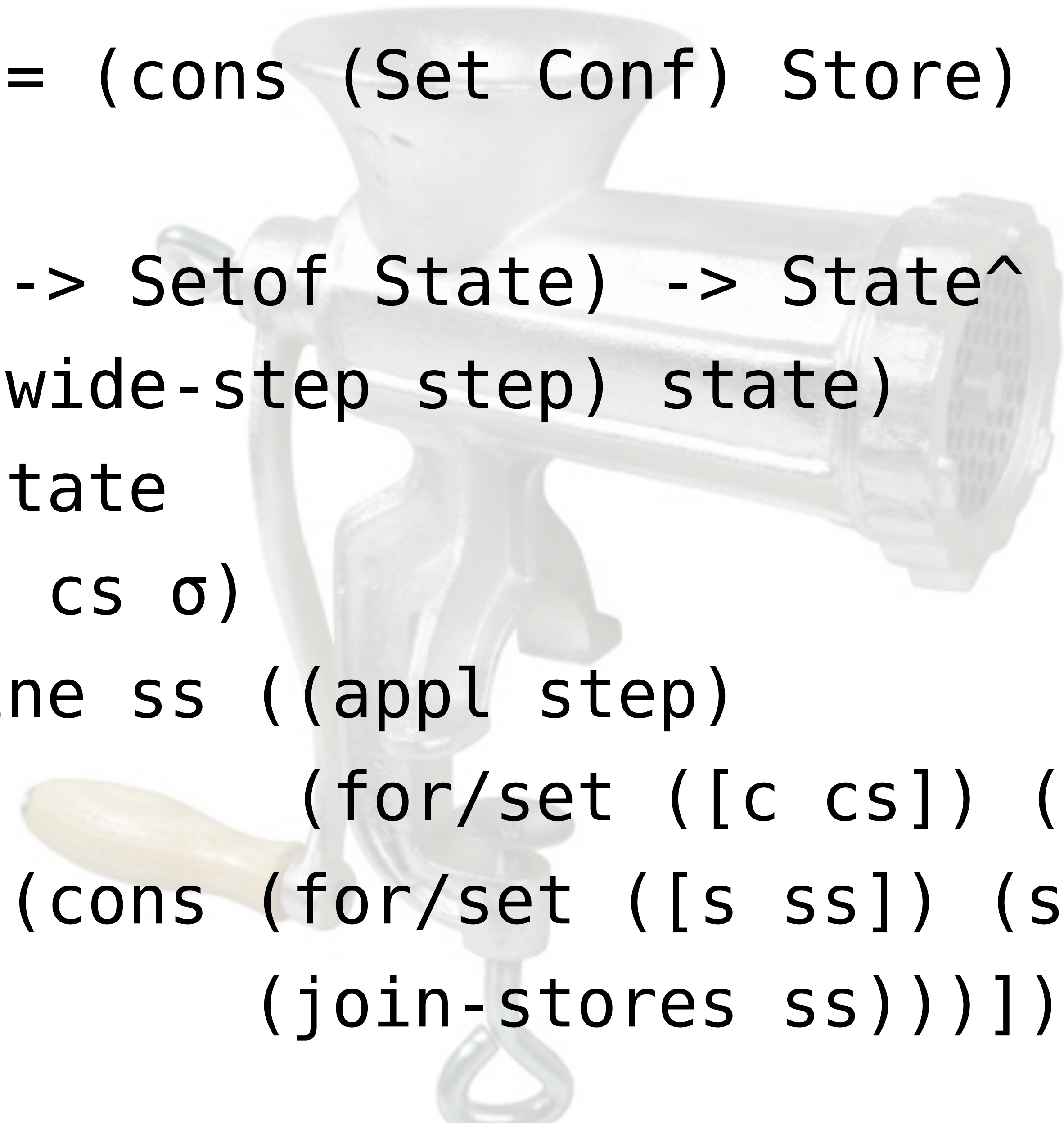
Time: ∞



<E_0,R_0,[0↦{8}],K_0>
<E_1,R_1,[1↦{6}],K_1>
<E_2,R_2,[1↦{3}],K_2>

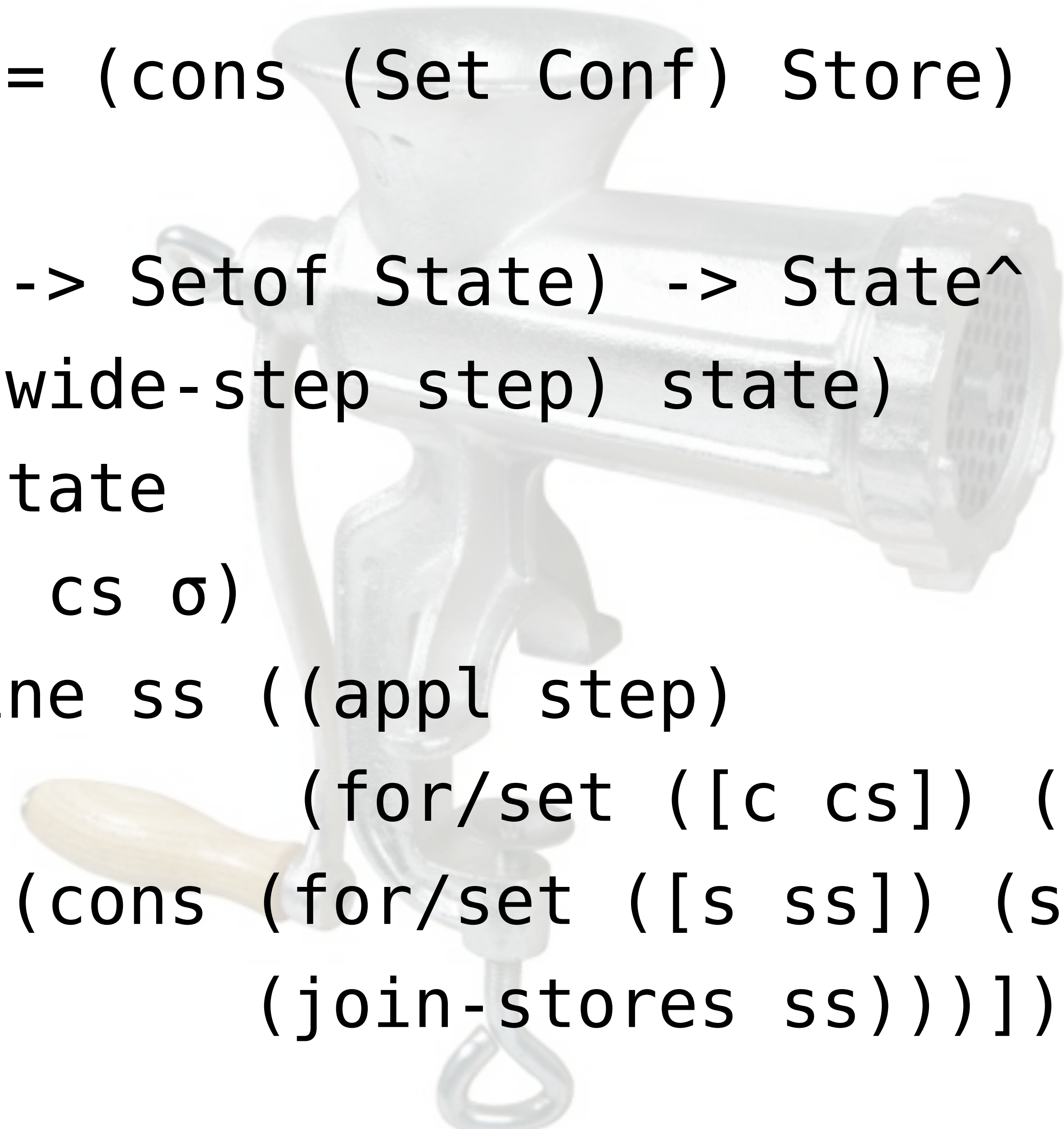

$\{<E_1,R_1,K_1>$
$<E_1,R_1,K_1>$
$<E_2,R_2,K_2>\},$
$[0 \mapsto \{8\}, 1 \mapsto \{3,6\}]$

Generic store widening



```
;; State^ = (cons (Set Conf) Store)

;; (State -> Setof State) -> State^ -> { State^ }
(define ((wide-step step) state)
  (match state
    [(cons cs σ)
     (define ss ((appl step)
                 (for/set ([c cs]) (c->s c σ))))
     (set (cons (for/set ([s ss]) (s->c s))
                (join-stores ss)))]))
```

Generic store widening

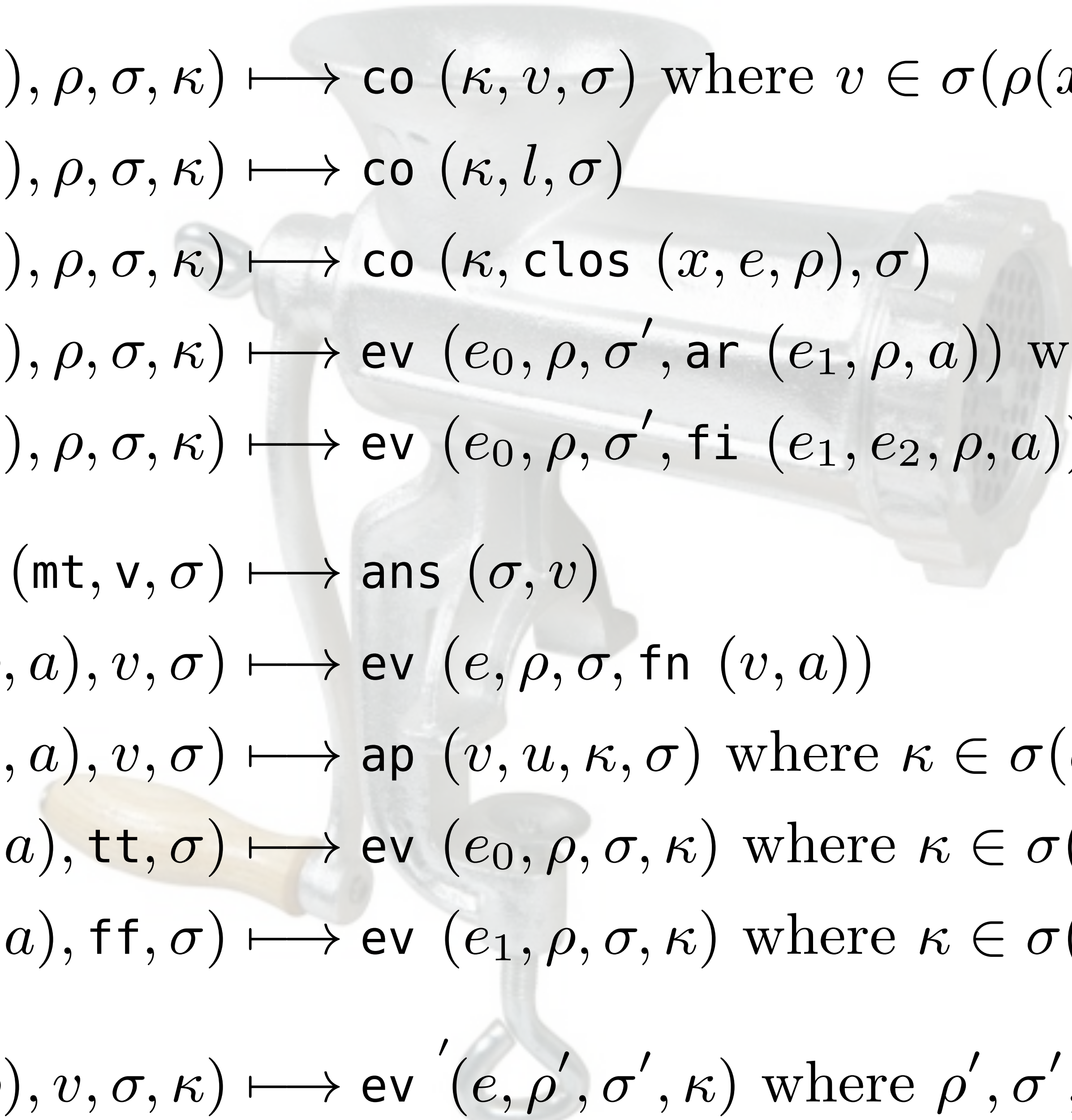


```
;; State^ = (cons (Set Conf) Store)

;; (State -> Setof State) -> State^ -> { State^ }
(define ((wide-step step) state)
  (match state
    [(cons cs σ)
     (define ss ((appl step)
                 (for/set ([c cs]) (c->s c σ))))
     (set (cons (for/set ([s ss]) (s->c s))
                (join-stores ss)))]))
```

Time: 551571ms (≈9.2m)

Precision Preserving Recipe

Lazy non-determinism



$$eval(e) = \{\varsigma \mid \texttt{ev}\ (e, \varnothing, \varnothing, \texttt{mt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\texttt{ev}\ (\texttt{var}\ (x), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x))$$

$$\texttt{ev}\ (\texttt{lit}\ (l), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, l, \sigma)$$

$$\texttt{ev}\ (\texttt{lam}\ (x, e), \rho, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, \texttt{clos}\ (x, e, \rho), \sigma)$$

$$\texttt{ev}\ (\texttt{app}\ (e_0, e_1), \rho, \sigma, \kappa) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{ar}\ (e_1, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)$$

$$\texttt{ev}\ (\texttt{if}\ (e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{fi}\ (e_1, e_2, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)$$

$$\texttt{co}\ (\texttt{mt}, \texttt{v}, \sigma) \longmapsto \texttt{ans}\ (\sigma, v)$$

$$\texttt{co}\ (\texttt{ar}\ (e, \rho, a), v, \sigma) \longmapsto \texttt{ev}\ (e, \rho, \sigma, \texttt{fn}\ (v, a))$$

$$\texttt{co}\ (\texttt{fn}\ (u, a), v, \sigma) \longmapsto \texttt{ap}\ (v, u, \kappa, \sigma) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{tt}, \sigma) \longmapsto \texttt{ev}\ (e_0, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{ff}, \sigma) \longmapsto \texttt{ev}\ (e_1, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)$$

$$\texttt{ap}\ (\texttt{clos}\ (x, e, \rho), v, \sigma, \kappa) \longmapsto \texttt{ev}\ {}'(e, \rho', \sigma', \kappa) \text{ where } \rho', \sigma', {}' = bind\ (\sigma, x, v)$$

$$\texttt{ap}\ (o, v, \sigma, \kappa) \longmapsto \texttt{co}\ (\kappa, v', \sigma) \text{ where } \kappa \in \sigma(a) \text{ and } v' \in \Delta(o, v)$$

Lazy non-determinism

$$eval(e) = \{\varsigma \mid \texttt{ev}\ (e, \varnothing, \varnothing, \texttt{mt}) \longmapsto\!\!\!\!\!\twoheadrightarrow \varsigma\} \text{ where}$$

$$\begin{aligned}
\texttt{ev}\ (\texttt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x)) \\
\texttt{ev}\ (\texttt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, l, \sigma) \\
\texttt{ev}\ (\texttt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, \texttt{clos}\ (x, e, \rho), \sigma) \\
\texttt{ev}\ (\texttt{app}\ (e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{ar}\ (e_1, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa) \\
\texttt{ev}\ (\texttt{if}\ (e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{fi}\ (e_1, e_2, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)
\end{aligned}$$

$$\begin{aligned}
\texttt{co}\ (\texttt{mt}, \texttt{v}, \sigma) &\longmapsto \texttt{ans}\ (\sigma, v) \\
\texttt{co}\ (\texttt{ar}\ (e, \rho, a), v, \sigma) &\longmapsto \texttt{ev}\ (e, \rho, \sigma, \texttt{fn}\ (v, a)) \\
\texttt{co}\ (\texttt{fn}\ (u, a), v, \sigma) &\longmapsto \texttt{ap}\ (v, u, \kappa, \sigma) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{tt}, \sigma) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{ff}, \sigma) &\longmapsto \texttt{ev}\ (e_1, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)
\end{aligned}$$

$$\begin{aligned}
\texttt{ap}\ (\texttt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \texttt{ev}\ {}'(e, \rho', \sigma', \kappa) \text{ where } \rho', \sigma', {}' = bind\ \ (\sigma, x, v) \\
\texttt{ap}\ (o, v, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v', \sigma) \text{ where } \kappa \in \sigma(a) \text{ and } v' \in \Delta(o, v)
\end{aligned}$$

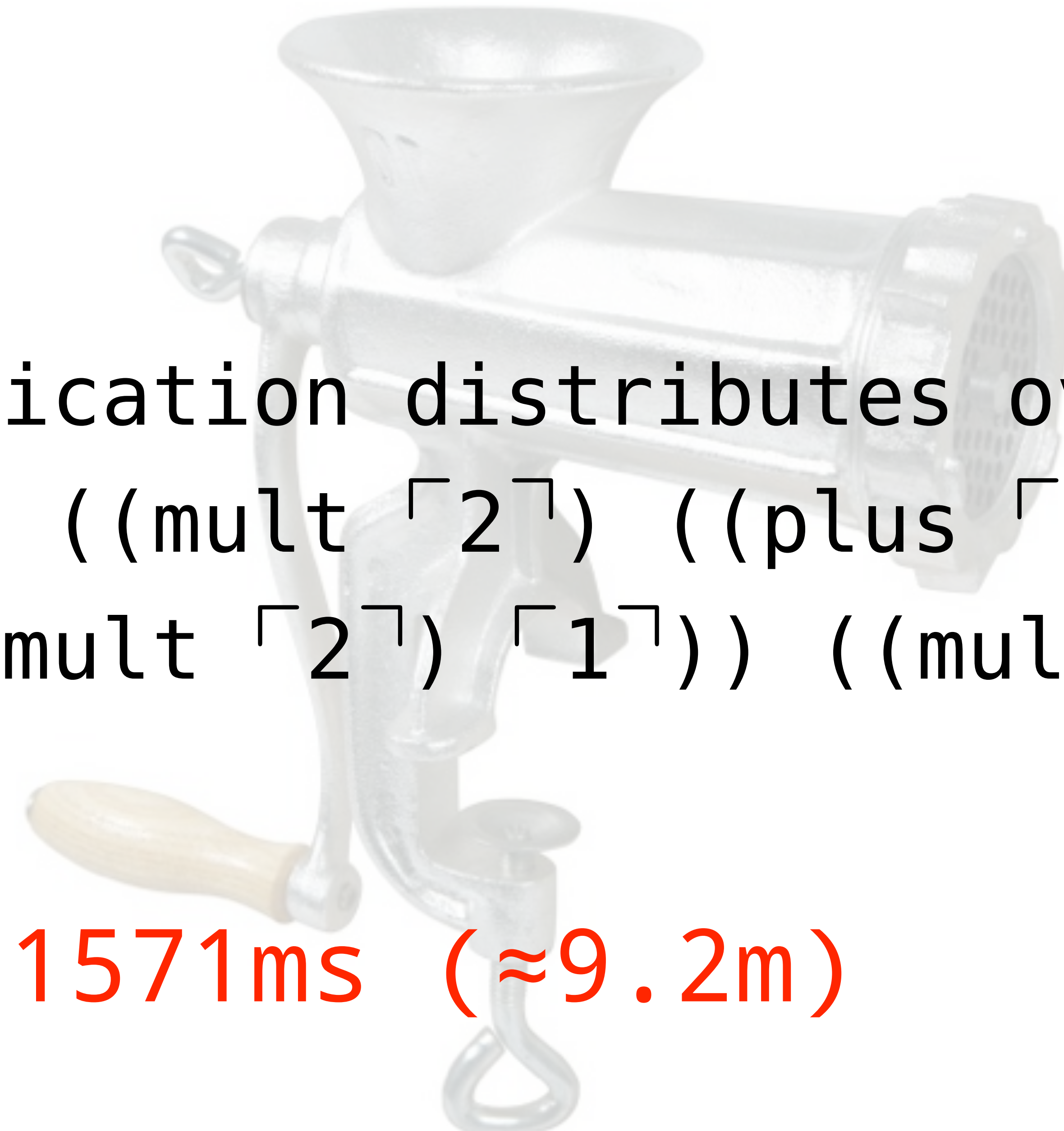
Lazy non-determinism

$$eval(e) = \{\varsigma \mid \texttt{ev}\ (e, \varnothing, \varnothing, \texttt{mt}) \longmapsto\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\begin{aligned}
\texttt{ev}\ (\texttt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x)) \\
\texttt{ev}\ (\texttt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, l, \sigma) \\
\texttt{ev}\ (\texttt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, \texttt{clos}\ (x, e, \rho), \sigma) \\
\texttt{ev}\ (\texttt{app}\ (e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{ar}\ (e_1, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa) \\
\texttt{ev}\ (\texttt{if}\ (e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma', \texttt{fi}\ (e_1, e_2, \rho, a)) \text{ where } a, \sigma' = push\ (\sigma, \kappa)
\end{aligned}$$

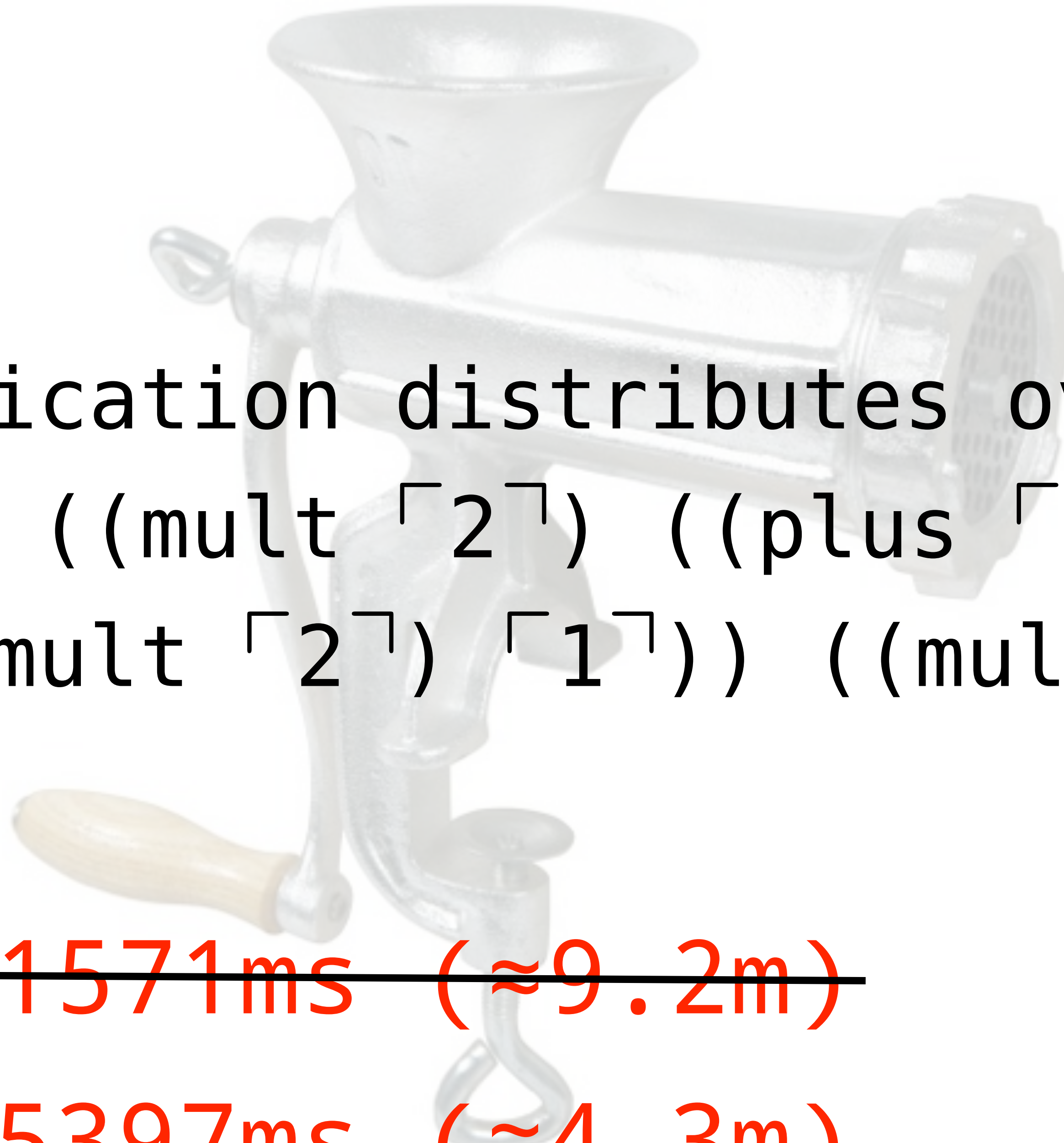
$$\begin{aligned}
\texttt{co}\ (\texttt{mt}, \texttt{v}, \sigma) &\longmapsto \texttt{ans}\ (\sigma, v) \\
\texttt{co}\ (\texttt{ar}\ (e, \rho, a), v, \sigma) &\longmapsto \texttt{ev}\ (e, \rho, \sigma, \texttt{fn}\ (v, a)) \\
\texttt{co}\ (\texttt{fn}\ (u, a), v, \sigma) &\longmapsto \texttt{ap}\ (v, u, \kappa, \sigma) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{tt}, \sigma) &\longmapsto \texttt{ev}\ (e_0, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a) \\
\texttt{co}\ (\texttt{fi}\ (e_0, e_1, \rho, a), \texttt{ff}, \sigma) &\longmapsto \texttt{ev}\ (e_1, \rho, \sigma, \kappa) \text{ where } \kappa \in \sigma(a)
\end{aligned}$$

$$\begin{aligned}
\texttt{ap}\ (\texttt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \texttt{ev}\ '(e, \rho', \sigma', \kappa) \text{ where } \rho', \sigma', ' = bind\ (\sigma, x, v) \\
\texttt{ap}\ (o, v, \sigma, \kappa) &\longmapsto \texttt{co}\ (\kappa, v', \sigma) \text{ where } \kappa \in \sigma(a) \text{ and } v' \in \Delta(o, v)
\end{aligned}$$



```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

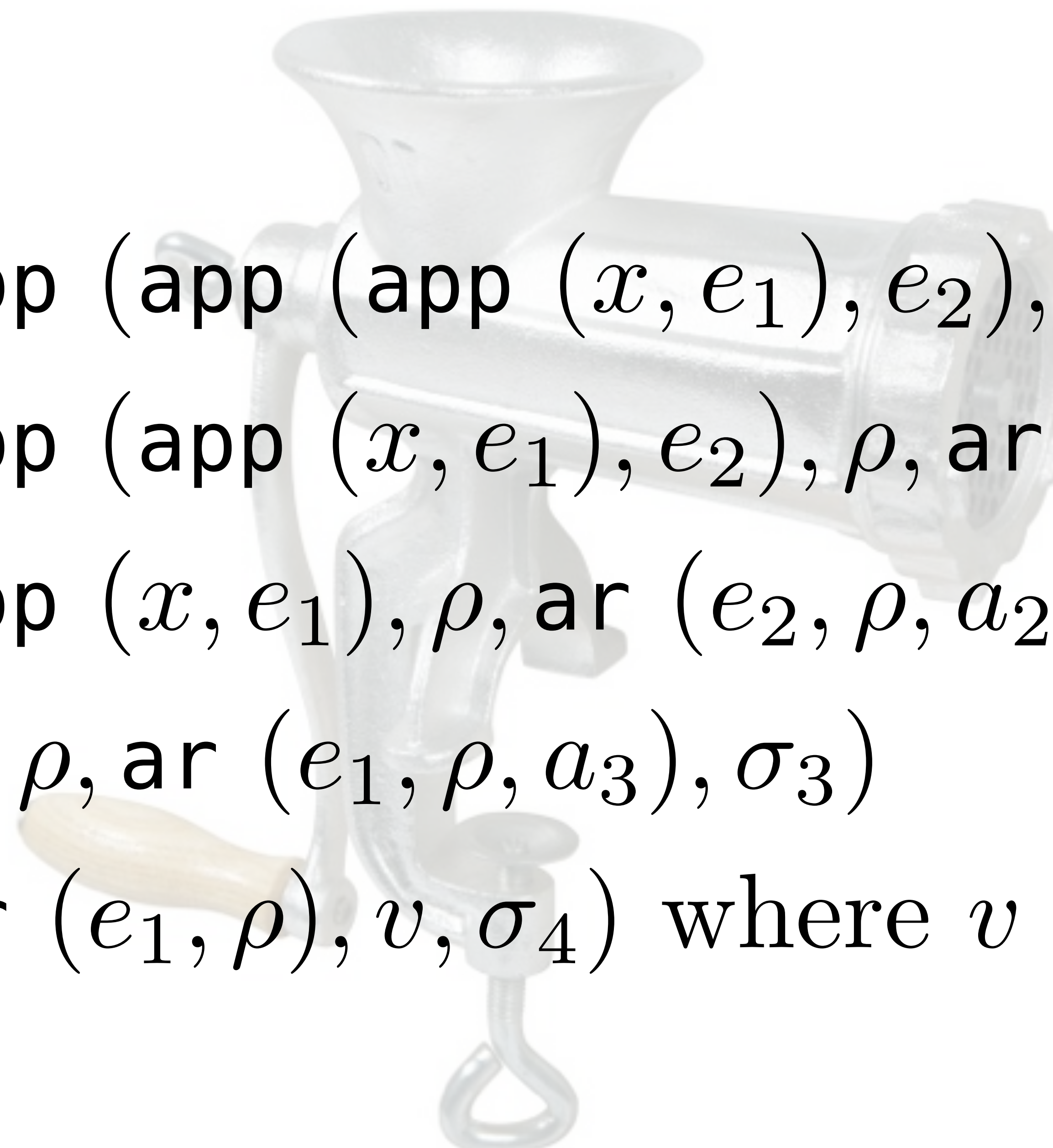
Time: 551571ms (≈9.2m)



```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

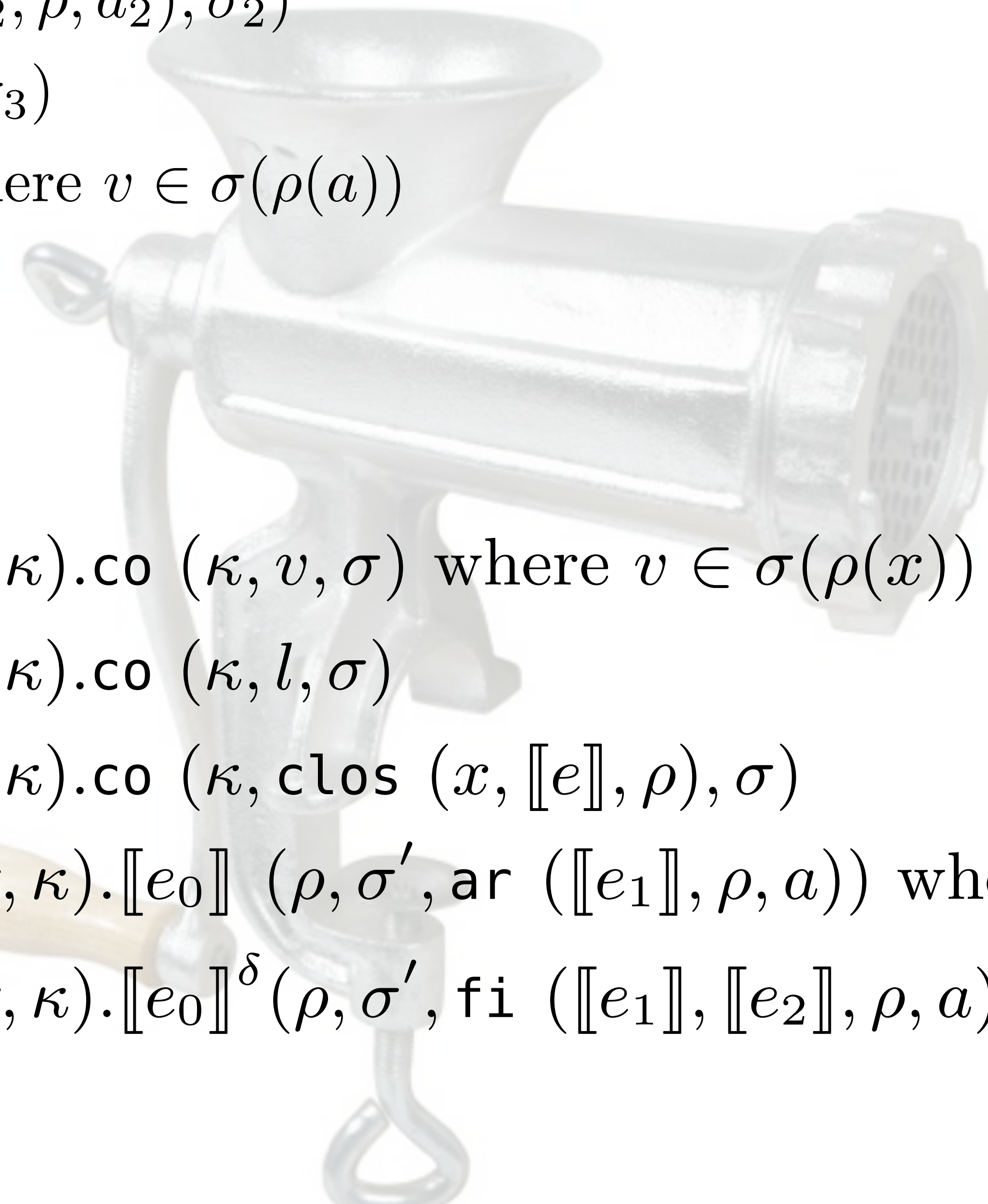
~~Time: 551571ms (≈9.2m)~~

Time: 255397ms (≈4.3m)

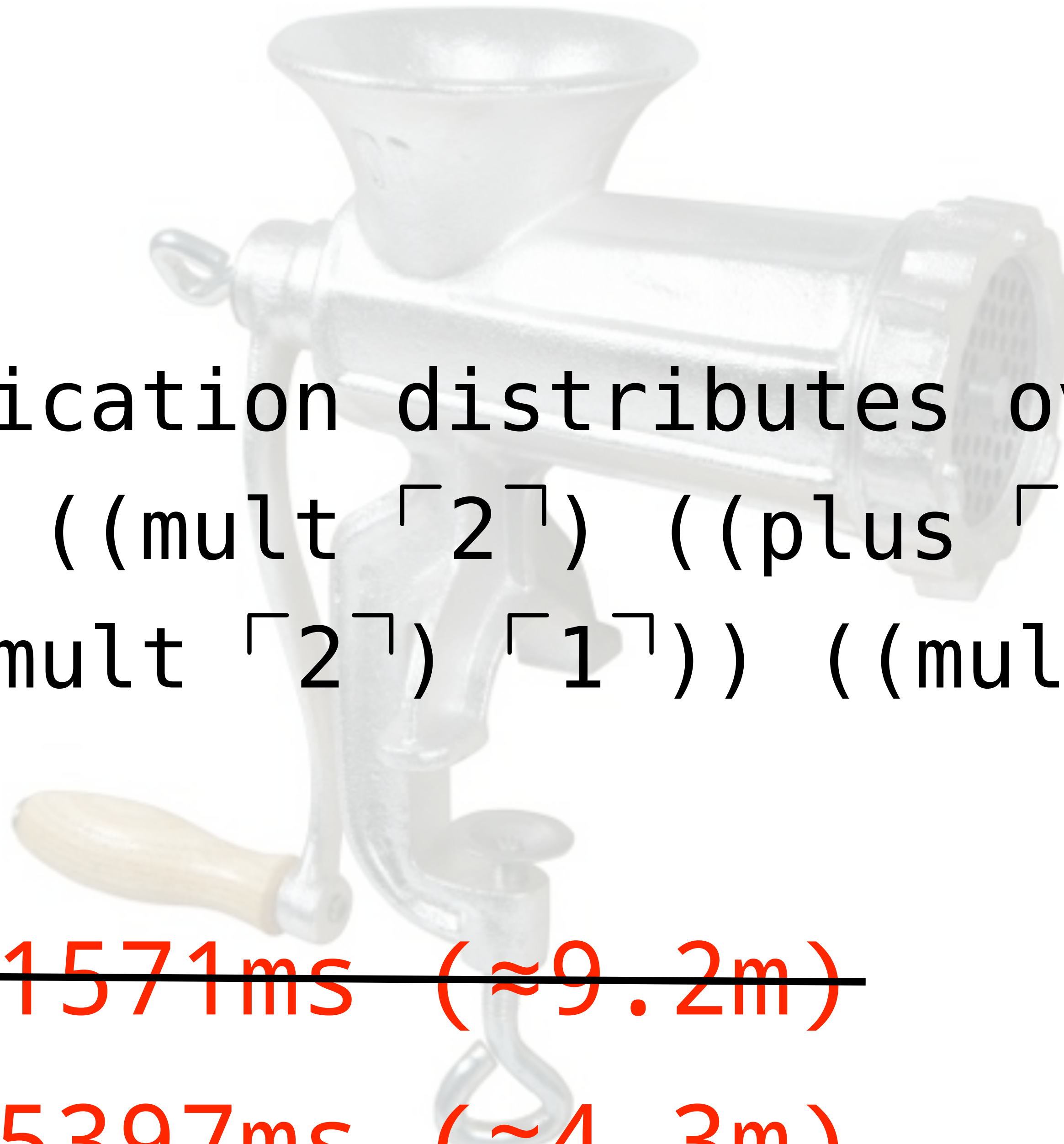


$$
\begin{aligned}
& \mathsf{ev}\ (\mathsf{app}\ (\mathsf{app}\ (\mathsf{app}\ (x, e_1), e_2), e_3), \rho, \kappa, \sigma_0) \\
\longmapsto\ & \mathsf{ev}\ (\mathsf{app}\ (\mathsf{app}\ (x, e_1), e_2), \rho, \mathsf{ar}\ (e_3, \rho, a_1), \sigma_1) \\
\longmapsto\ & \mathsf{ev}\ (\mathsf{app}\ (x, e_1), \rho, \mathsf{ar}\ (e_2, \rho, a_2), \sigma_2) \\
\longmapsto\ & \mathsf{ev}\ (x, \rho, \mathsf{ar}\ (e_1, \rho, a_3), \sigma_3) \\
\longmapsto\ & \mathsf{co}\ (\mathsf{ar}\ (e_1, \rho), v, \sigma_4) \text{ where } v \in \sigma(\rho(a))
\end{aligned}
$$

$$
\begin{aligned}
& \texttt{ev}\ (\texttt{app}\ (\texttt{app}\ (\texttt{app}\ (x, e_1), e_2), e_3), \rho, \kappa, \sigma_0) \\
\longmapsto\ & \texttt{ev}\ (\texttt{app}\ (\texttt{app}\ (x, e_1), e_2), \rho, \texttt{ar}\ (e_3, \rho, a_1), \sigma_1) \\
\longmapsto\ & \texttt{ev}\ (\texttt{app}\ (x, e_1), \rho, \texttt{ar}\ (e_2, \rho, a_2), \sigma_2) \\
\longmapsto\ & \texttt{ev}\ (x, \rho, \texttt{ar}\ (e_1, \rho, a_3), \sigma_3) \\
\longmapsto\ & \texttt{co}\ (\texttt{ar}\ (e_1, \rho), v, \sigma_4) \text{ where } v \in \sigma(\rho(a))
\end{aligned}
$$



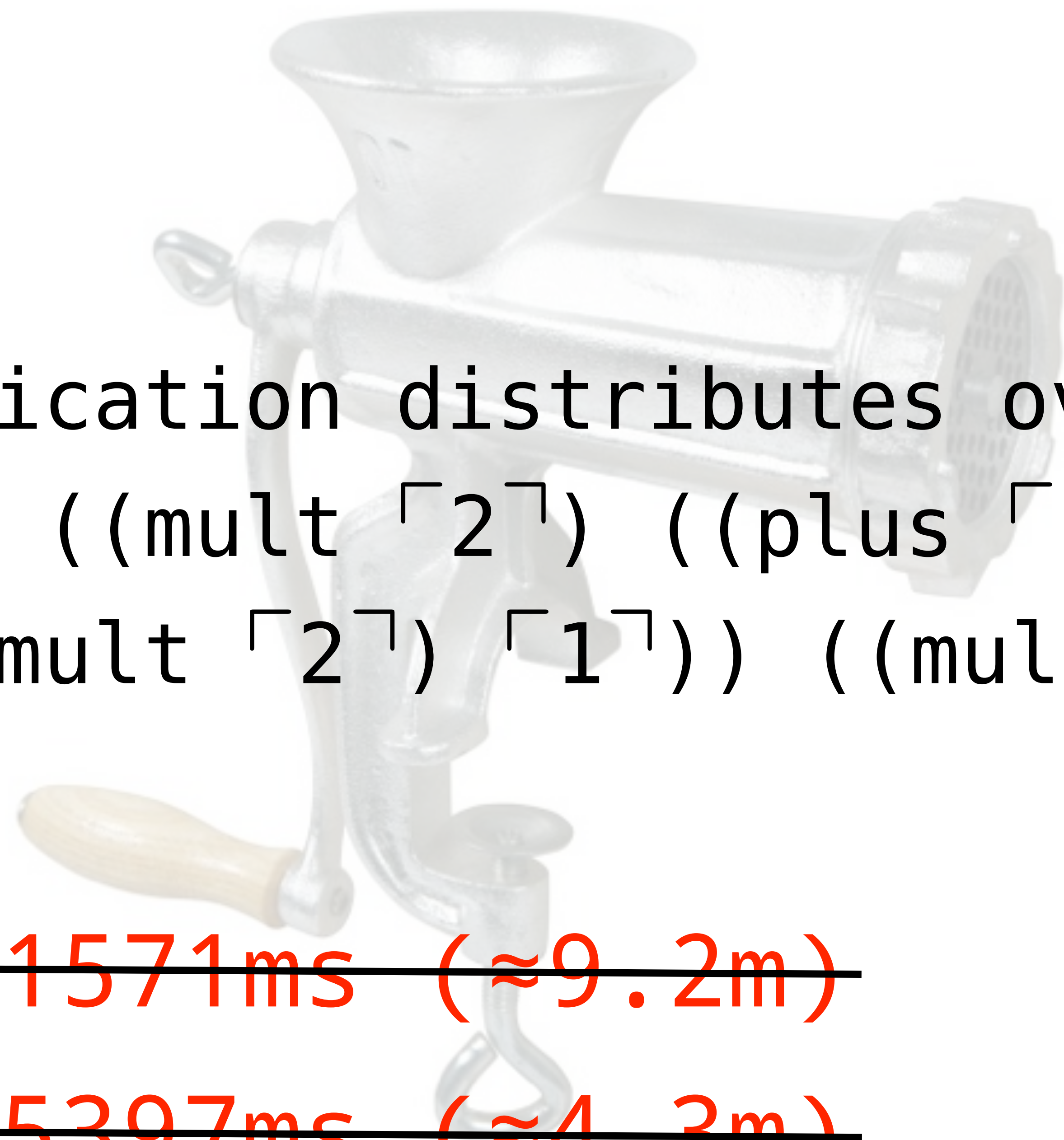
$$
\begin{aligned}
[\![\texttt{var}\ (x)]\!] &= \lambda(\rho, \sigma, \kappa).\texttt{co}\ (\kappa, v, \sigma) \text{ where } v \in \sigma(\rho(x)) \\
[\![\texttt{lit}\ (l)]\!] &= \lambda(\rho, \sigma, \kappa).\texttt{co}\ (\kappa, l, \sigma) \\
[\![\texttt{lam}\ (x, e)]\!] &= \lambda(\rho, \sigma, \kappa).\texttt{co}\ (\kappa, \texttt{clos}\ (x, [\![e]\!], \rho), \sigma) \\
[\![\texttt{app}\ (e_0, e_1)]\!] &= \lambda\ (\rho, \sigma, \kappa).[\![e_0]\!]\ (\rho, \sigma', \texttt{ar}\ ([\![e_1]\!], \rho, a)) \text{ where } a, \sigma' = \mathit{push}\ (\sigma, \kappa) \\
[\![\texttt{if}\ (e_0, e_1, e_2)]\!] &= \lambda\ (\rho, \sigma, \kappa).[\![e_0]\!]^{\delta}(\rho, \sigma', \texttt{fi}\ ([\![e_1]\!], [\![e_2]\!], \rho, a)) \text{ where } a, \sigma' = \mathit{push}\ (\sigma, \kappa)
\end{aligned}
$$



```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

~~Time: 551571ms (≈9.2m)~~

Time: 255397ms (≈4.3m)



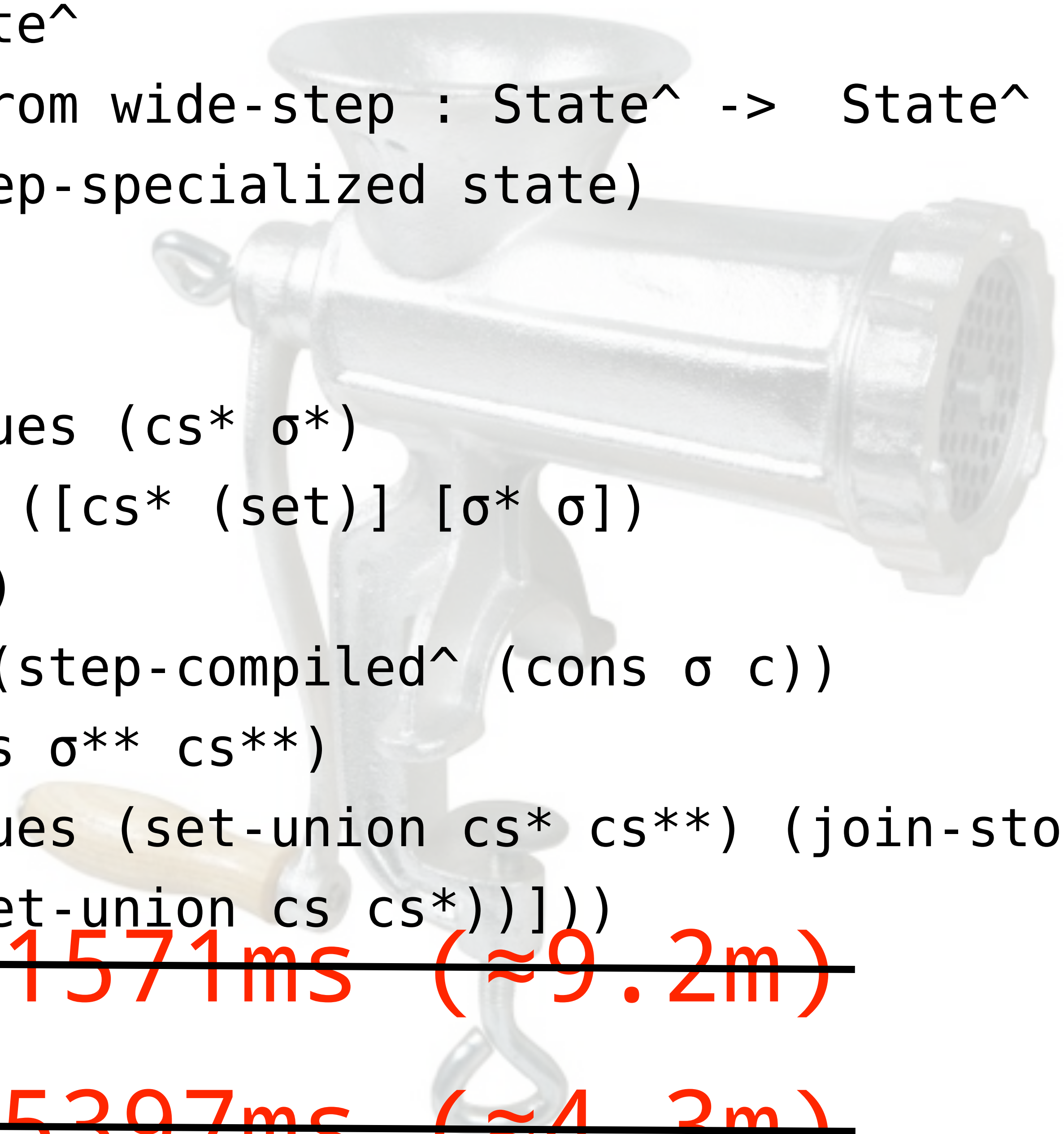
```
;; multiplication distributes over addition
((church=? ((mult ⌜2⌝) ((plus ⌜1⌝) ⌜3⌝)))
 ((plus ((mult ⌜2⌝) ⌜1⌝)) ((mult ⌜2⌝) ⌜3⌝))))
```

~~Time: 551571ms (≈9.2m)~~

~~Time: 255397ms (≈4.3m)~~

Time: 31173ms (≈.5m)

Specialized fixpoint computation



```
;; State^ -> State^
;; Specialized from wide-step : State^ ->  State^  ≈ State^ -> State^
(define (wide-step-specialized state)
  (match state
    [(cons σ cs)
     (define-values (cs* σ*)
       (for/fold ([cs* (set)] [σ* σ])
         ([c cs])
         (match (step-compiled^ (cons σ c))
           [(cons σ** cs**)
            (values (set-union cs* cs**) (join-store σ* σ**))])))
     (cons σ* (set-union cs cs*))]))
```

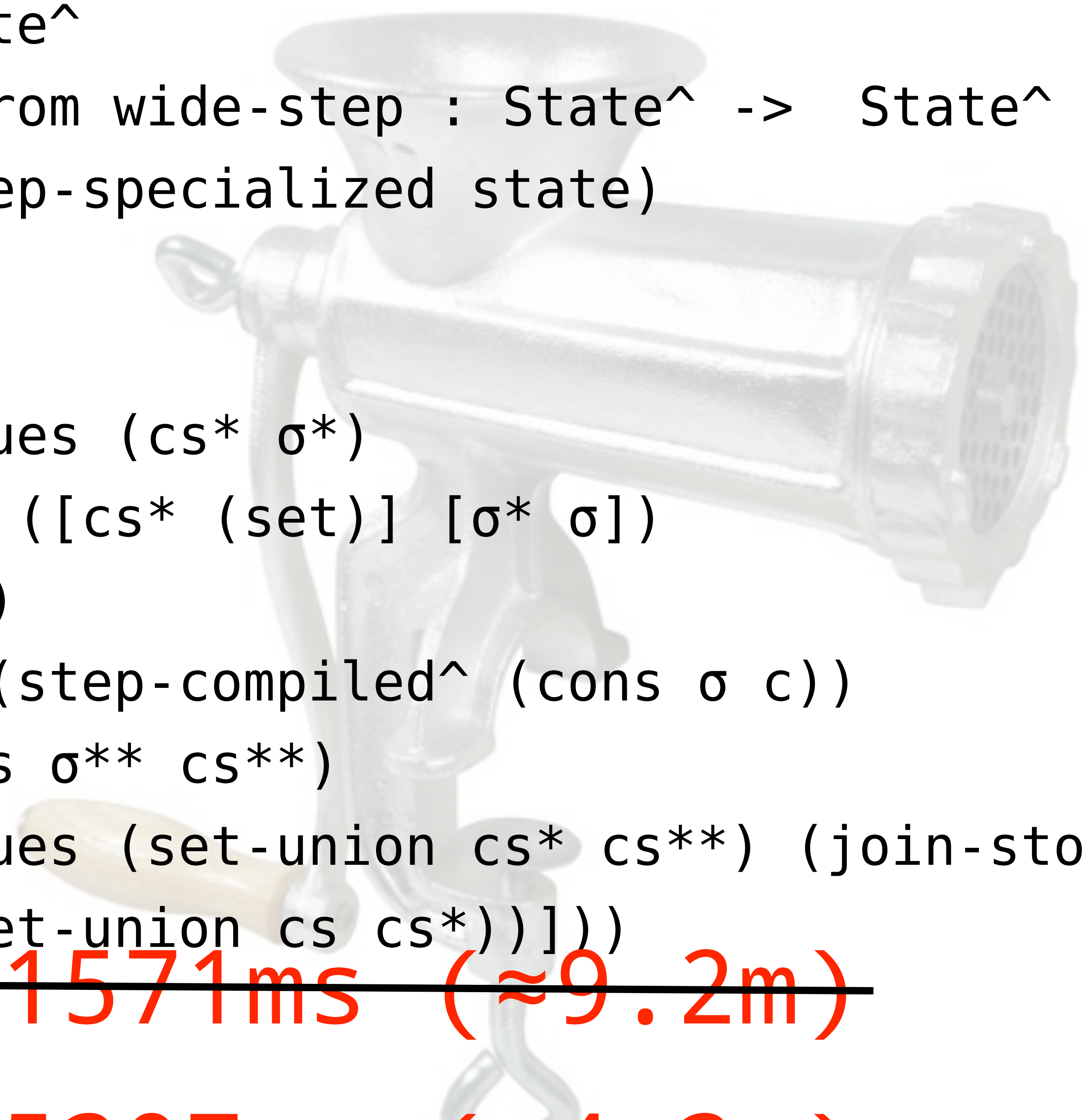
~~Time: 551571ms (≈9.2m)~~

~~Time: 255397ms (≈4.3m)~~

Time: 31173ms (≈.5m)

Specialized fixpoint computation

```
;; State^ -> State^
;; Specialized from wide-step : State^ ->  State^  ≈ State^ -> State^
(define (wide-step-specialized state)
  (match state
    [(cons σ cs)
     (define-values (cs* σ*)
       (for/fold ([cs* (set)] [σ* σ])
         ([c cs])
         (match (step-compiled^ (cons σ c))
           [(cons σ** cs**)
            (values (set-union cs* cs**) (join-store σ* σ**))])))
     (cons σ* (set-union cs cs*))]))
```



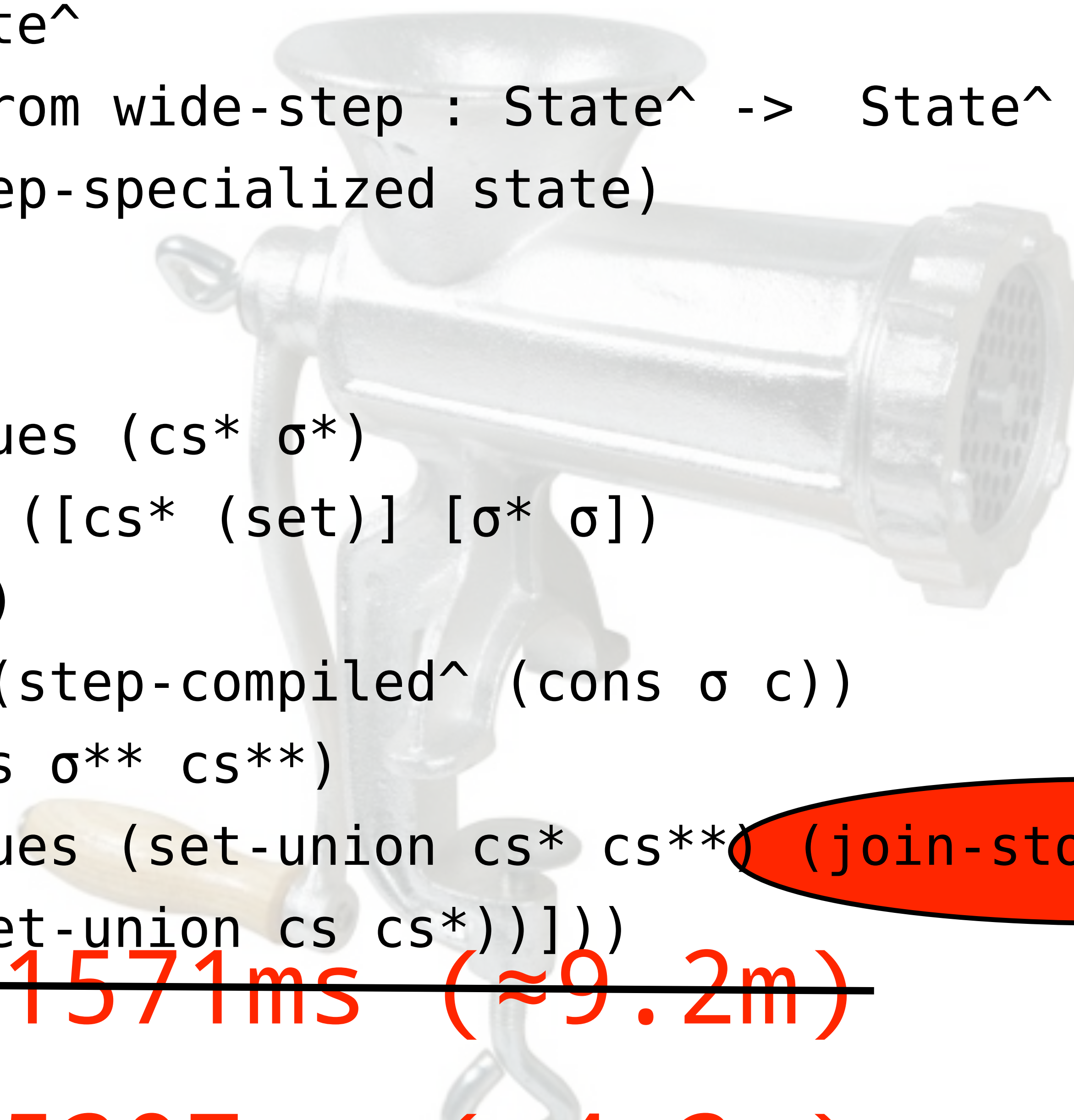
~~Time: 551571ms (≈9.2m)~~

~~Time: 255397ms (≈4.3m)~~

~~Time: 31173ms (≈.5m)~~

Time: 14212ms

Specialized fixpoint computation



```
;; State^ -> State^
;; Specialized from wide-step : State^ ->  State^  ≈ State^ -> State^
(define (wide-step-specialized state)
  (match state
    [(cons σ cs)
     (define-values (cs* σ*)
       (for/fold ([cs* (set)] [σ* σ])
         ([c cs])
         (match (step-compiled^ (cons σ c))
           [(cons σ** cs**)
            (values (set-union cs* cs**) (join-store σ* σ**))])))
     (cons σ* (set-union cs cs*))]))
```

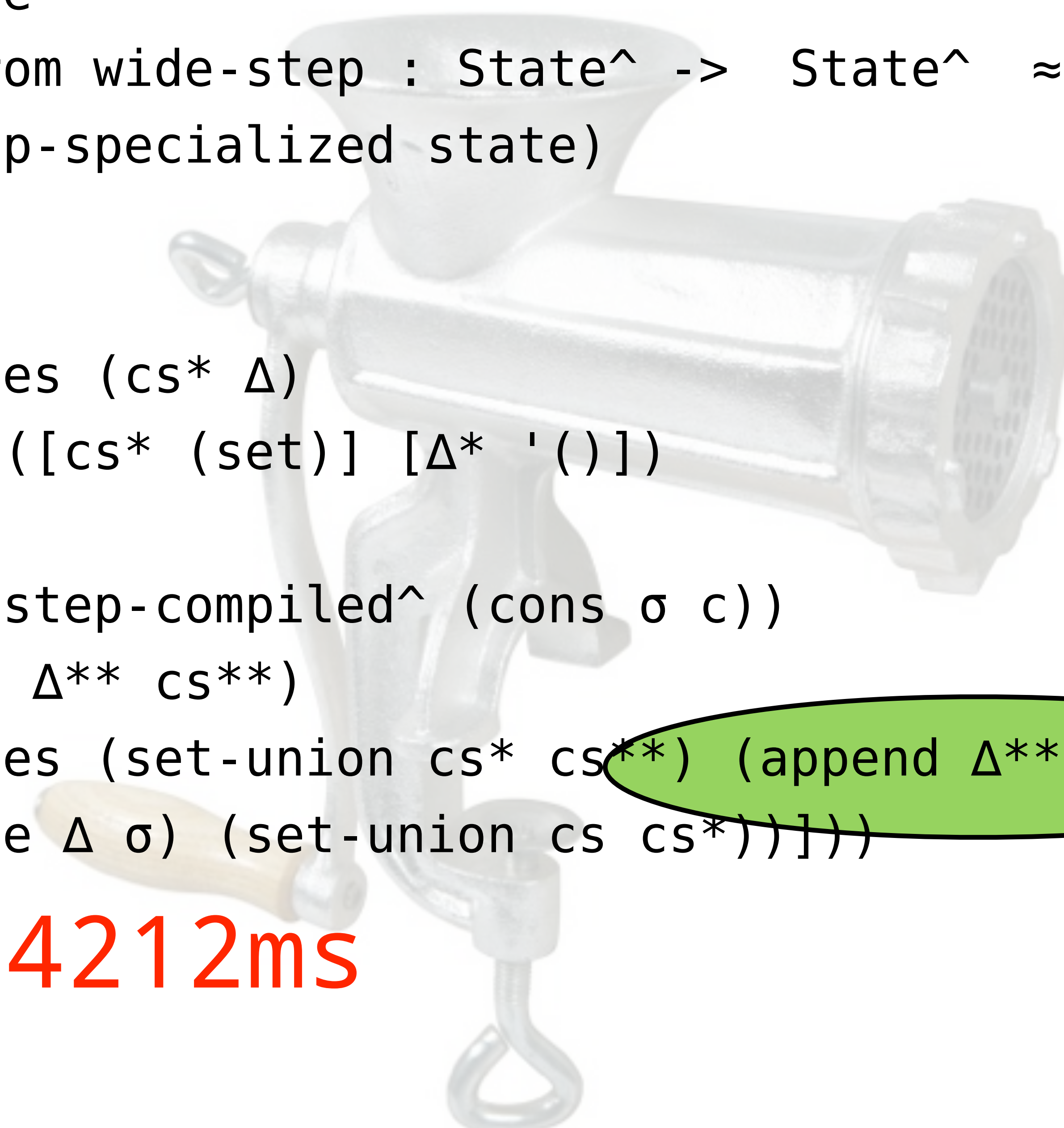
~~Time: 551571ms (≈9.2m)~~

~~Time: 255397ms (≈4.3m)~~

~~Time: 31173ms (≈.5m)~~

Time: 14212ms

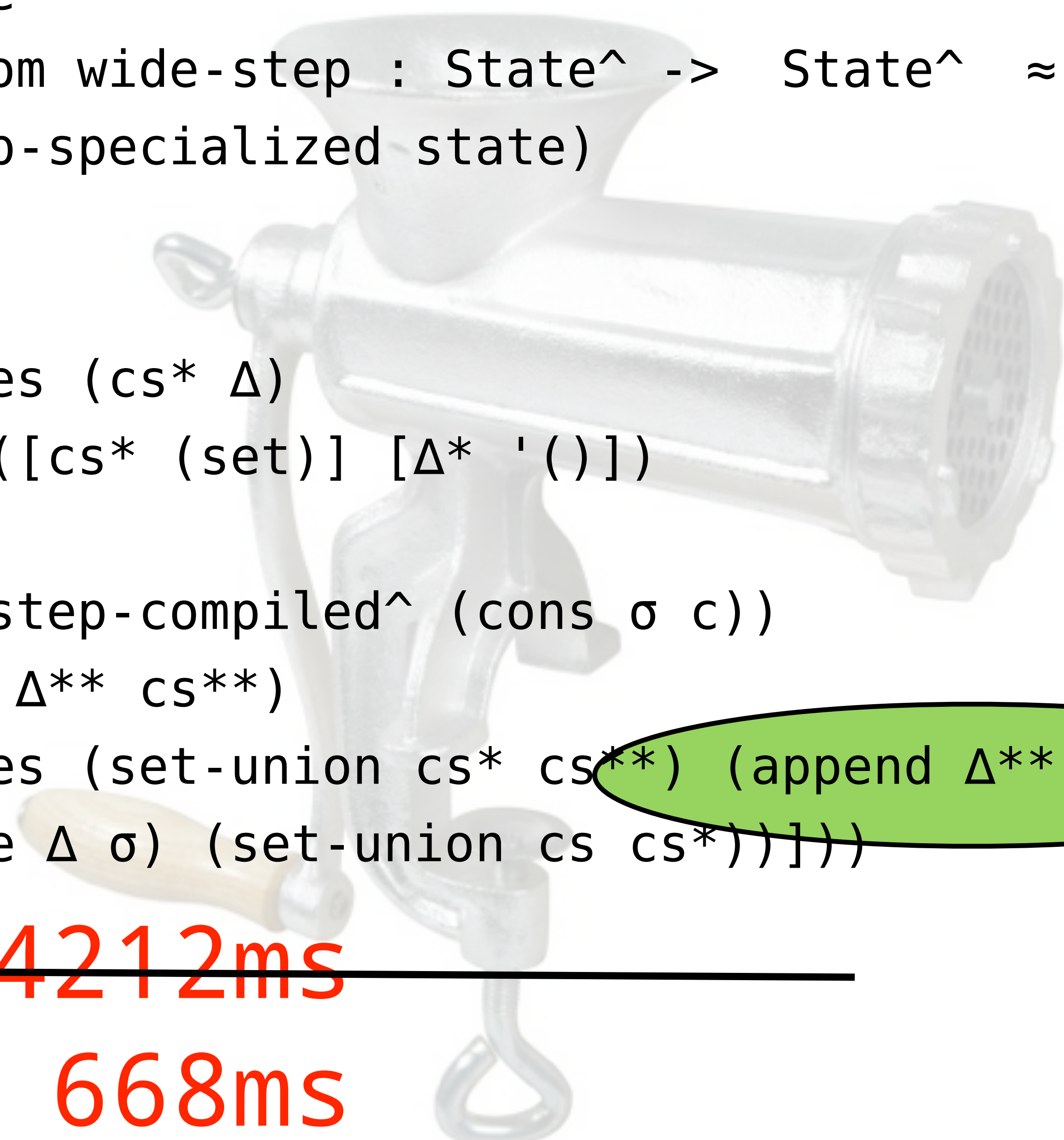
Computing with store diffs



```
;; State^ -> State^
;; Specialized from wide-step : State^ ->  State^  ≈ State^ -> State^
(define (wide-step-specialized state)
  (match state
    [(cons σ cs)
     (define-values (cs* Δ)
       (for/fold ([cs* (set)] [Δ* '()])
         ([c cs])
         (match (step-compiled^ (cons σ c))
           [(cons Δ** cs**)
            (values (set-union cs* cs**) (append Δ** Δ*))])))
     (cons (update Δ σ) (set-union cs cs*))]))
```

Time: 14212ms

Computing with store diffs

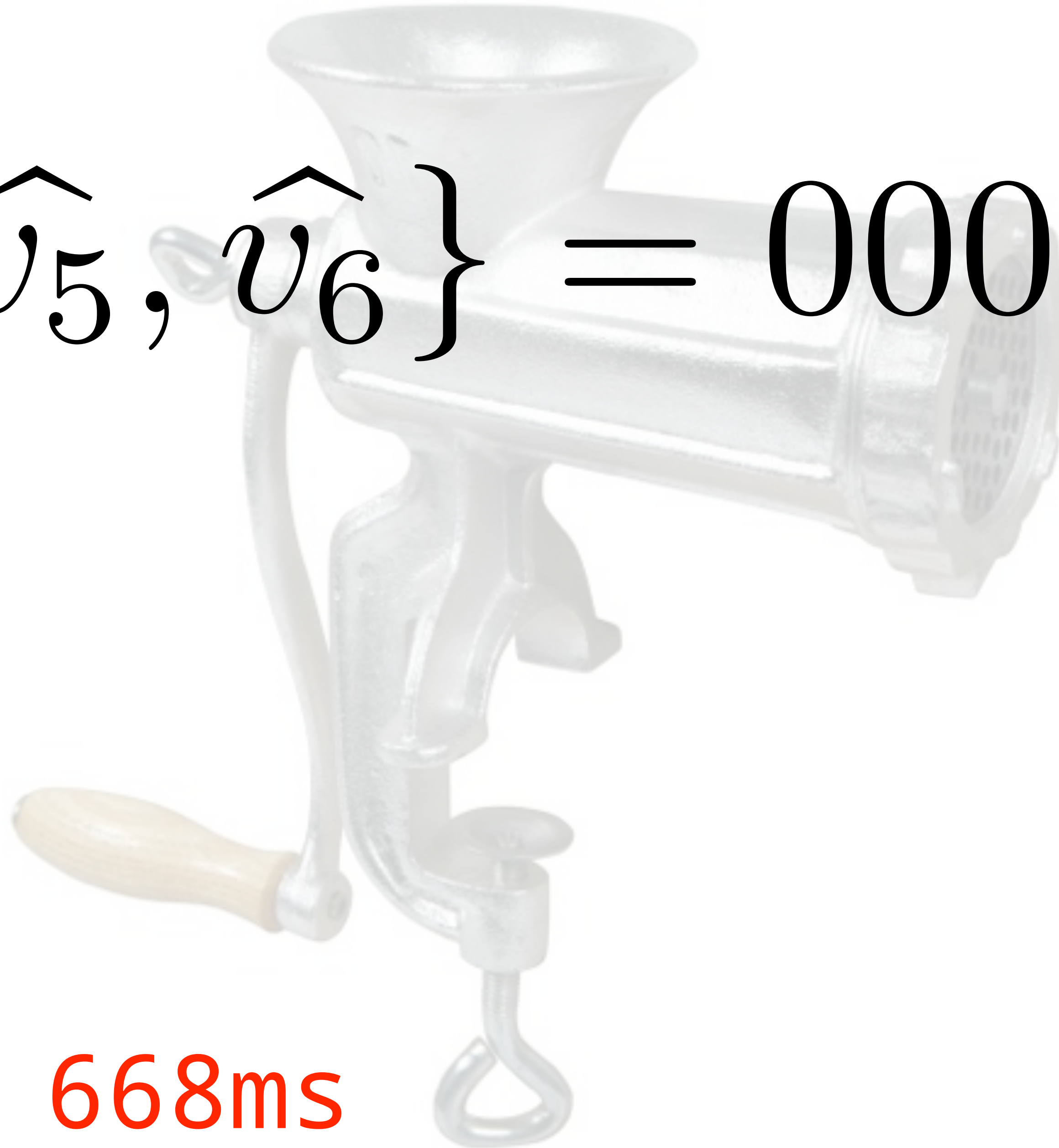


```
;; State^ -> State^
;; Specialized from wide-step : State^ ->  State^  ≈ State^ -> State^
(define (wide-step-specialized state)
  (match state
    [(cons σ cs)
     (define-values (cs* Δ)
       (for/fold ([cs* (set)] [Δ* '()])
         ([c cs])
         (match (step-compiled^ (cons σ c))
           [(cons Δ** cs**)
            (values (set-union cs* cs**) (append Δ** Δ*))])))
     (cons (update Δ σ) (set-union cs cs*))]))
```

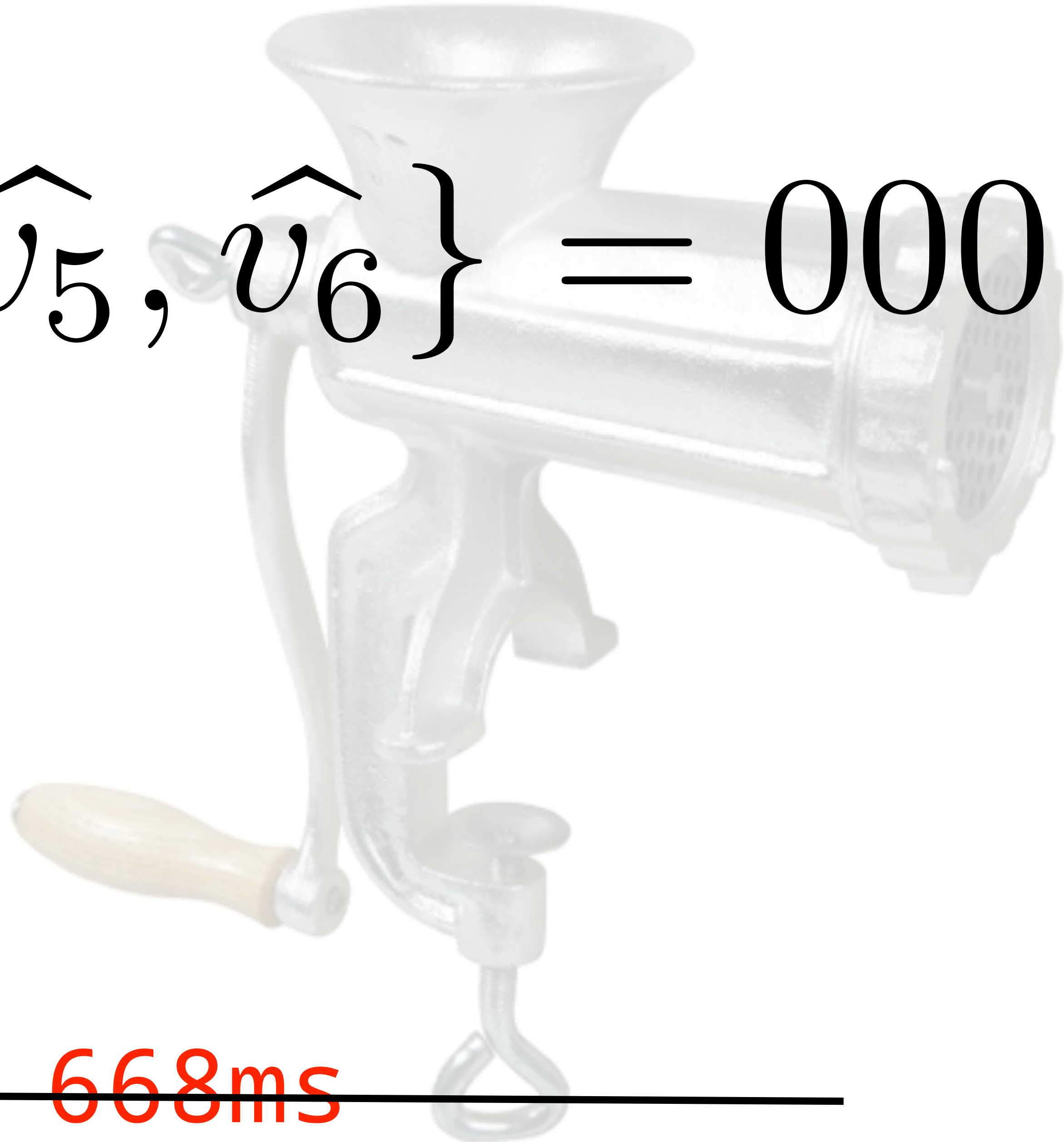
~~Time: 14212ms~~

Time: 668ms

$$\{\widehat{v_3}, \widehat{v_5}, \widehat{v_6}\} = 0001011$$



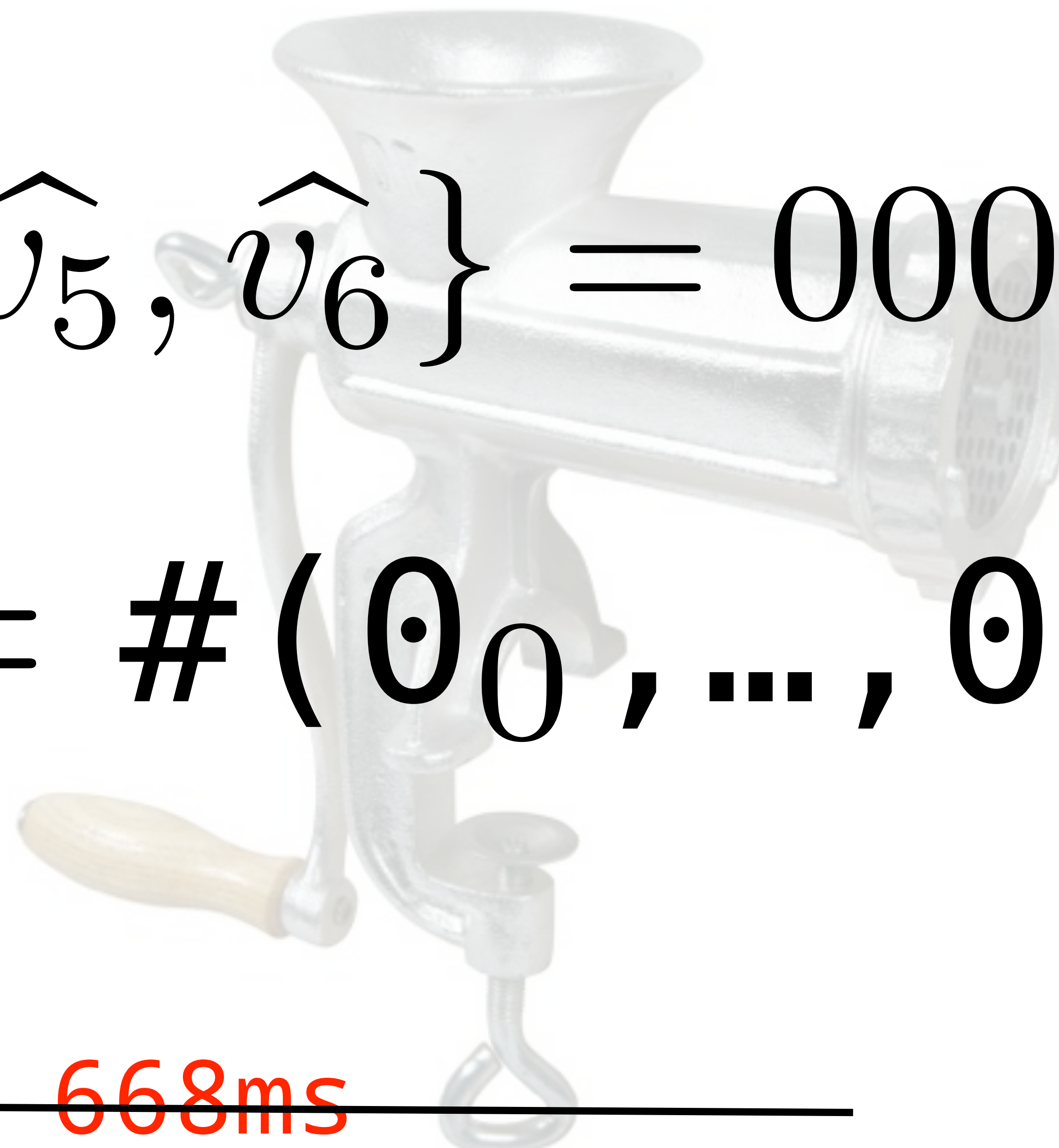
Time: 668ms



$$\{\widehat{v_3}, \widehat{v_5}, \widehat{v_6}\} = 0001011$$

~~Time: 668ms~~

Time: 342ms

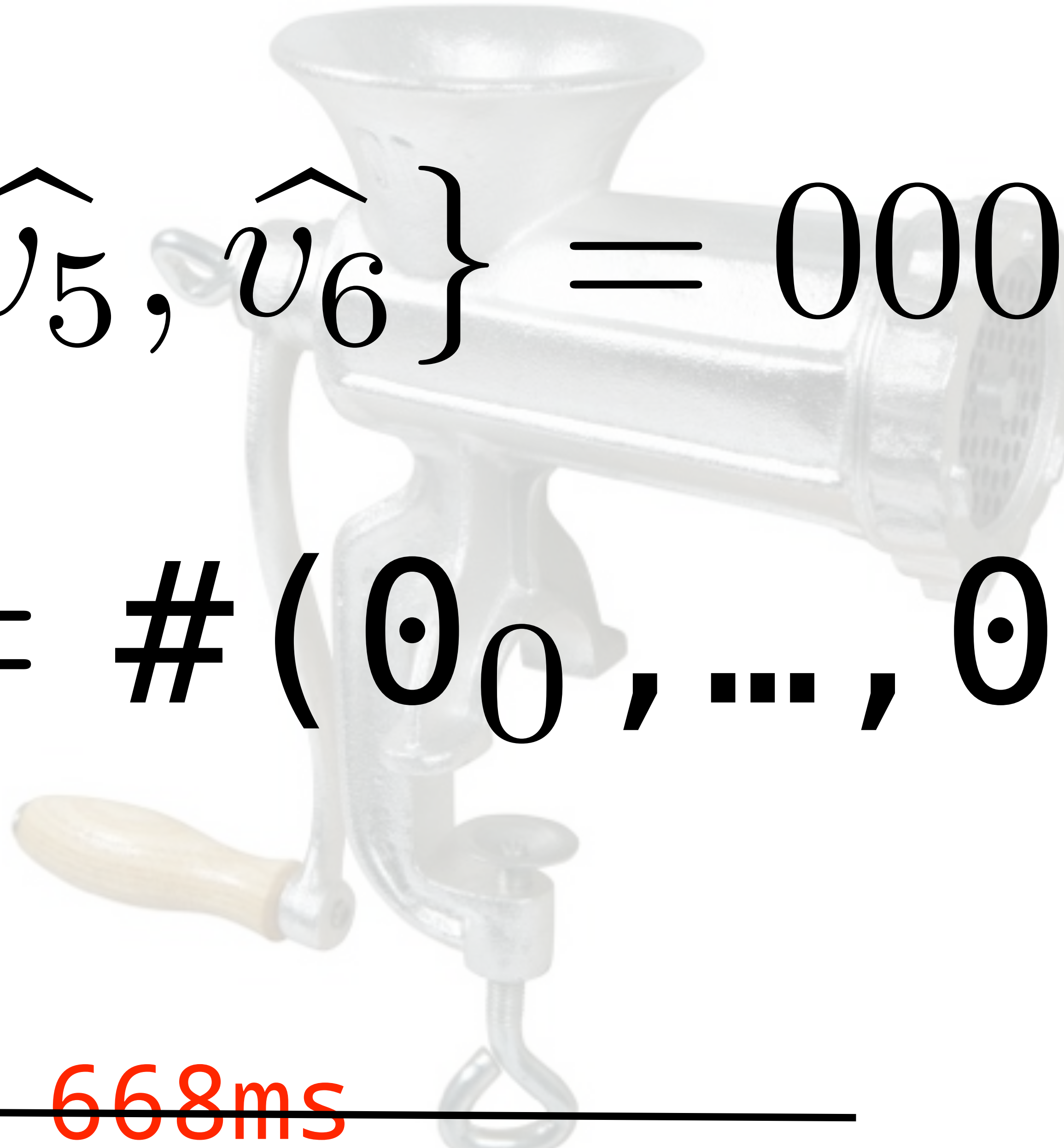


$$\{\widehat{v_3}, \widehat{v_5}, \widehat{v_6}\} = 0001011$$

$$\sigma = \#(0_0,\ldots,0_{|P|})$$

~~Time: 668ms~~

Time: 342ms



$$\{\widehat{v_3}, \widehat{v_5}, \widehat{v_6}\} = 0001011$$

$$\sigma = \#(0_0, \ldots, 0_{|P|})$$

~~Time: 668ms~~

~~Time: 342ms~~

Time: 112ms

Precision Preserving Recipe

- ★ Lazy non-determinism
- ★ Abstract compilation
- ★ Specialized fixpoint
- ★ Store diffs
- ★ Finite sets as bit vectors
- ★ Pre-allocation

Precision Preserving Recipe

≈ 5000x improvement

- ★ Lazy non-determinism
- ★ Abstract compilation
- ★ Specialized fixpoint
- ★ Store diffs
- ★ Finite sets as bit vectors
- ★ Pre-allocation

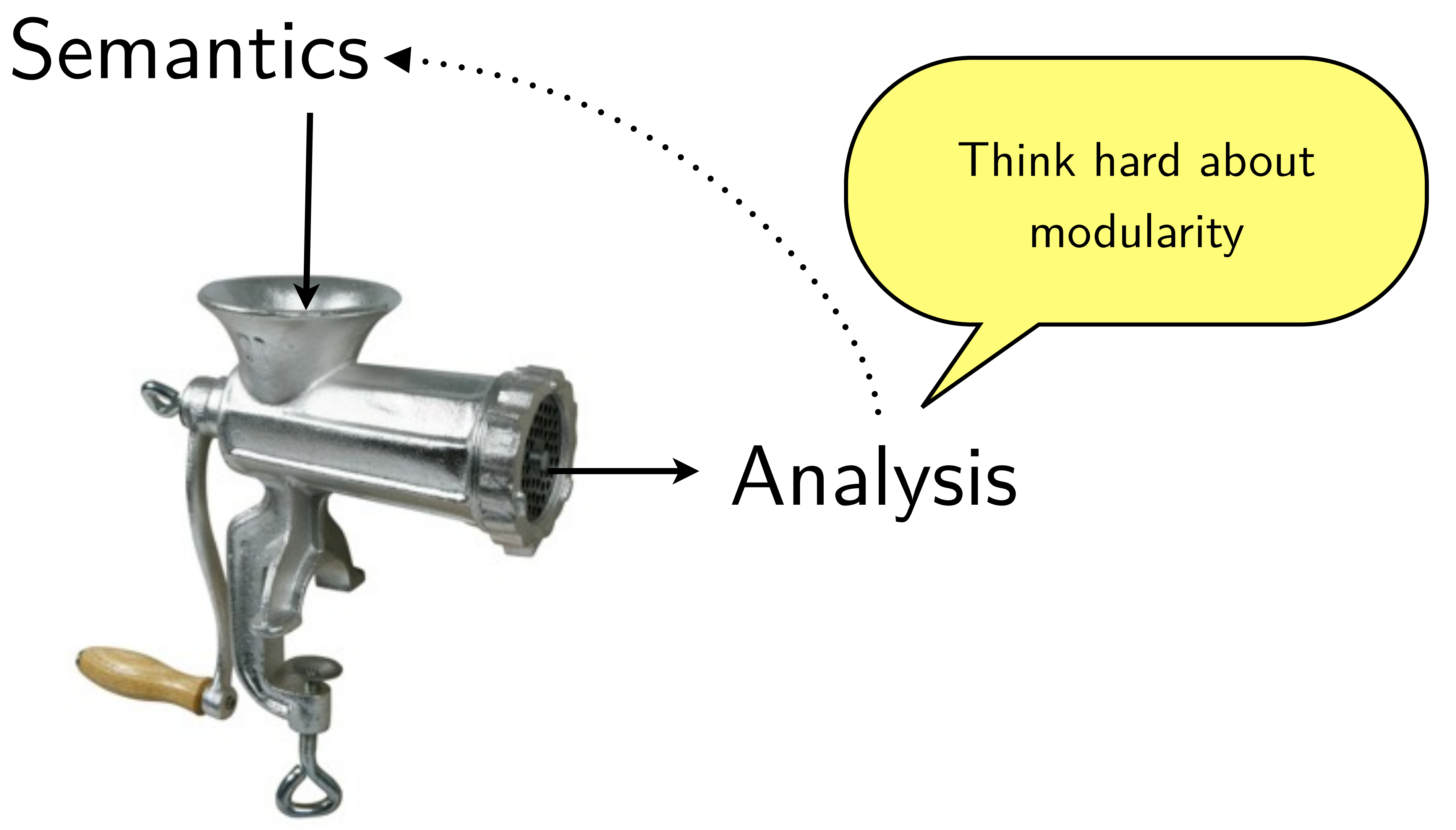
Modularity

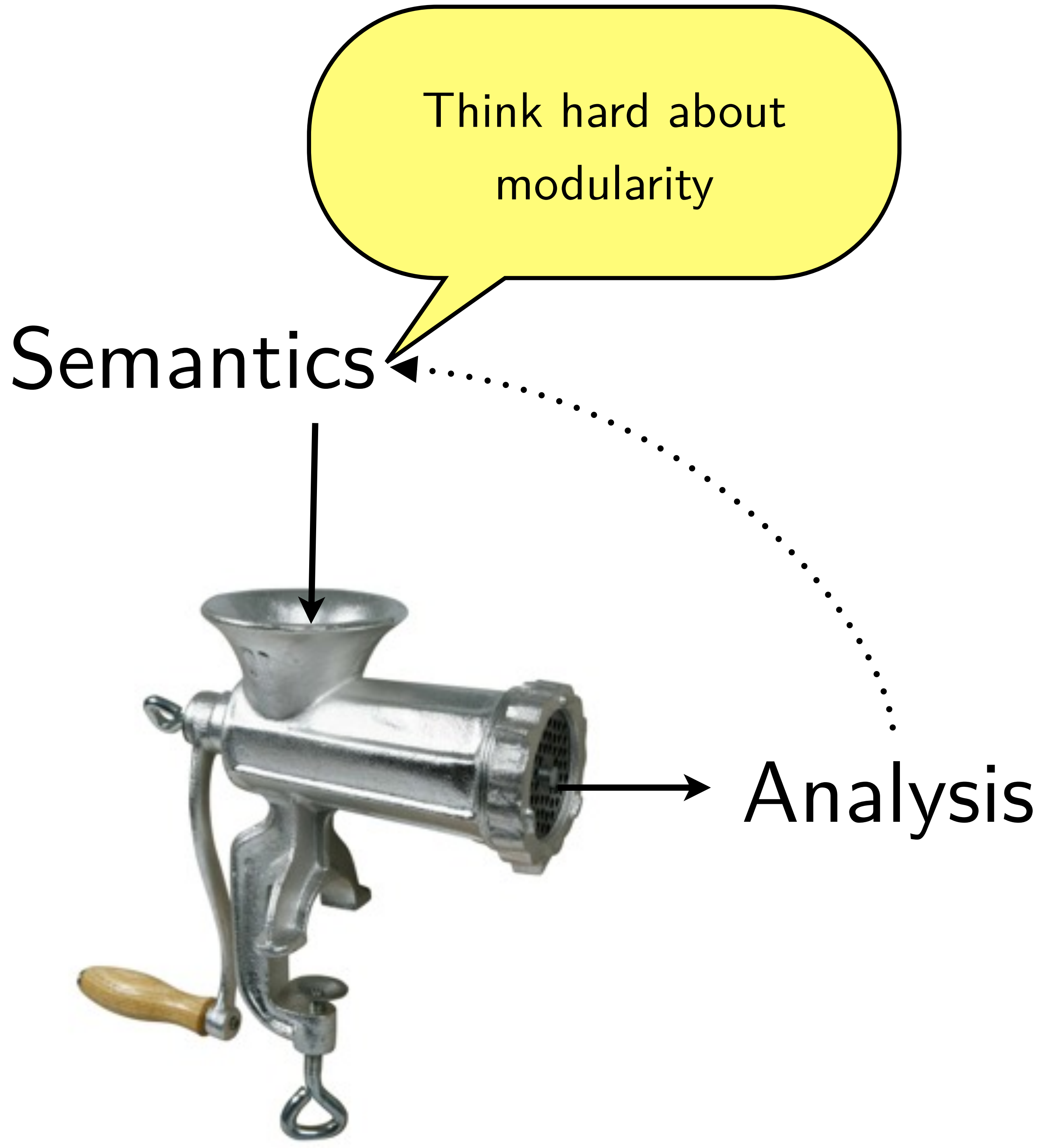
- ⋆ Some programs are open
- ⋆ Good components in bad languages
- ⋆ Programs are big; analysis is hard
- ⋆ Libraries matter

Analysis

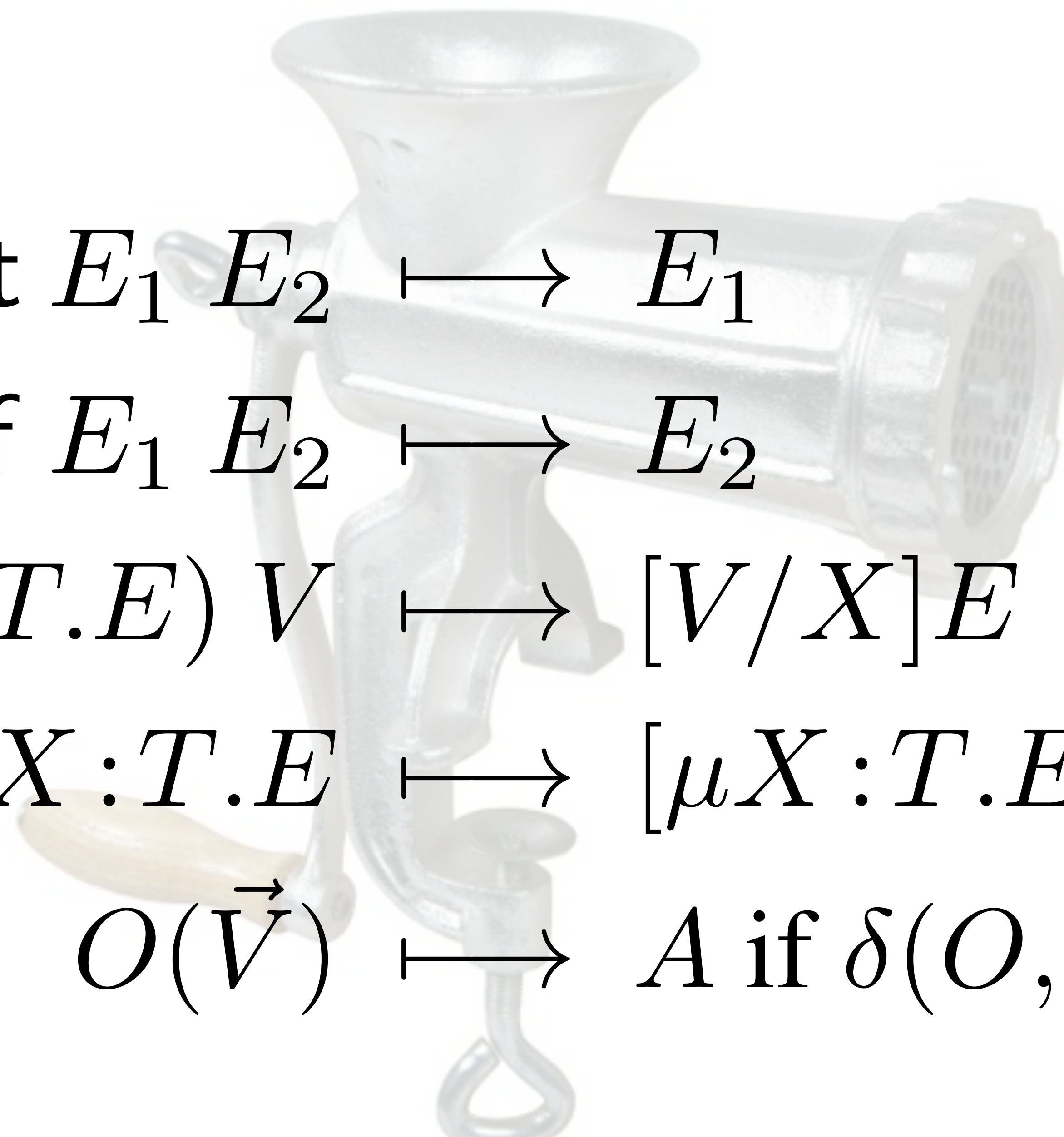
Think hard about modularity

Analysis


Semantics
Think hard about modularity
Analysis

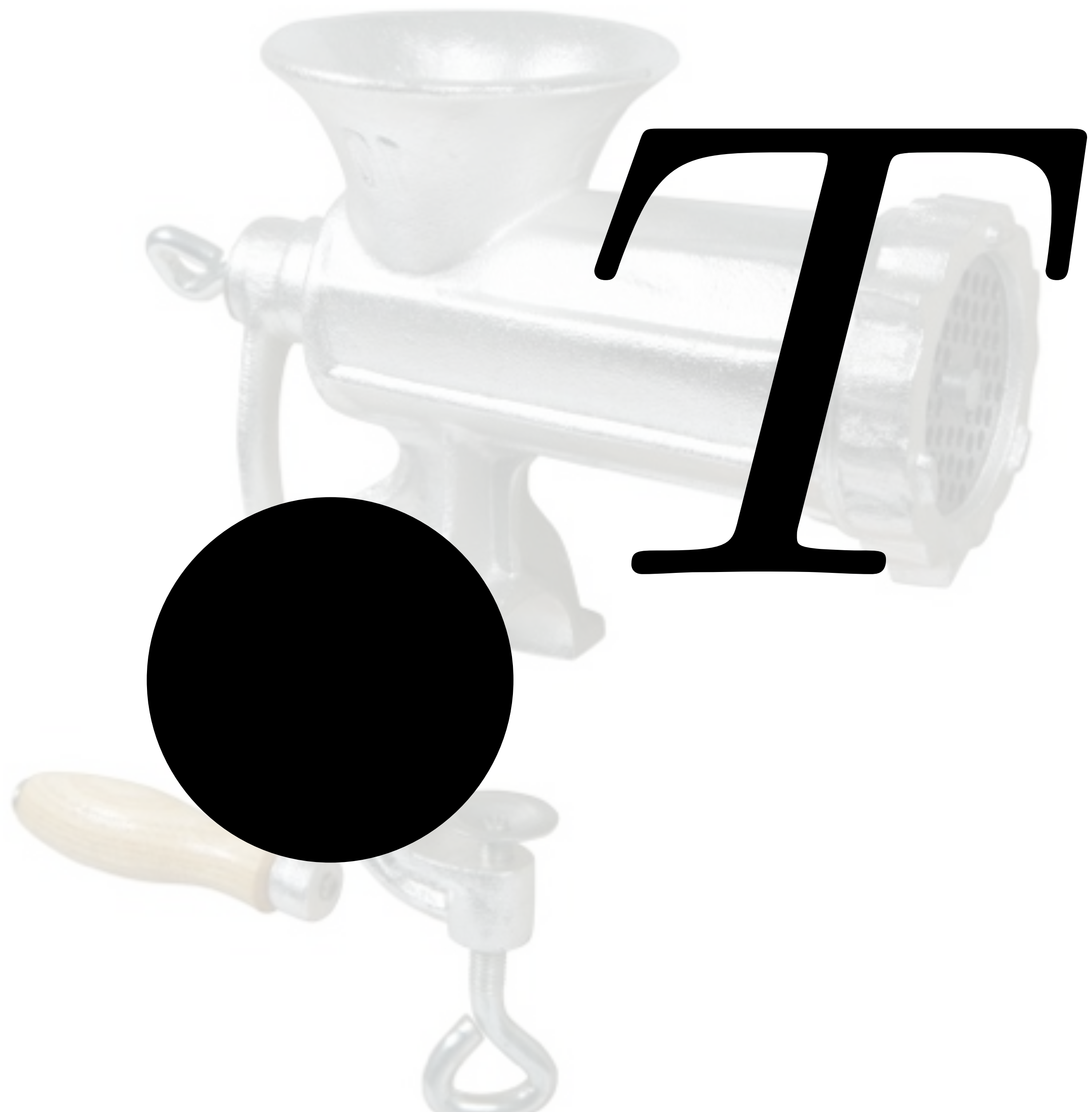

Think hard about modularity
Semantics
Analysis

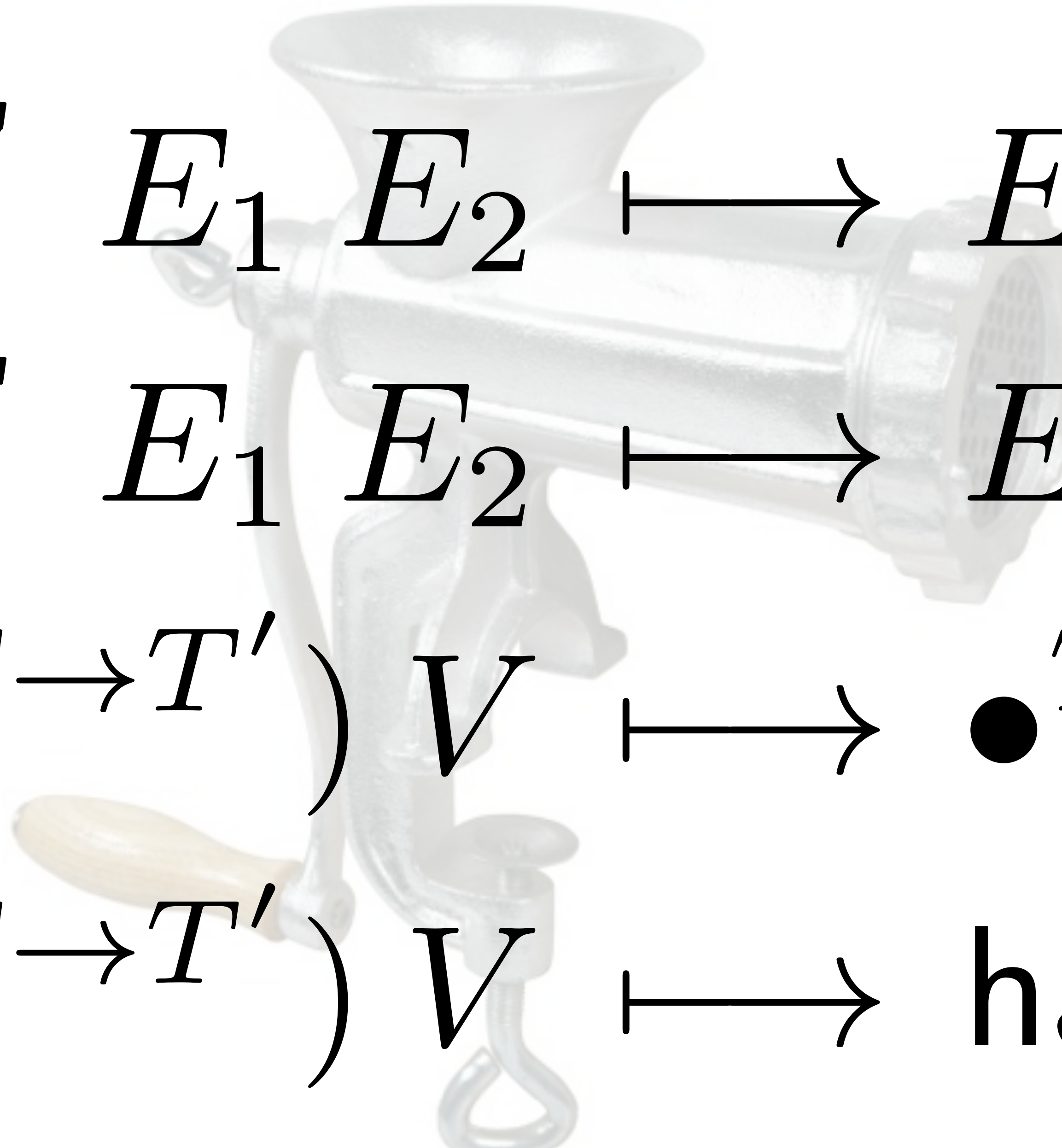
PCF



$$
\begin{aligned}
\mathsf{if}\ \mathsf{tt}\ E_1\ E_2 &\longmapsto E_1 \\
\mathsf{if}\ \mathsf{ff}\ E_1\ E_2 &\longmapsto E_2 \\
(\lambda X\!:\!T.E)\ V &\longmapsto [V/X]E \\
\mu X\!:\!T.E &\longmapsto [\mu X\!:\!T.E/X]E \\
O(\vec{V}) &\longmapsto A \text{ if } \delta(O, \vec{V}) = A
\end{aligned}
$$

Symbolic PCF


T

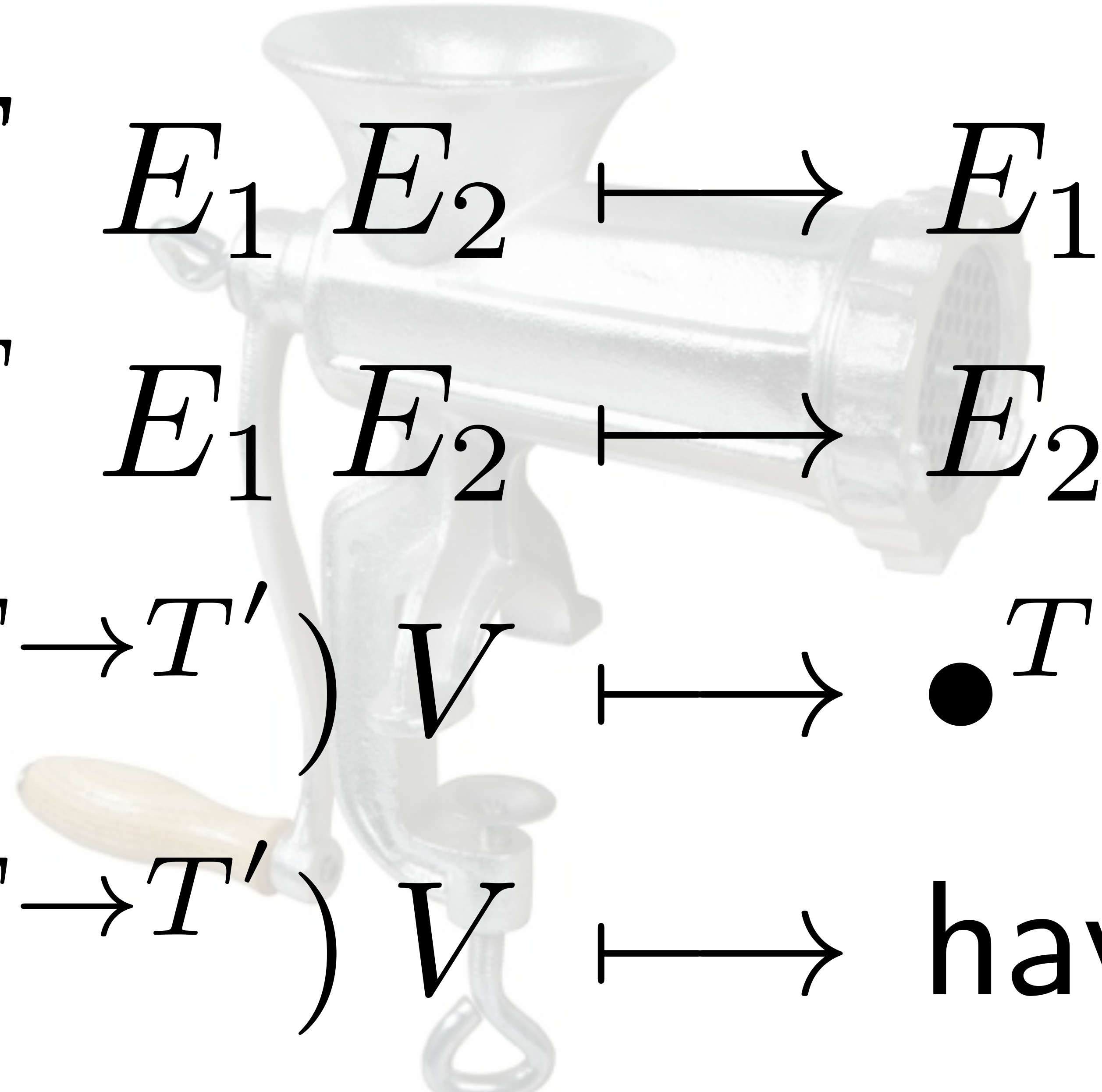


$$\mathsf{if}\ \bullet^{T}\ E_1\ E_2 \longmapsto E_1$$

$$\mathsf{if}\ \bullet^{T}\ E_1\ E_2 \longmapsto E_2$$

$$(\bullet^{T \to T'})\ V \longmapsto \bullet^{T'}$$

$$(\bullet^{T \to T'})\ V \longmapsto \mathsf{havoc}_T\ V$$



$$\text{if } \bullet^{T}\ E_1\, E_2 \longmapsto E_1$$

$$\text{if } \bullet^{T}\ E_1\, E_2 \longmapsto E_2$$

$$(\bullet^{T \to T'})\ V \longmapsto \bullet^{T'}$$

$$(\bullet^{T \to T'})\ V \longmapsto \mathsf{havoc}_T\ V$$

$$\mathsf{havoc}_B = \mu \mathsf{x}.\mathsf{x}$$

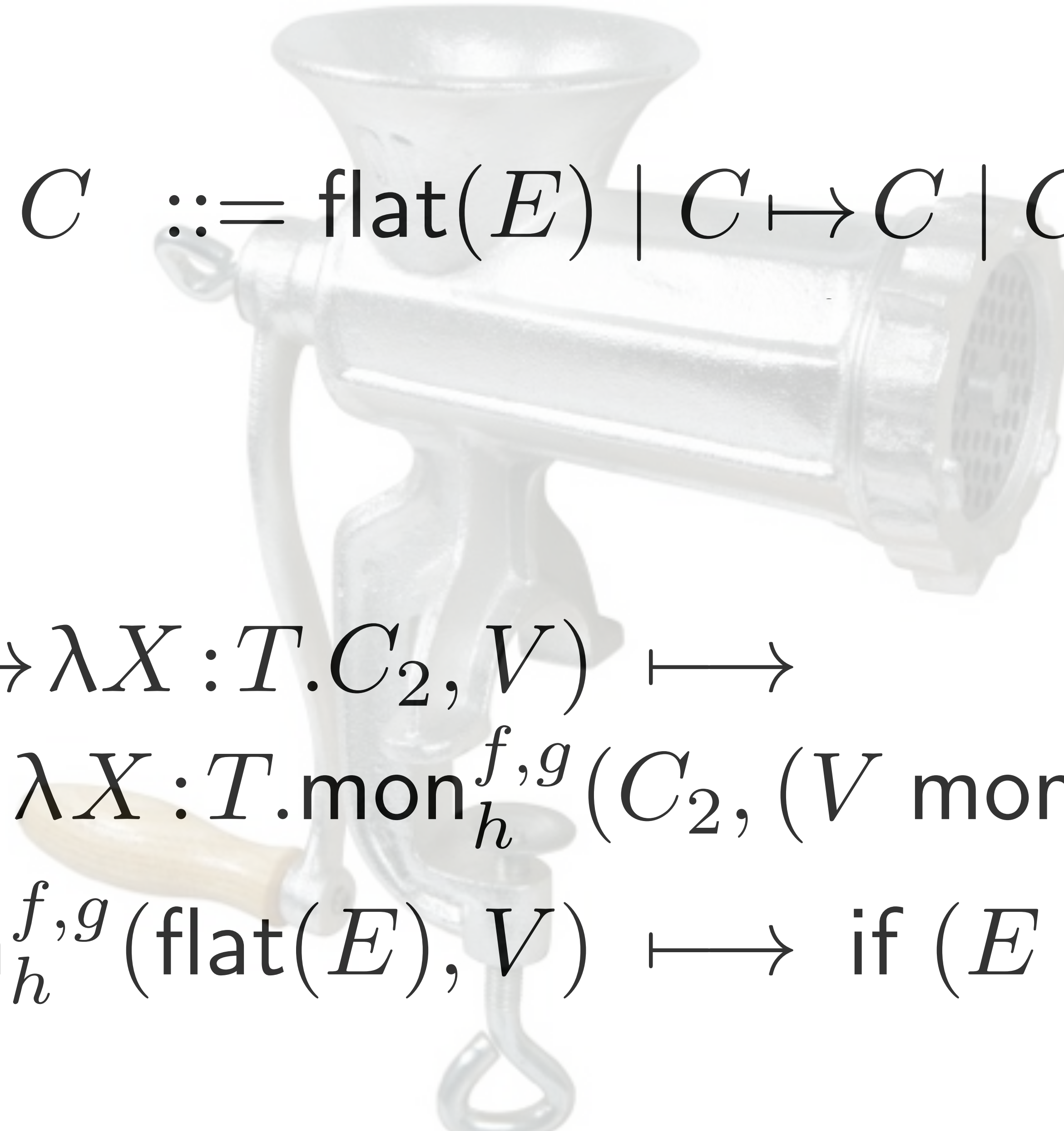
$$\mathsf{havoc}_{T \to T'} = \lambda \mathsf{x}\!:\!T \to T'.\mathsf{havoc}_{T'}(\mathsf{x}\, \bullet^{T})$$

$$
\begin{array}{rcl}
\textsf{if}\ \textsf{tt}\ E_1\ E_2 & \longmapsto & E_1 \\
\textsf{if}\ \textsf{ff}\ E_1\ E_2 & \longmapsto & E_2 \\
(\lambda X\!:\!T.E)\ V & \longmapsto & [V/X]E \\
\mu X\!:\!T.E & \longmapsto & [\mu X\!:\!T.E/X]E \\
O(\vec{V}) & \longmapsto & A\ \textsf{if}\ \delta(O,\vec{V}) = A \\
\textsf{if}\ \bullet^{T}\ E_1\ E_2 & \longmapsto & E_1 \\
\textsf{if}\ \bullet^{T}\ E_1\ E_2 & \longmapsto & E_2 \\
(\bullet^{T\to T'})\ V & \longmapsto & \bullet^{T'} \\
(\bullet^{T\to T'})\ V & \longmapsto & \textsf{havoc}_T\ V
\end{array}
$$

Semantics

Type-based Modular Analysis

Contract PCF

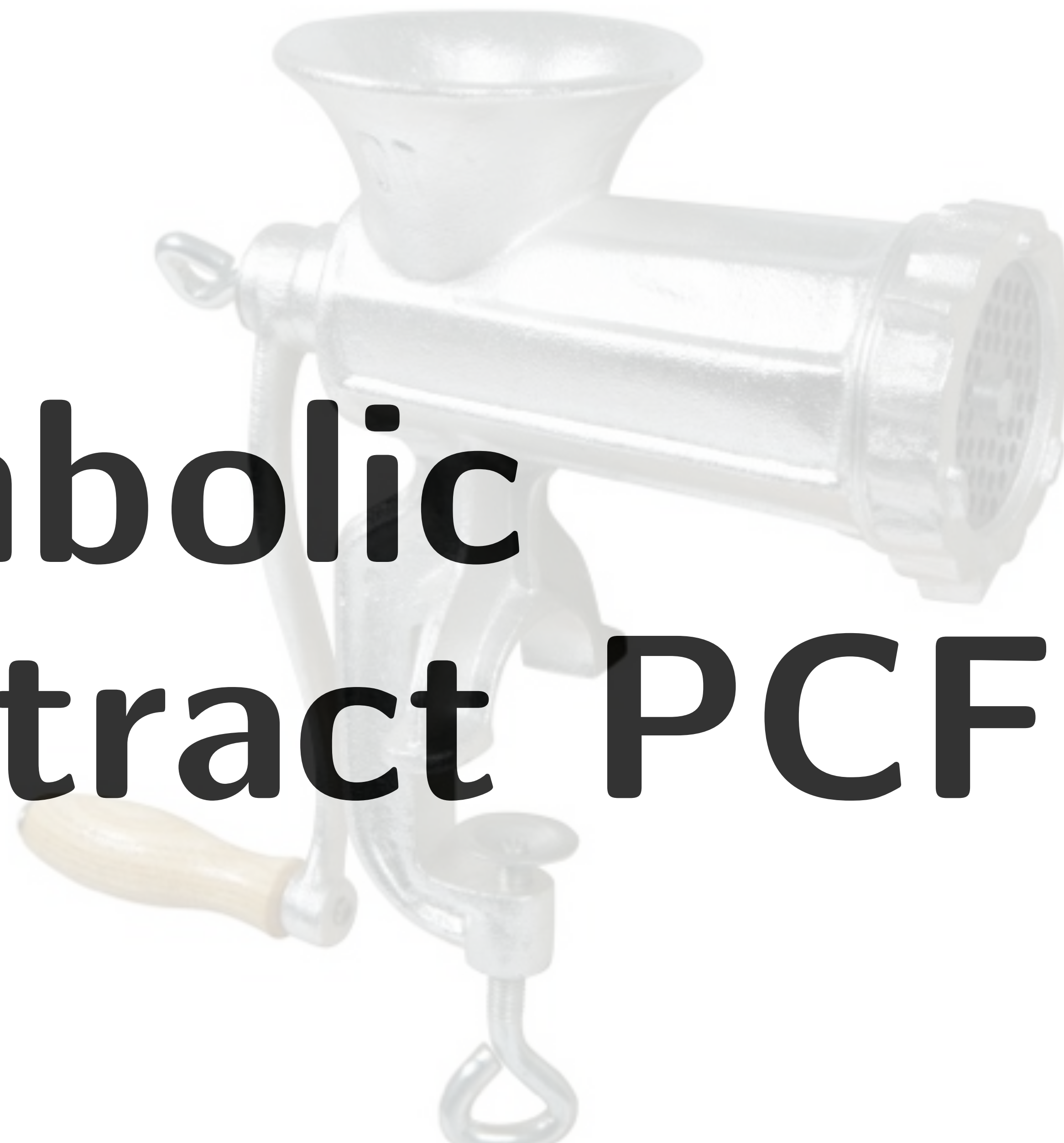


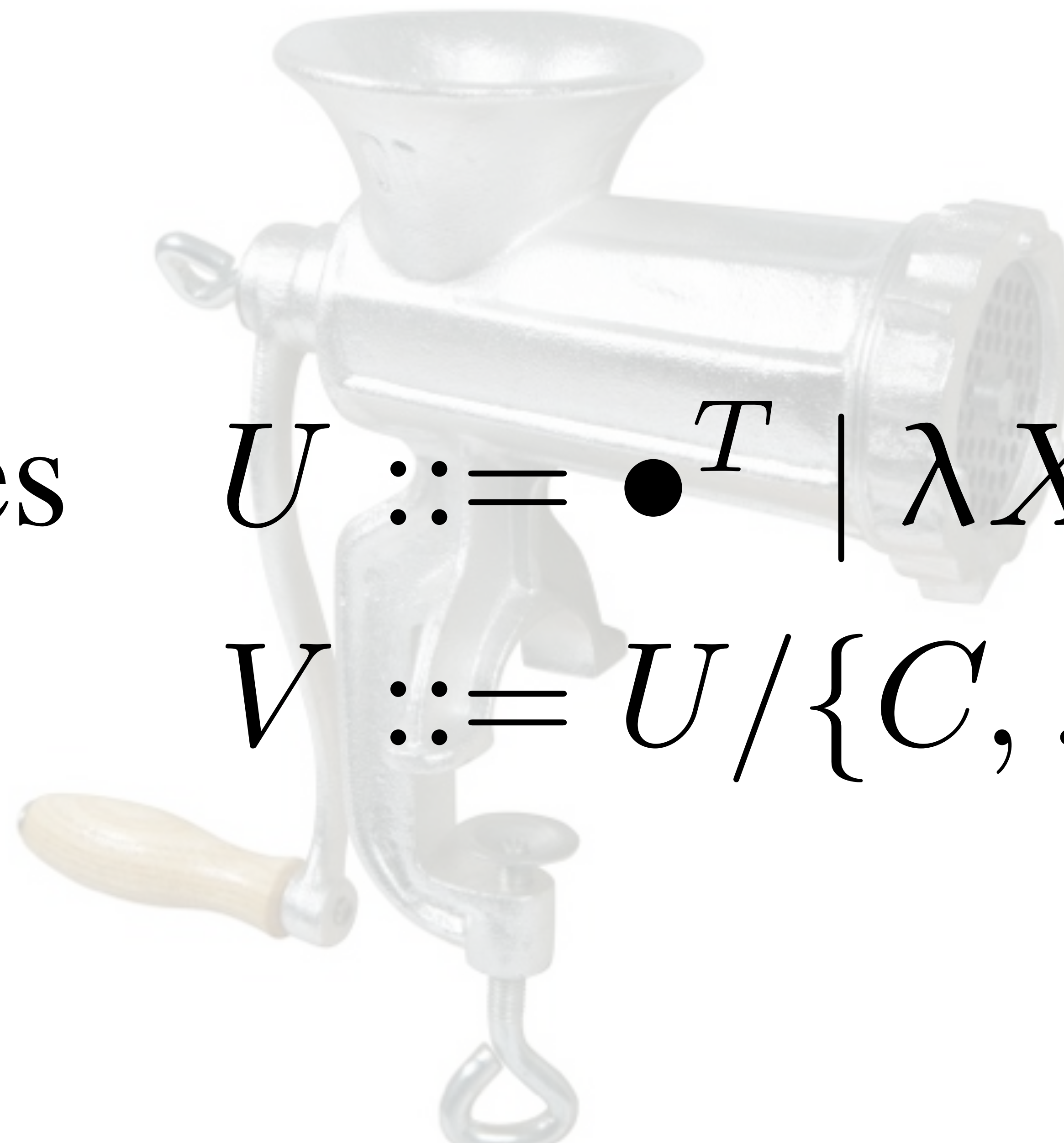
Contracts $\quad C \;::=\; \mathsf{flat}(E) \mid C \mapsto C \mid C \mapsto \lambda X{:}T.C$

$$\mathsf{mon}_h^{f,g}(C_1 \mapsto \lambda X{:}T.C_2, V) \longmapsto \lambda X{:}T.\mathsf{mon}_h^{f,g}(C_2, (V\ \mathsf{mon}_h^{g,f}(C_1, X)))$$

$$\mathsf{mon}_h^{f,g}(\mathsf{flat}(E), V) \longmapsto \mathsf{if}\ (E\ V)\ V\ \mathsf{blame}_h^f$$

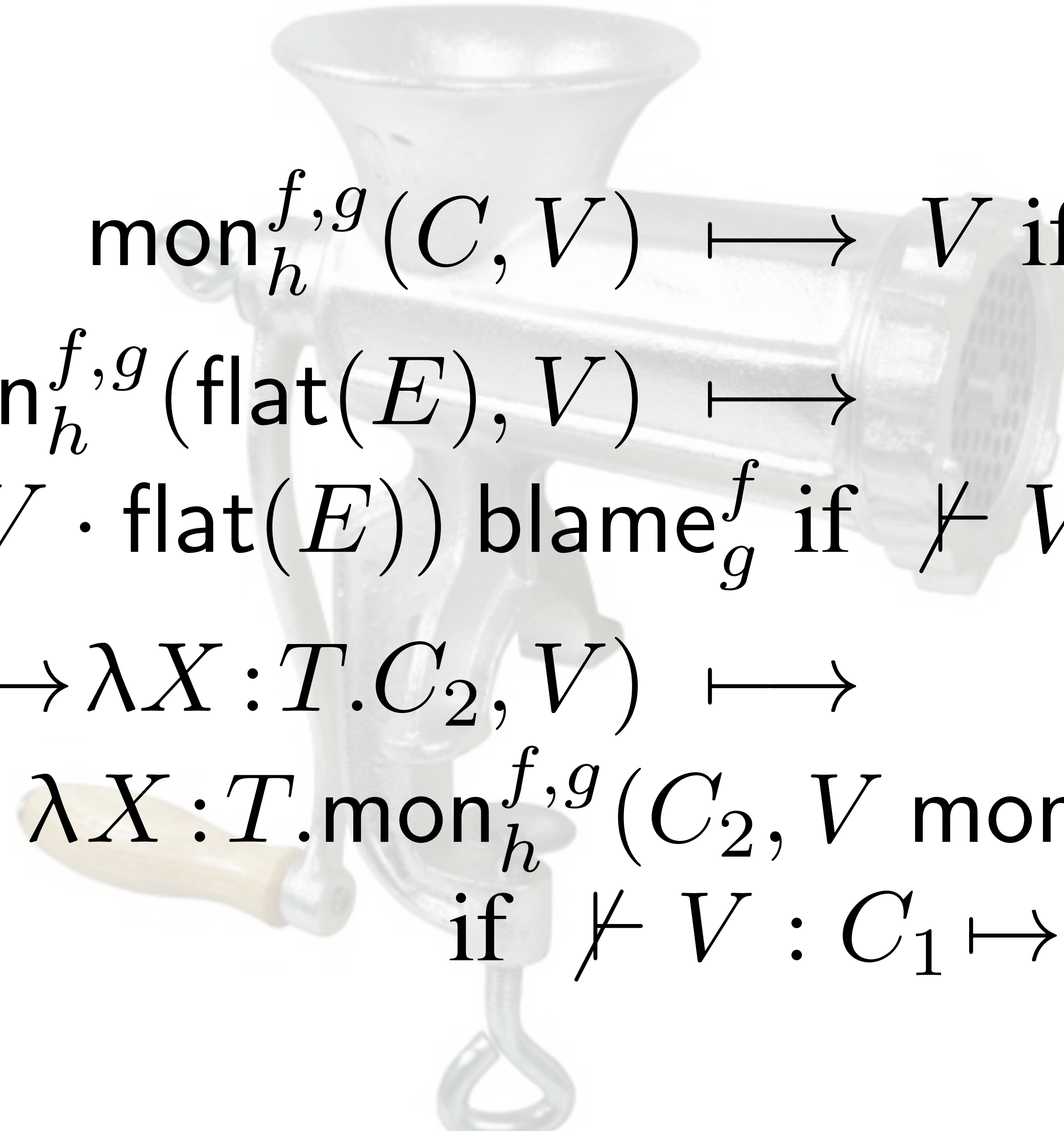
Symbolic Contract PCF





$$\text{Prevalues} \quad U ::= \bullet^{T} \mid \lambda X{:}T.E \mid 0 \mid 1 \mid$$

$$\text{Values} \quad V ::= U/\{C, \ldots\}$$



$$\mathsf{mon}_h^{f,g}(C, V) \;\longmapsto\; V \text{ if} \vdash V : C \;✓$$

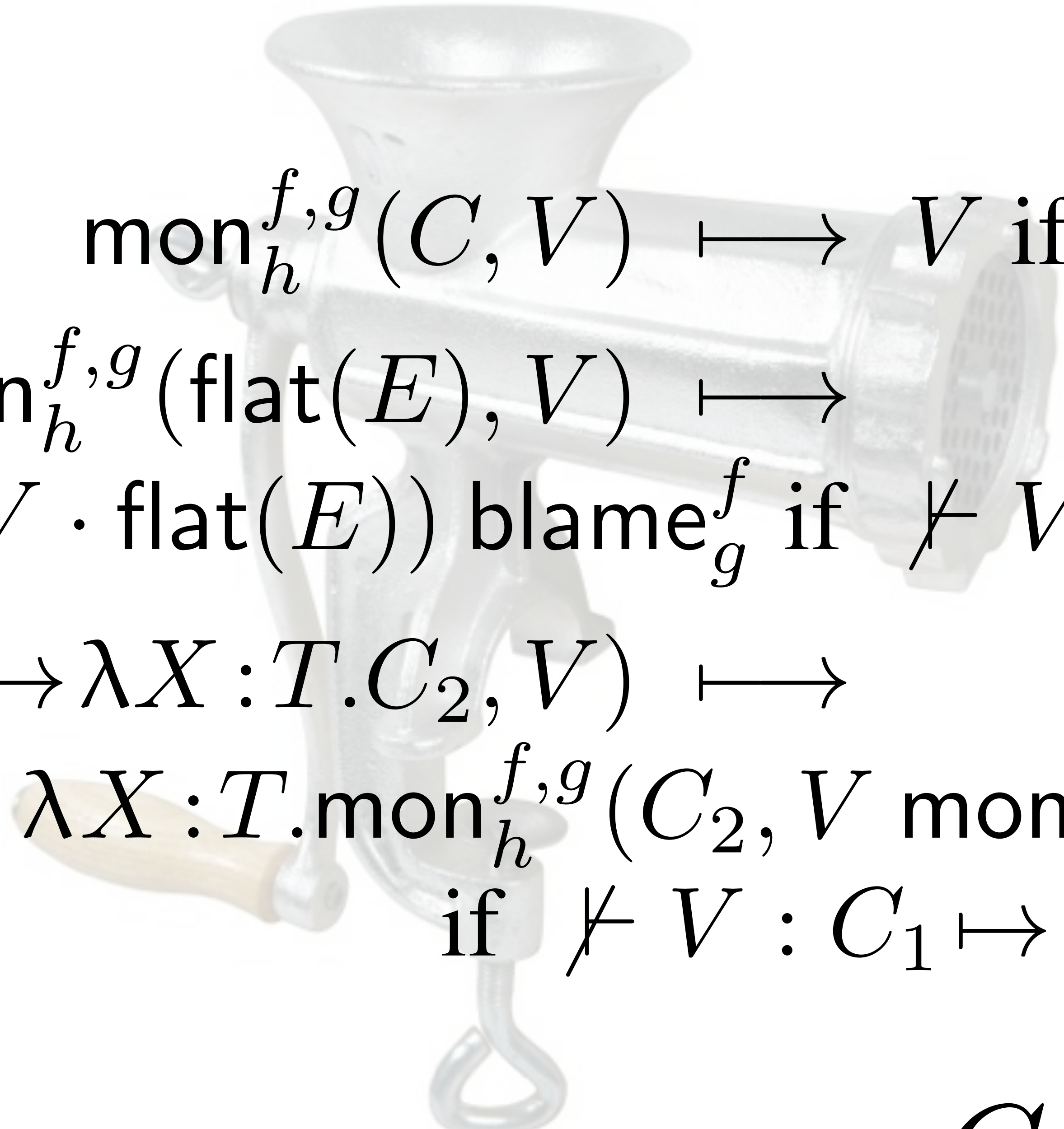
$$\mathsf{mon}_h^{f,g}(\mathsf{flat}(E), V) \;\longmapsto$$

$$\mathsf{if}\ (E\ V)\ (V \cdot \mathsf{flat}(E))\ \mathsf{blame}_g^f \text{ if } \nvdash V : \mathsf{flat}(E) \;✓$$

$$\mathsf{mon}_h^{f,g}(C_1 \mapsto \lambda X{:}T.C_2, V) \;\longmapsto$$

$$\lambda X{:}T.\mathsf{mon}_h^{f,g}(C_2, V\ \mathsf{mon}_h^{g,f}(C_1, X))$$

$$\text{if } \nvdash V : C_1 \mapsto \lambda X{:}T.C_2 \;✓$$

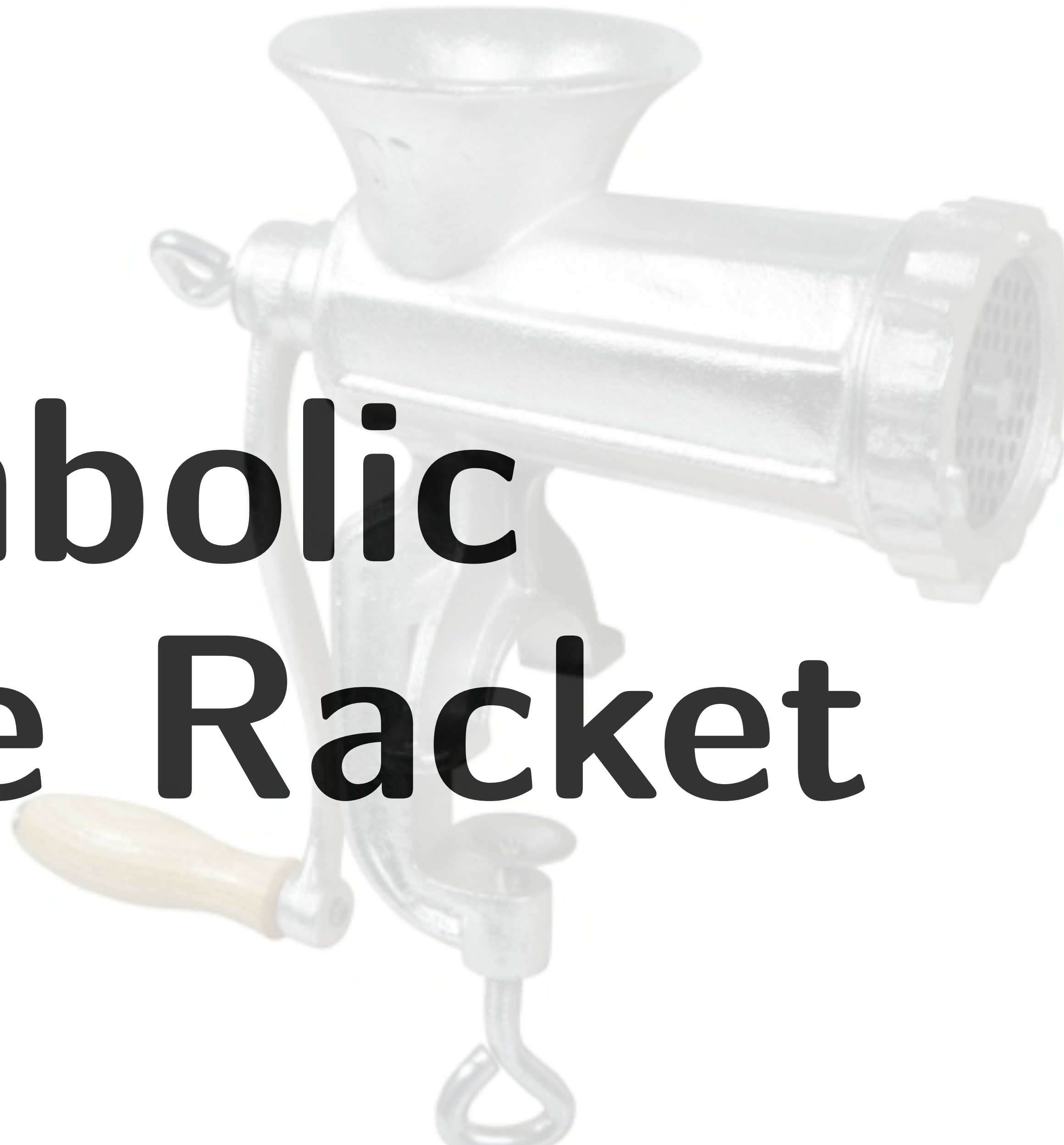


$$\mathsf{mon}_h^{f,g}(C,V) \longmapsto V \text{ if} \vdash V : C\,✓$$

$$\mathsf{mon}_h^{f,g}(\mathsf{flat}(E),V) \longmapsto$$
$$\mathsf{if}\ (E\,V)\ (V \cdot \mathsf{flat}(E))\ \mathsf{blame}_g^f \text{ if } \not\vdash V : \mathsf{flat}(E)\,✓$$

$$\mathsf{mon}_h^{f,g}(C_1 \mapsto \lambda X{:}T.C_2, V) \longmapsto$$
$$\lambda X{:}T.\mathsf{mon}_h^{f,g}(C_2, V\ \mathsf{mon}_h^{g,f}(C_1, X))$$
$$\text{if } \not\vdash V : C_1 \mapsto \lambda X{:}T.C_2\,✓$$

$$\frac{C \in \mathcal{C}}{\vdash V/\mathcal{C} : C\,✓}$$



Symbolic Core Racket

$$
\begin{array}{lcl}
P, Q & ::= & \vec{M}\,E \\
M, N & ::= & (\mathsf{module}\ f\ C\ V) \\
E, E' & ::= & f^\ell \mid X \mid A \mid E\ E^\ell \mid \mathsf{if}\ E\ E\ E \mid O\ \vec{E}^\ell \mid \mu X.E \\
 & \mid & \mathsf{mon}_\ell^{\ell,\ell}(C, E) \\
U & ::= & n \mid \mathsf{tt} \mid \mathsf{ff} \mid (\lambda X.E) \mid \bullet \mid (V,V) \mid \mathsf{empty} \\
V & ::= & U/\mathcal{C} \\
C, D & ::= & X \mid C \mapsto \lambda X.C \mid \mathsf{flat}(E) \\
 & \mid & \langle C,C\rangle \mid C \vee C \mid C \wedge C \mid \mu X.C \\
O & ::= & \mathsf{add1} \mid \mathsf{car} \mid \mathsf{cdr} \mid \mathsf{cons} \mid + \mid = \mid o? \mid \ldots \\
o? & ::= & \mathsf{nat?} \mid \mathsf{bool?} \mid \mathsf{empty?} \mid \mathsf{cons?} \mid \mathsf{proc?} \mid \mathsf{false?} \\
A & ::= & V \mid \mathcal{E}[\mathsf{blame}^\ell_\ell]
\end{array}
$$

$$
\begin{array}{rcll}
((\lambda X.E)\ V)^\ell & \longmapsto & [V/X]E & \\
(V\ V')^\ell & \longmapsto & \mathsf{blame}^\ell_\Lambda & \text{if } \delta(\mathsf{proc?}, V) \ni \mathsf{ff} \\
(O\ \vec{V})^\ell & \longmapsto & A & \text{if } \delta(O^\ell, \vec{V}) \ni A \\
\mathsf{if}\ V\ E\ E' & \longmapsto & E & \text{if } \delta(\mathsf{false?}, V) \ni \mathsf{ff} \\
\mathsf{if}\ V\ E\ E' & \longmapsto & E' & \text{if } \delta(\mathsf{false?}, V) \ni \mathsf{tt}
\end{array}
$$

$$
\begin{array}{l}
\delta(\mathsf{add1}, n) \ni n+1 \\
\delta(+, n, m) \ni n+m \\
\delta(\mathsf{car}, (V, V')) \ni V \\
\delta(\mathsf{cdr}, (V, V')) \ni V'
\end{array}
$$

$$
\begin{array}{lcl}
\vdash V : o?\checkmark & \Longrightarrow & \delta(o?, V) \ni \mathsf{tt} \\
\vdash V : o?\,\text{✗} & \Longrightarrow & \delta(o?, V) \ni \mathsf{ff} \\
\vdash V : o?\,\mathbf{?} & \Longrightarrow & \delta(o?, V) \ni \bullet/\{\mathsf{flat}(\mathsf{bool?})\} \\
\vdash V : \mathsf{nat?}\checkmark & \Longrightarrow & \delta(\mathsf{add1}, V) \ni \bullet/\{\mathsf{flat}(\mathsf{nat?})\} \\
\vdash V : \mathsf{nat?}\,\text{✗} & \Longrightarrow & \delta(\mathsf{add1}^\ell, V) \ni \mathsf{blame}^\ell_{\mathsf{add1}} \\
\vdash V : \mathsf{nat?}\,\mathbf{?} & \Longrightarrow & \delta(\mathsf{add1}, V) \ni \bullet/\mathsf{flat}(\mathsf{nat?}) \\
 & & \wedge\ \delta(\mathsf{add1}^\ell, V) \ni \mathsf{blame}^\ell_{\mathsf{add1}} \\
\vdash V : \mathsf{cons?}\checkmark & \Longrightarrow & \delta(\mathsf{car}, V) \ni \pi_1(V) \\
\vdash V : \mathsf{cons?}\,\text{✗} & \Longrightarrow & \delta(\mathsf{car}^\ell, V) \ni \mathsf{blame}^\ell_{\mathsf{car}} \\
\vdash V : \mathsf{cons?}\,\mathbf{?} & \Longrightarrow & \delta(\mathsf{car}, V) \ni \pi_1(V) \\
 & & \wedge\ \delta(\mathsf{car}^\ell, V) \ni \mathsf{blame}^\ell_{\mathsf{car}} \\
\text{otherwise} & & \delta(O^\ell, \vec{V}) \ni \mathsf{blame}^\ell_\Lambda
\end{array}
$$

$$
\begin{array}{ll}
\vec{M} \vdash f^f \longmapsto V & \text{if } (\mathsf{module}\ f\ C\ V) \in \vec{M} \\
\vec{M} \vdash f^g \longmapsto \mathsf{mon}_f^{f,g}(C, V) & \text{if } (\mathsf{module}\ f\ C\ V) \in \vec{M} \\
\vec{M} \vdash f^g \longmapsto \mathsf{mon}_f^{f,g}(C, \bullet \cdot C) & \text{if } (\mathsf{module}\ f\ C\ \bullet) \in \vec{M}
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{mon}_h^{f,g}(C, V) \longmapsto V \cdot C & \text{if } C \text{ is flat and} \vdash V : C\checkmark \\
\mathsf{mon}_h^{f,g}(C, V) \longmapsto \mathsf{blame}^f_h & \text{if } C \text{ is flat and} \vdash V : C\,\text{✗} \\
\mathsf{mon}_h^{f,g}(C, V) \longmapsto \mathsf{if}\ (\textsc{fc}(C)\ V)\ (V \cdot C)\ \mathsf{blame}^f_h & \\
 & \text{if } C \text{ is flat and} \vdash V : C\,\mathbf{?}
\end{array}
$$

$$
\begin{array}{rcl}
\textsc{fc}(\mu X.C) & = & \mu X.\textsc{fc}(C) \\
\textsc{fc}(X) & = & X \\
\textsc{fc}(\mathsf{flat}(E)) & = & E \\
\textsc{fc}(C_1 \wedge C_2) & = & \lambda y.\mathsf{if}\ (\textsc{fc}(C_1)\ y)\ (\textsc{fc}(C_2)\ y)\ \mathsf{ff} \\
\textsc{fc}(C_1 \vee C_2) & = & \lambda y.\mathsf{if}\ (\textsc{fc}(C_1)\ y)\ \mathsf{tt}\ (\textsc{fc}(C_2)\ y) \\
\textsc{fc}(\langle C_1, C_2\rangle) & = & \\
\multicolumn{3}{l}{\lambda y.(\mathsf{and}\ (\mathsf{cons?}\ y)\ (\textsc{fc}(C_1)\ (\mathsf{car}\ y))\ (\textsc{fc}(C_2)\ (\mathsf{cdr}\ y)))}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{mon}_h^{f,g}(C \mapsto \lambda X.D, V) \longmapsto \\
\qquad (\lambda X.\mathsf{mon}_h^{f,g}(D, (V\ \mathsf{mon}_h^{g,f}(C, X)))) \\
\qquad\qquad \text{if } \delta(\mathsf{proc?}, V) \ni \mathsf{tt} \\
\mathsf{mon}_h^{f,g}(C \mapsto \lambda X.D, V) \longmapsto \mathsf{blame}^f_h \quad \text{if } \delta(\mathsf{proc?}, V) \ni \mathsf{ff}
\end{array}
$$

$$
\begin{array}{lll}
\mathsf{mon}(\langle C, D\rangle, V) & \longmapsto & \\
\multicolumn{3}{r}{(\mathsf{cons}\ \mathsf{mon}(C, \mathsf{car}\ V')\ \mathsf{mon}(D, \mathsf{cdr}\ V'))} \\
\multicolumn{3}{r}{\text{if } \delta(\mathsf{cons?}, V) \ni \mathsf{tt} \text{ and } V' = V \cdot \mathsf{flat}(\mathsf{cons?})} \\
\mathsf{mon}_h^{f,g}(\langle C, D\rangle, V) & \longmapsto \mathsf{blame}^f_h & \text{if } \delta(\mathsf{cons?}, V) \ni \mathsf{ff} \\
\mathsf{mon}(\mu X.C, V) & \longmapsto \mathsf{mon}([\mu X.C/X]C, V) & \\
\mathsf{mon}(C \wedge D, V) & \longmapsto \mathsf{mon}(D, \mathsf{mon}(C, V)) & \\
\mathsf{mon}(C \vee D, V) & \longmapsto \mathsf{if}\ (\textsc{fc}(C)\ V)\ (V \cdot C)\ \mathsf{mon}(D, V) & \\
 & & \text{if } \vdash V : C\,\mathbf{?} \\
\mathsf{mon}(C \vee D, V) & \longmapsto V & \text{if } \vdash V : C\checkmark \\
\mathsf{mon}(C \vee D, V) & \longmapsto \mathsf{mon}(D, V) & \text{if } \vdash V : C\,\text{✗}
\end{array}
$$

Modular Contract Analysis

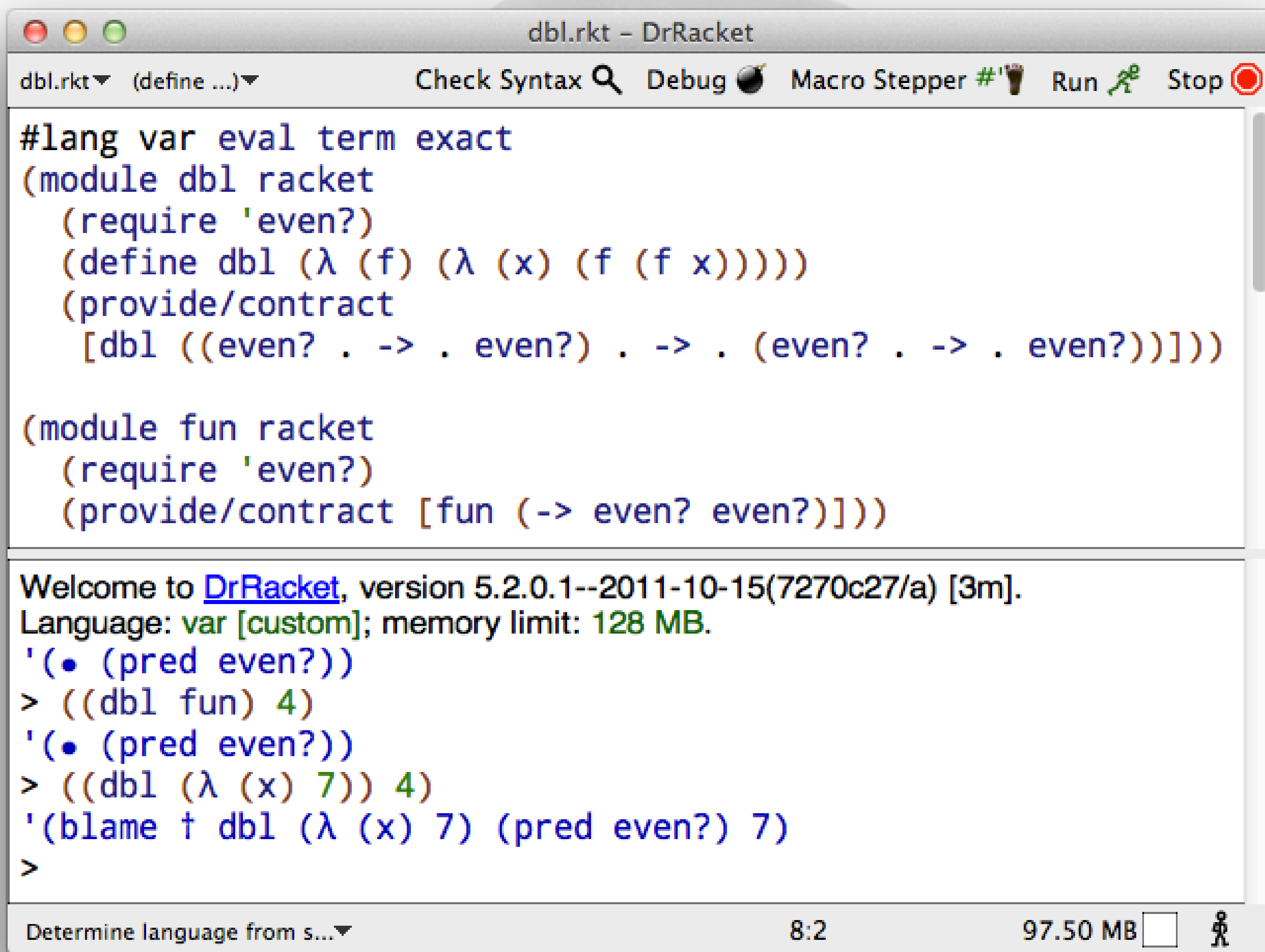
$$
\begin{array}{lcl}
\widehat{V}\ V' & \longmapsto & \bullet/\{[\widehat{V}/X]D \mid (C \mapsto \lambda X.D) \in \mathcal{C}\} \\
 & & \qquad \text{if } \delta(\mathsf{proc?}, \widehat{V}) \ni \mathsf{tt} \\
\widehat{V}\ V' & \longmapsto & \mathsf{havoc}\ V' \qquad \text{if } \delta(\mathsf{proc?}, \widehat{V}) \ni \mathsf{tt}
\end{array}
$$

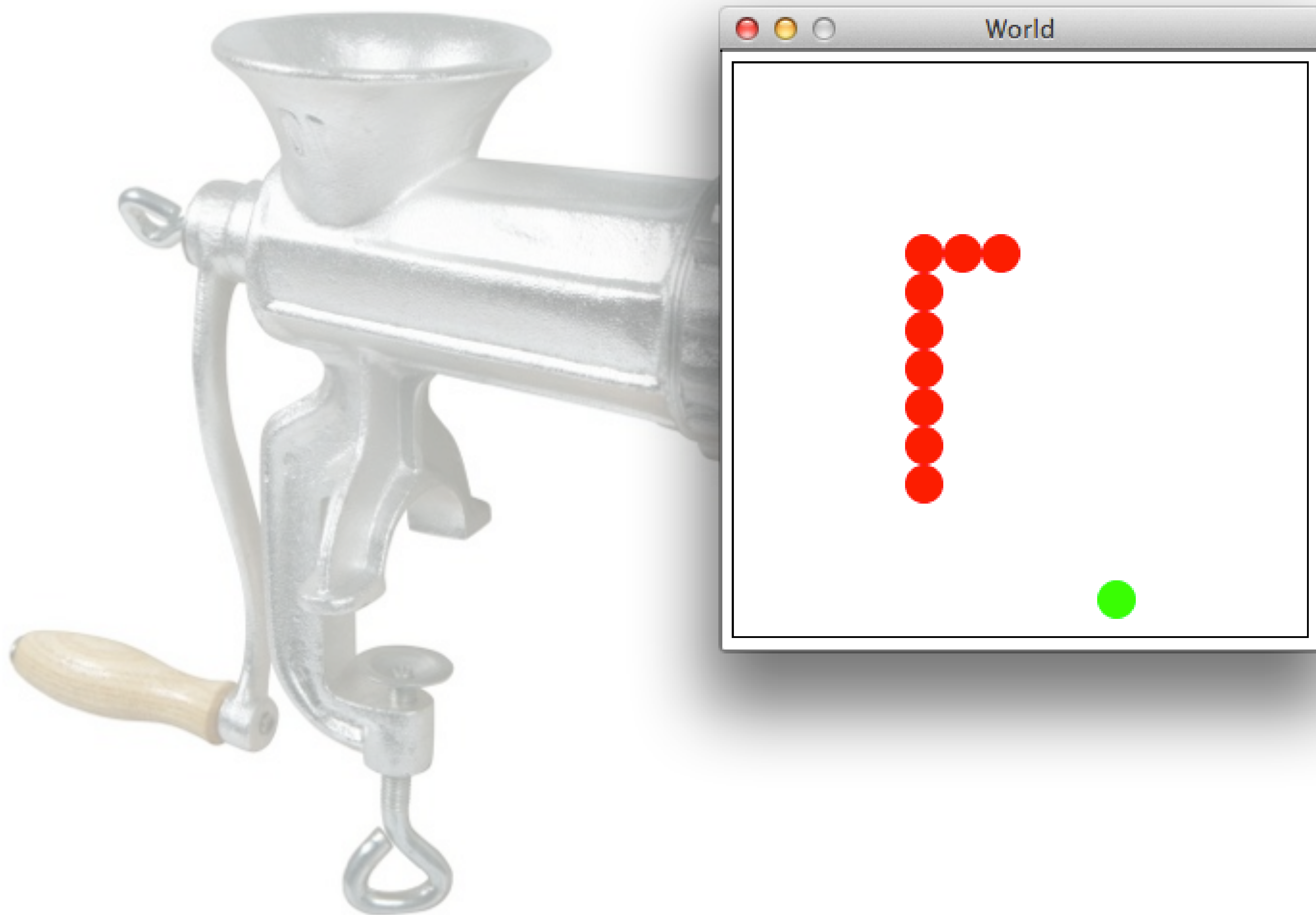
$$
\mathsf{havoc} = \mu \mathsf{y}.(\lambda \mathsf{x}.\textsc{amb}(\{\mathsf{y}\ (\mathsf{x}\ \bullet), \mathsf{y}\ (\mathsf{car}\ \mathsf{x}), \mathsf{y}\ (\mathsf{cdr}\ \mathsf{x})\}))
$$

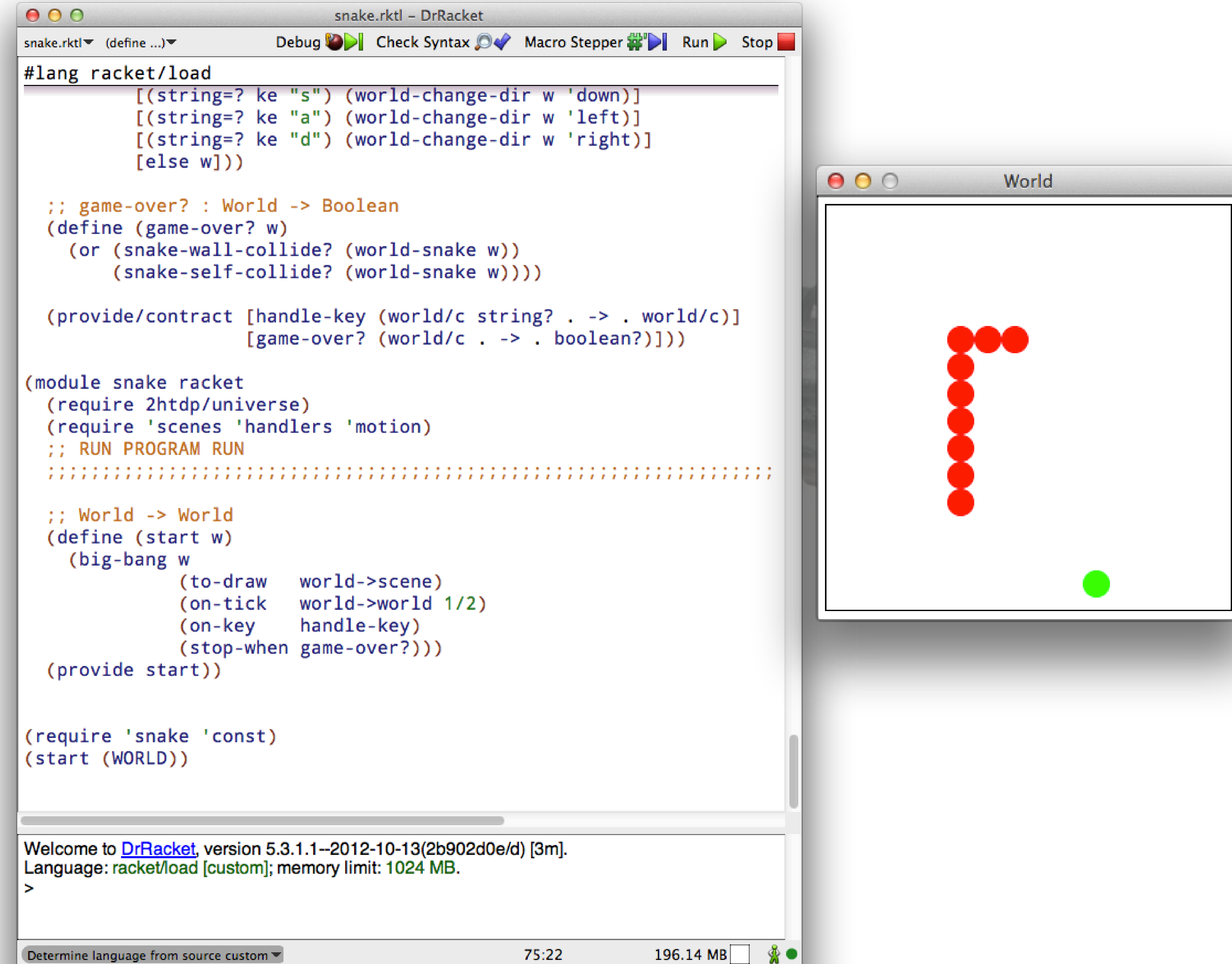
$$
\begin{array}{lcl}
\textsc{amb}(\{E\}) & = & E \\
\textsc{amb}(\{E, E_1, \ldots\}) & = & \mathsf{if}\ \bullet\ E\ \textsc{amb}(\{E_1, \ldots\})
\end{array}
$$

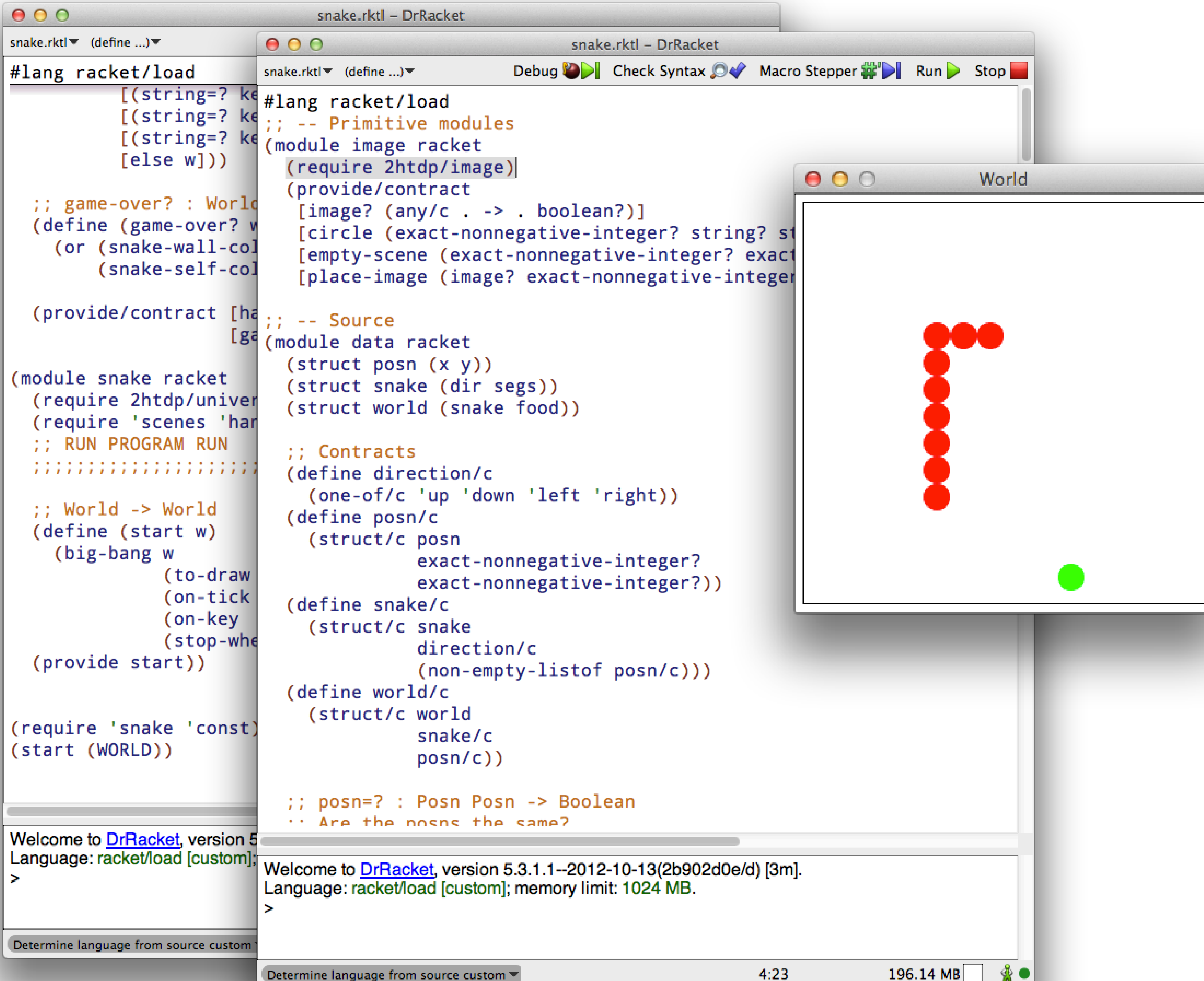
$$
\begin{array}{lcl}
\bullet/\mathcal{C} \uplus \{C_1 \vee C_2\} & \longmapsto & \bullet/\mathcal{C} \cup \{C_i\} \quad i \in \{1, 2\} \\
\bullet/\mathcal{C} \uplus \{\mu X.C\} & \longmapsto & \bullet/\mathcal{C} \cup \{[\mu X.C/X]C\}
\end{array}
$$

Interactive verification environment




World

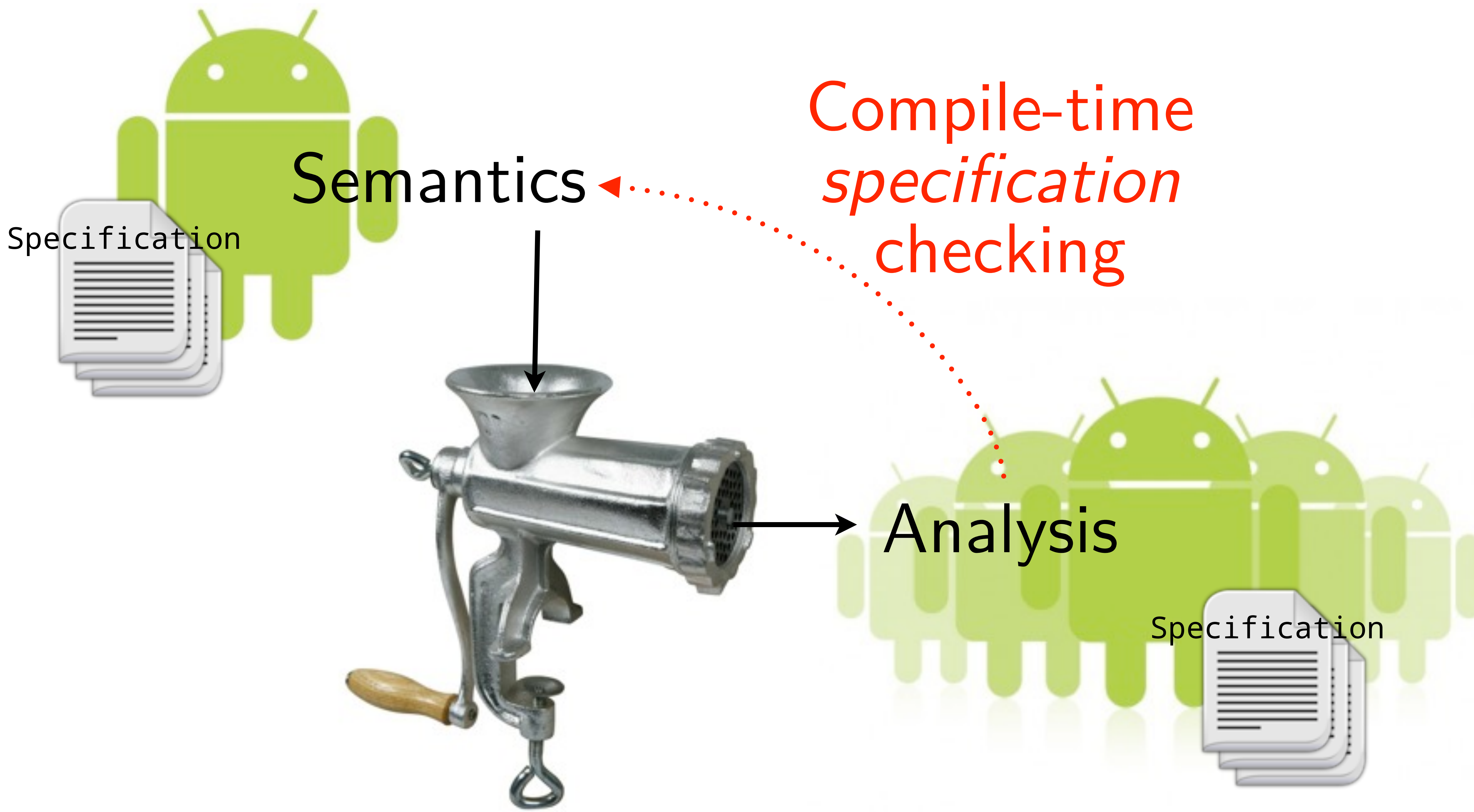

snake.rktl – DrRacket
snake.rktl (define ...)
Debug
Check Syntax
Macro Stepper
Run
Stop
#lang racket/load
[(string=? ke "s") (world-change-dir w 'down)]
[(string=? ke "a") (world-change-dir w 'left)]
[(string=? ke "d") (world-change-dir w 'right)]
[else w]))
;; game-over? : World -> Boolean
(define (game-over? w)
(or (snake-wall-collide? (world-snake w))
(snake-self-collide? (world-snake w))))
(provide/contract [handle-key (world/c string? . -> . world/c)]
[game-over? (world/c . -> . boolean?)]))
(module snake racket
(require 2htdp/universe)
(require 'scenes 'handlers 'motion)
;; RUN PROGRAM RUN
;;;
;; World -> World
(define (start w)
(big-bang w
(to-draw world->scene)
(on-tick world->world 1/2)
(on-key handle-key)
(stop-when game-over?)))
(provide start))
(require 'snake 'const)
(start (WORLD))
Welcome to DrRacket, version 5.3.1.1--2012-10-13(2b902d0e/d) [3m].
Language: racket/load [custom]; memory limit: 1024 MB.
>
Determine language from source custom
75:22
196.14 MB
World

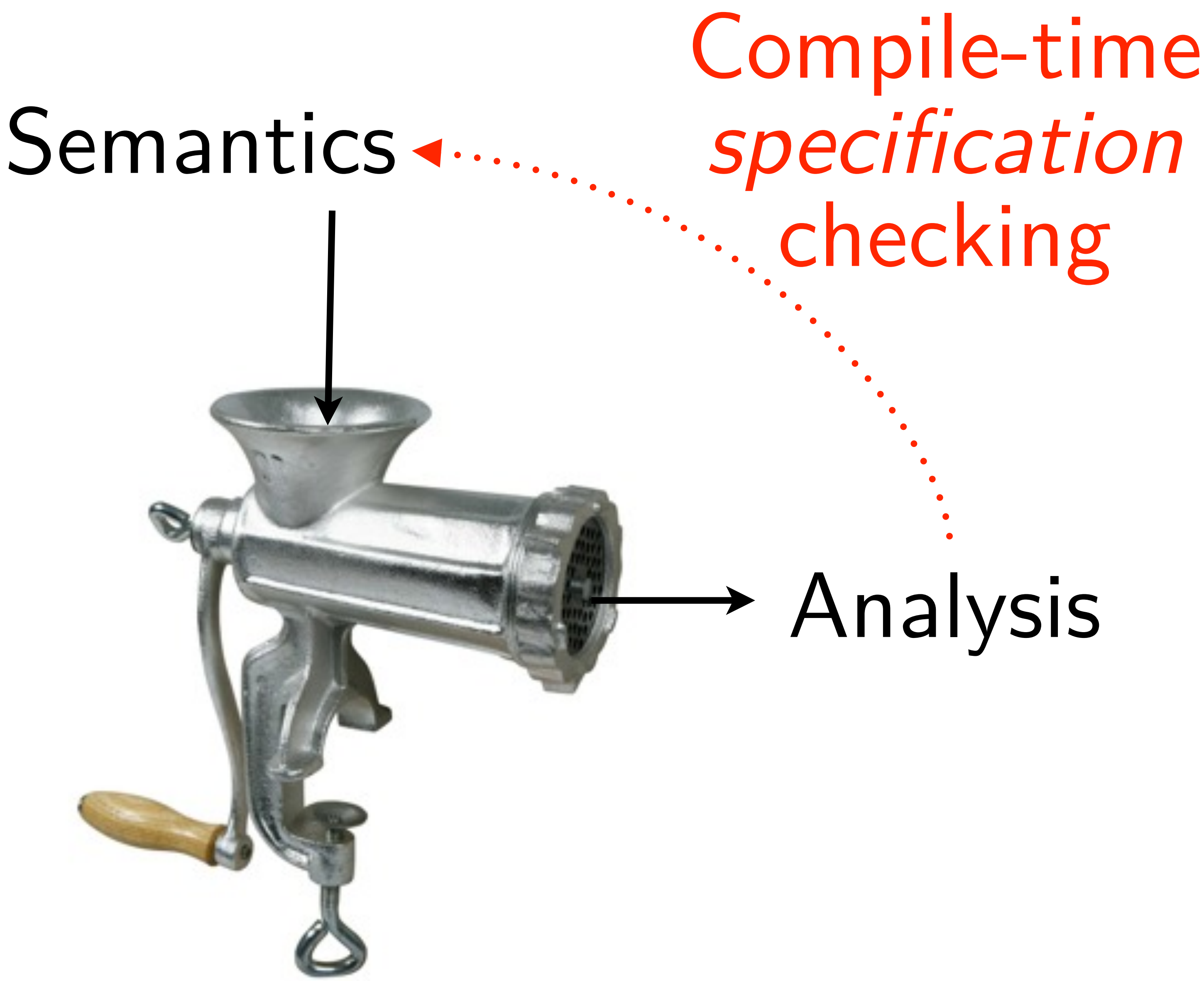

snake.rktl – DrRacket
snake.rktl (define ...)
#lang racket/load
[(string=? ke
[(string=? ke
[(string=? ke
[else w]))
;; game-over? : Worl
(define (game-over? w
(or (snake-wall-co
(snake-self-co
(provide/contract [ha
[ga
(module snake racket
(require 2htdp/unive
(require 'scenes 'ha
;; RUN PROGRAM RUN
;; World -> World
(define (start w)
(big-bang w
(to-draw
(on-tick
(on-key
(stop-whe
(provide start))
(require 'snake 'const)
(start (WORLD))
Welcome to DrRacket, version 5
Language: racket/load [custom];
>
Determine language from source custom
snake.rktl – DrRacket
snake.rktl (define ...) Debug Check Syntax Macro Stepper Run Stop
#lang racket/load
;; -- Primitive modules
(module image racket
(require 2htdp/image)
(provide/contract
[image? (any/c . -> . boolean?)]
[circle (exact-nonnegative-integer? string? st
[empty-scene (exact-nonnegative-integer? exact
[place-image (image? exact-nonnegative-integer
;; -- Source
(module data racket
(struct posn (x y))
(struct snake (dir segs))
(struct world (snake food))
;; Contracts
(define direction/c
(one-of/c 'up 'down 'left 'right))
(define posn/c
(struct/c posn
exact-nonnegative-integer?
exact-nonnegative-integer?))
(define snake/c
(struct/c snake
direction/c
(non-empty-listof posn/c)))
(define world/c
(struct/c world
snake/c
posn/c))
;; posn=? : Posn Posn -> Boolean
Welcome to DrRacket, version 5.3.1.1--2012-10-13(2b902d0e/d) [3m].
Language: racket/load [custom]; memory limit: 1024 MB.
>
Determine language from source custom
4:23
196.14 MB
World

Behavioral contracts

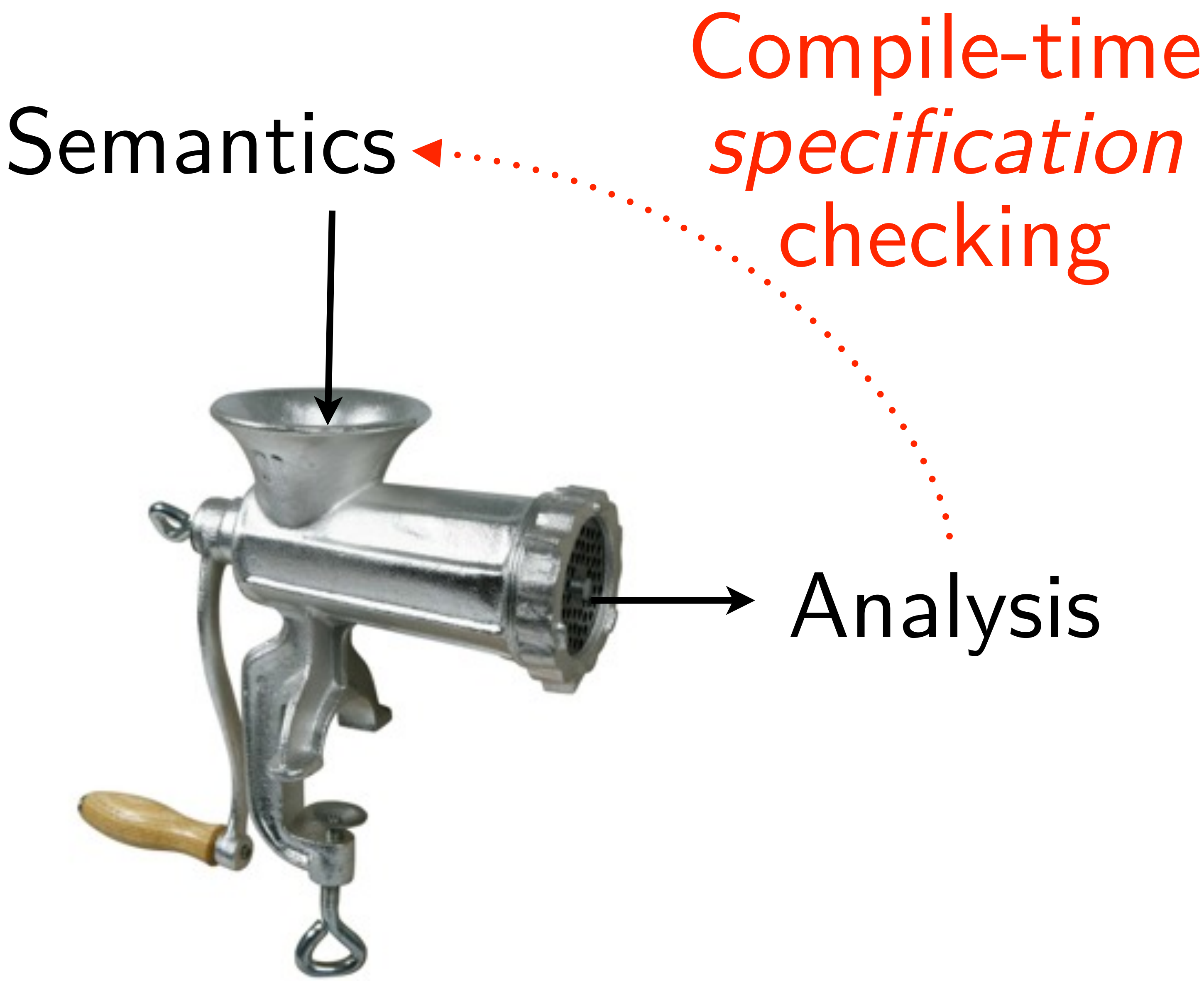
Specify pre- & post-conditions as predicates

```
@SafeSocket(url)
Socket openURL(@OnWhiteList(wl) URL url)

class OnWhiteList extends Contract<List<URL>> {
  bool checkContract(List<URL> wl, URL u) {...}
}
```


Compile-time
specification
checking
Semantics
Specification
Analysis
Specification



- ★ Fast design & development times
- ★ Fast analysis times
- ★ Modular
- ★ Handles libraries
- ★ Verifies rich properties



- ★ Fast design & development times
- ★ Fast analysis times
- ★ Modular
- ★ Handles libraries
- ★ Verifies rich properties

Thank you