Automating Abstract Interpretation of Abstract Machines

Thesis Defense of

J. Ian Johnson

2015, March 30

People make their own languages.

People make their own languages.

They write interpreters.

People make their own languages.

They write interpreters.

Higher-order. Dynamically typed.

People make their own languages.

They write interpreters.

Higher-order. Dynamically typed.

Some languages become popular.

People make their own languages.

They write interpreters.

Higher-order. Dynamically typed.

Some languages become popular.

Lots of code that *does stuff*.

People make their own languages.

They

High ped.

Some languages become popular.

Lots of code that *does stuff*.

So many new languages!
So many new interpreters!

~ Hal Abelson, foreword to EOPLv3

People make their own languages.

They

Hig ped.

Som gages become popular.

Lots of code that *does stuff*.

So many ways to go wrong!

~ me, right now

Two questions a dev asks:

Two questions a dev asks:

Does it do what it's supposed to do?

Two questions a dev asks:

Does it do what it's supposed to do?

Does it *not* do what it's *not* supposed to do?

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Test?

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Test. But what else?

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Test. But what else?

Static types

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Test. But what else?

Static ~~types~~

Two questions a dev asks:

Does it do what it's supposed to do?

Test. Test. Test.

Does it *not* do what it's *not* supposed to do?

Test. But what else?

Static analysis

Static analysis is wonderful!

Understand code

Static analysis is wonderful!

Understand code

- Value flow
- Control flow
- Type inference

Static analysis is wonderful!

Understand code

- Value flow
- Control flow
- Type inference

Harden code

Static analysis is wonderful!

Understand code

- Value flow
- Control flow
- Type inference

Harden code

- Crash-freedom
- Contract verification
- Information flow
- Data race detection

Static analysis is wonderful!

Understand code

- Value flow
- Control flow
- Type inference

Harden code

- Crash-freedom
- Contract verification
- Information flow
- Data race detection

Improve code

Static analysis is wonderful!

Understand code

- Value flow
- Control flow
- Type inference

Harden code

- Crash-freedom
- Contract verification
- Information flow
- Data race detection

Improve code

- no runtime enforcement
- optimization

Static analysis is terrible!

It's too hard to make right

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

It's too hard to understand

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

It's too hard to understand

It's too easy to waste time

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

It's too hard to understand

It's too easy to waste time

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

It's too hard to understand

It's too easy to waste time

Static analysis is terrible!

It's too hard to make right

It's too hard to make fast

It's too hard to make precise

It's too hard to understand

It's too easy to waste time

Static analysis is terrible!

t's too hard to make right

t's too hard to make fast

t's too hard to make precise

t's too hard to understand

t's too easy to waste time

Static analysis is terrible!

s too hard to make right

s too hard to make fast

s too hard to make precise

s too hard to understand

s too easy to waste time

Static analysis is terrible!

too hard to make right

too hard to make fast

too hard to make precise

too hard to understand

too easy to waste time

Static analysis is terrible!

- hard to make right
- hard to make fast
- hard to make precise
- hard to understand
- easy to waste time

Static analysis is terrible!

ard to	make right
ard to	make fast
ard to	make precise
ard to	understand
asy to	waste time

Static analysis is terrible!

to make right

to make fast

to make precise

to understand

to waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is terrible!

make right

make fast

make precise

understand

waste time

Static analysis is doable!

make right

make fast

make precise

understand

waste time

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

```
make right:

make fast:

make precise:

understand:

waste time:
```

Static analysis is doable!

```
make right:

make fast:

make precise:

understand:

waste time:
```

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

```
make right:

make fast:

make precise:

understand:

waste time:
```

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

make right:

make fast:

make precise:

understand:

waste time:

Static analysis is doable!

```
make right:systematic/automatic

make fast:

make precise:

understand:

waste time:
```

Static analysis is doable!

```
make right:systematic/automatic

make fast:systematize folklore

make precise:

understand:

waste time:
```

Static analysis is doable!

```
make right:systematic/automatic

make fast:systematize folklore

make precise:memoize = pushdown

understand:

waste time:
```

Static analysis is doable!

```
make right:systematic/automatic

make fast:systematize folklore

make precise:memoize = pushdown

understand:interpreters

waste time:
```

Static analysis is doable!

```
make right:systematic/automatic

make fast:systematize folklore

make precise:memoize = pushdown

understand:interpreters

waste time:already wasted
```

Static analysis is doable!

make right:systematic/automatic

make fast:systematize folklore

make pr[illegible]ushdown

understand:interpreters

waste time:already wasted

I made a language

Thesis:

Precise and performant analyses for higher-order languages can be systematically and algorithmically constructed from their semantics.

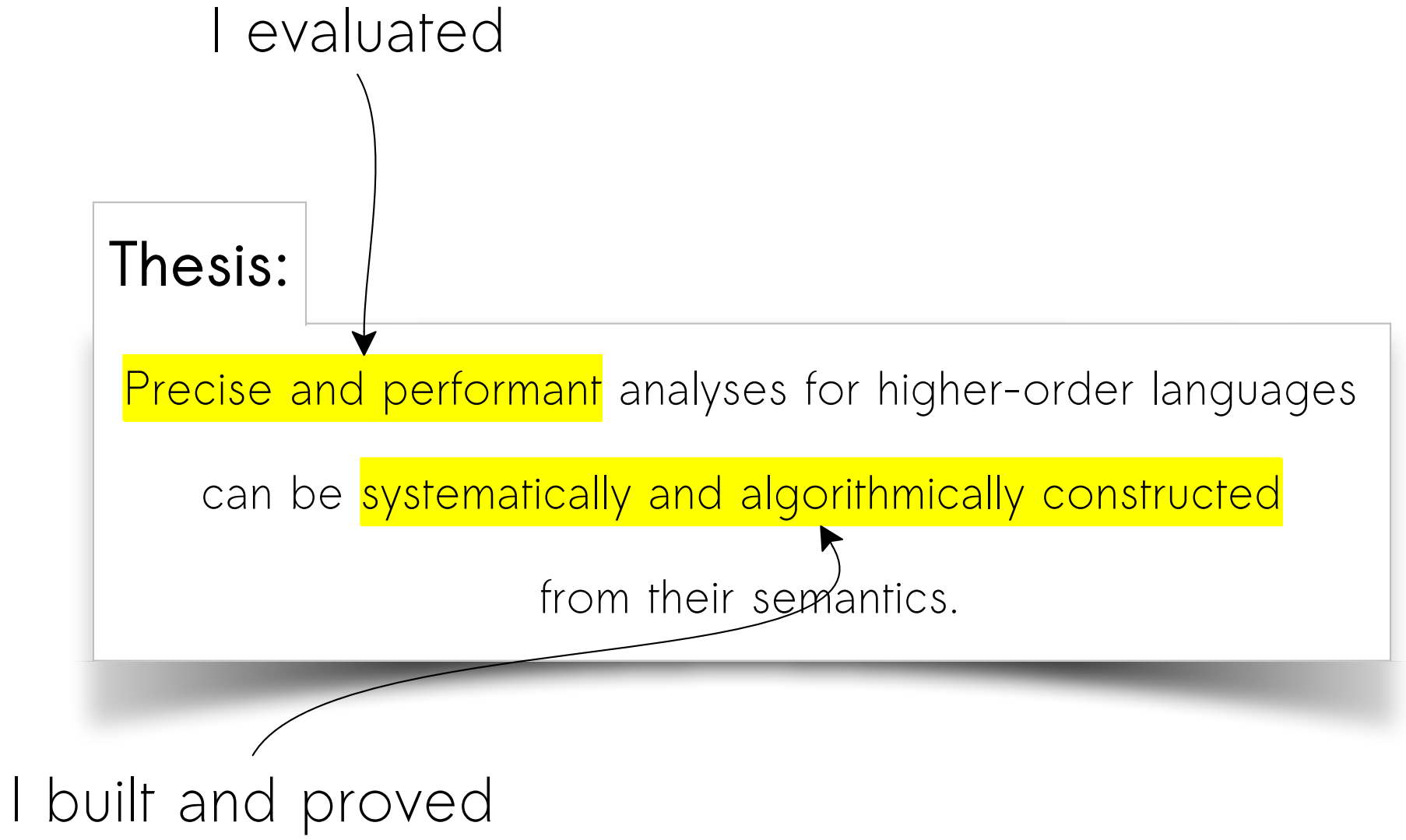
Thesis:

Precise and performant analyses for higher-order languages

can be systematically and algorithmically constructed

from their semantics.

I built and proved


I evaluated
Thesis:
Precise and performant analyses for higher-order languages
can be systematically and algorithmically constructed
from their semantics.
I built and proved

Thesis:

Precise and performant analyses for higher-order languages can be systematically and algorithmically constructed from their semantics.

What *is* semantics?

Meaning of programs

What *is* semantics?

Intensional Meaning of programs

What *is* semantics?

Intensional Meaning of programs

What *is* semantics?

Intensional Meaning of programs

What *is* semantics?

Intensional Meaning of programs

What *is* semantics?

Intensional Meaning of programs

What *is* semantics?

Intensional meaning of programs

What *is* semantics?

Intensional meaning of programs

What *is* semantics?

Intensional meaning of programs

What *is* semantics?

Intensional meaning of programs

What *is* semantics?

Intensional meaning of programs

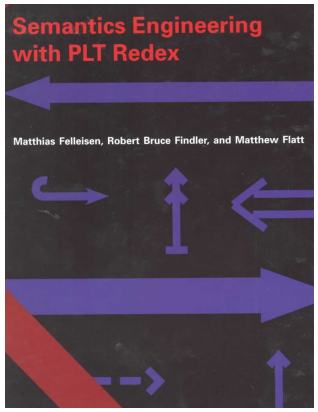
What *is* semantics?

Intensional meaning of programs

What *is* semantics?

Intensional meaning of programs

Abstract machines

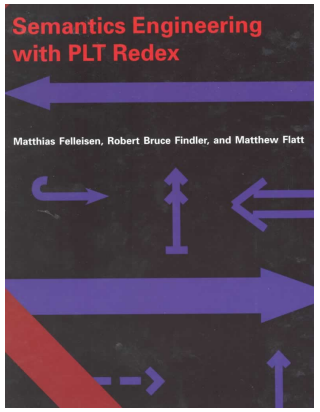


What *is* semantics?

Intensional meaning of programs

Abstract machines

Reasonably efficient

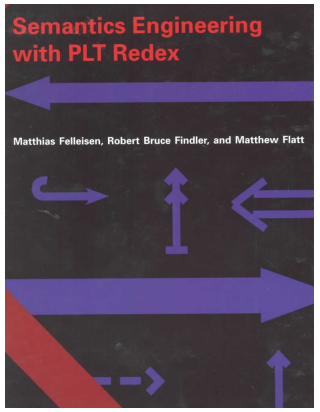


What *is* semantics?

Intensional meaning of programs

Abstract machines

- Reasonably efficient
- Approachable

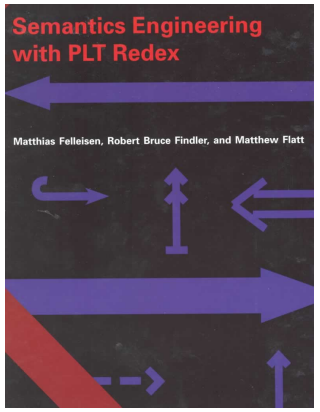


What *is* semantics?

Intensional meaning of programs

Abstract machines

- Reasonably efficient
- Approachable
- Easily abstracted



Abstract
abstract machines?

Abstracting Abstract Machines

David Van Horn *
Northeastern University
dvanhorn@ccs.neu.edu

Matthew Might
University of Utah
might@cs.utah.edu

Abstract

We describe a derivational approach to abstract interpretation that yields novel and transparently sound static analyses when applied to well-established abstract machines. To demonstrate the technique and support our claim, we transform the CEK machine of Felleisen and Friedman, a lazy variant of Krivine's machine, and the stack-inspecting CM machine of Clements and Felleisen into abstract interpretations of themselves. The resulting analyses bound temporal ordering of program events; predict return-flow and stack-inspection behavior; and approximate the flow and evaluation of by-need parameters. For all of these machines, we find that a series of well-known concrete machine refactorings, plus a technique we call store-allocated continuations, leads to machines that abstract into static analyses simply by bounding their stores. We demonstrate that the technique scales up uniformly to allow static analysis of realistic language features, including tail calls, conditionals, side effects, exceptions, first-class continuations, and even garbage collection.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Program analysis, Operational semantics; F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic—Lambda calculus and related systems

General Terms Languages, Theory

Keywords abstract machines, abstract interpretation

1. Introduction

Abstract machines such as the CEK machine and Krivine's machine are first-order state transition systems that represent the core of a real language implementation. Semantics-based program analysis, on the other hand, is concerned with safely approximating intensional properties of such a machine as it runs a program. It seems natural then to want to systematically derive analyses from machines to approximate the core of realistic run-time systems.

Our goal is to develop a technique that enables direct abstract interpretations of abstract machines by methods for transforming a given machine description into another that computes its finite approximation.

* Supported by the National Science Foundation under grant 0937060 to the Computing Research Association for the CIFellow Project.

ICFP'10, September 27–29, 2010, Baltimore, Maryland, USA.

We demonstrate that the technique of refactoring a machine with **store-allocated continuations** allows a direct structural abstraction[1] by bounding the machine's store. Thus, we are able to convert semantic techniques used to model language features into static analysis techniques for reasoning about the behavior of those very same features. By abstracting well-known machines, our technique delivers static analyzers that can reason about by-need evaluation, higher-order functions, tail calls, side effects, stack structure, exceptions and first-class continuations.

The basic idea behind store-allocated continuations is not new. SML/NJ has allocated continuations in the heap for well over a decade [28]. At first glance, modeling the program stack in an abstract machine with store-allocated continuations would not seem to provide any real benefit. Indeed, for the purpose of defining the meaning of a program, there is no benefit, because the meaning of the program does not depend on the stack-implementation strategy. Yet, a closer inspection finds that store-allocating continuations eliminate recursion from the definition of the state-space of the machine. With no recursive structure in the state-space, an abstract machine becomes eligible for conversion into an abstract interpreter through a simple structural abstraction.

To demonstrate the applicability of the approach, we derive abstract interpreters of:

- a call-by-value λ-calculus with state and control based on the CESK machine of Felleisen and Friedman [13],
- a call-by-need λ-calculus based on a tail-recursive, lazy variant of Krivine's machine derived by Ager, Danvy and Midtgaard [1], and
- a call-by-value λ-calculus with stack inspection based on the CM machine of Clements and Felleisen [3];

and use abstract garbage collection to improve precision [25].

Overview

In Section 2, we begin with the CEK machine and attempt a structural abstract interpretation, but find ourselves blocked by two recursive structures in the machine: environments and continuations. We make three refactorings to:

1. store-allocate bindings,
2. store-allocate continuations, and
3. time-stamp machine states;

resulting in the CESK, CESK*, and time-stamped CESK* machines, respectively. The time-stamps encode the history (context) of the machine's execution and facilitate context-sensitive abstractions. We then demonstrate that the time-stamped machine abstracts directly into a parameterized, sound and computable static analysis.

[1] A structural abstraction distributes component-, point-, and member-wise.

Abstracting Abstract Machines

David Van Horn *
Northeastern University
dvanhorn@ccs.neu.edu

Matthew Might
University of Utah
might@cs.utah.edu

Abstract

We describe a derivational approach to abstract interpretation that yields novel and transparently sound static analyses when applied to well-established abstract machines. To demonstrate the technique and support our claim, we transform the CEK machine of Felleisen and Friedman, a lazy variant of Krivine's machine, and the stack-inspecting CM machine of Clements and Felleisen into abstract interpretations of themselves. The resulting analyses bound temporal ordering of program events; predict return-flow and stack-inspection behavior; and approximate the flow and evaluation of by-need parameters. For all of these machines, we find that a series of well-
technique we call sto
that abstract into stat
We demonstrate that
static analysis of rea
conditionals, side effi
even garbage collecti

Categories and Subj
of Programs]: Sema
analysis, Operational
Formal Languages]:
related systems

General Terms Lar

Keywords abstract

x 1000

1. Introduction

Abstract machines such as the CEK machine and Krivine's machine are first-order state transition systems that represent the core of a real language implementation. Semantics-based program analysis, on the other hand, is concerned with safely approximating intensional properties of such a machine as it runs a program. It seems natural then to want to systematically derive analyses from machines to approximate the core of realistic run-time systems.

Our goal is to develop a technique that enables direct abstract interpretations of abstract machines by methods for transforming a given machine description into another that computes its finite approximation.

* Supported by the National Science Foundation under grant 0937060 to the Computing Research Association for the CIFellow Project.

ICFP'10, September 27–29, 2010, Baltimore, Maryland, USA.

We demonstrate that the technique of refactoring a machine with **store-allocated continuations** allows a direct structural abstraction[1] by bounding the machine's store. Thus, we are able to convert semantic techniques used to model language features into static analysis techniques for reasoning about the behavior of those very same features. By abstracting well-known machines, our technique delivers static analyzers that can reason about by-need evaluation, higher-order functions, tail calls, side effects, stack structure, exceptions and first-class continuations.

The basic idea behind store-allocated continuations is not new. SML/NJ has allocated continuations in the heap for well over a
gram stack in an ab-
ions would not seem
rpose of defining the
because the meaning
implementation strat-
-allocating continua-
of the state-space of
e state-space, an ab-
n into an abstract in-
on.
approach, we derive

control based on the
[13],

-recursive, lazy vari-
ant of Krivine's machine derived by Ager, Danvy and Midtgaard [1], and

- a call-by-value λ-calculus with stack inspection based on the CM machine of Clements and Felleisen [3];

and use abstract garbage collection to improve precision [25].

Overview

In Section 2, we begin with the CEK machine and attempt a structural abstract interpretation, but find ourselves blocked by two recursive structures in the machine: environments and continuations. We make three refactorings to:

1. store-allocate bindings,
2. store-allocate continuations, and
3. time-stamp machine states;

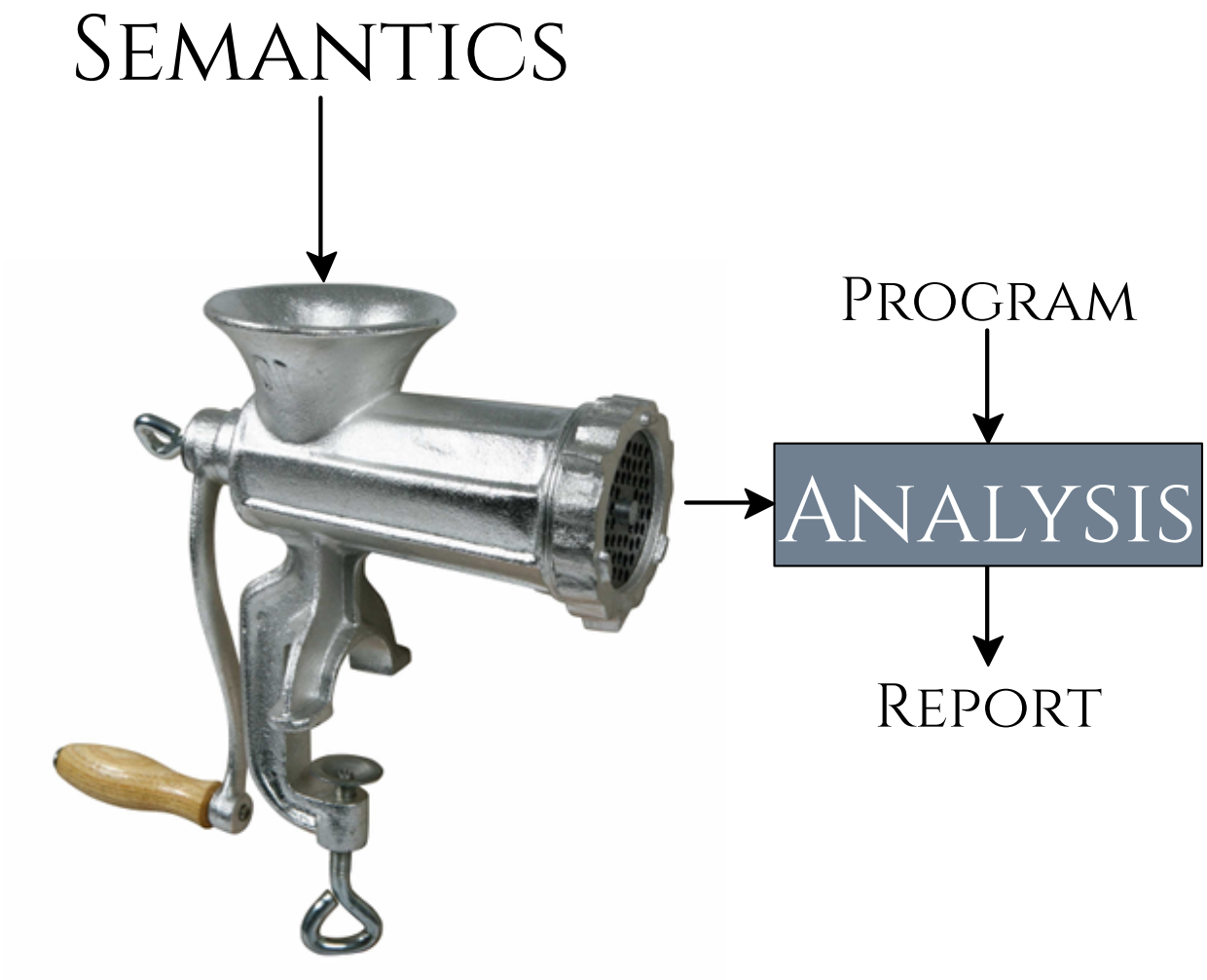
resulting in the CESK, CESK*, and time-stamped CESK* machines, respectively. The time-stamps encode the history (context) of the machine's execution and facilitate context-sensitive abstractions. We then demonstrate that the time-stamped machine abstracts directly into a parameterized, sound and computable static analysis.

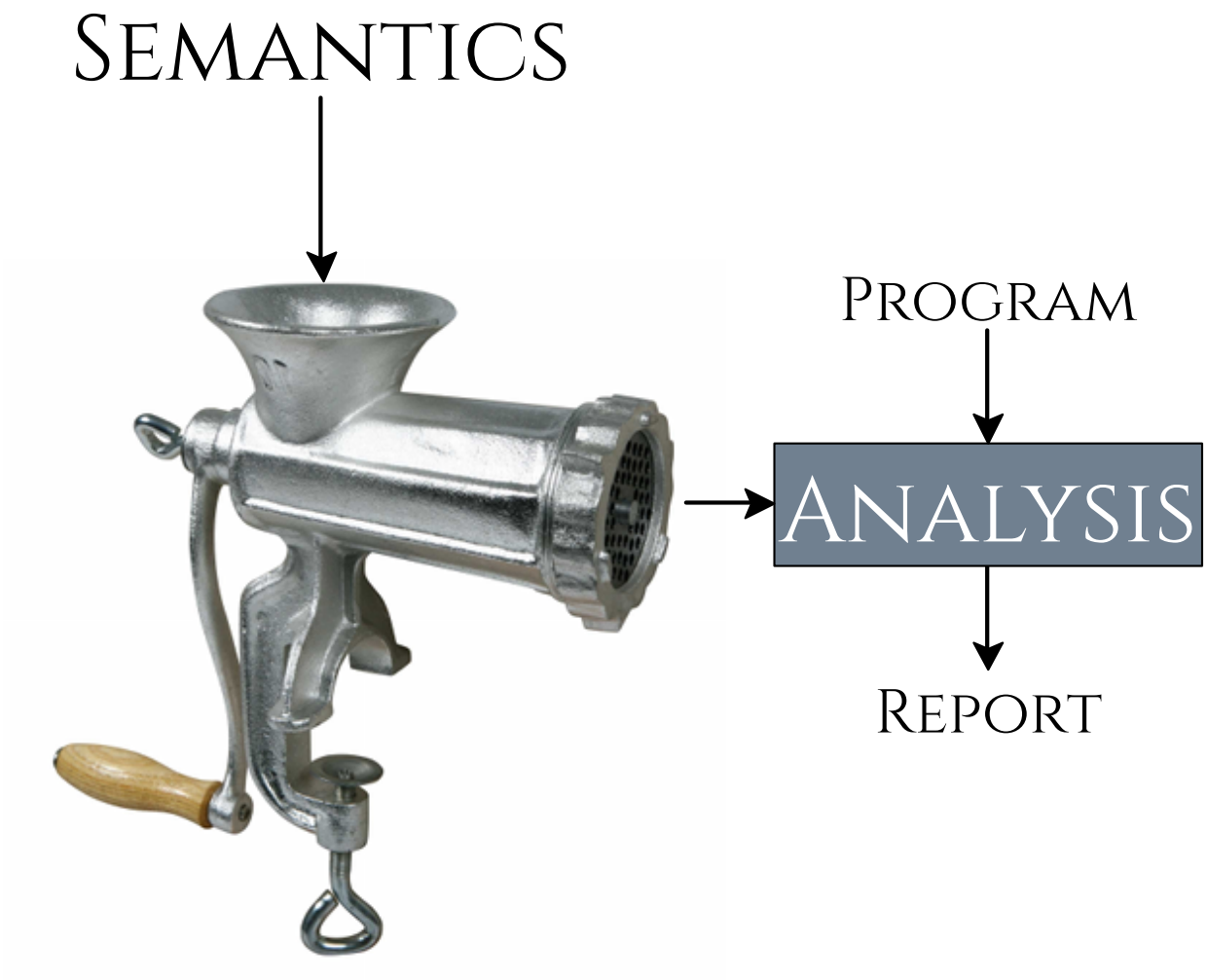
[1] A structural abstraction distributes component-, point-, and member-wise.

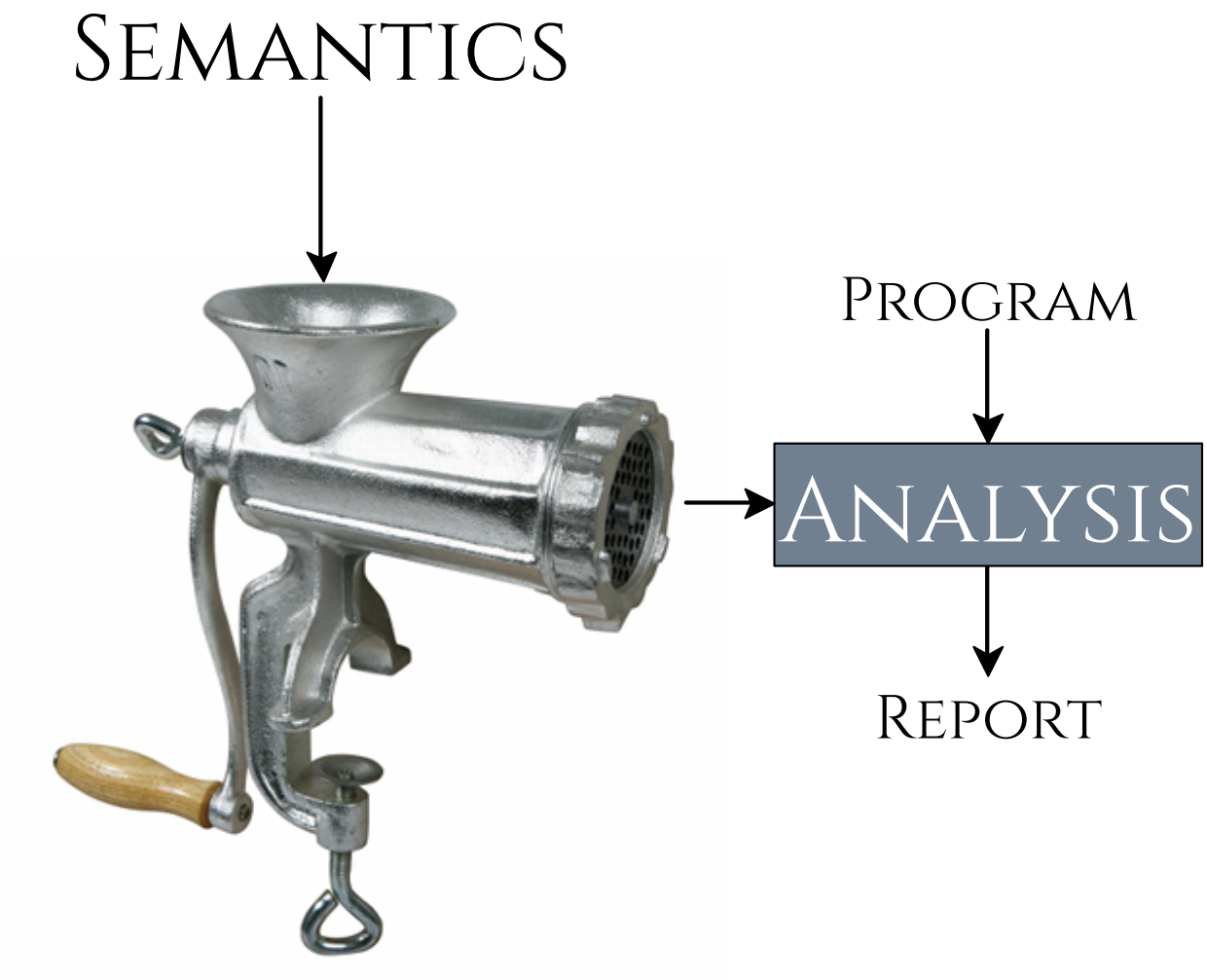
Semantics

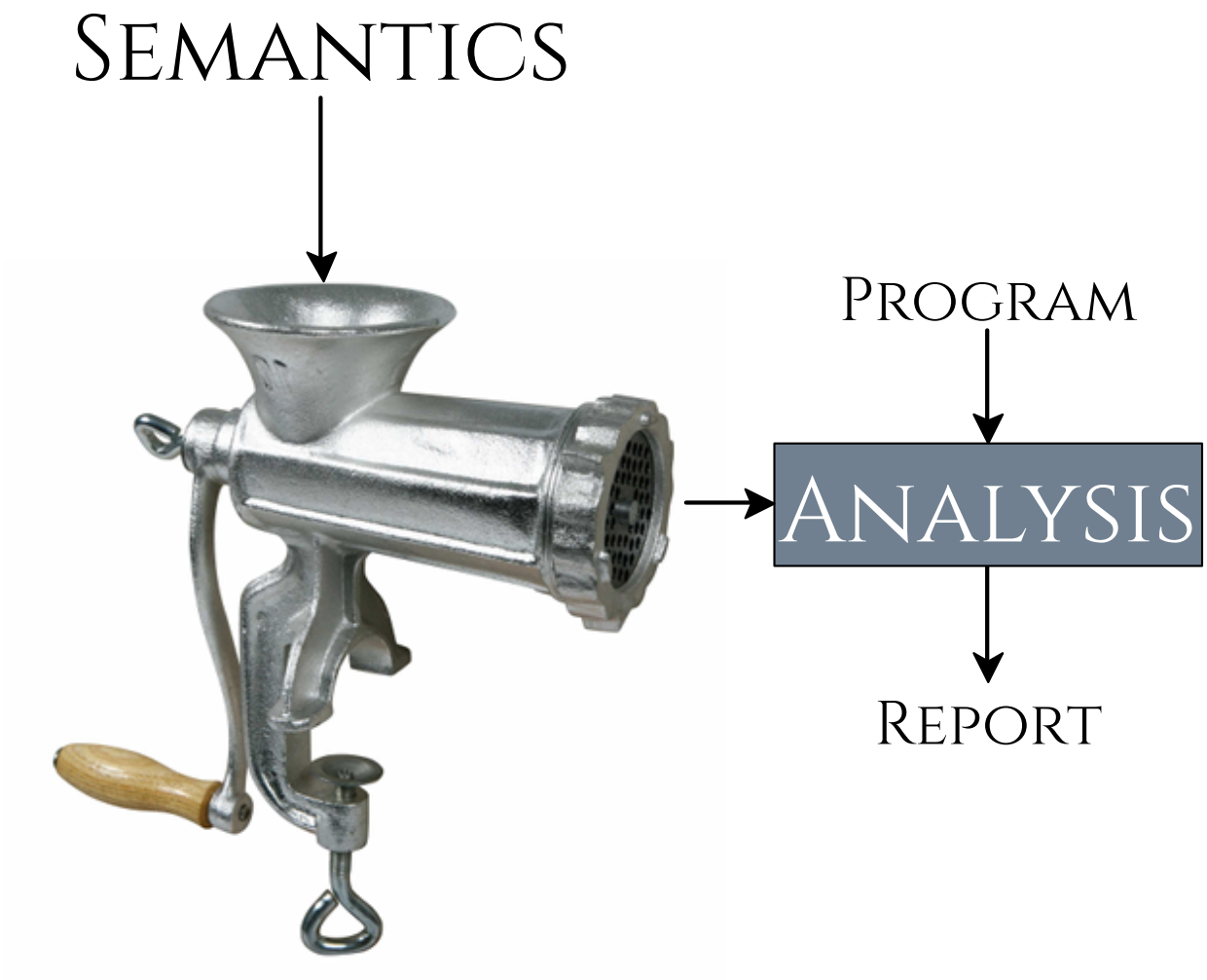
→Analysis

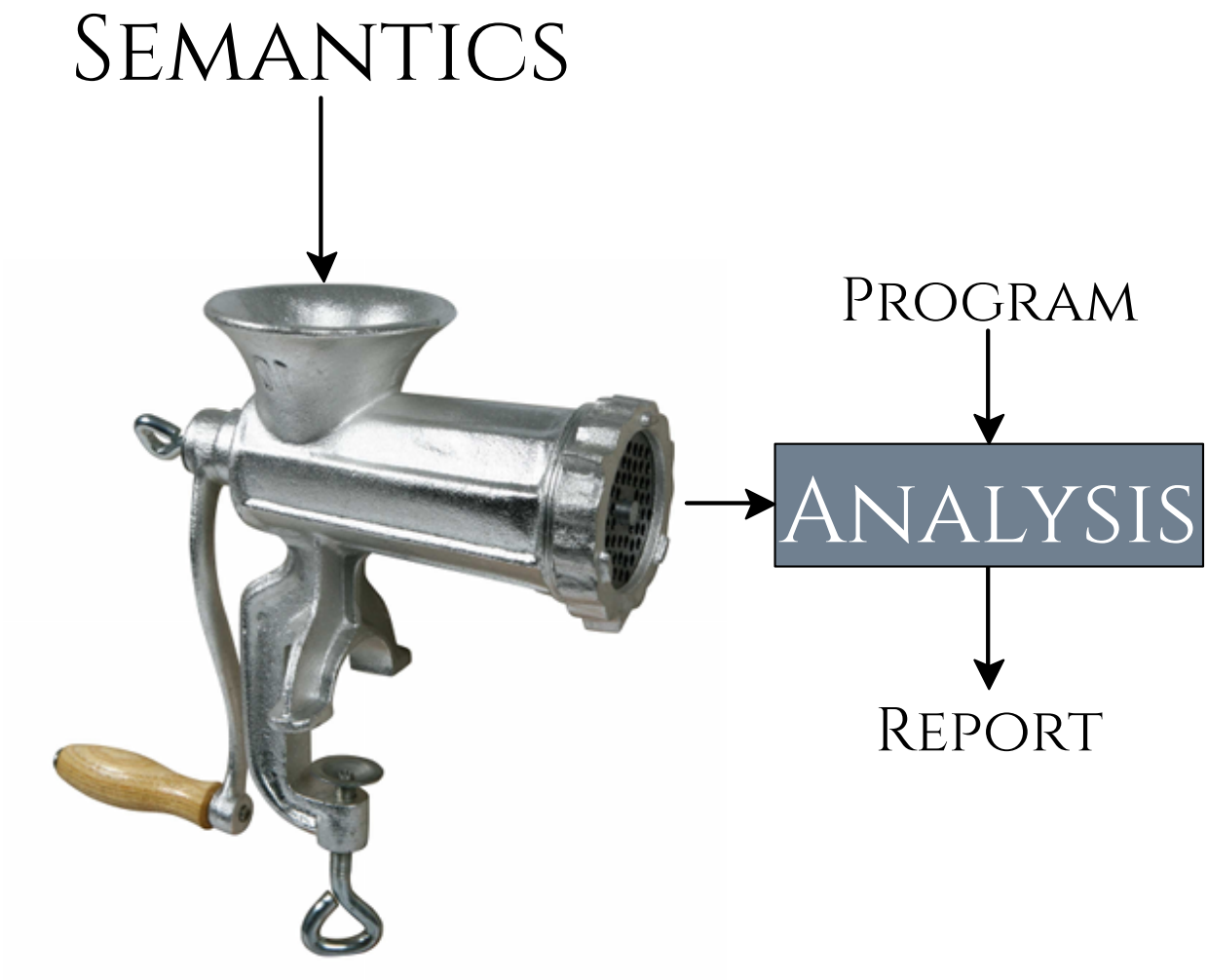

JS
SEMANTICS
ANALYSIS

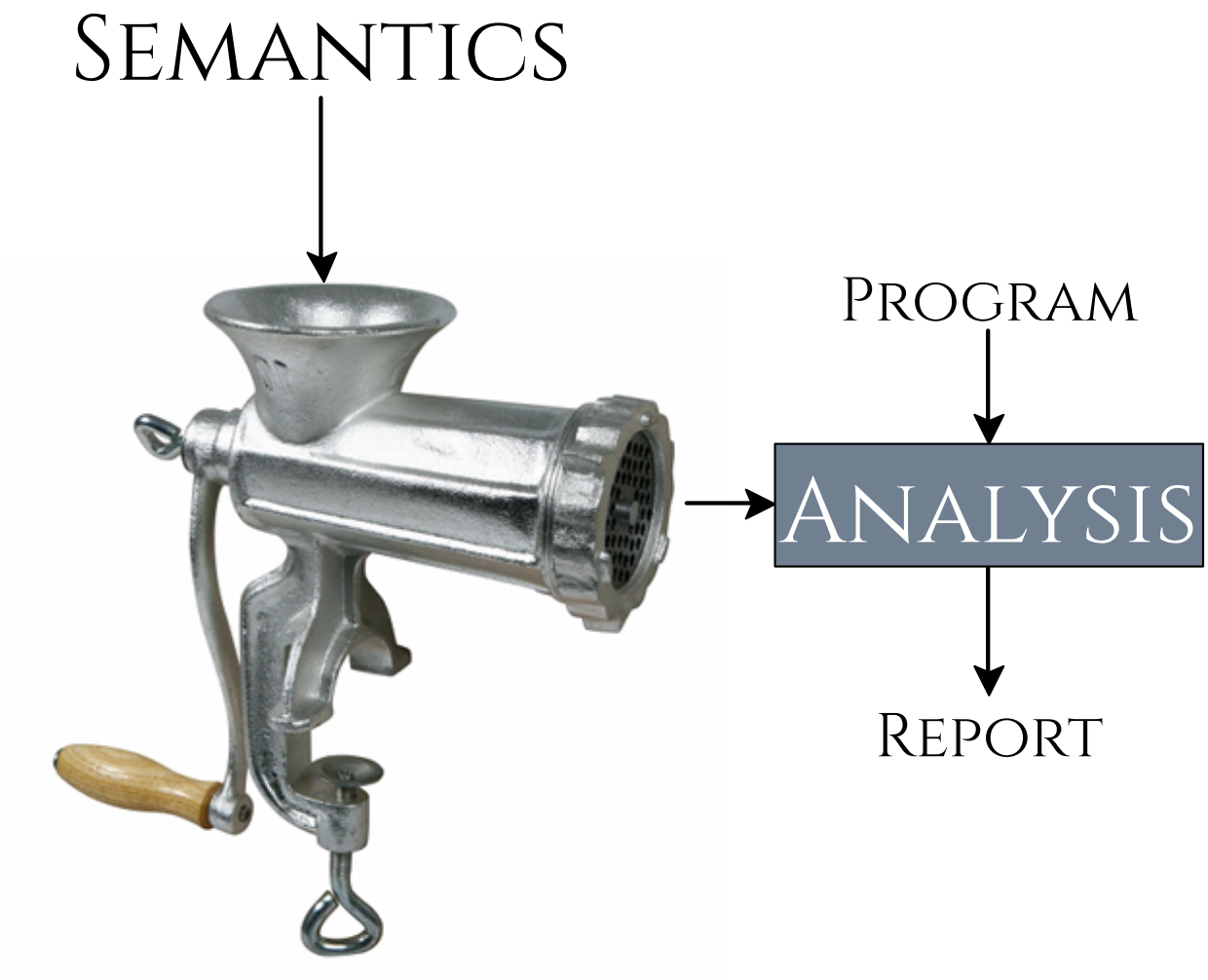

Semantics
Program
Analysis
Report

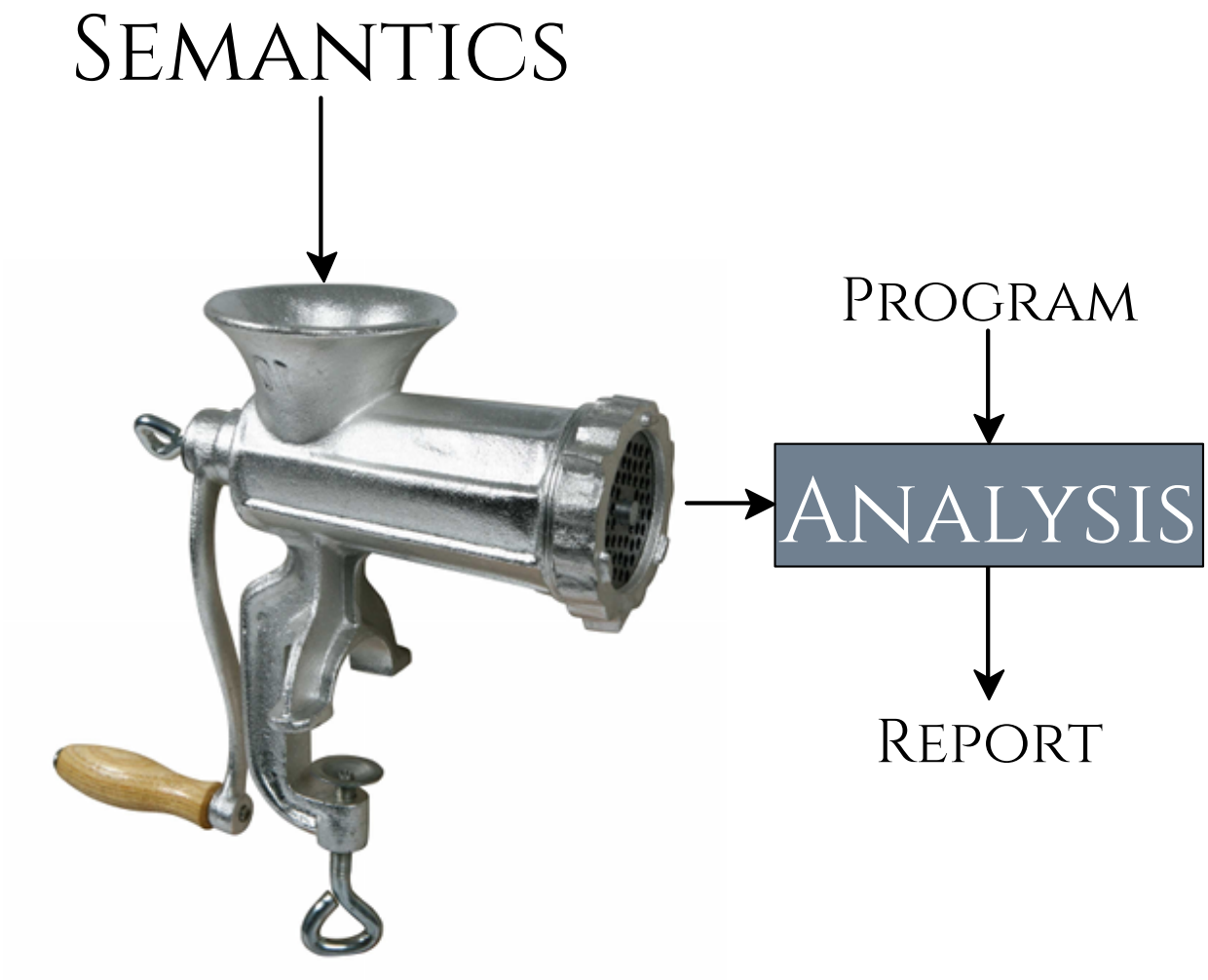

Semantics
Program
Analysis
Report

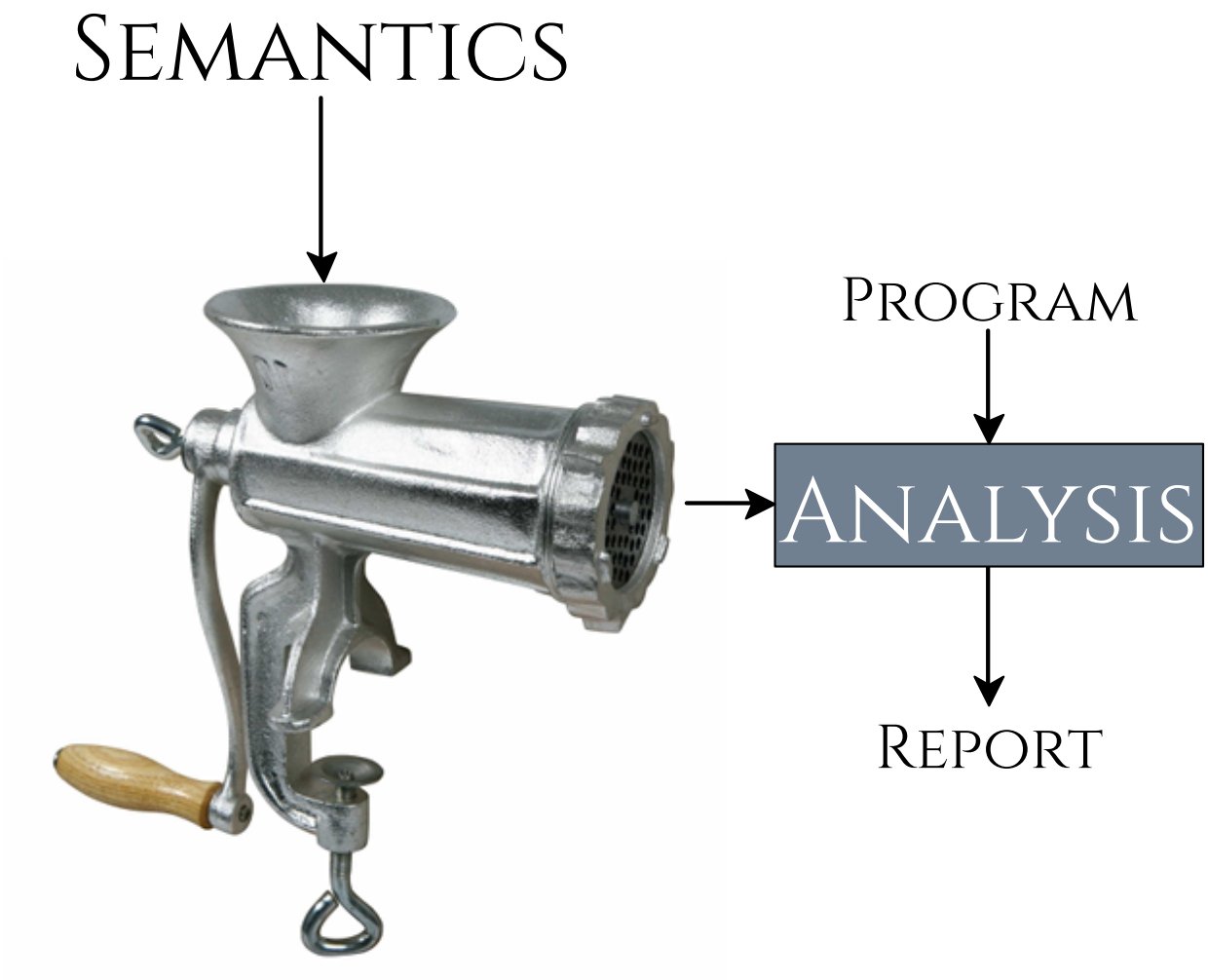

Semantics
Program
Analysis
Report

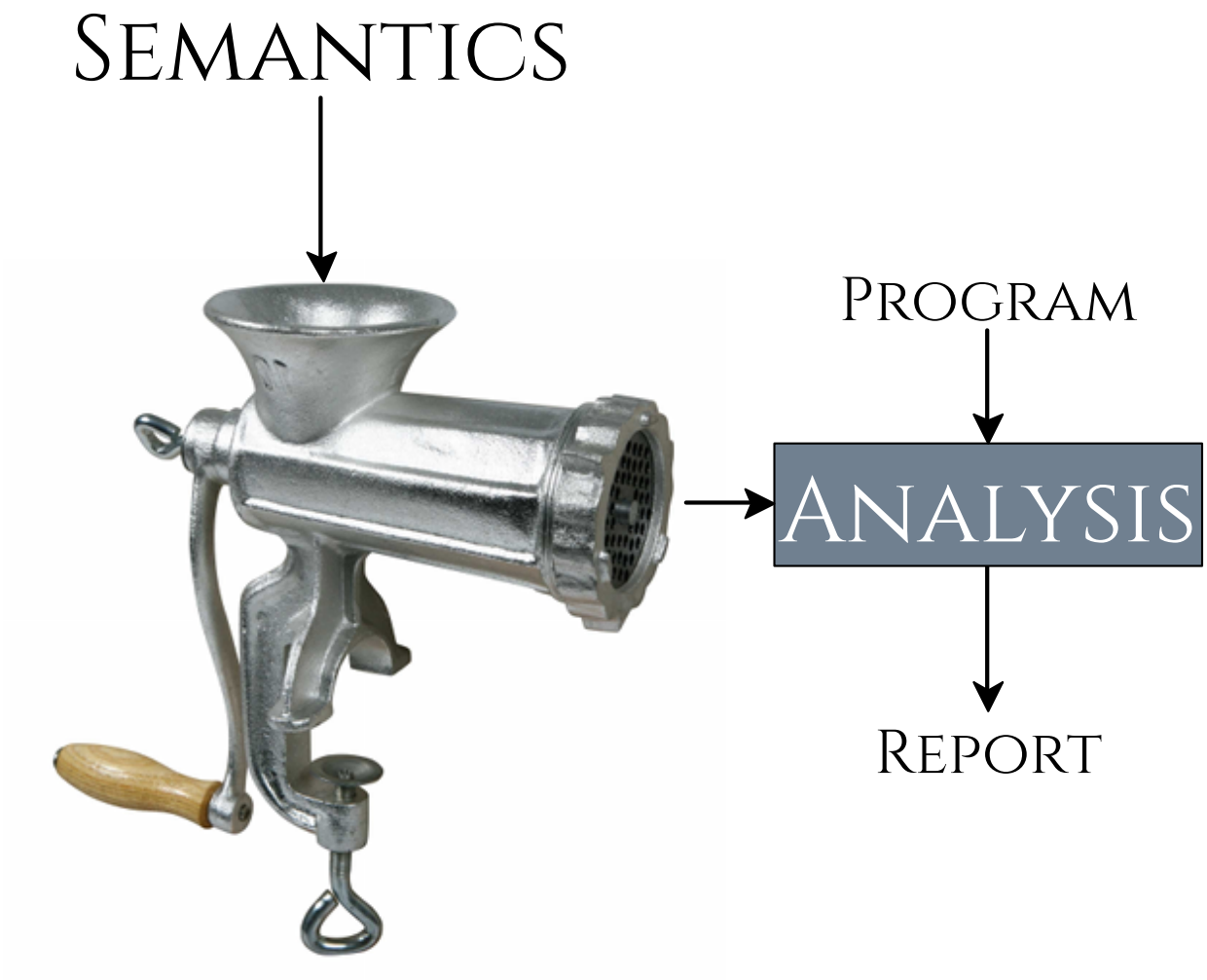

Semantics
Program
Analysis
Report

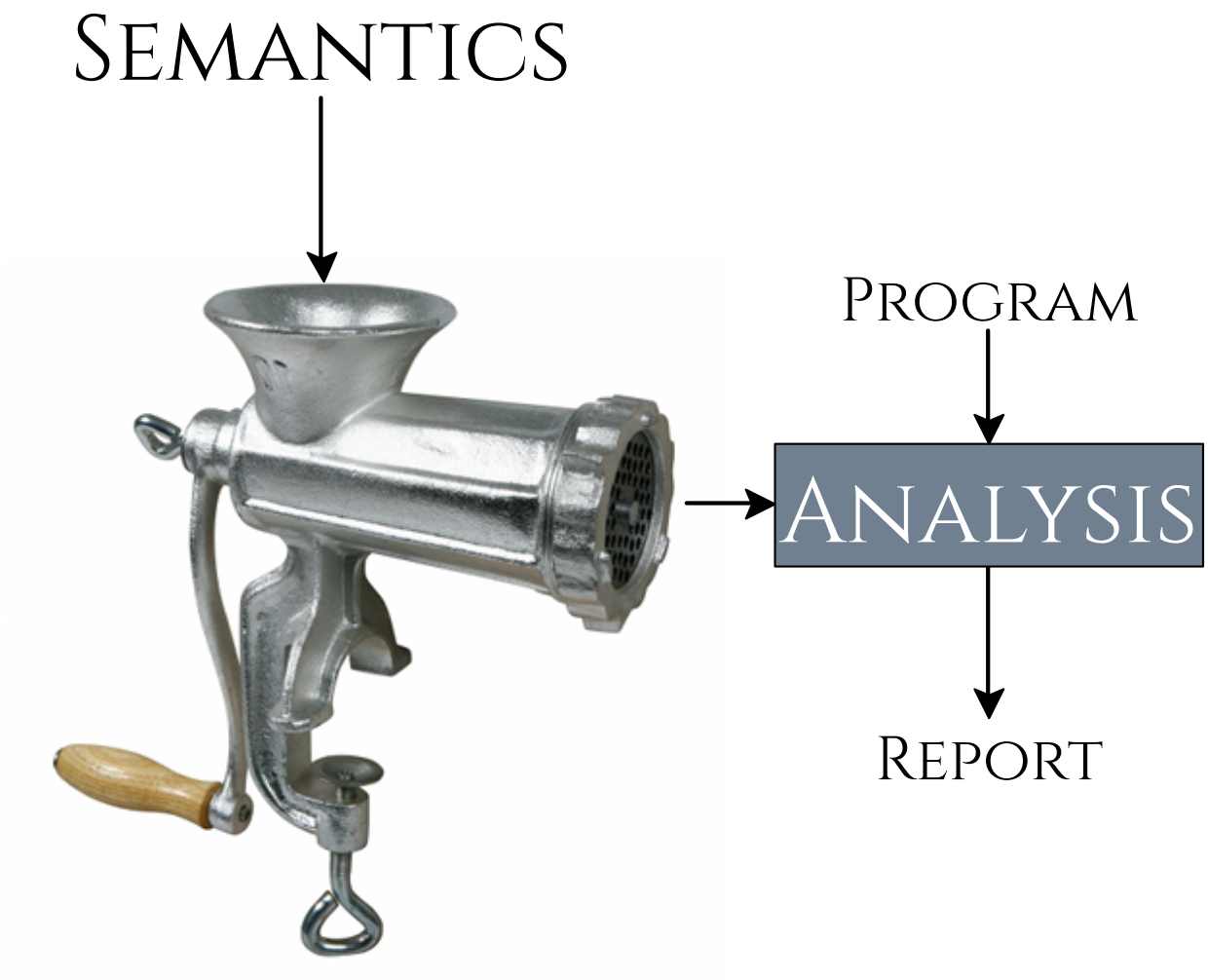

Semantics
Program
Analysis
Report

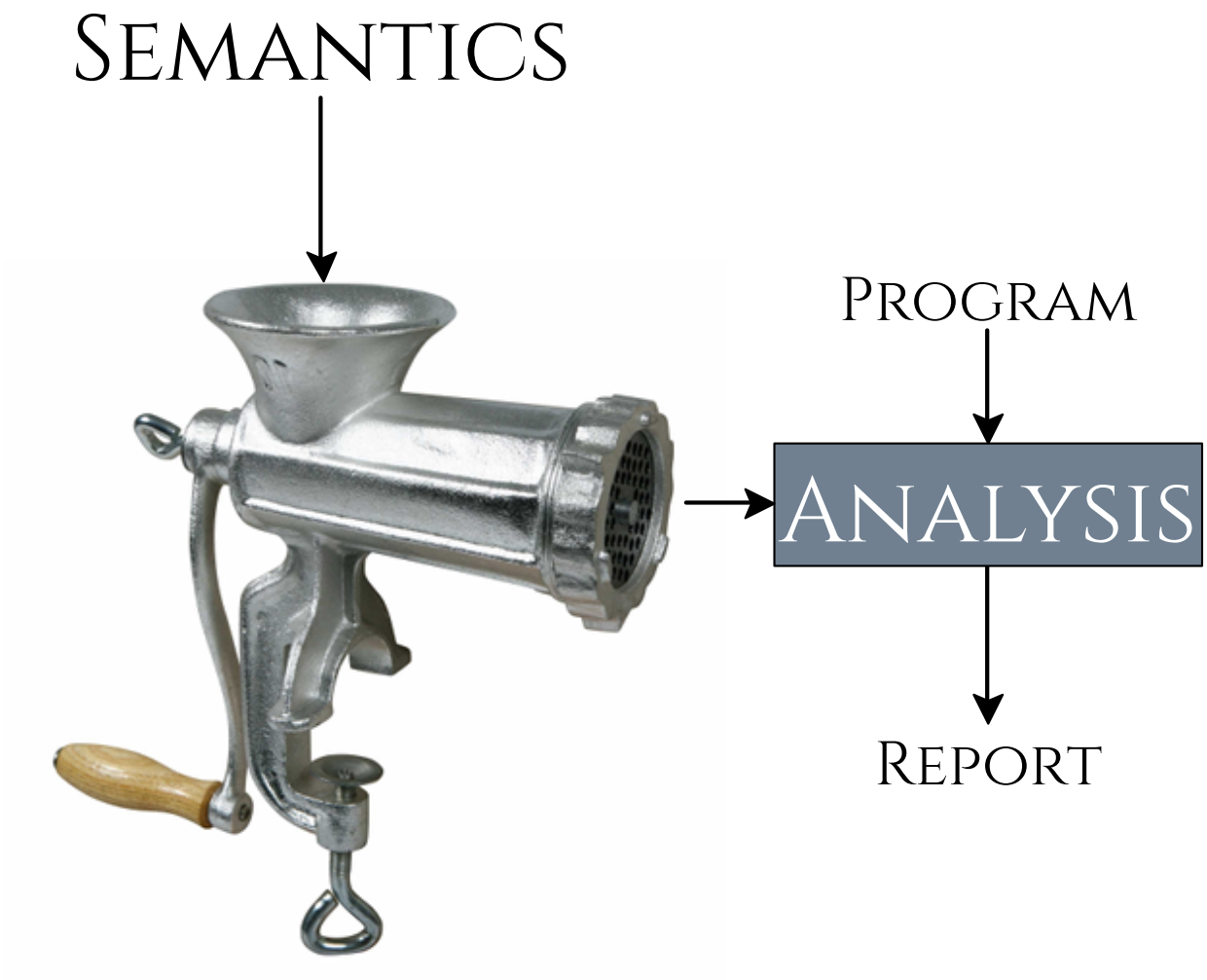

Semantics
Program
Analysis
Report

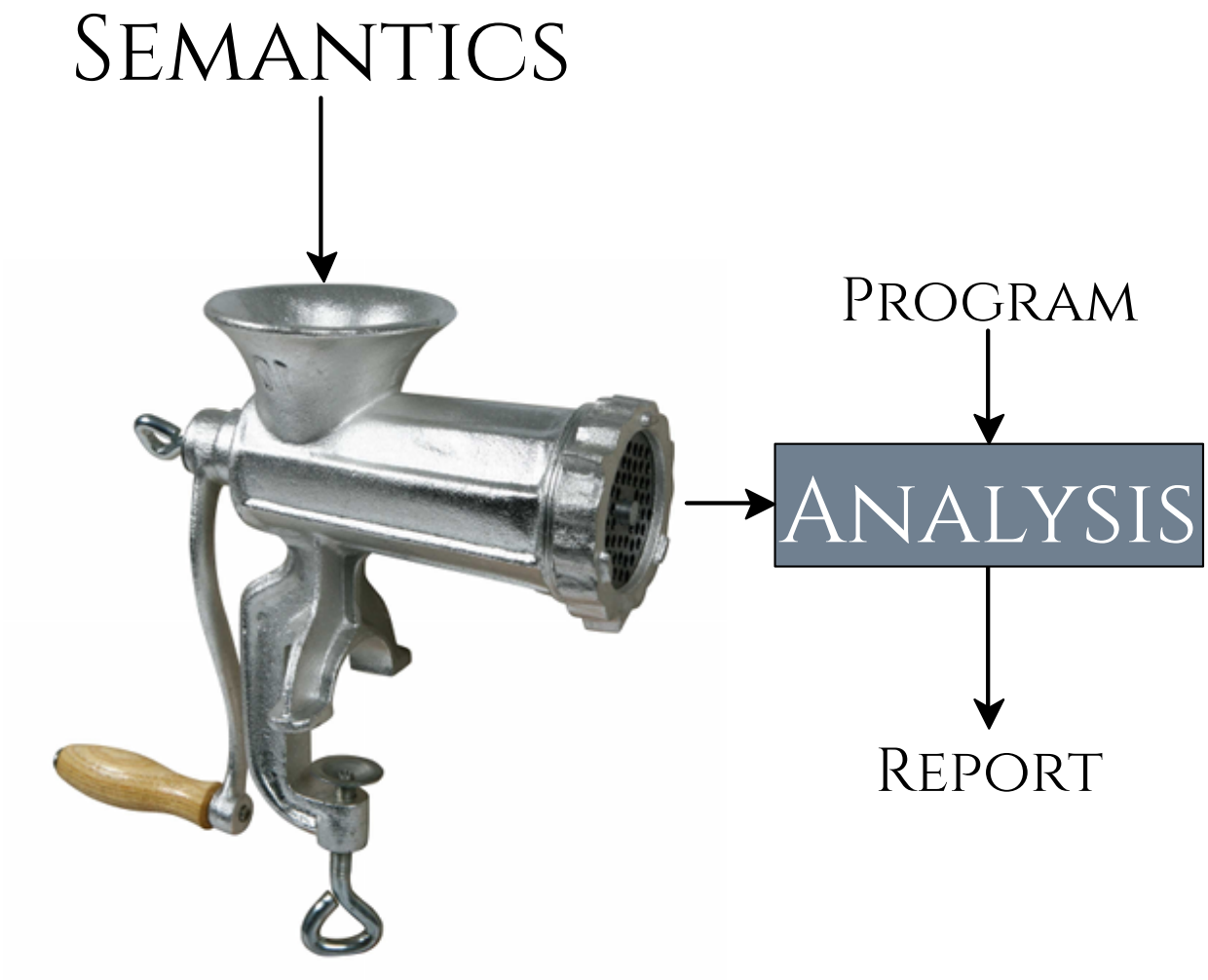

Semantics
Program
Analysis
Report

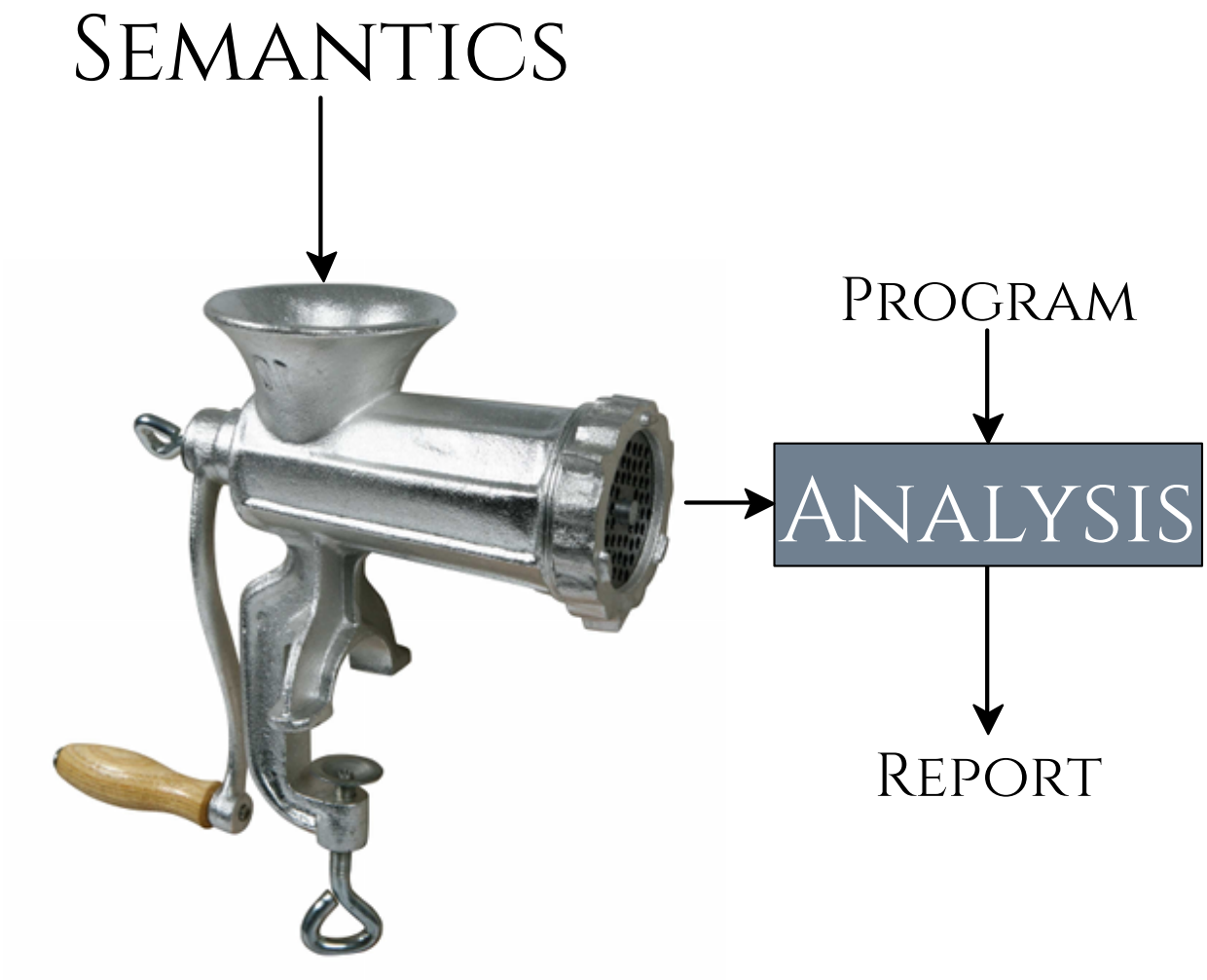

Semantics
Program
Analysis
Report

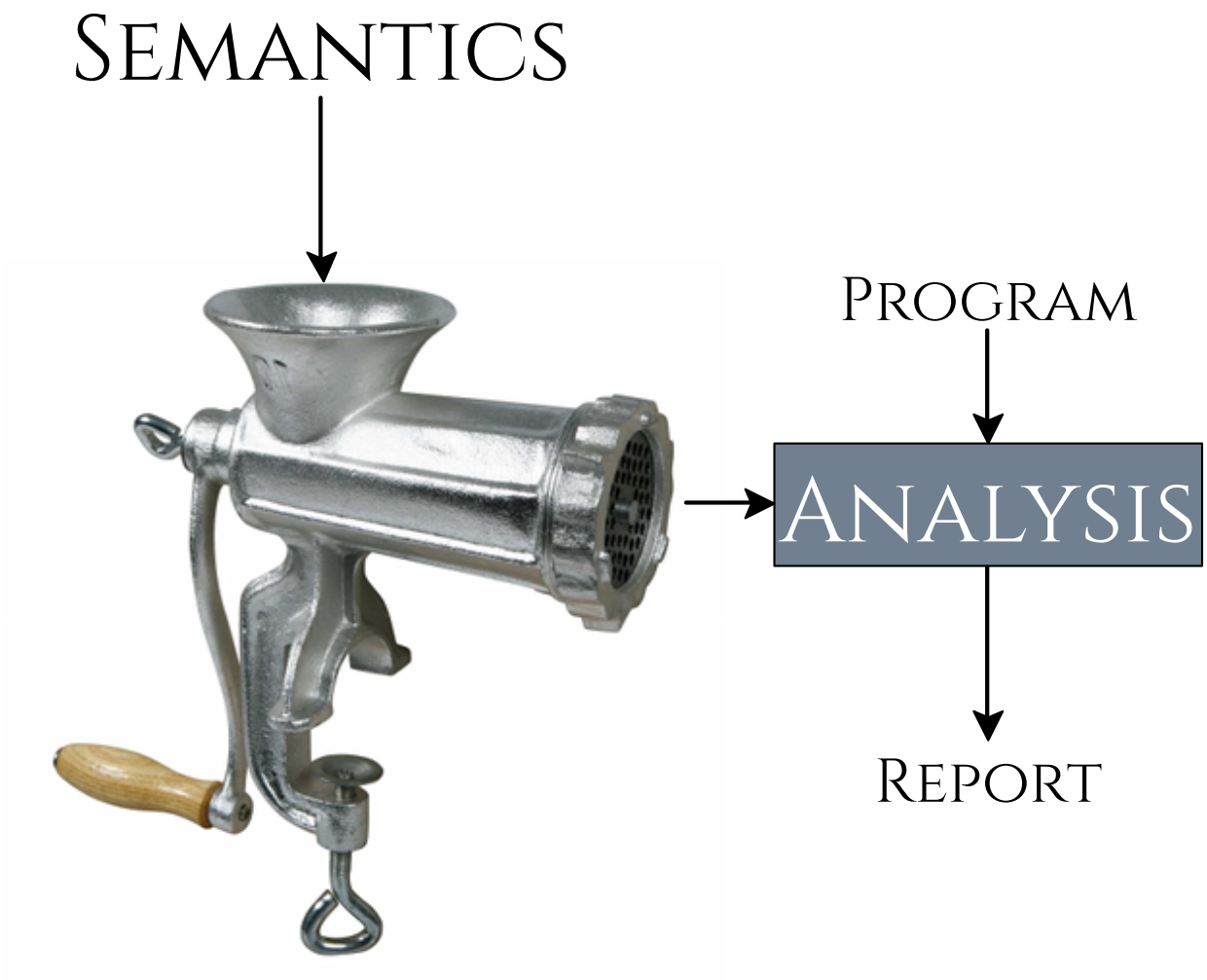

Semantics
Program
Analysis
Report

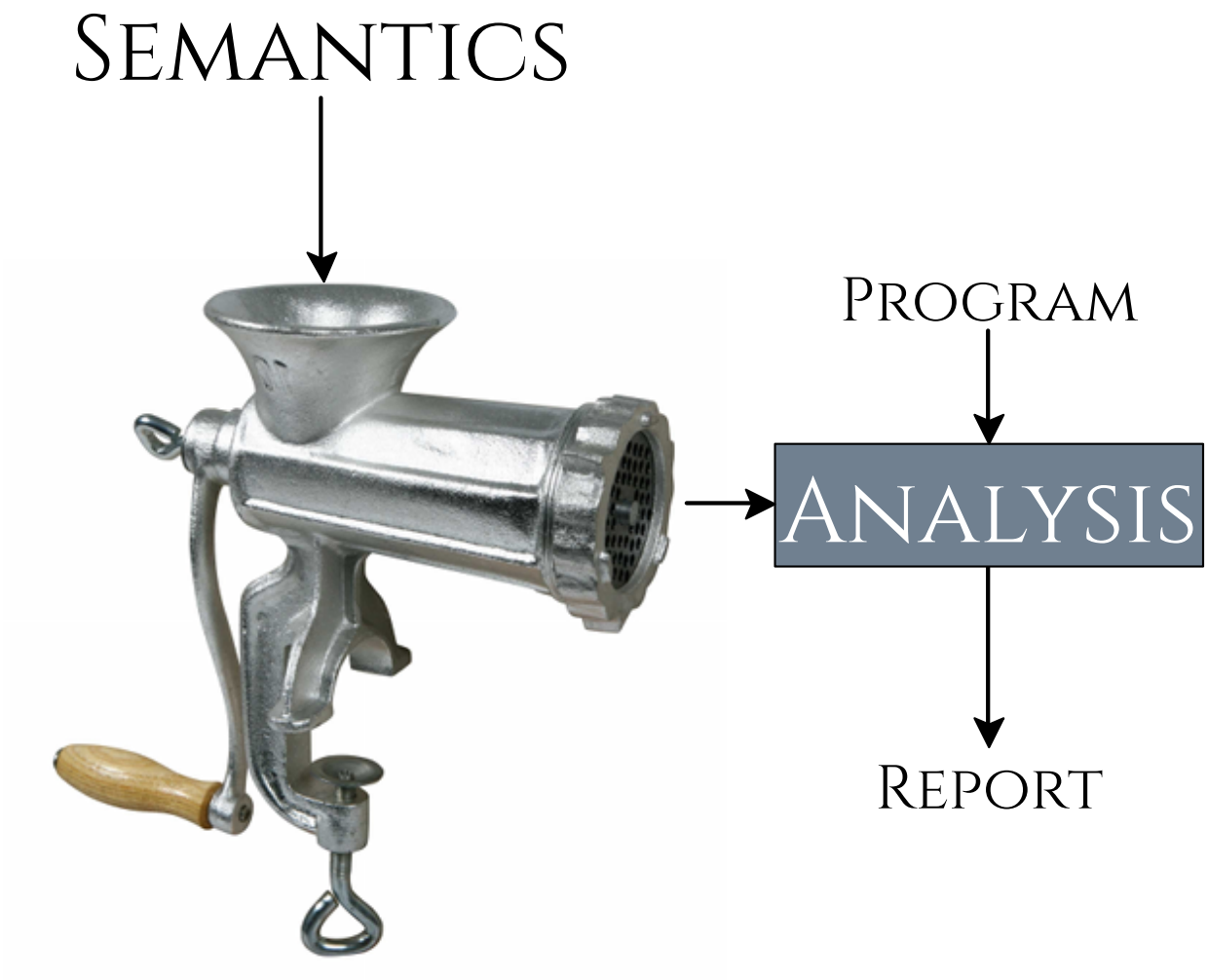

Semantics
Program
Analysis
Report

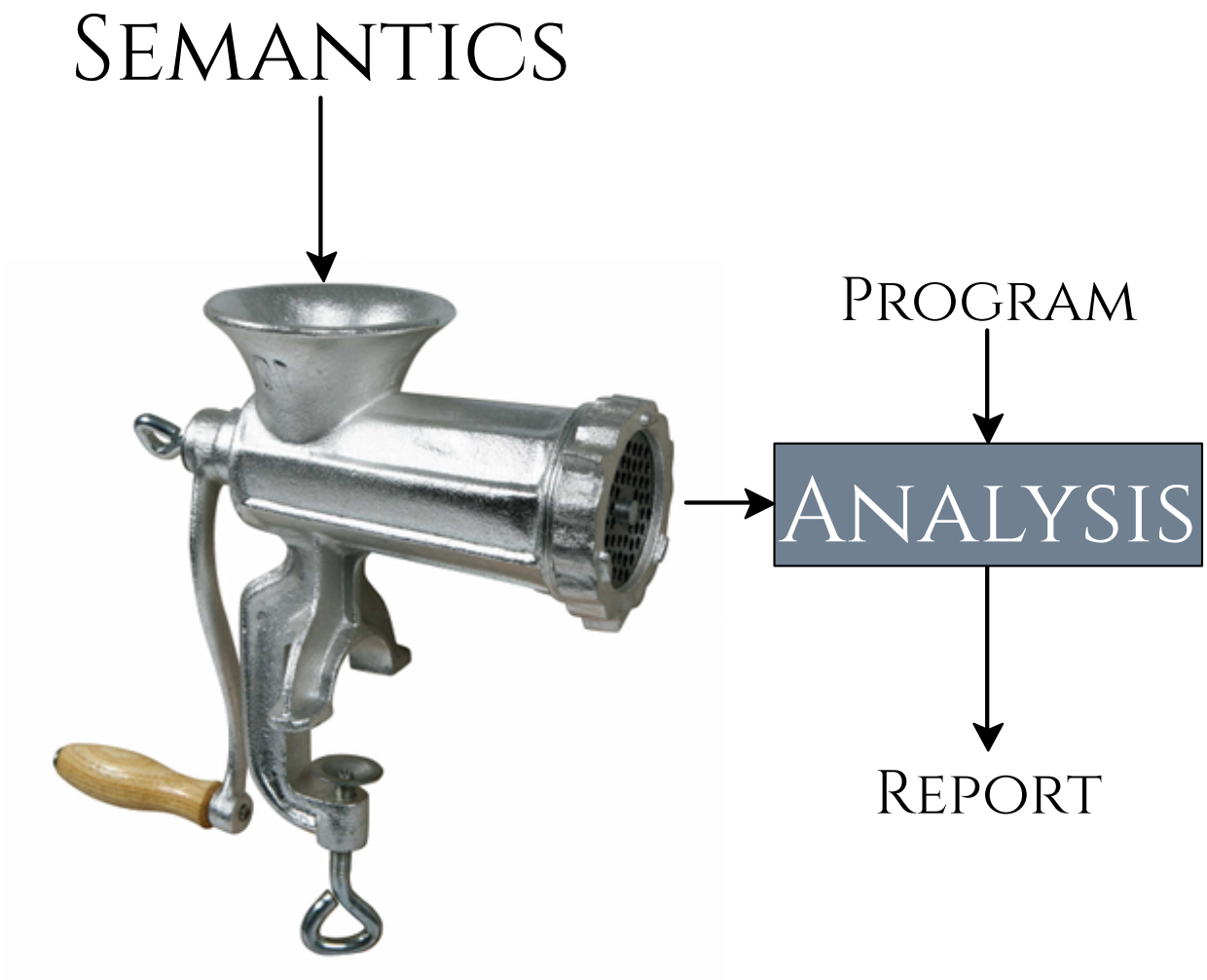

Semantics
Program
Analysis
Report

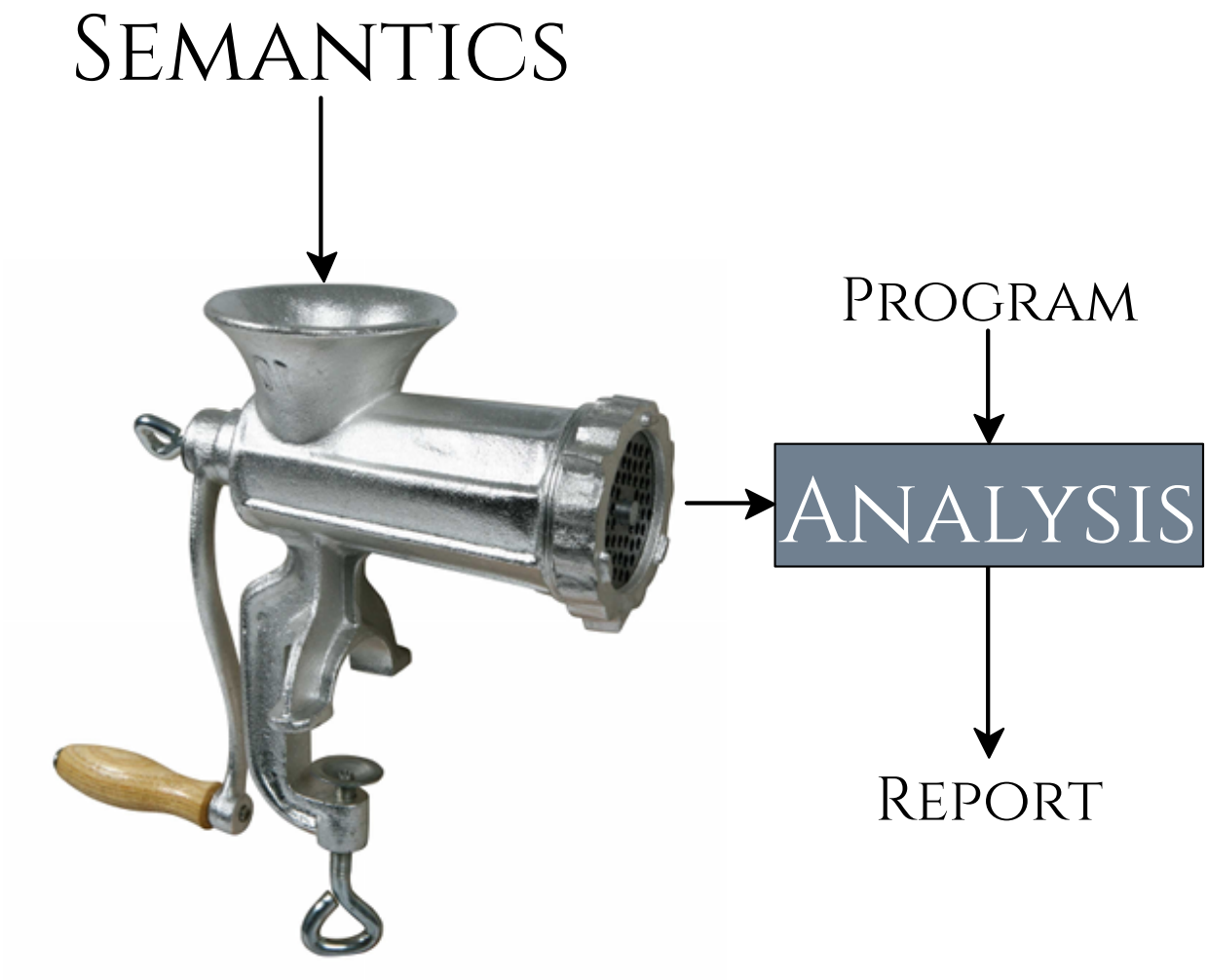

Semantics
Program
Analysis
Report

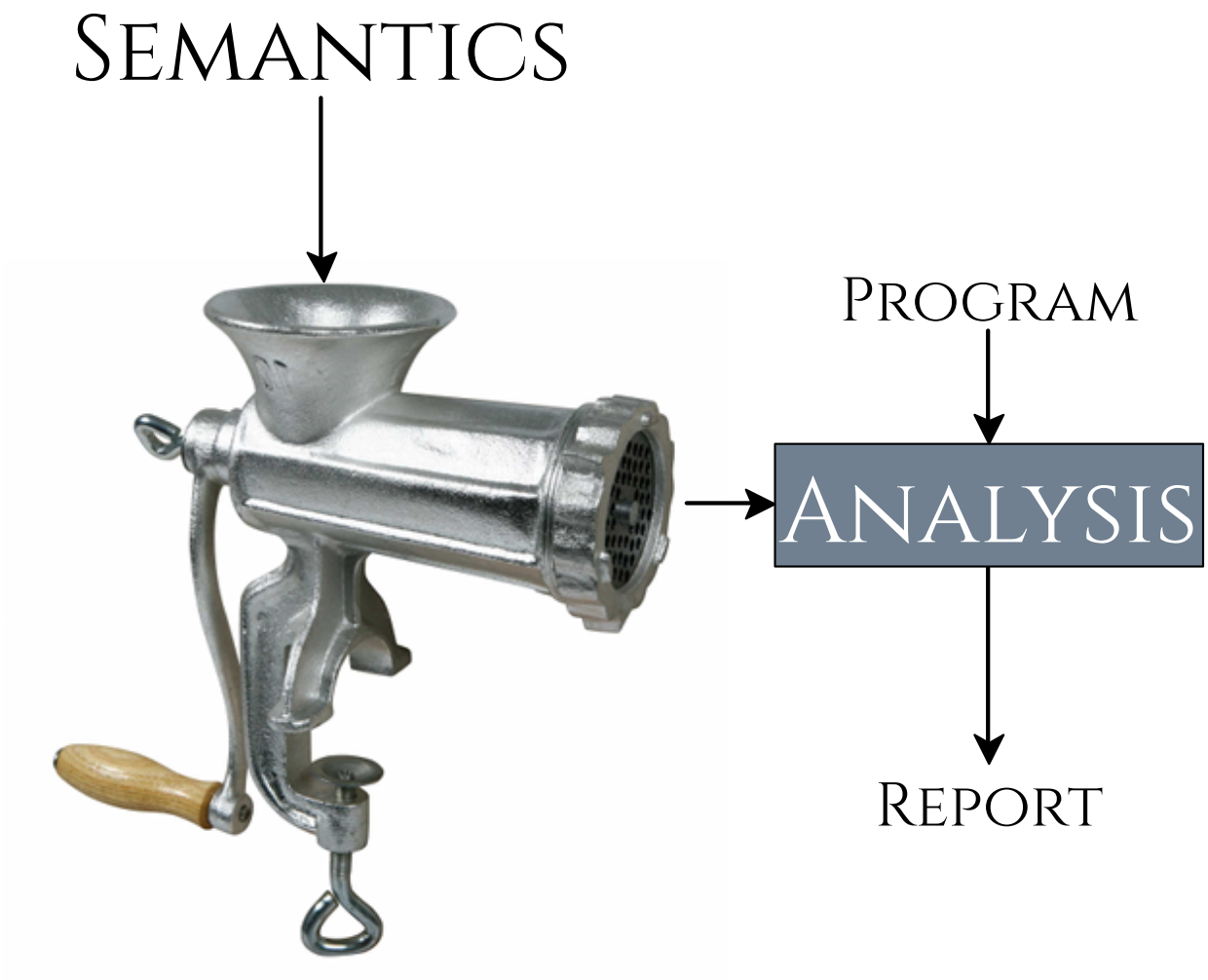

Semantics
Program
Analysis
Report

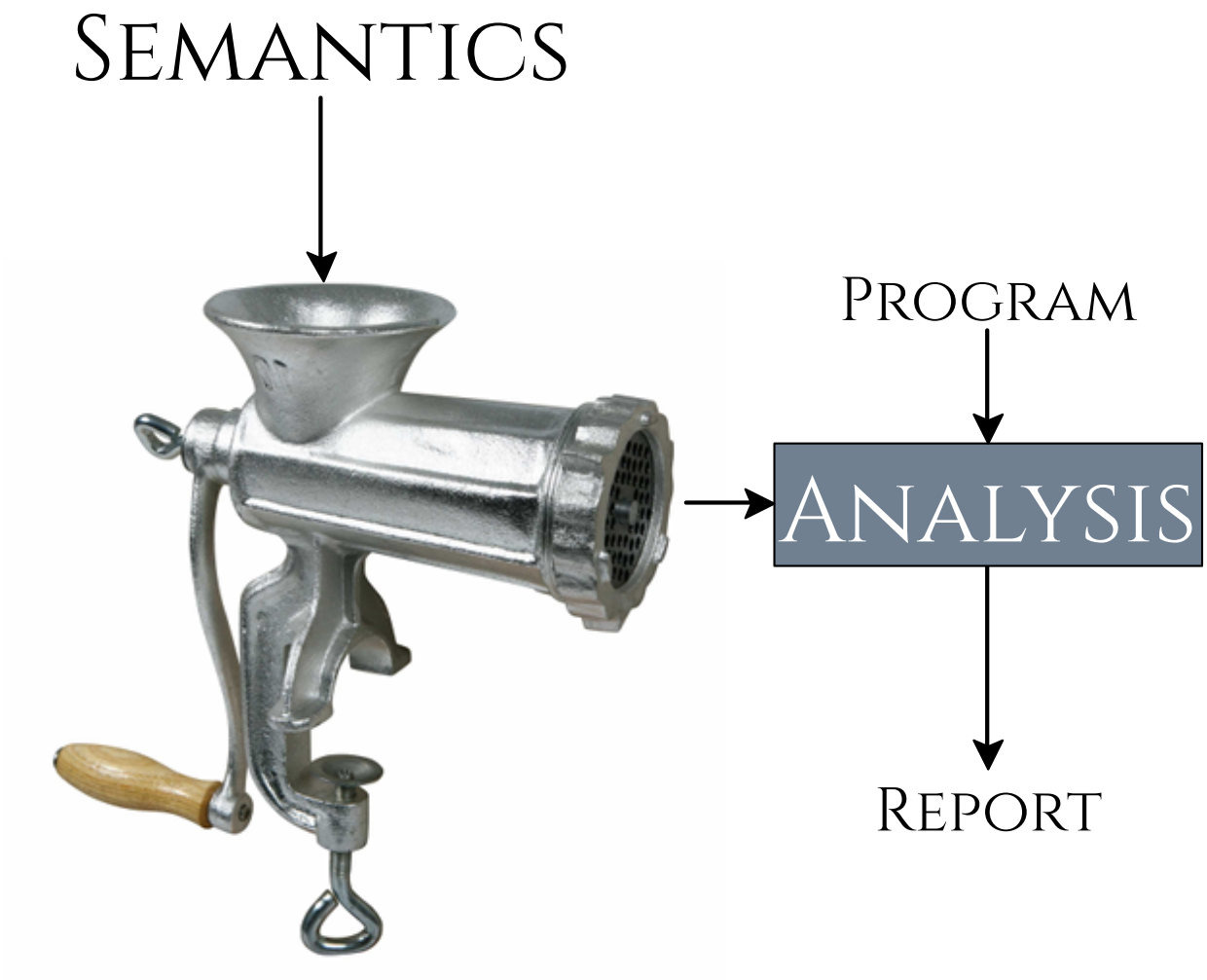

Semantics
Program
Analysis
Report

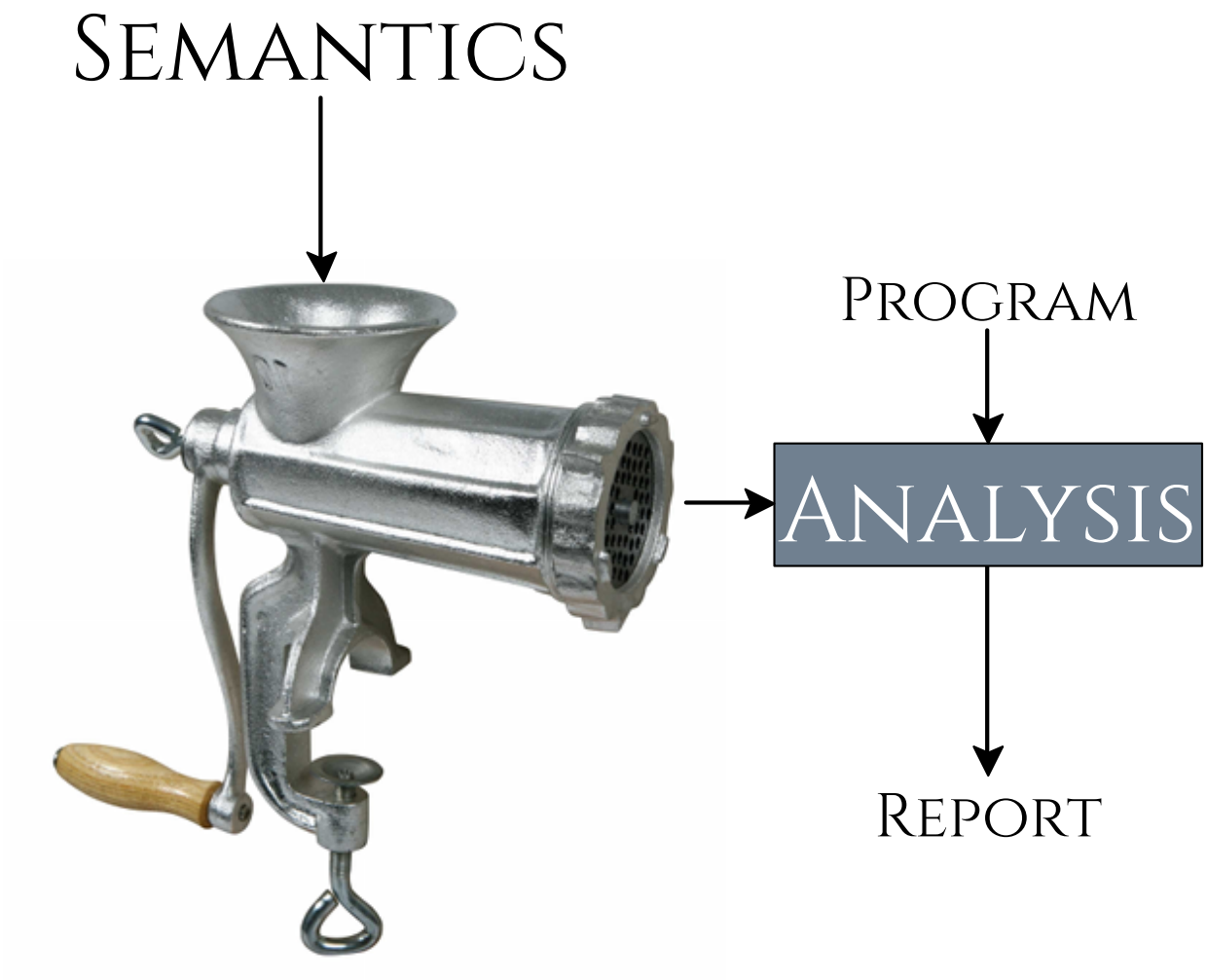

Semantics
Program
Analysis
Report

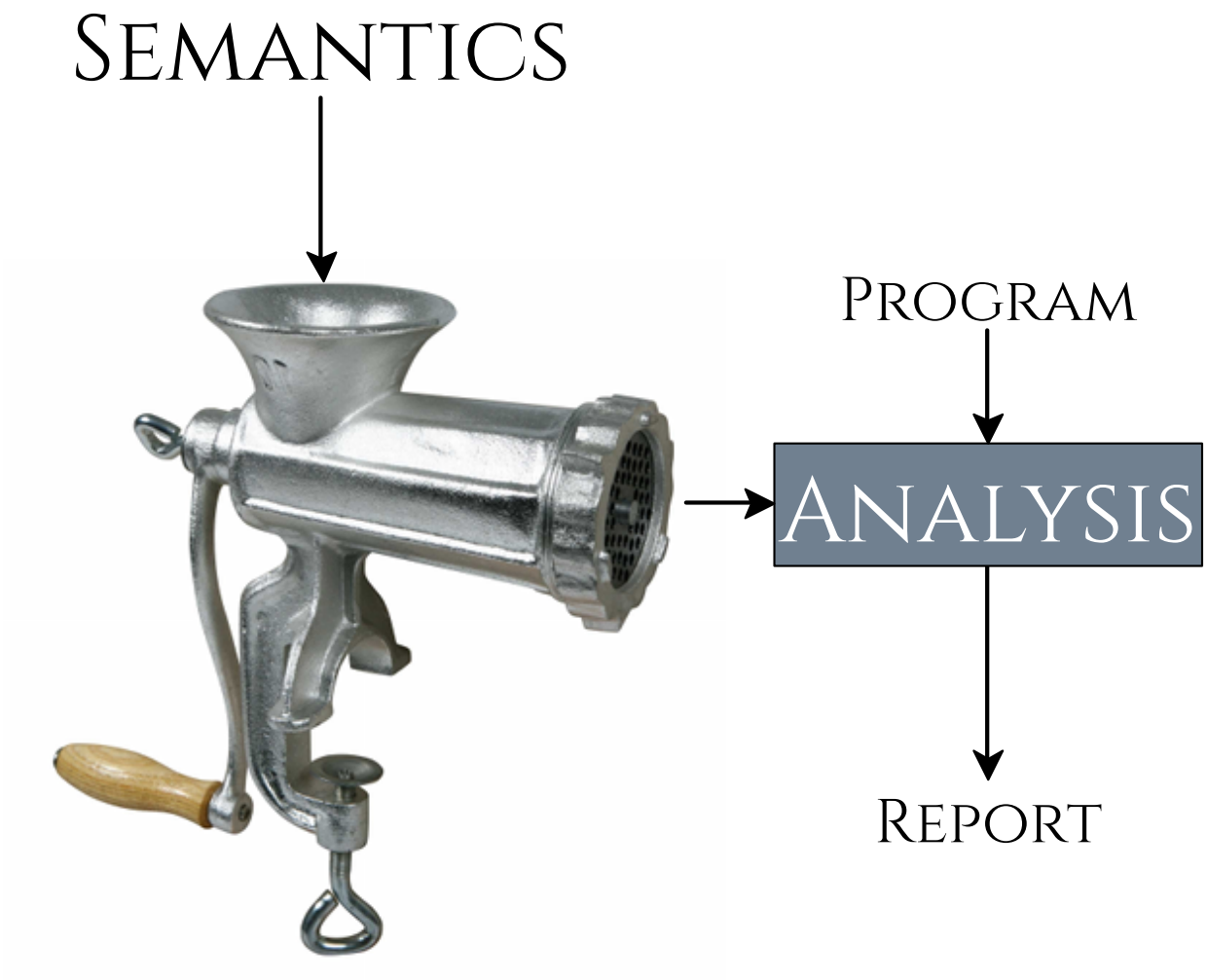

Semantics
Program
Analysis
Report

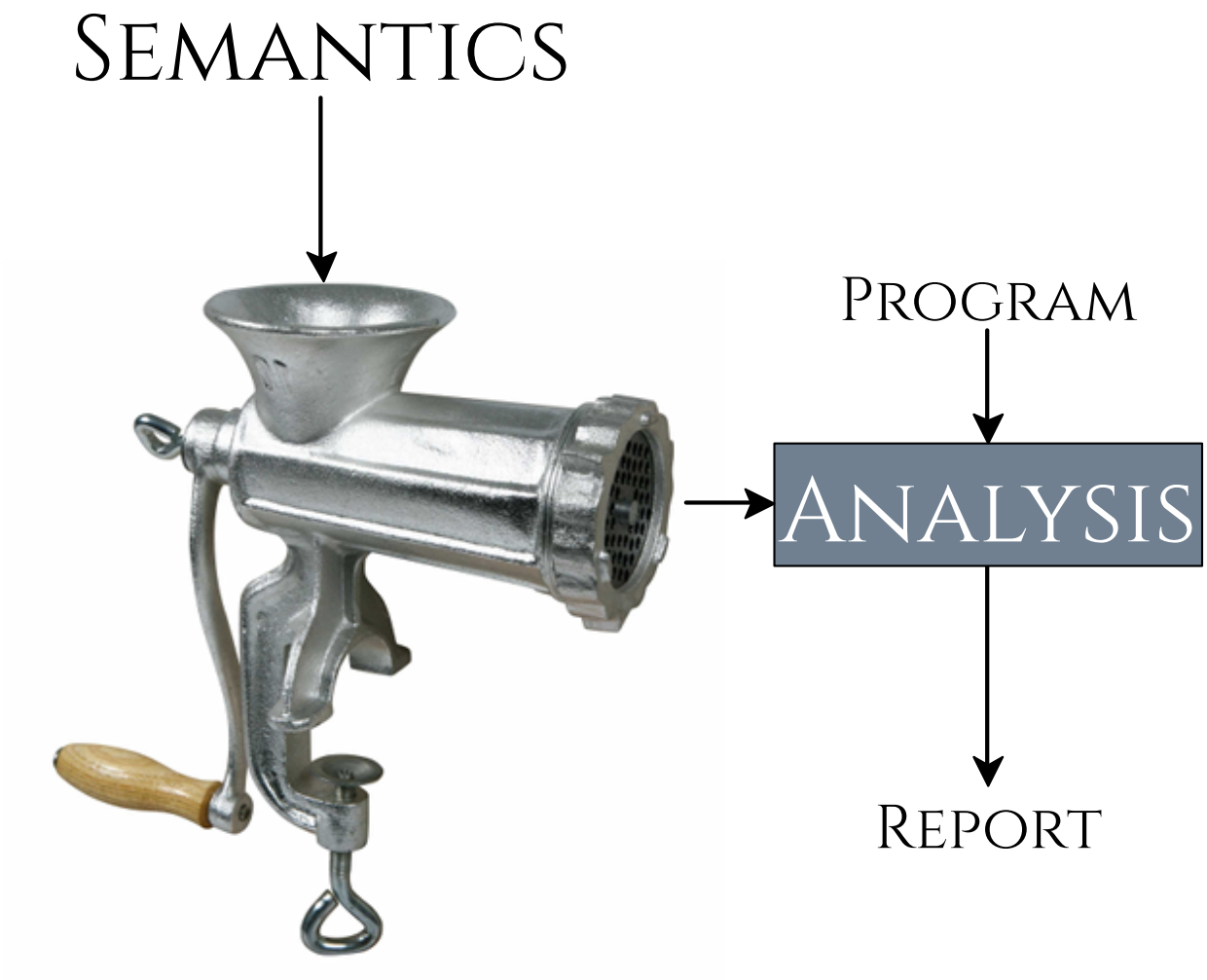

Semantics
Program
Analysis
Report

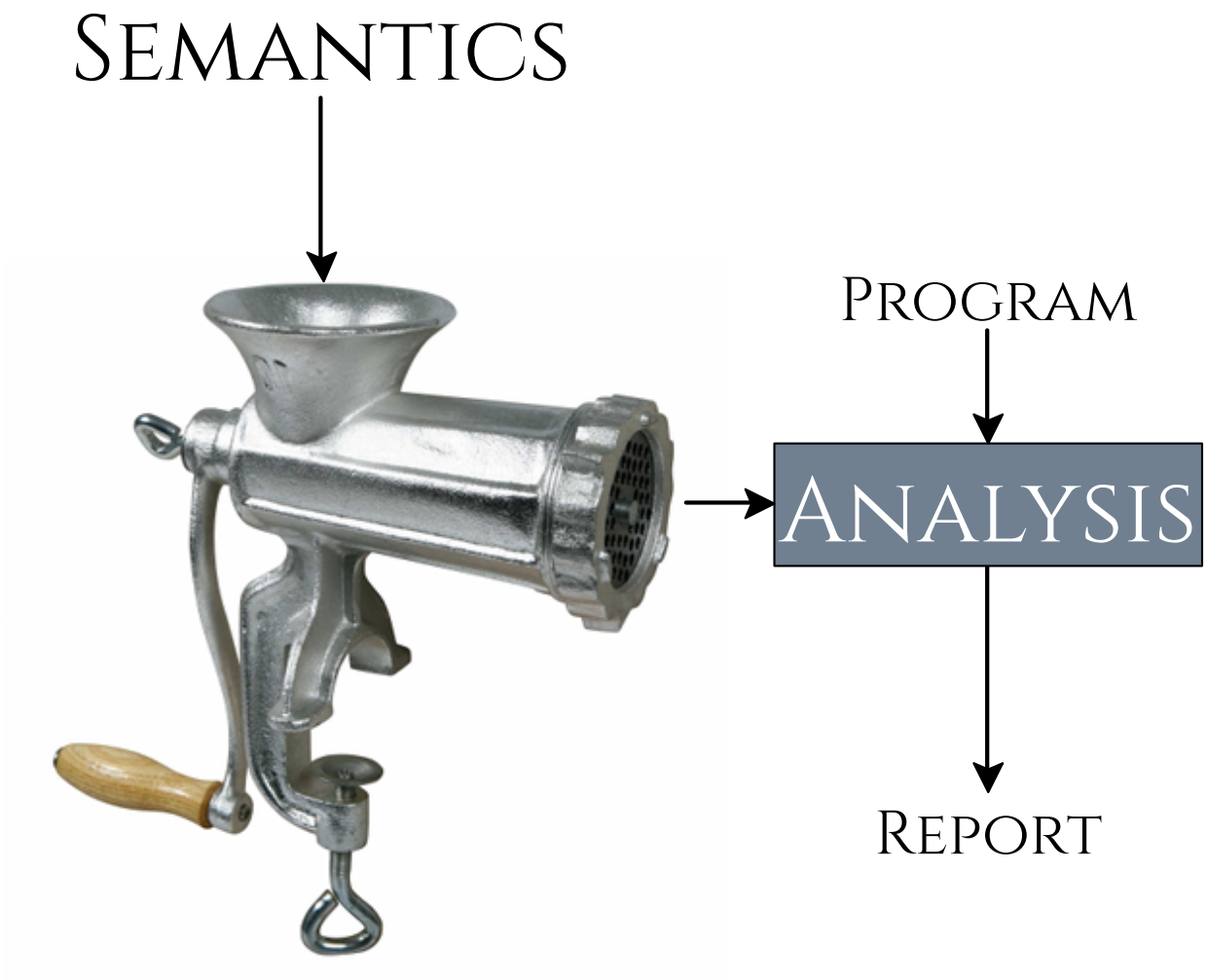

Semantics
Program
Analysis
Report

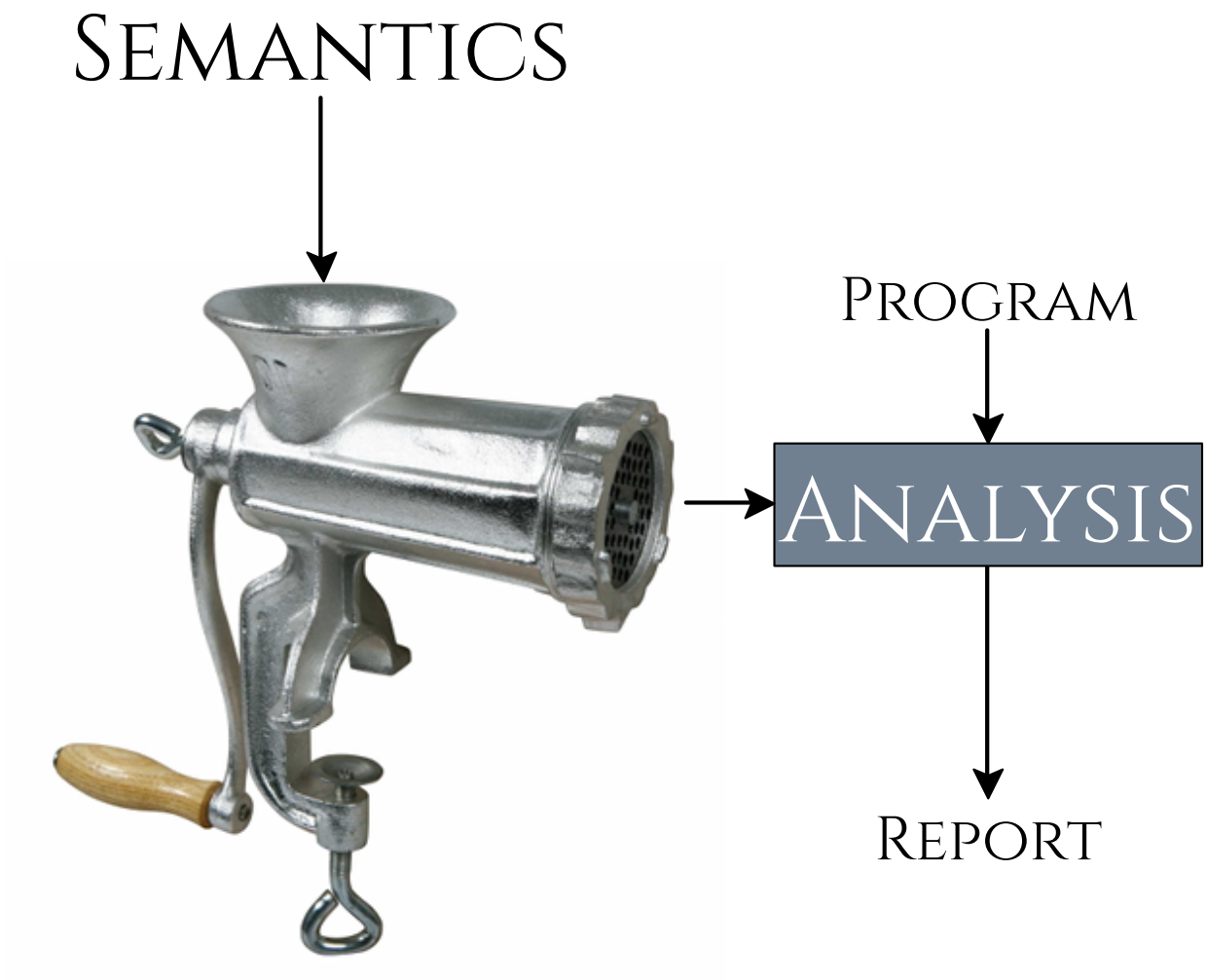

Semantics
Program
Analysis
Report

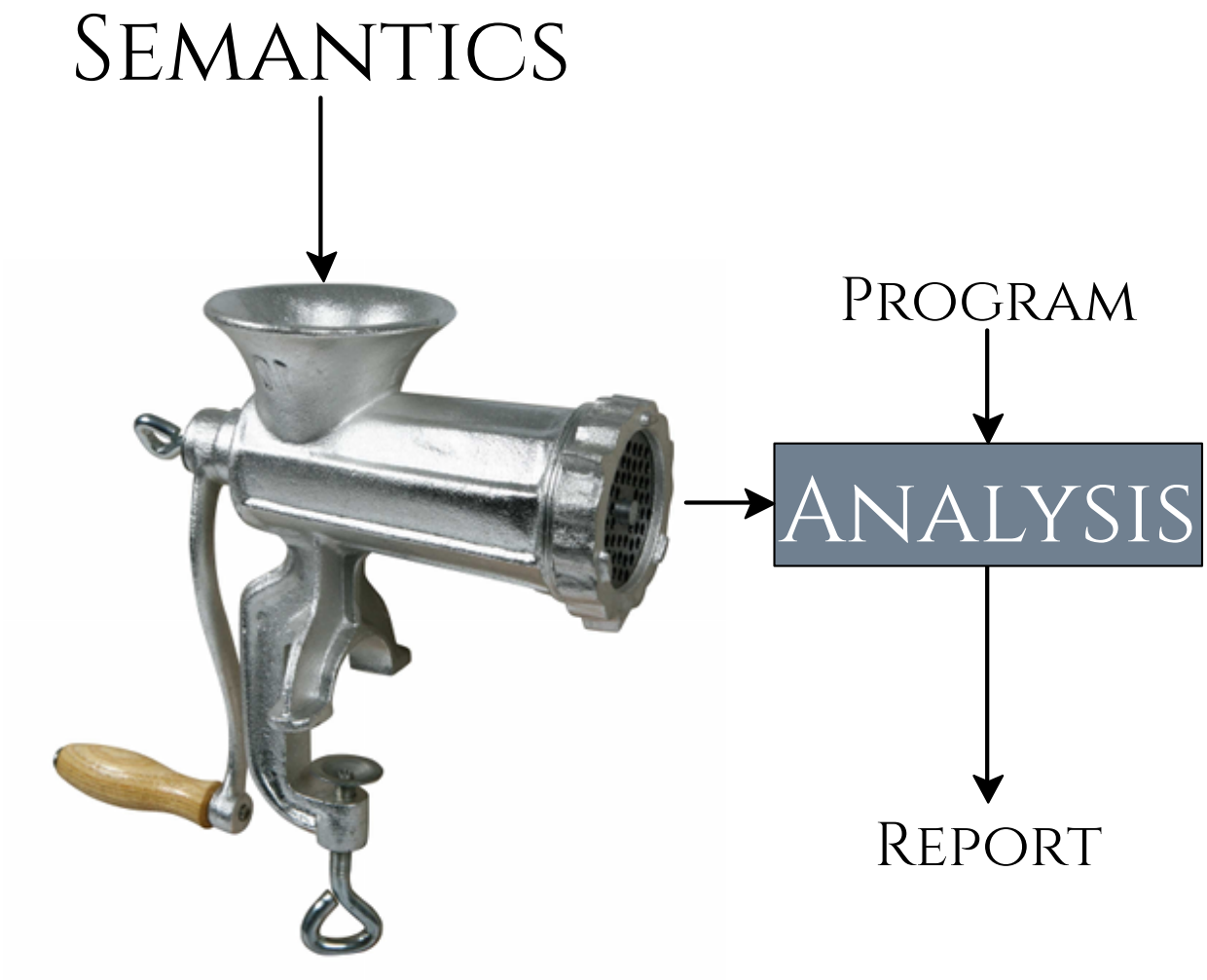

Semantics
Program
Analysis
Report

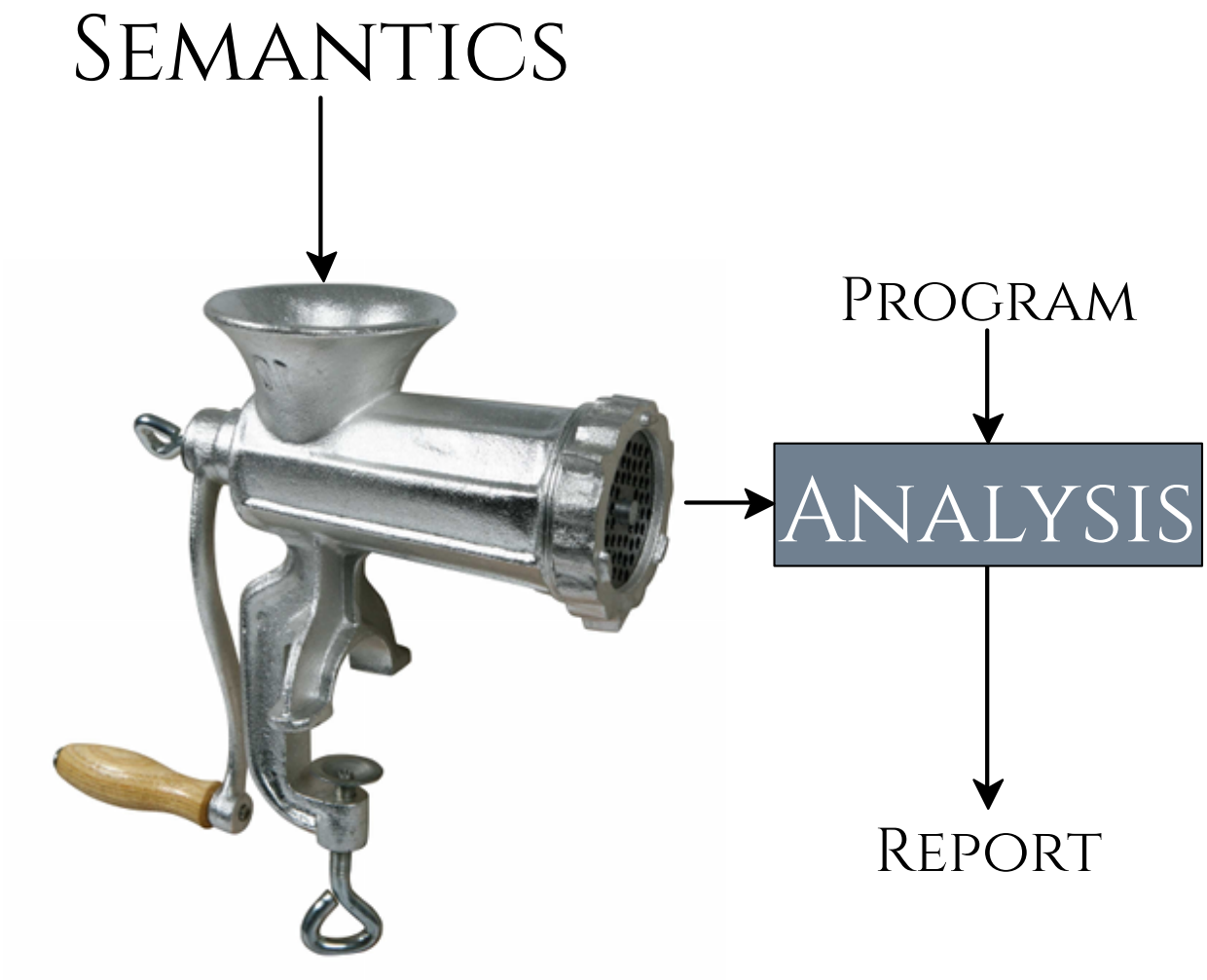

Semantics
Program
Analysis
Report

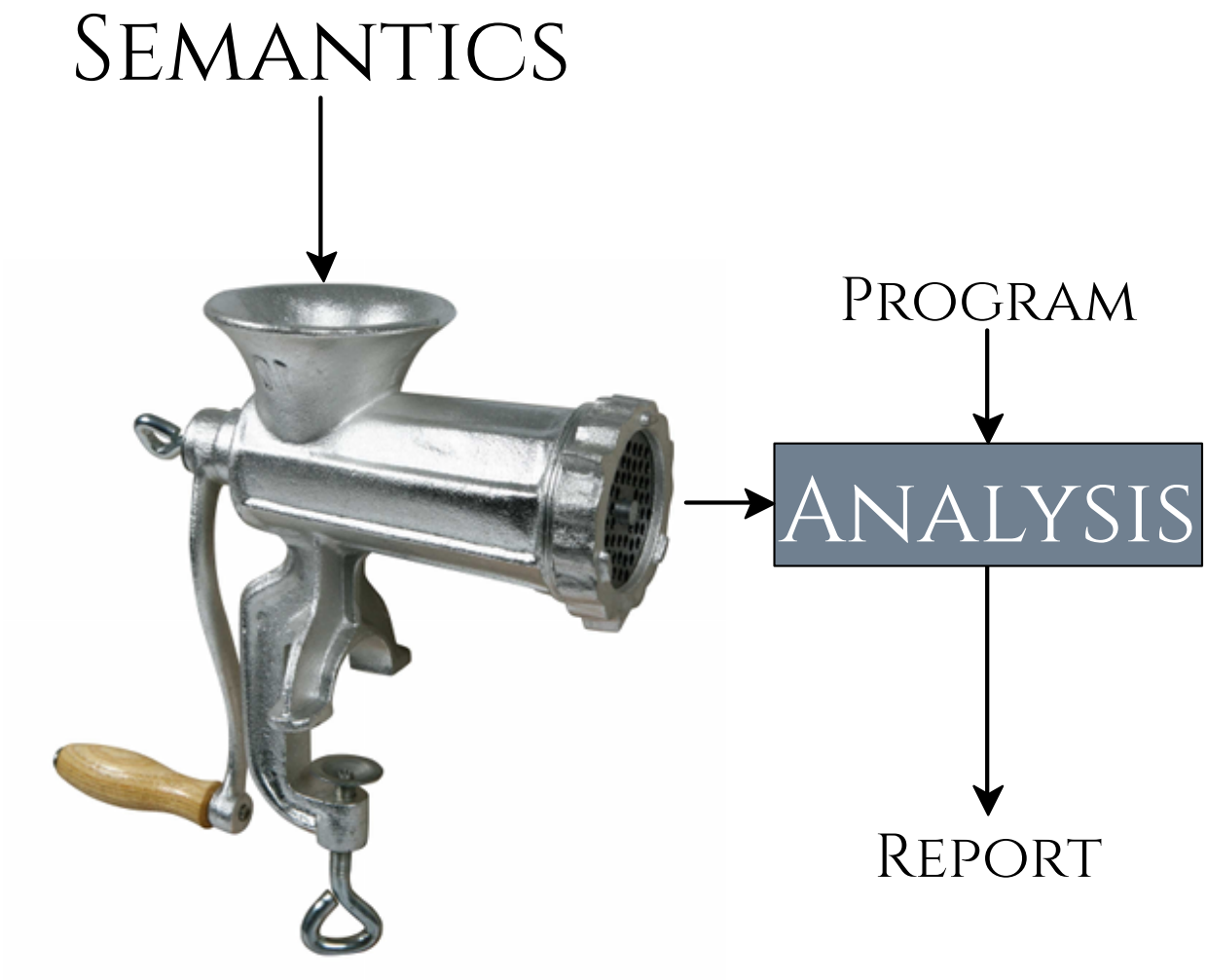

Semantics
Program
Analysis
Report

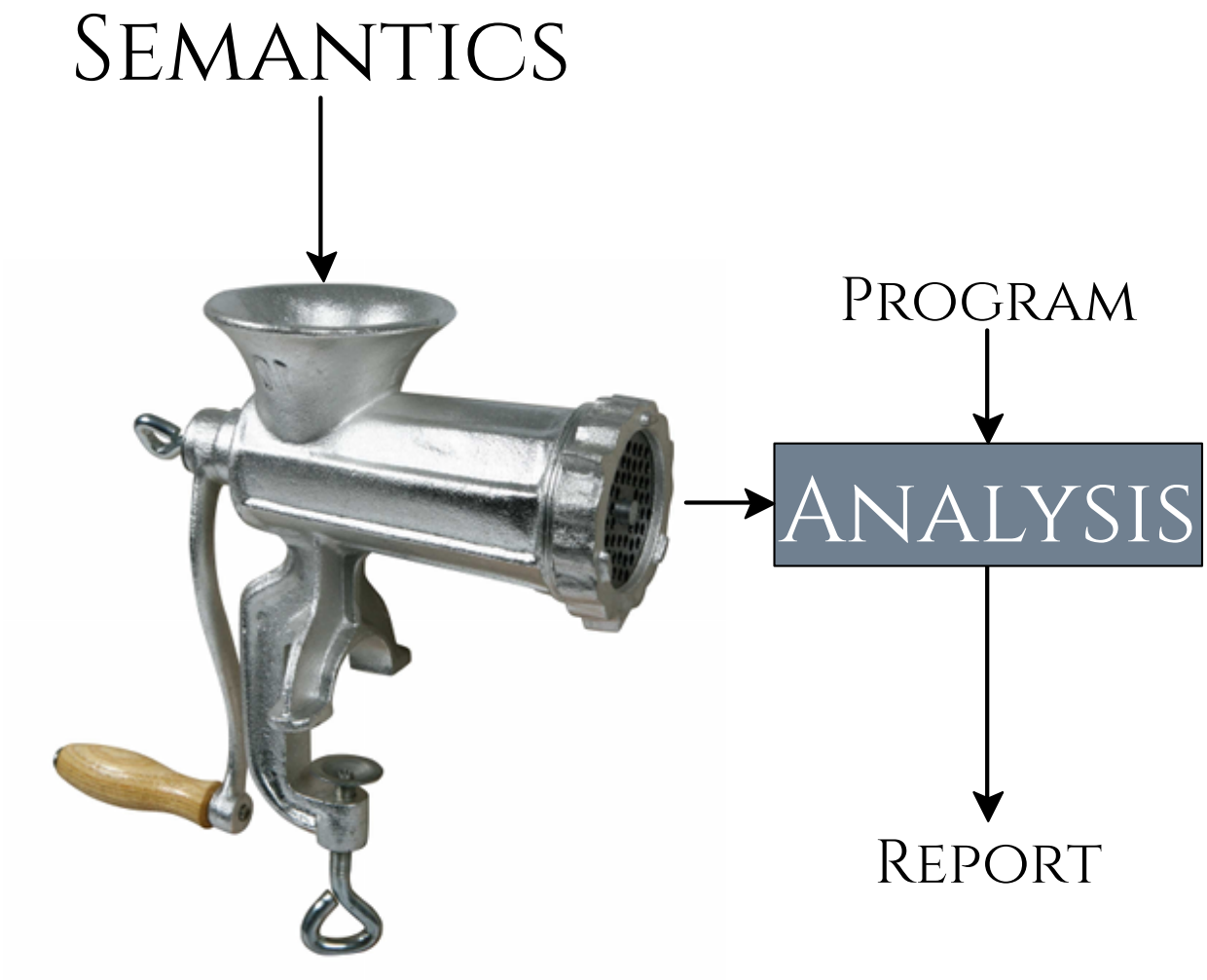

Semantics
Program
Analysis
Report

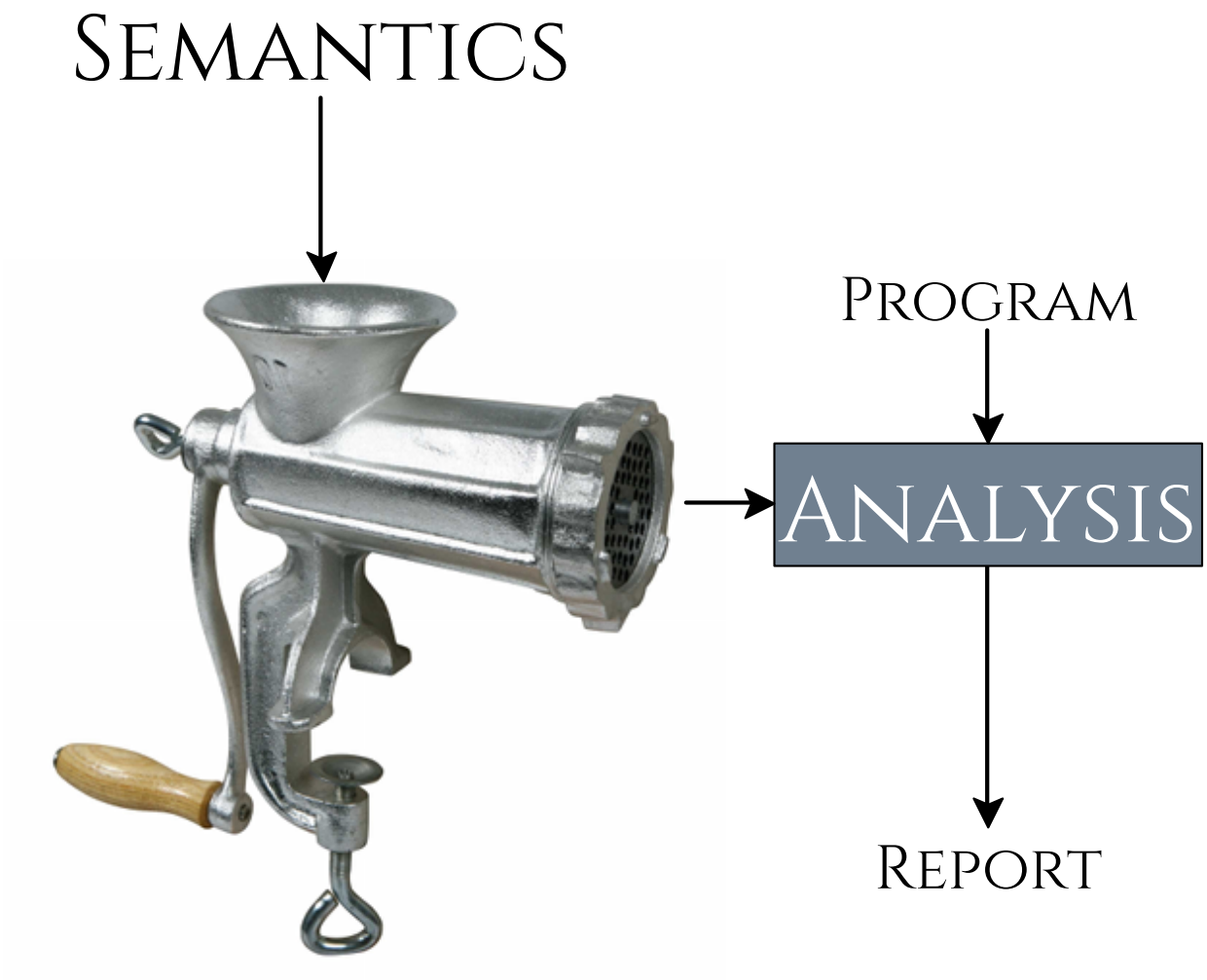

Semantics
Program
Analysis
Report

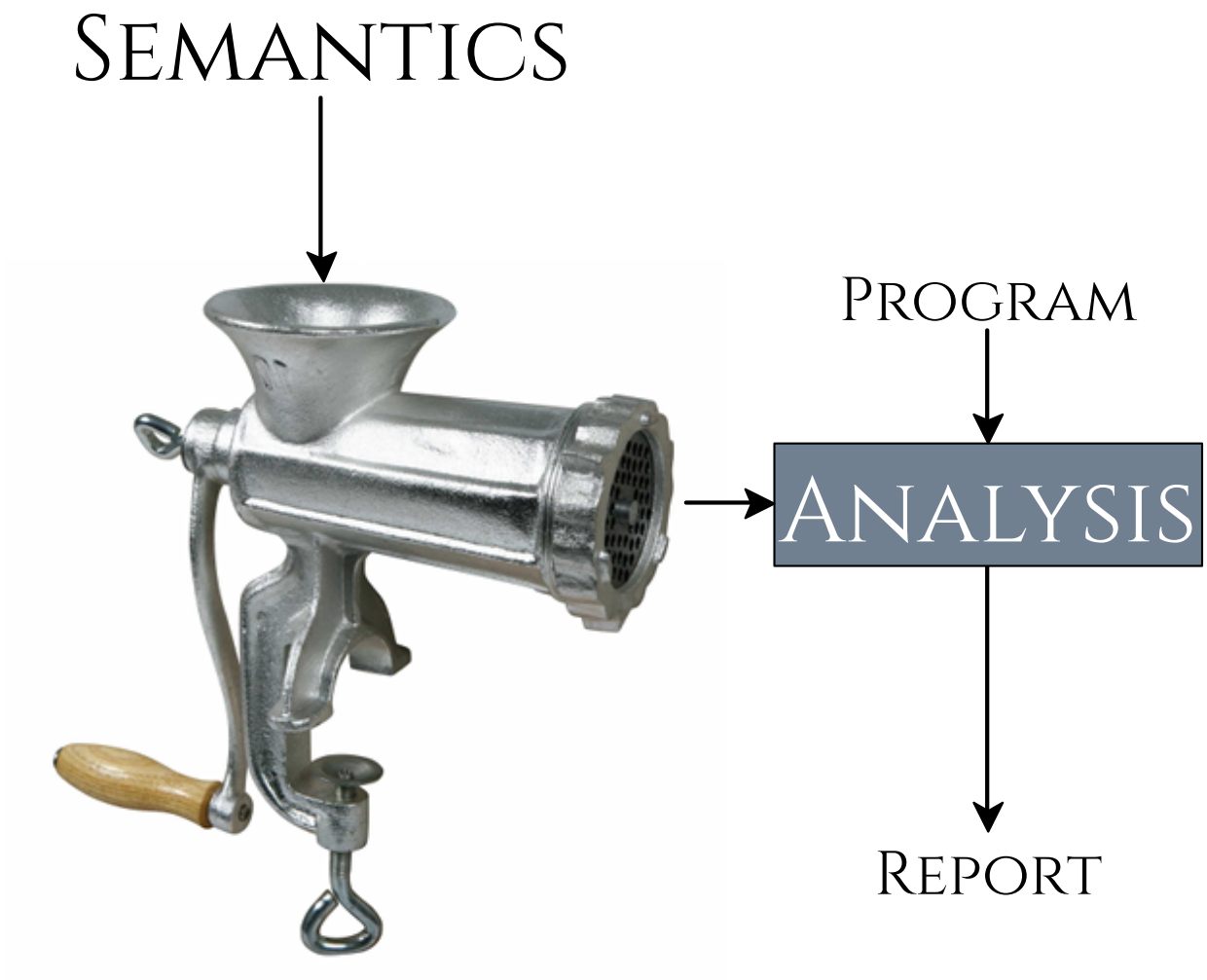

Semantics
Program
Analysis
Report

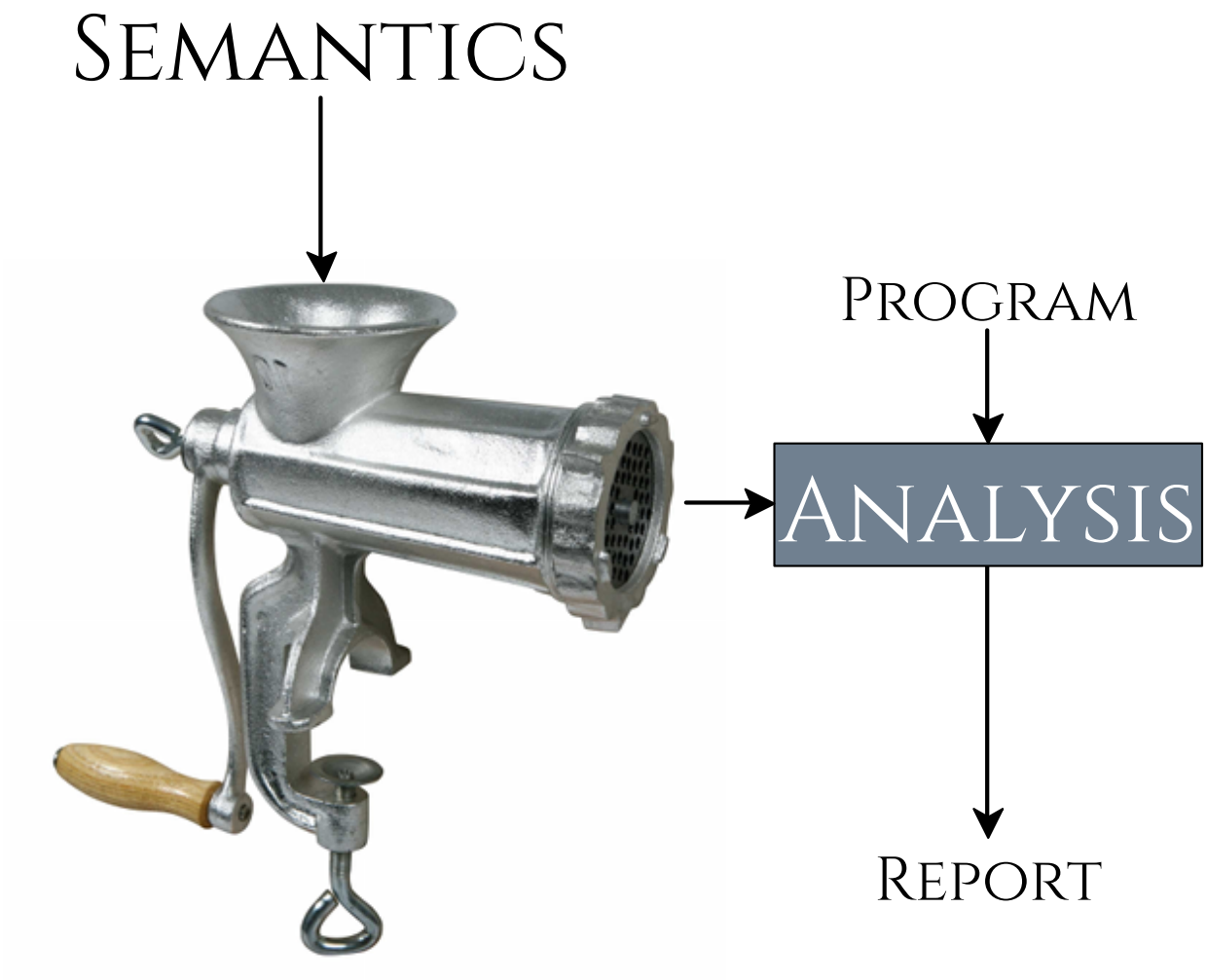

Semantics
Program
Analysis
Report

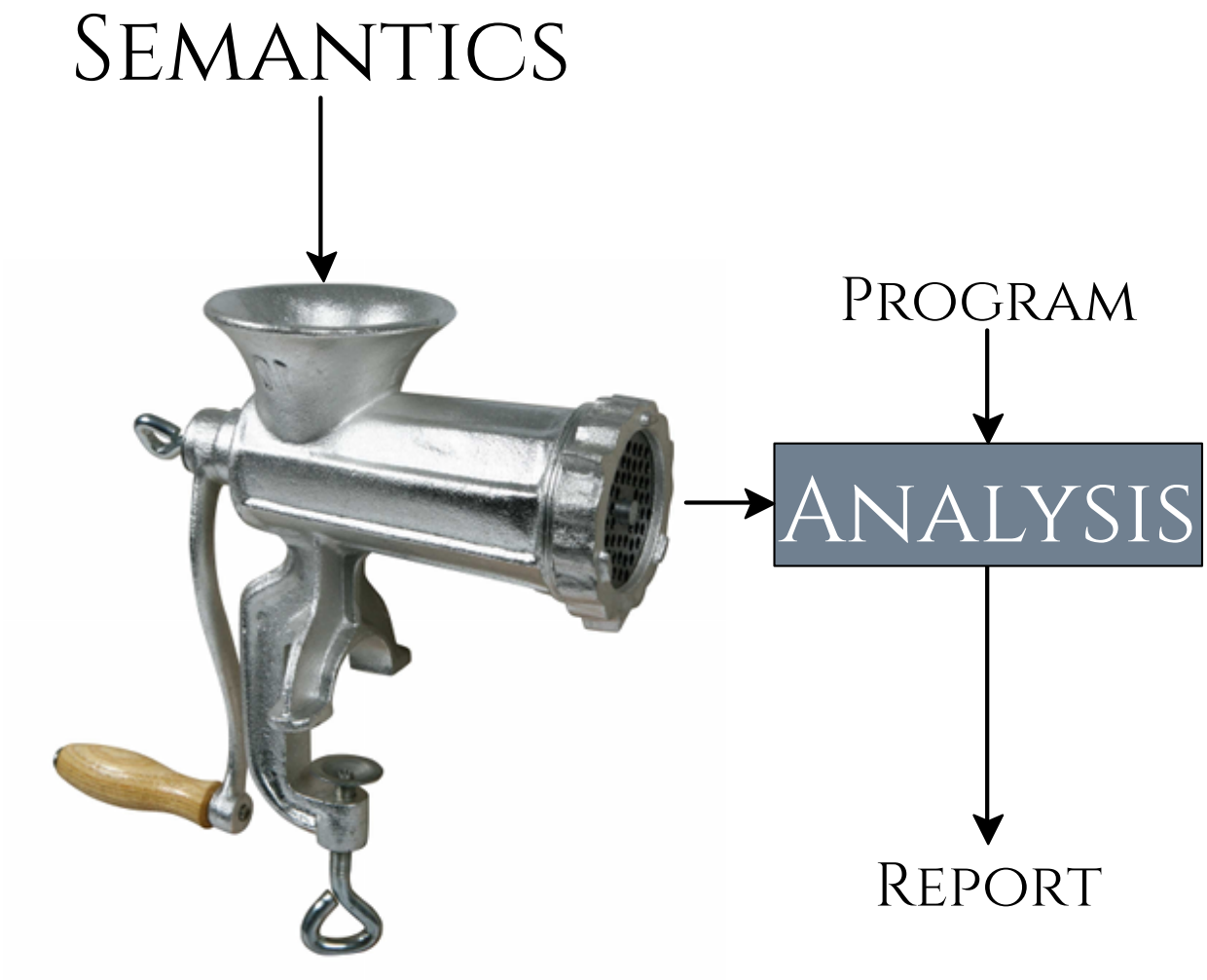

Semantics
Program
Analysis
Report

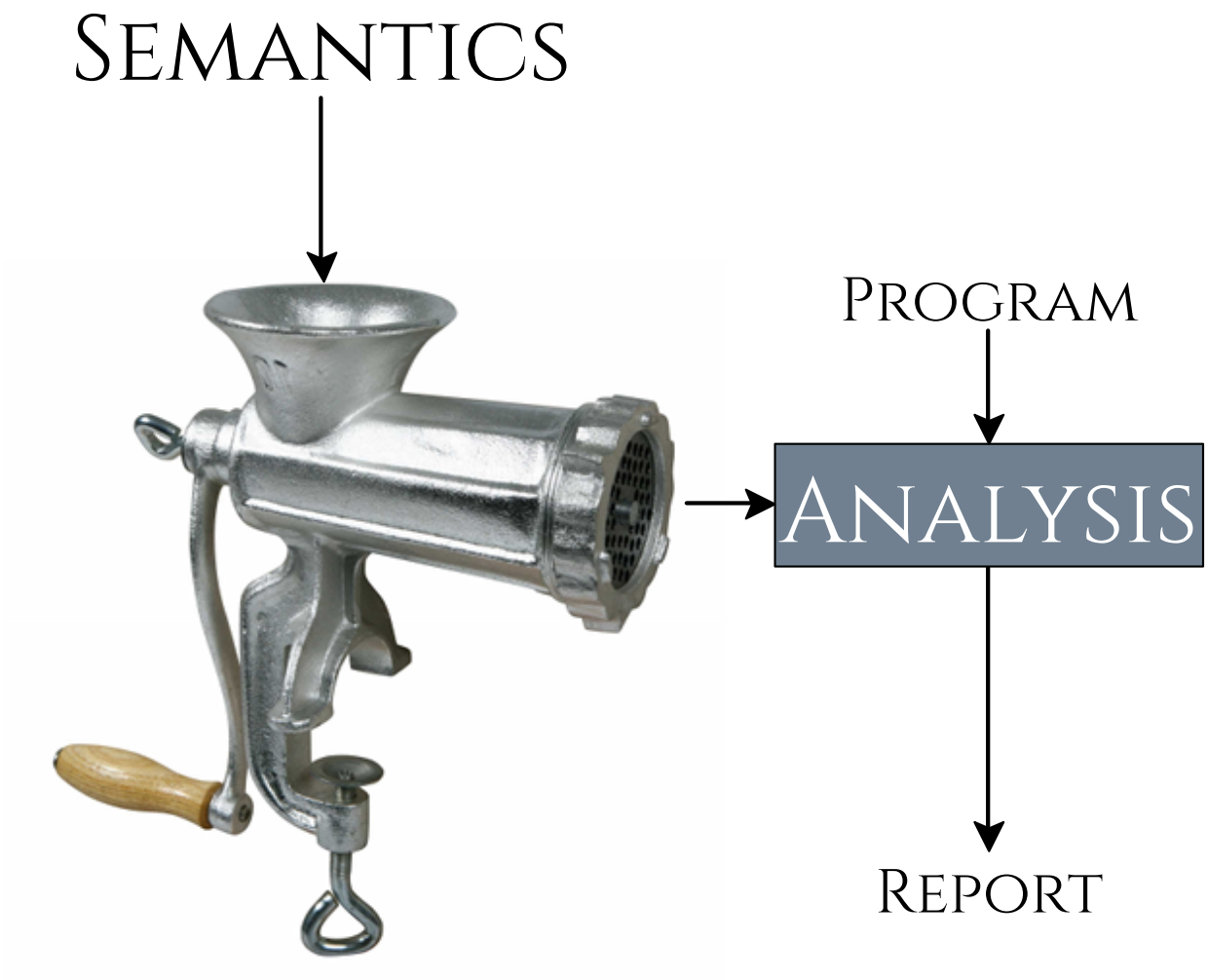

Semantics
Program
Analysis
Report

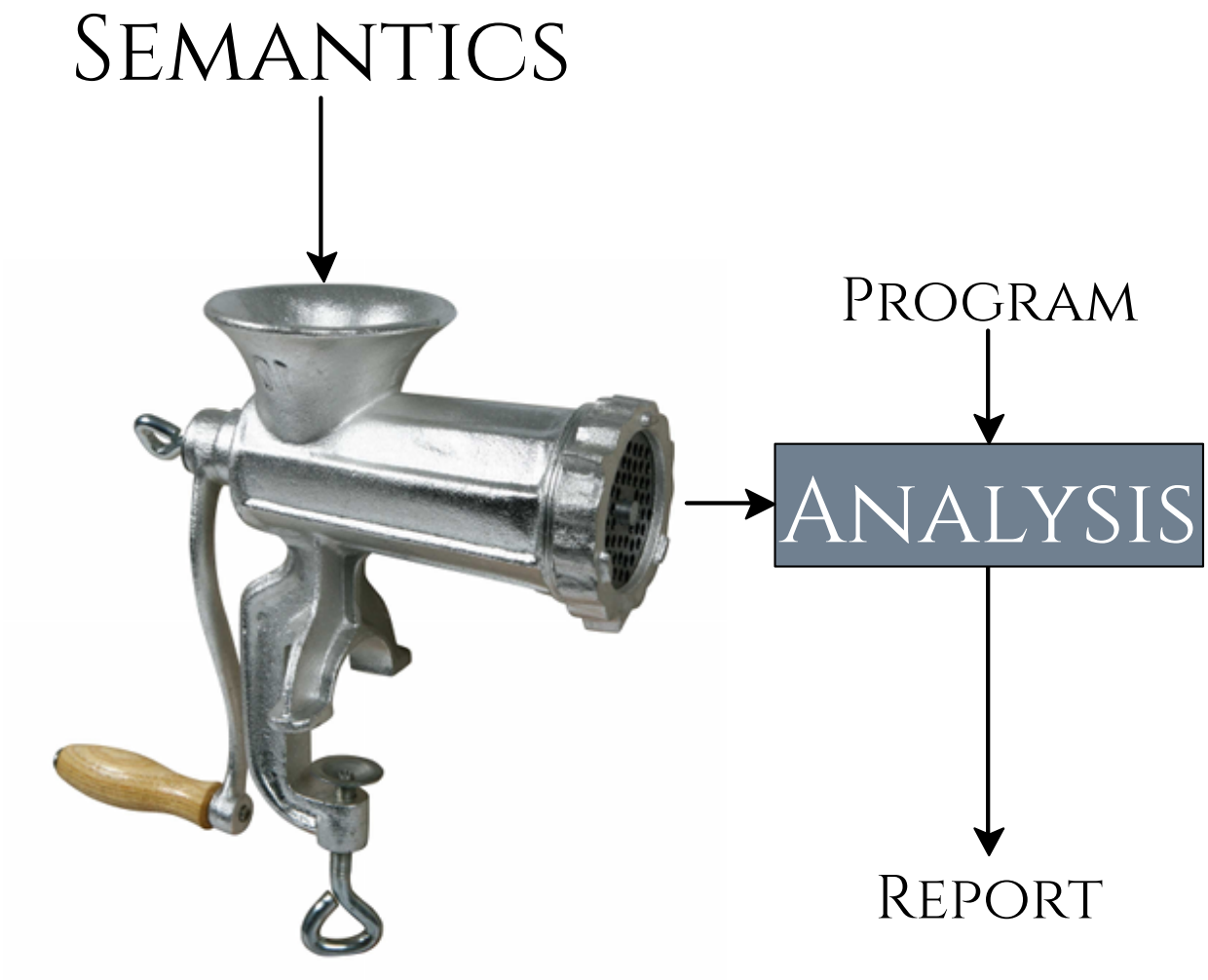

Semantics
Program
Analysis
Report

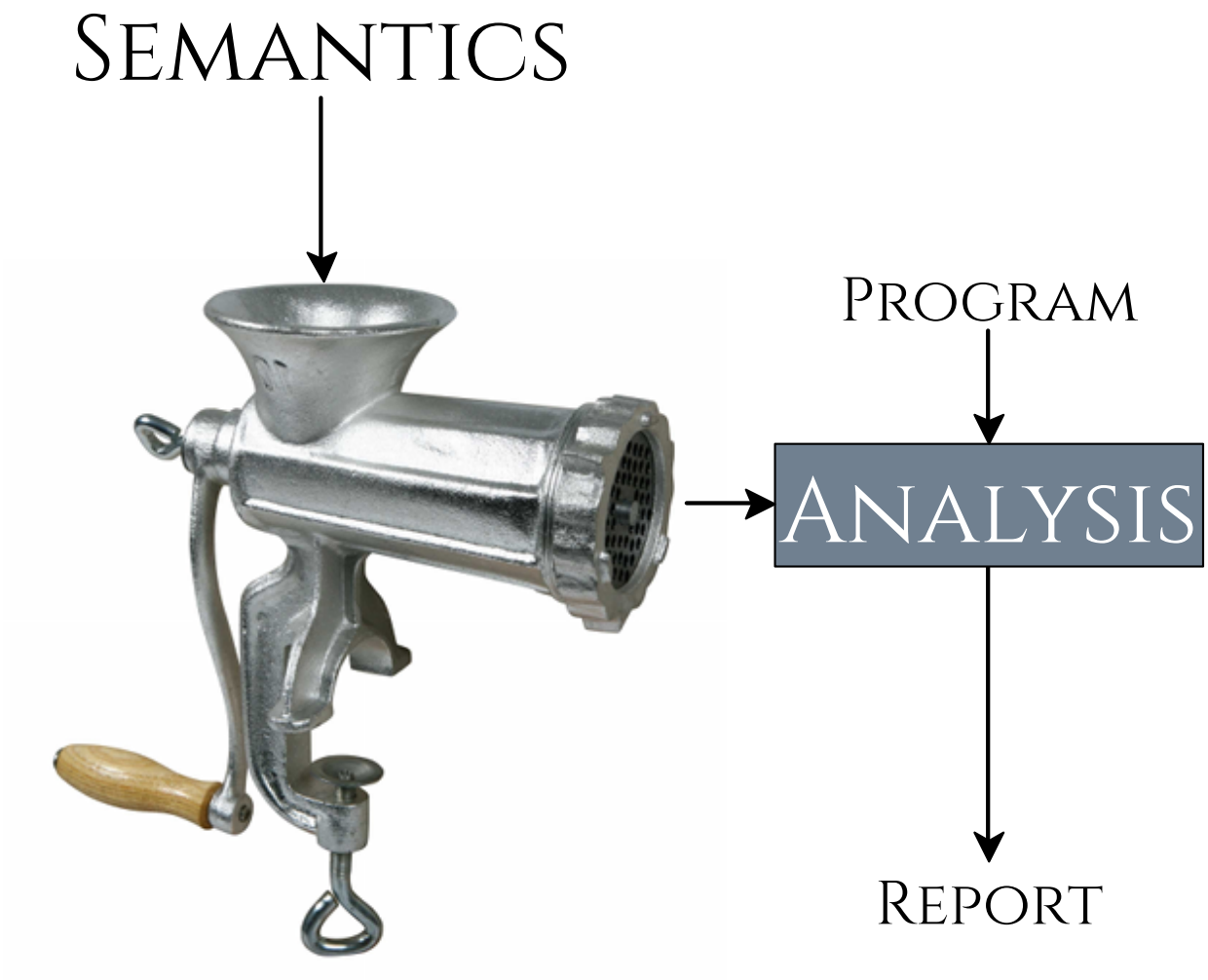

Semantics
Program
Analysis
Report

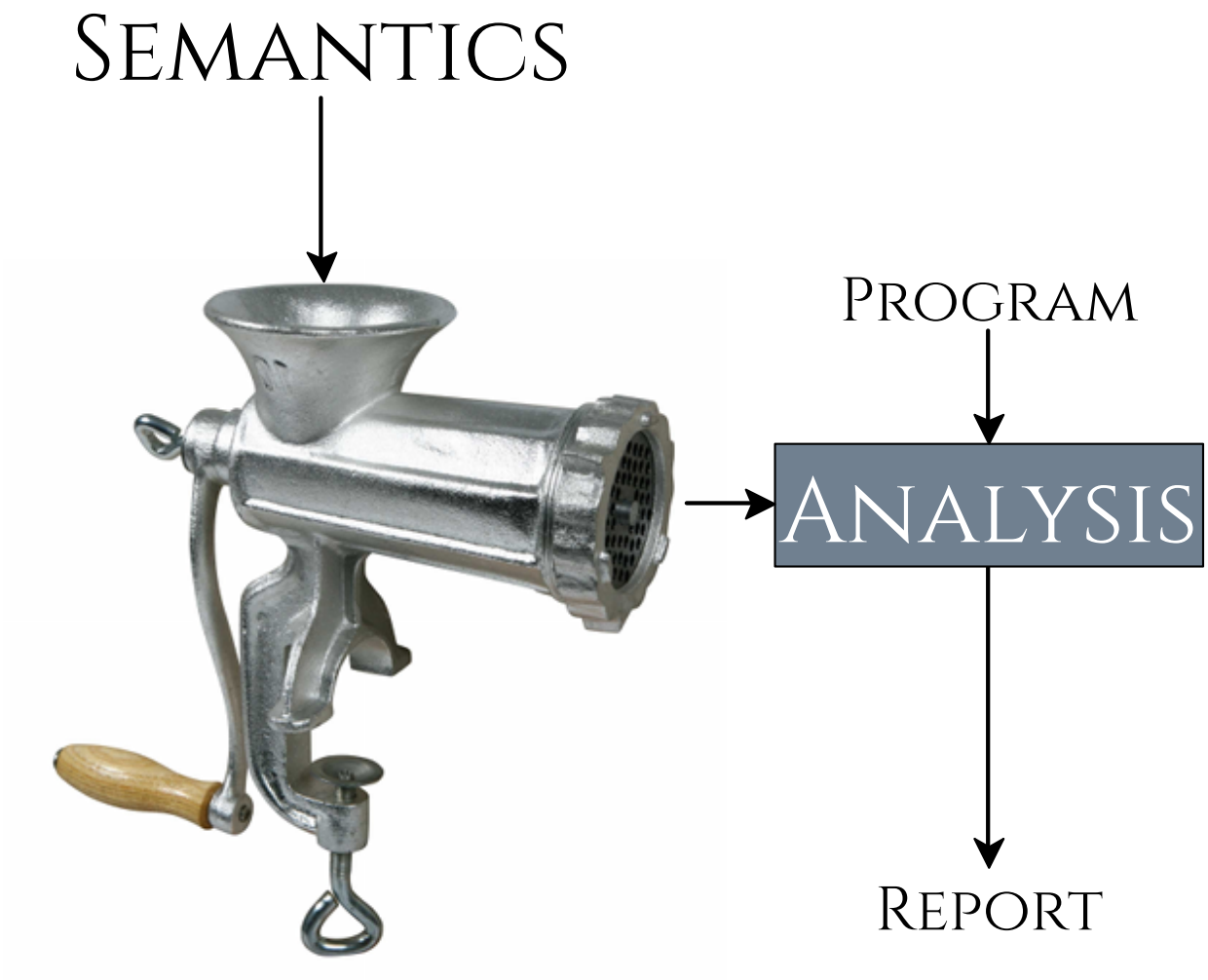

Semantics
Program
Analysis
Report

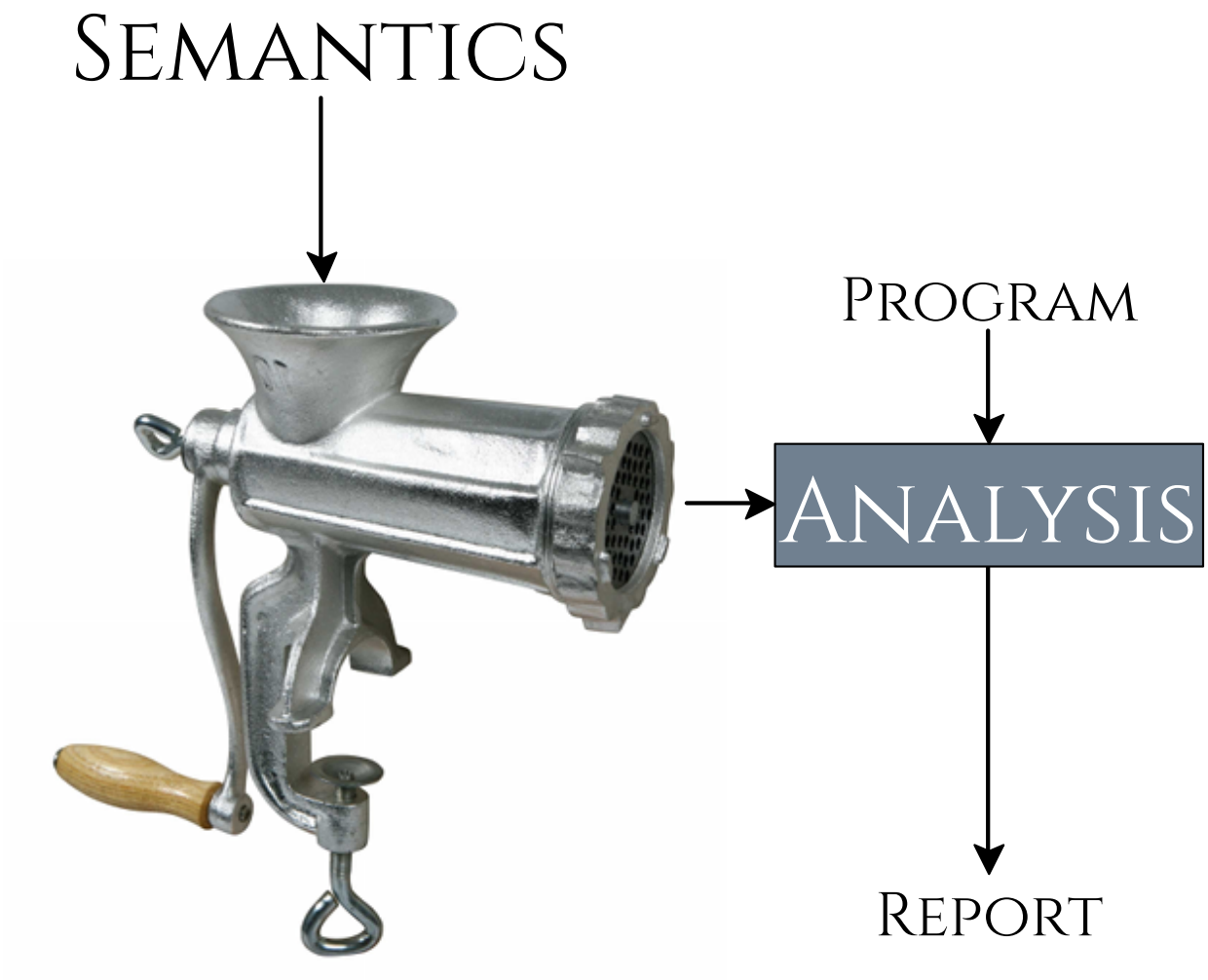

Semantics
Program
Analysis
Report

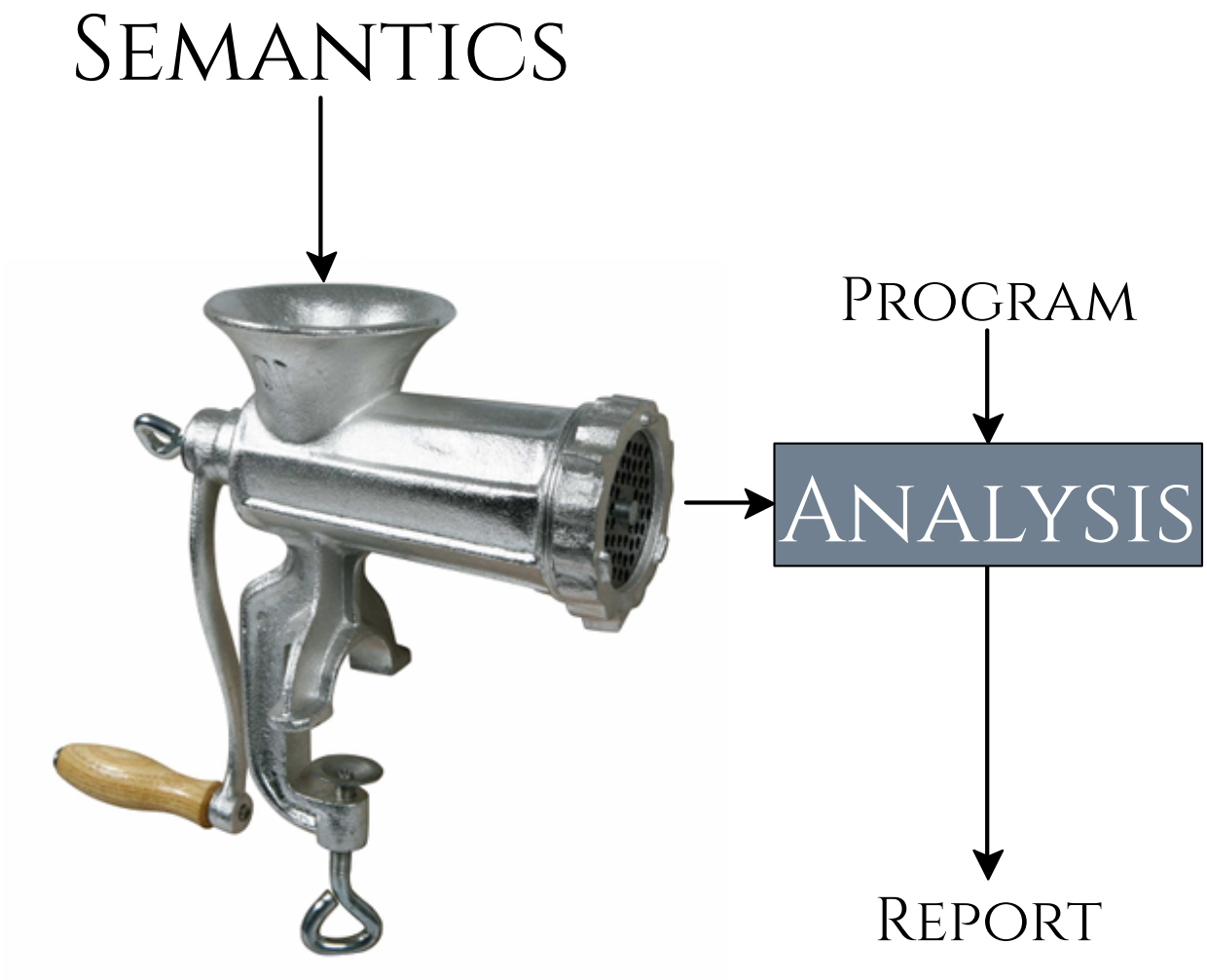

Semantics
Program
Analysis
Report

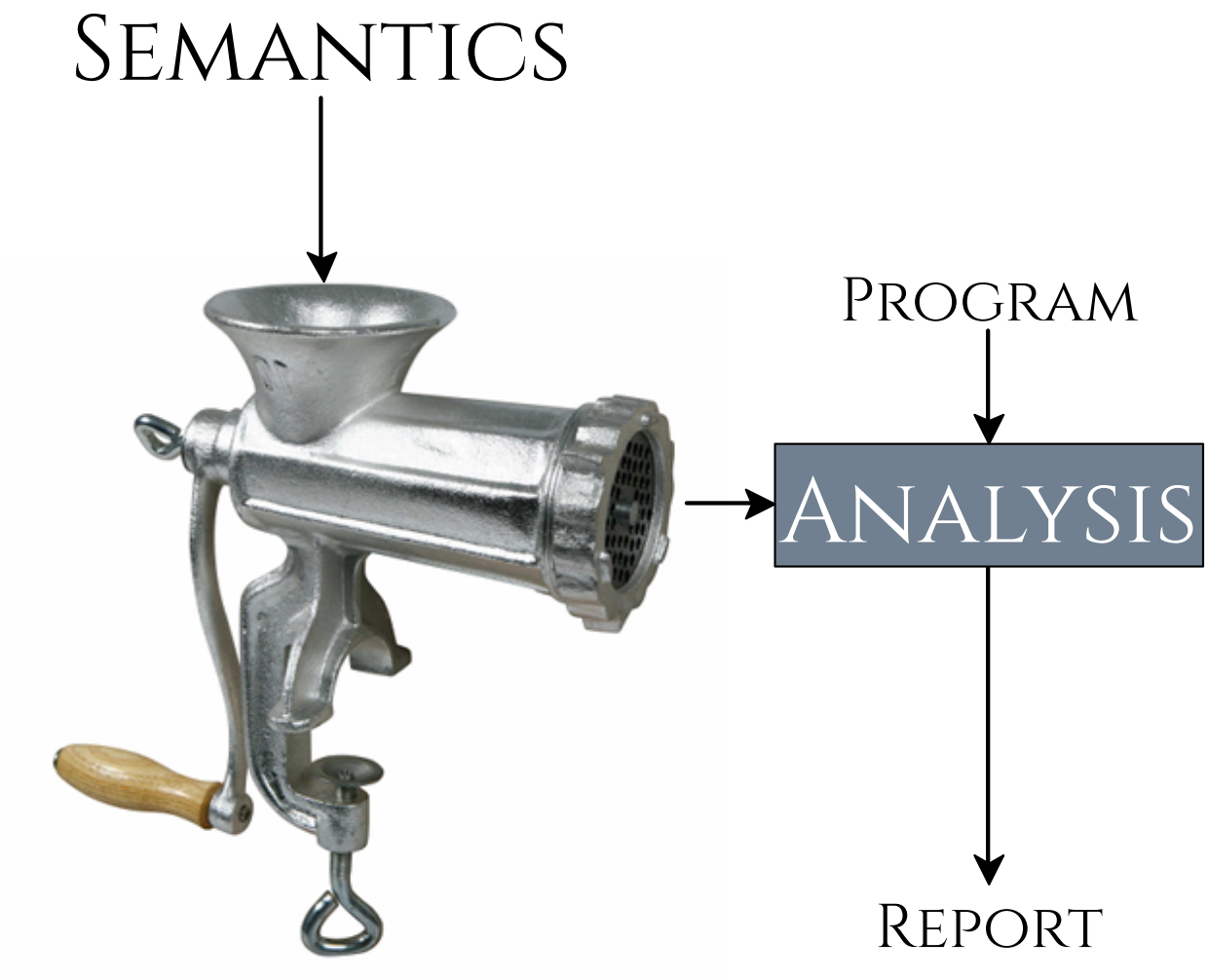

Semantics
Program
Analysis
Report

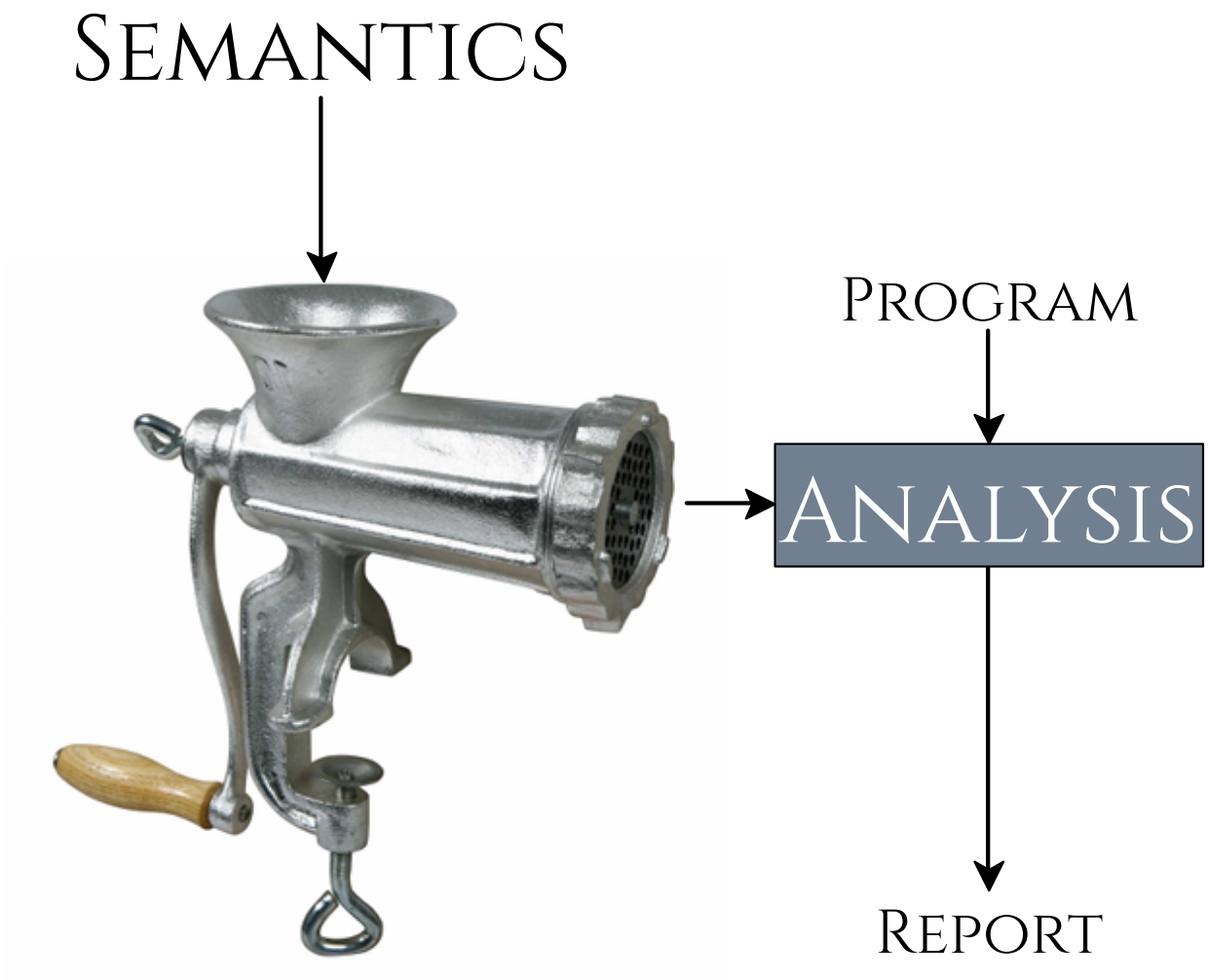

Semantics
Program
Analysis
Report

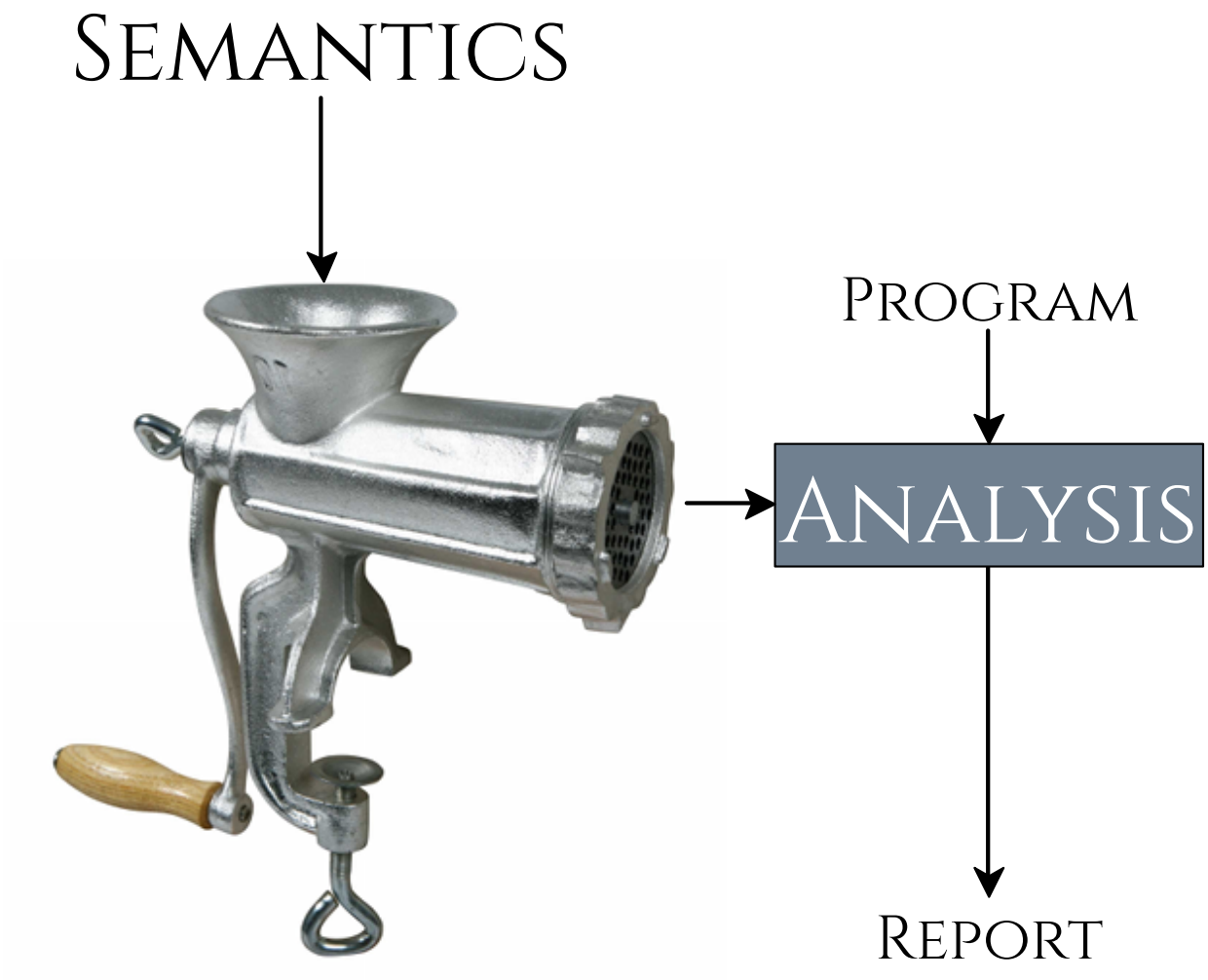

Semantics
Program
Analysis
Report

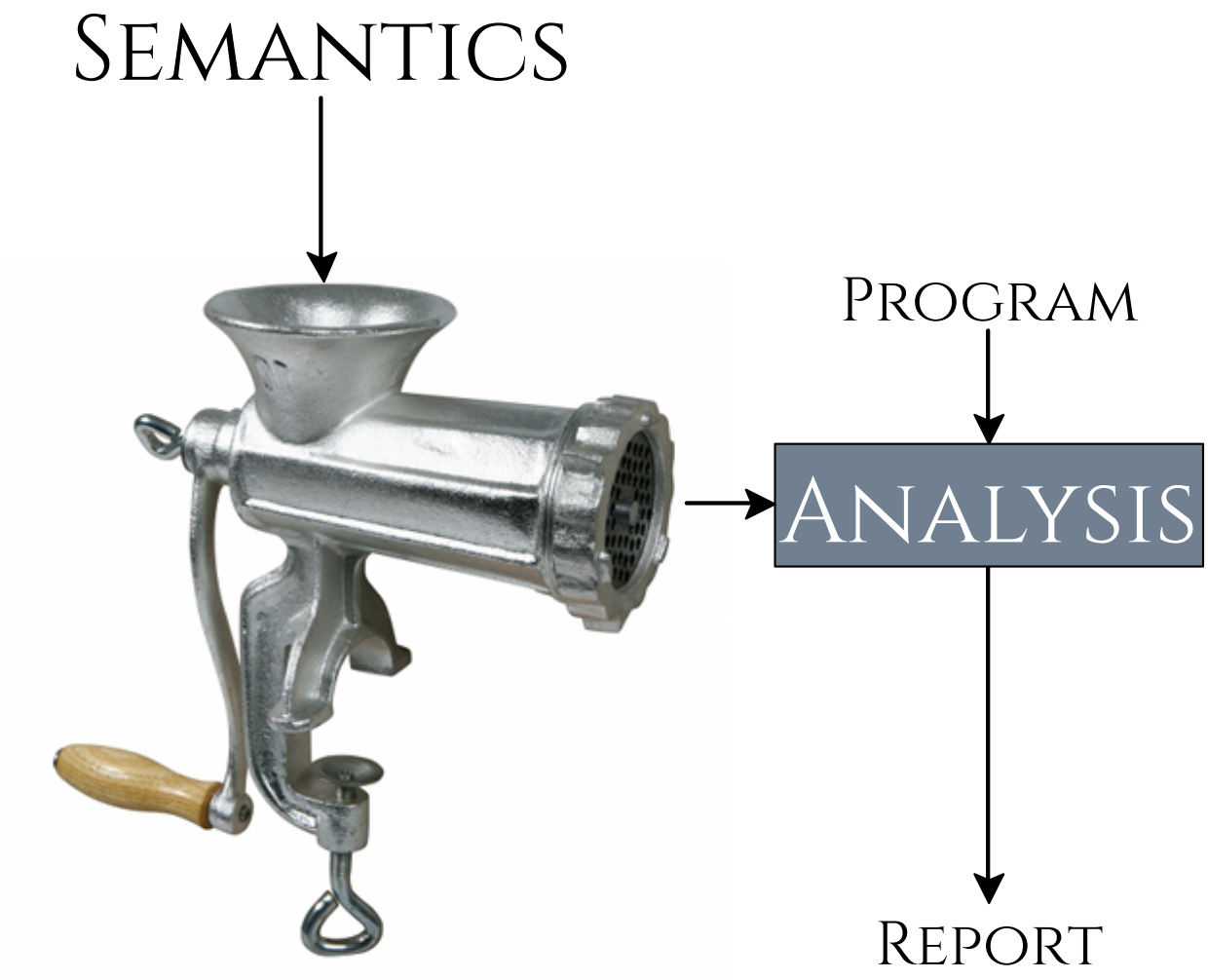

Semantics
Program
Analysis
Report

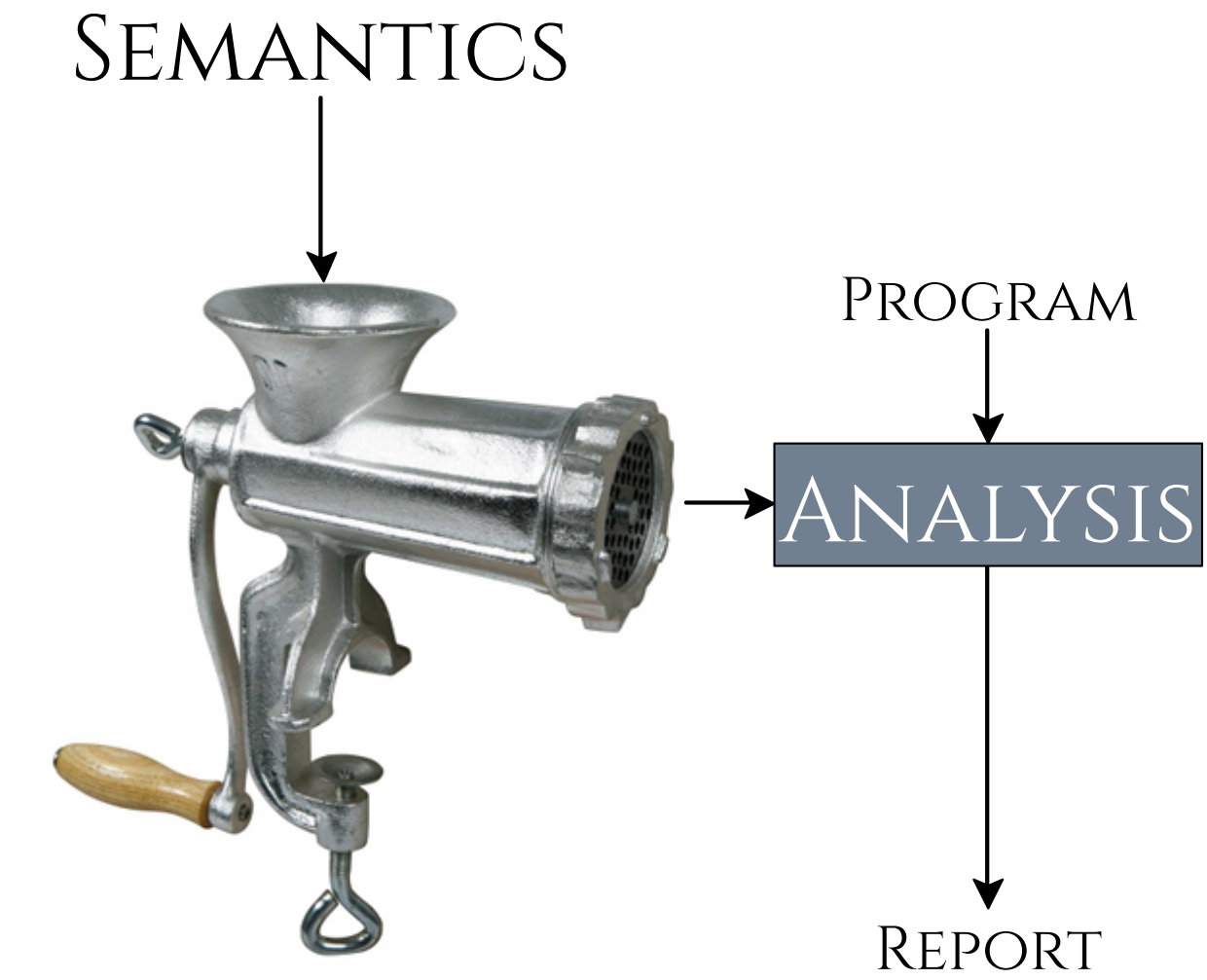

Semantics
Program
Analysis
Report

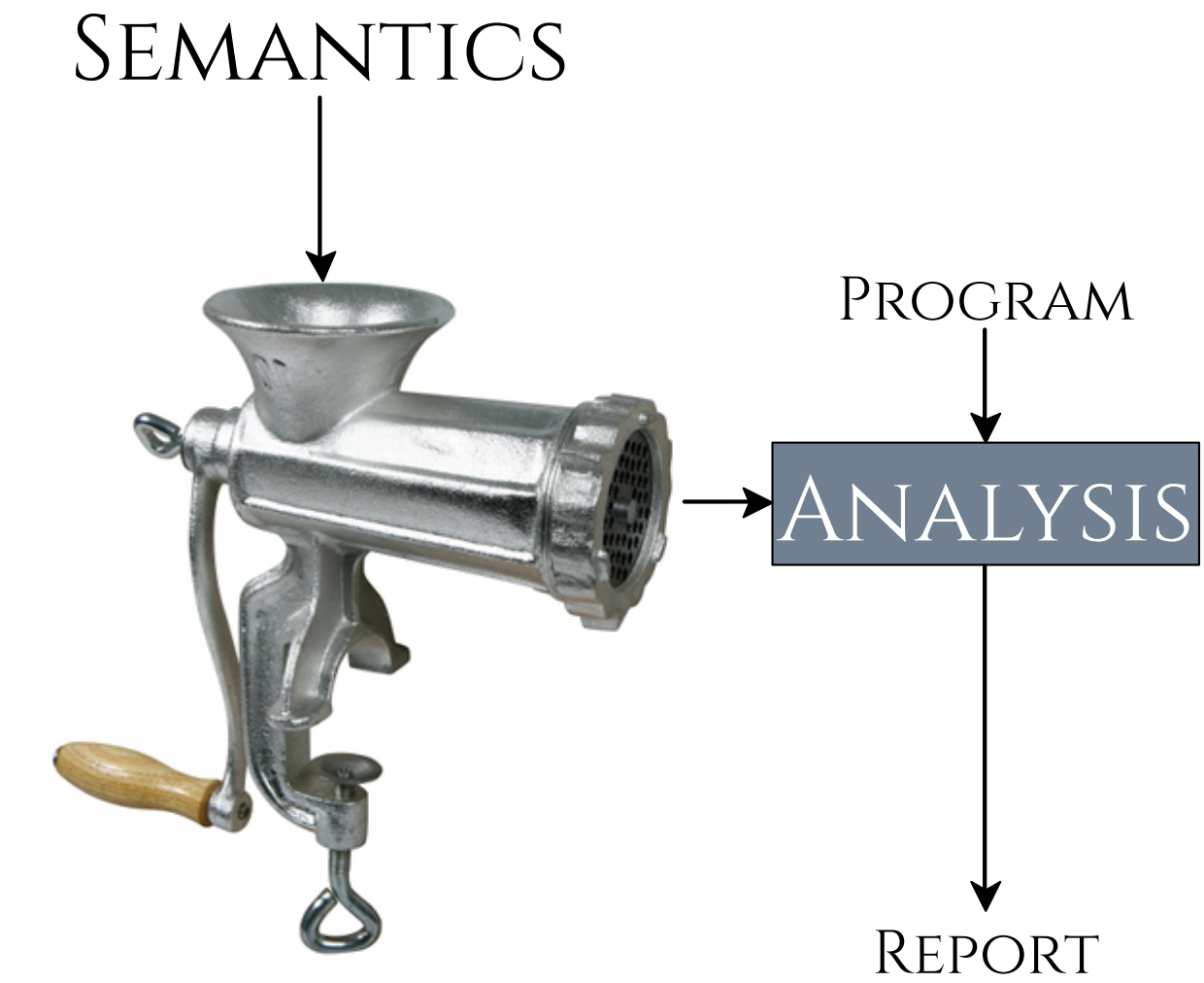

Semantics
Program
Analysis
Report

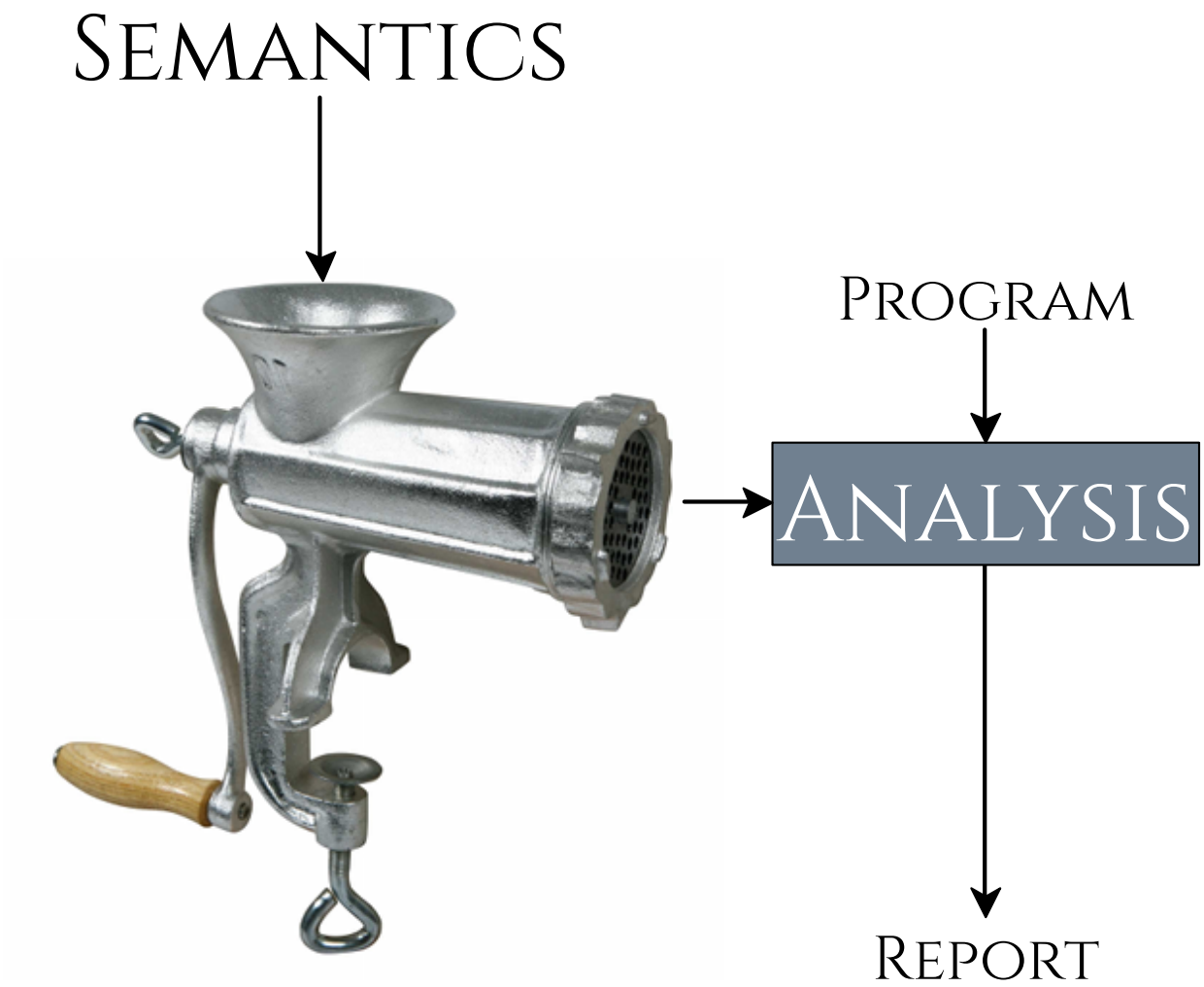

Semantics
Program
Analysis
Report

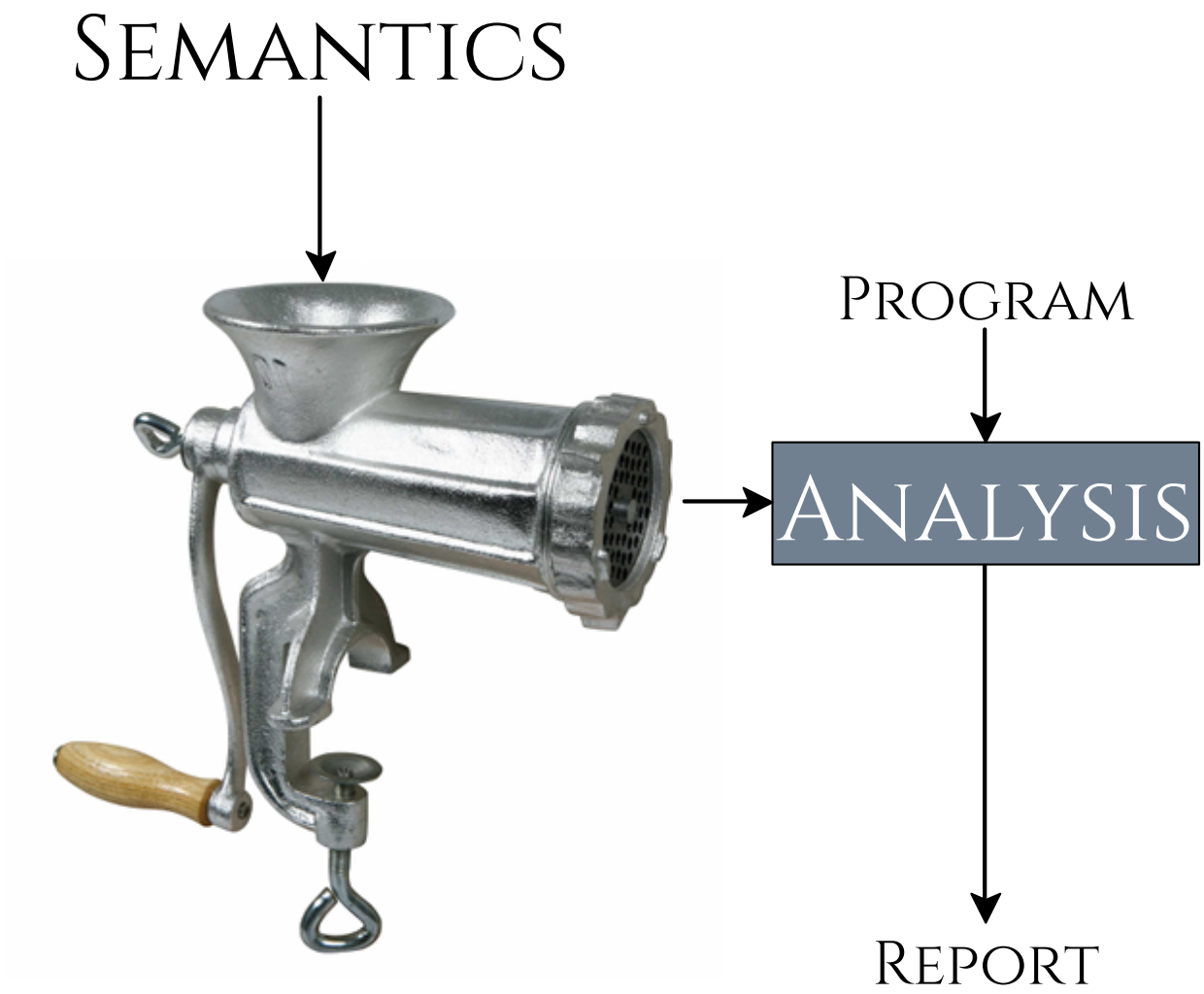

Semantics
Program
Analysis
Report

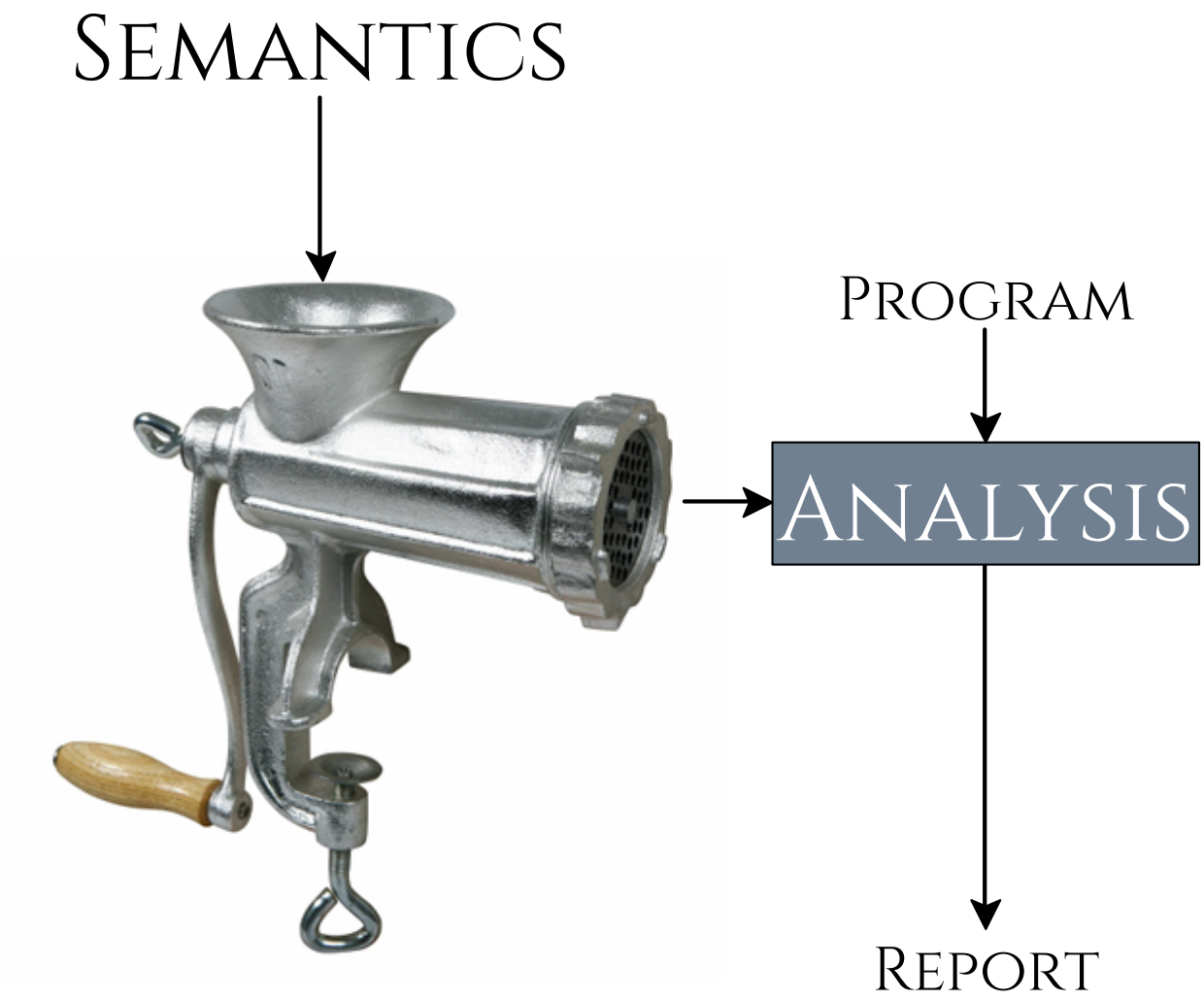

Semantics
Program
Analysis
Report

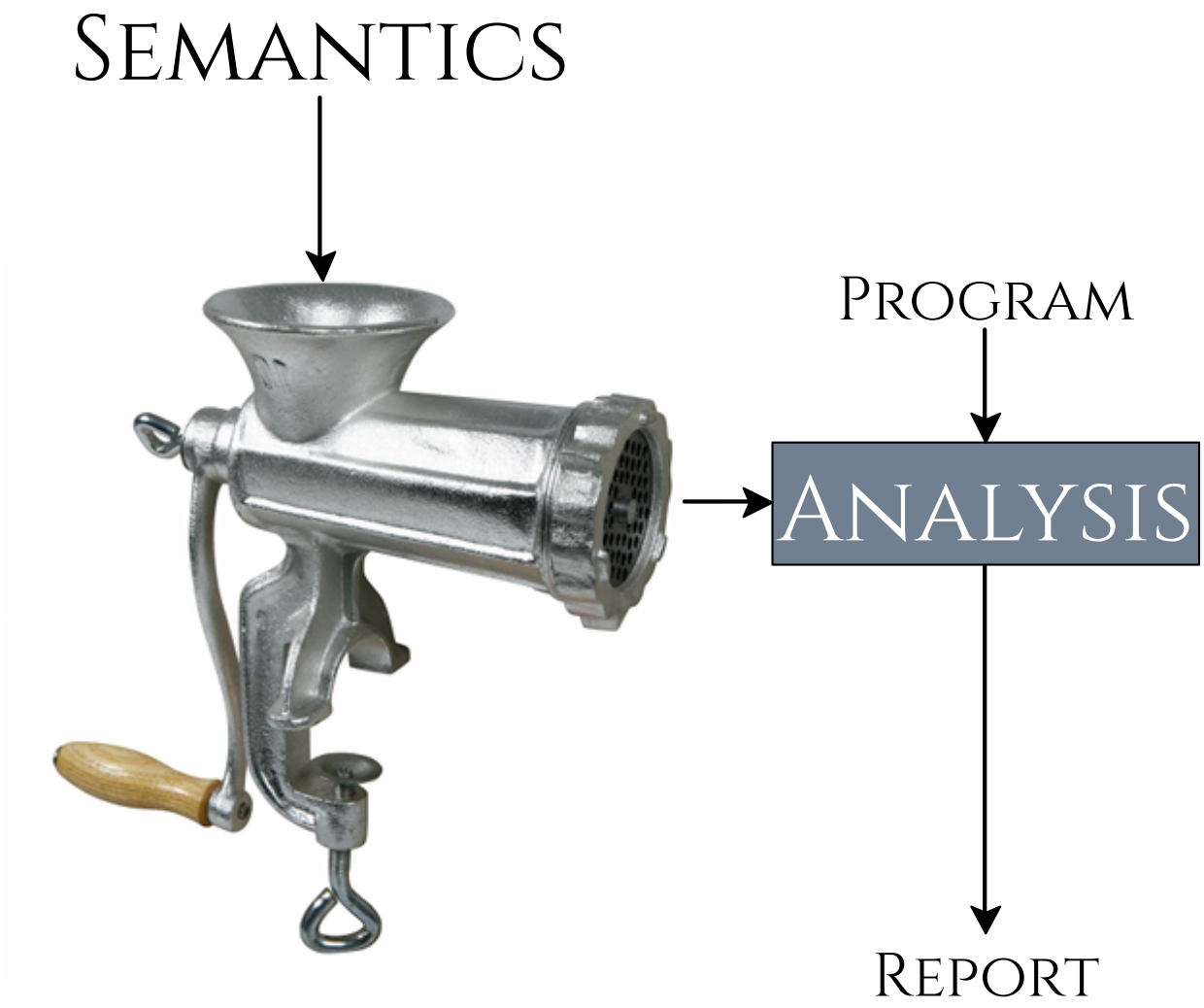

Semantics
Program
Analysis
Report

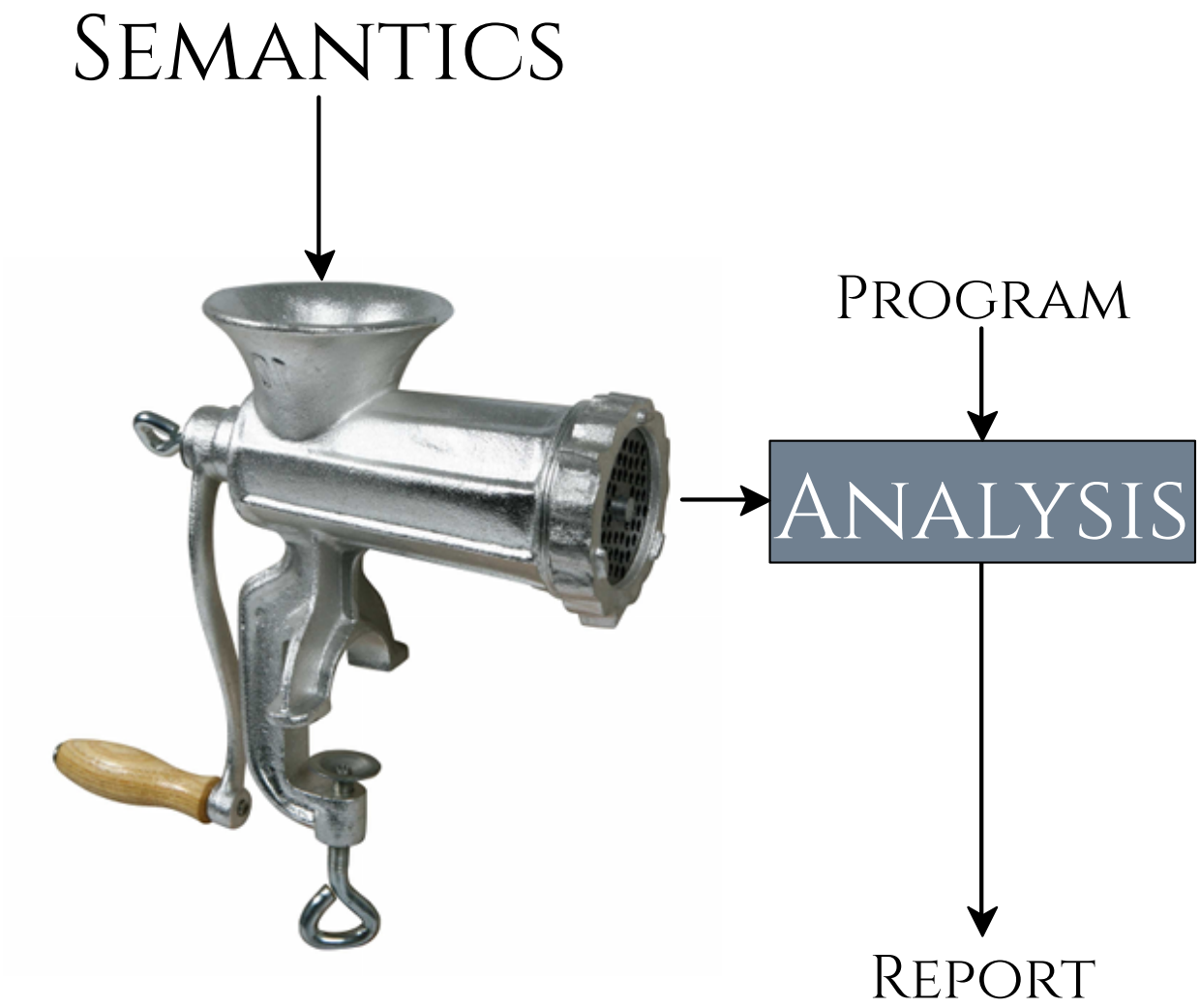

Semantics
Program
Analysis
Report

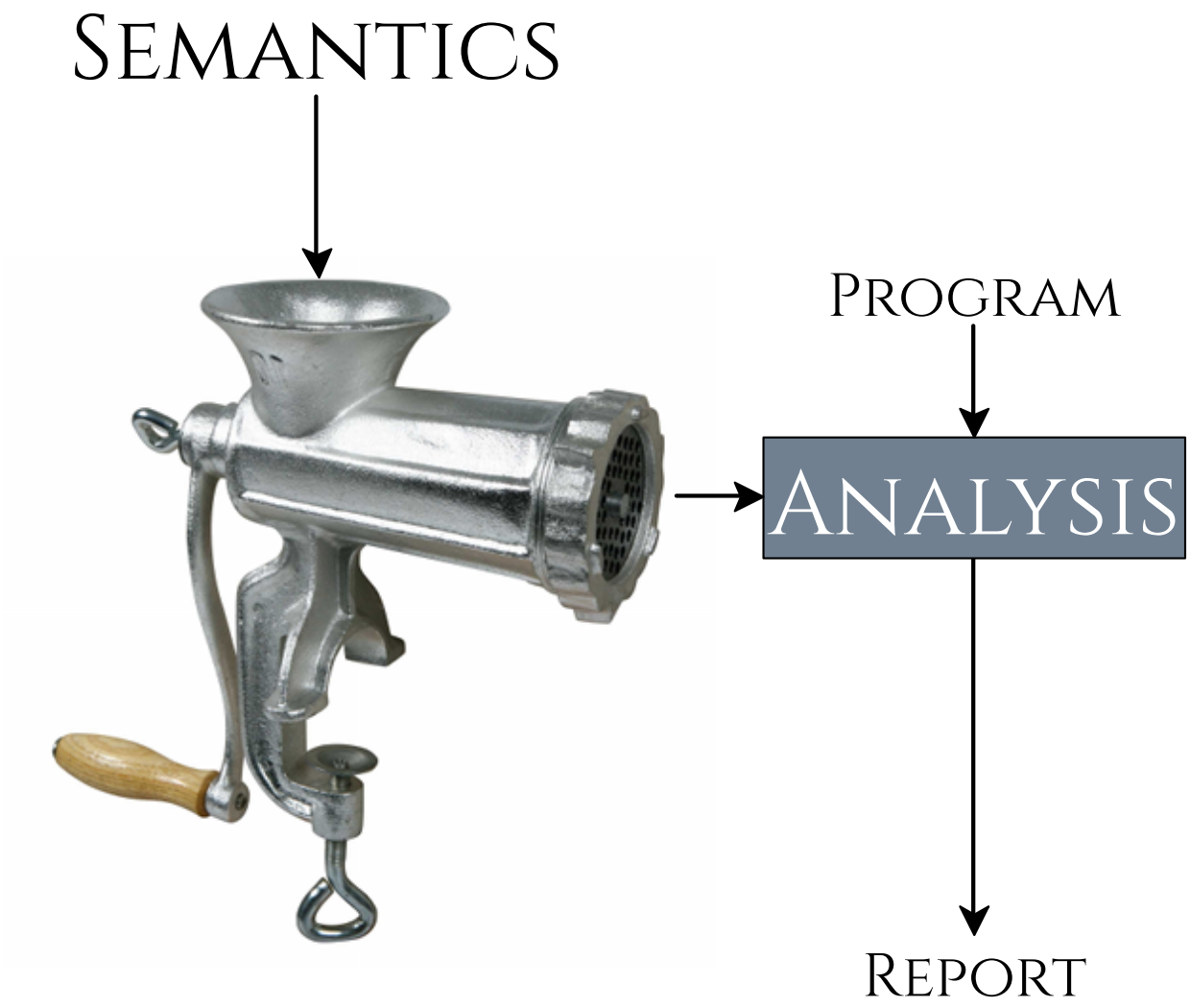

Semantics
Program
Analysis
Report

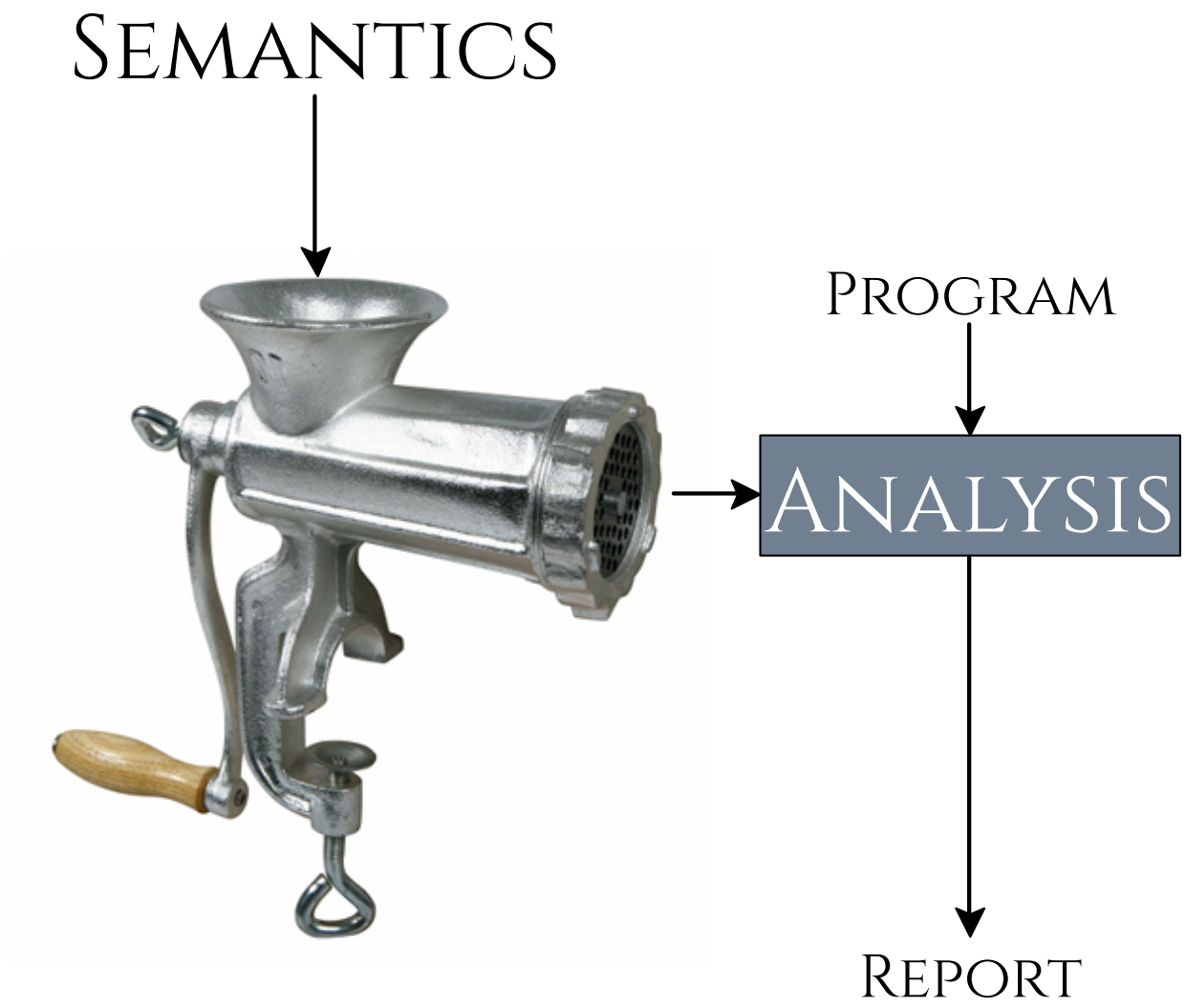

Semantics
Program
Analysis
Report

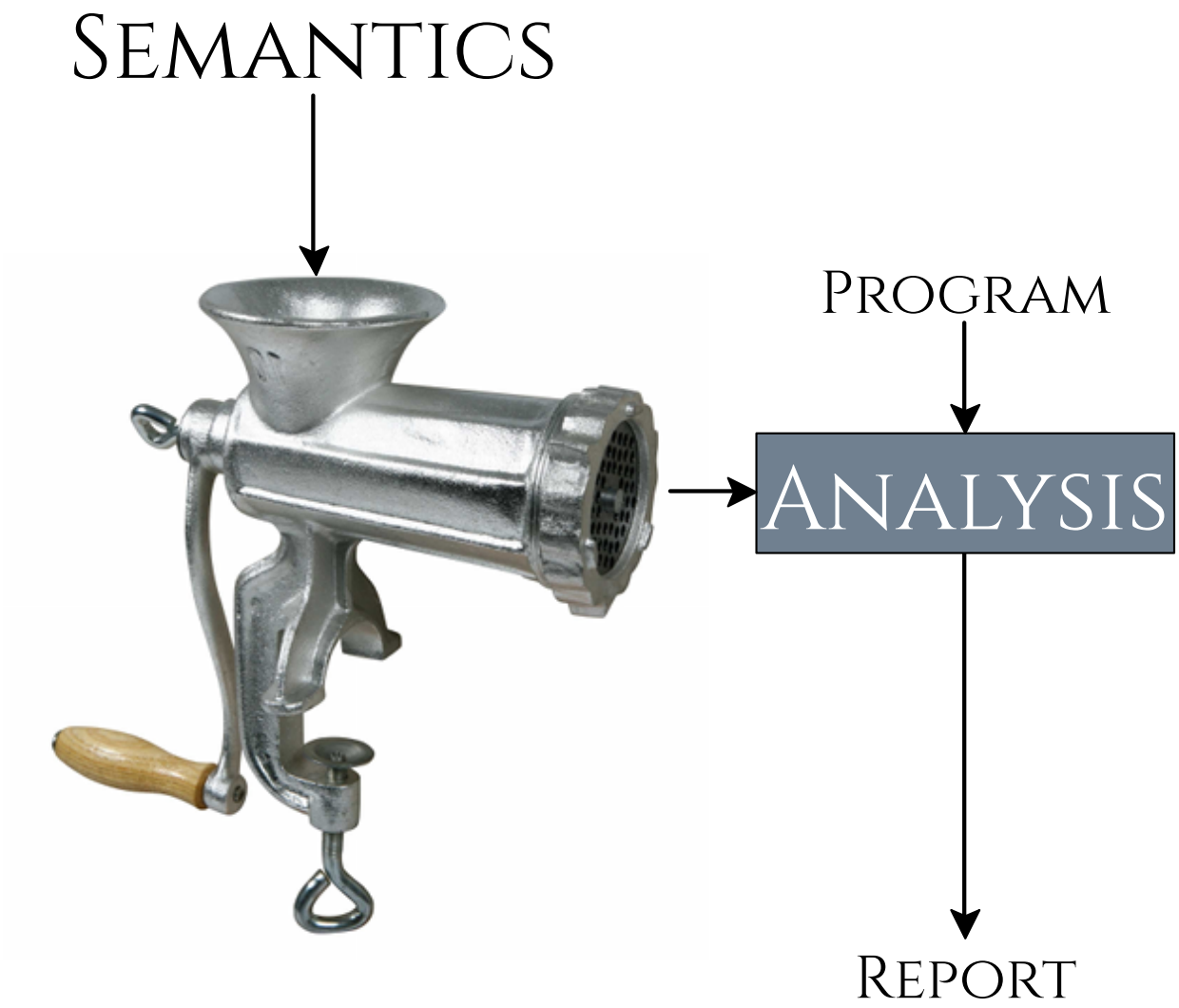

Semantics
Program
Analysis
Report

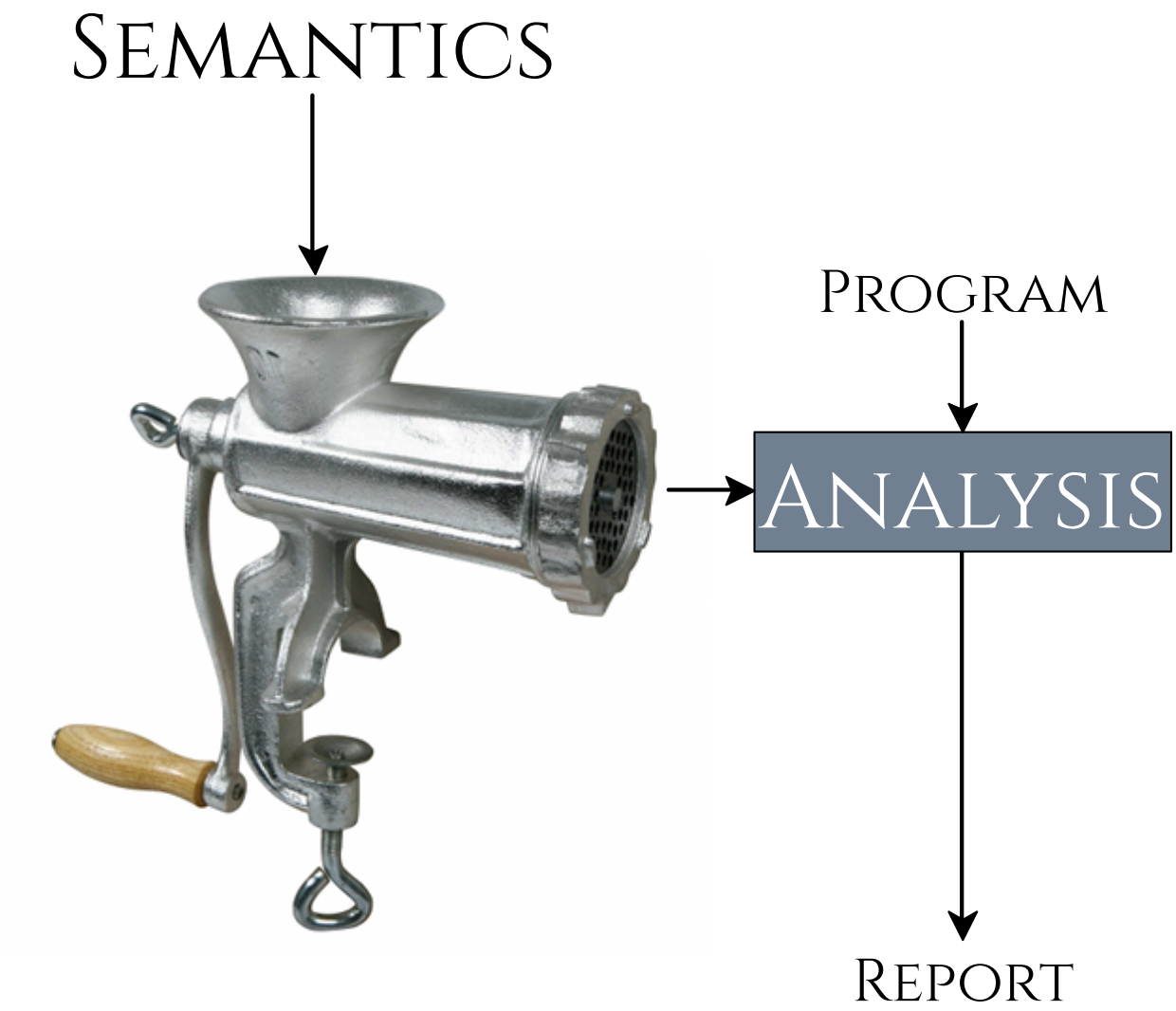

Semantics
Program
Analysis
Report

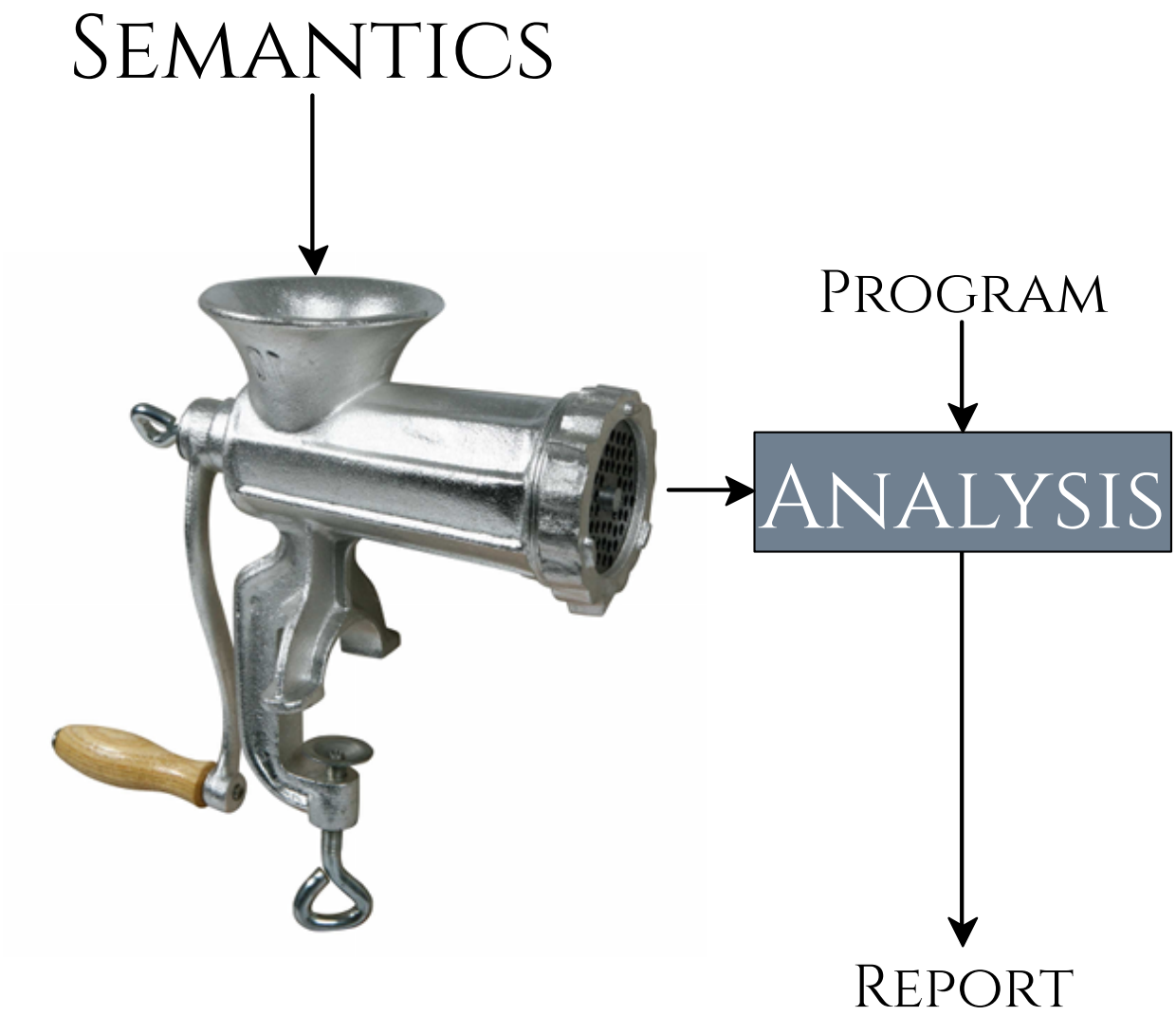

Semantics
Program
Analysis
Report

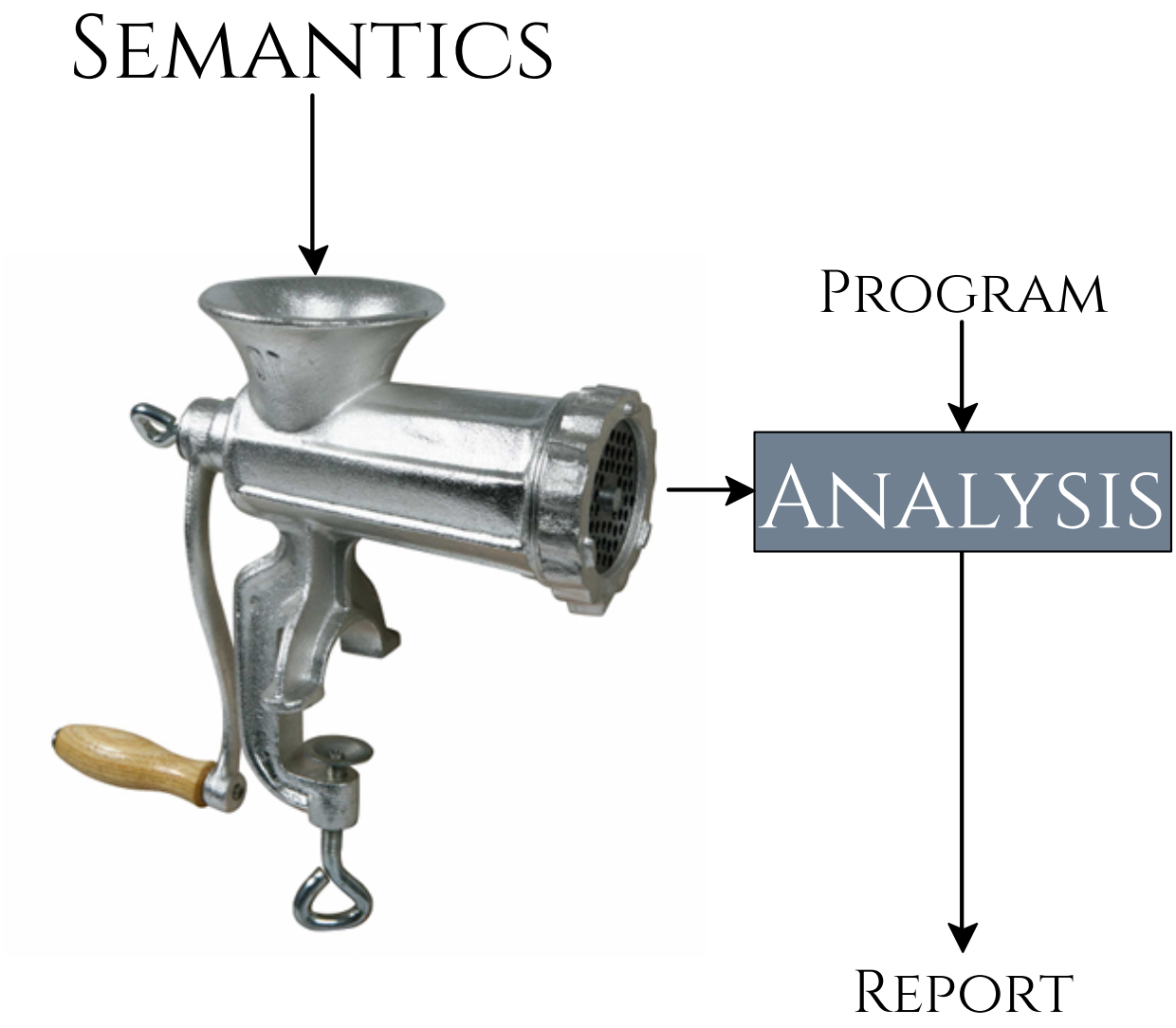

Semantics
Program
Analysis
Report

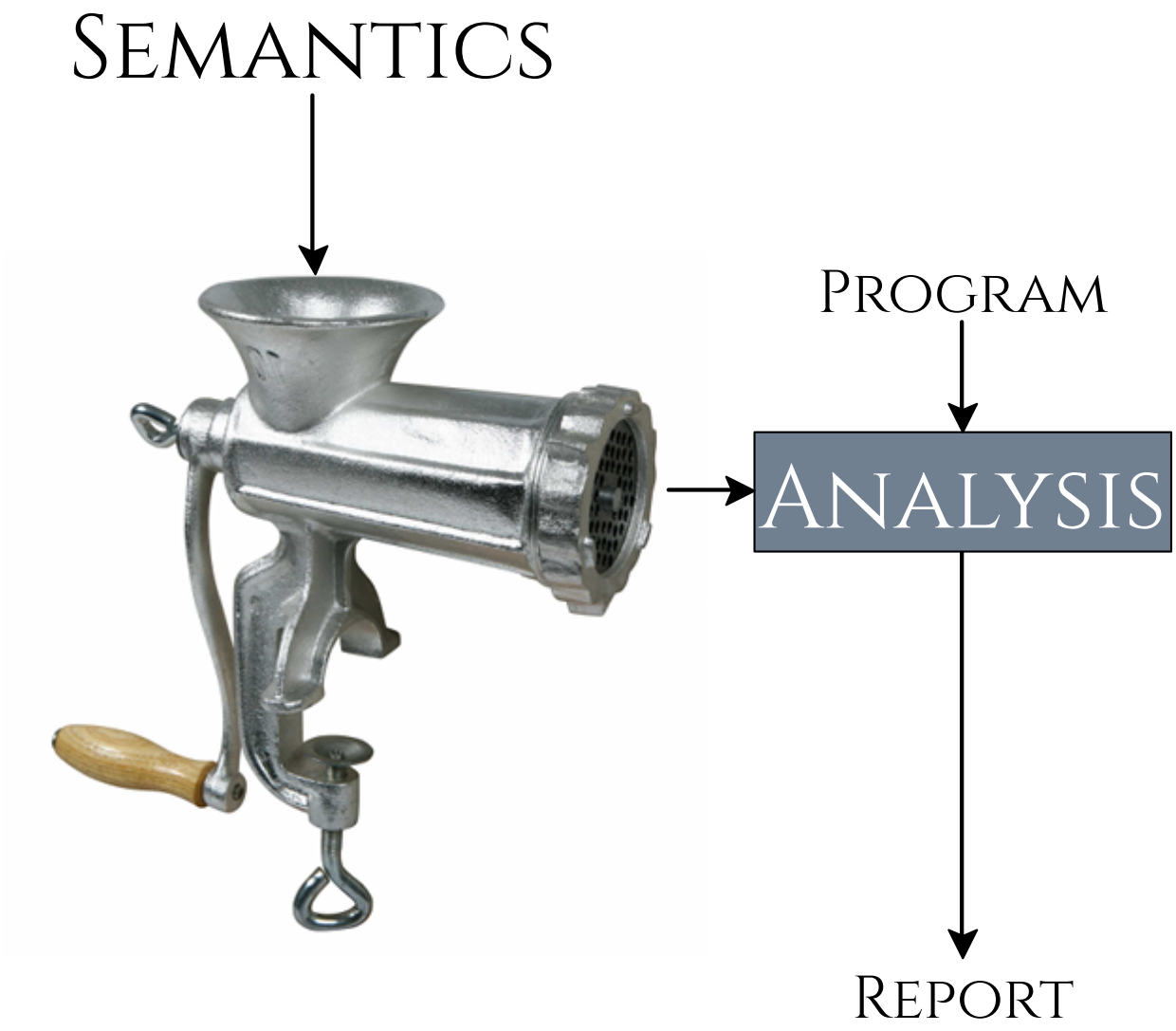

Semantics
Program
Analysis
Report

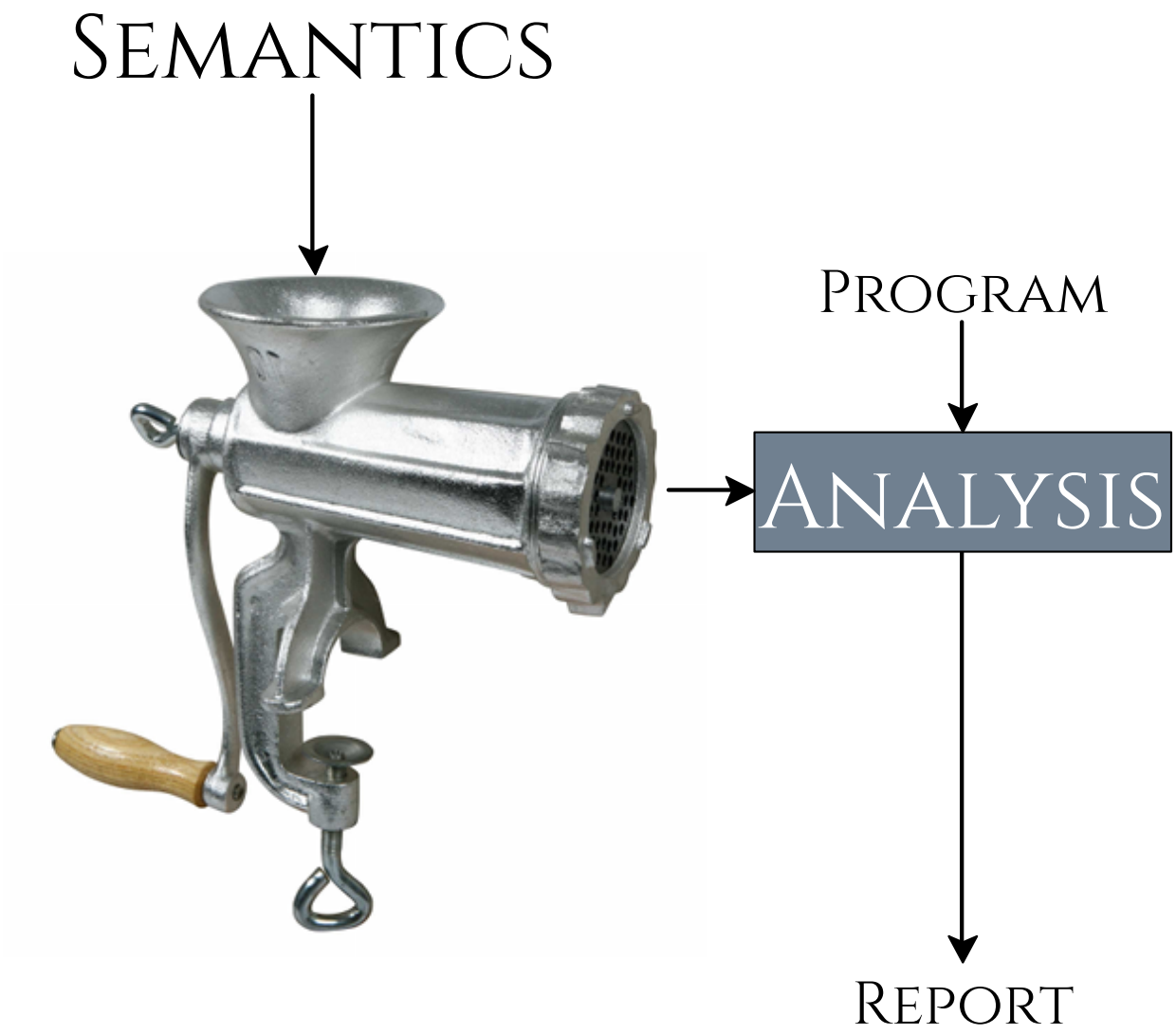

Semantics
Program
Analysis
Report

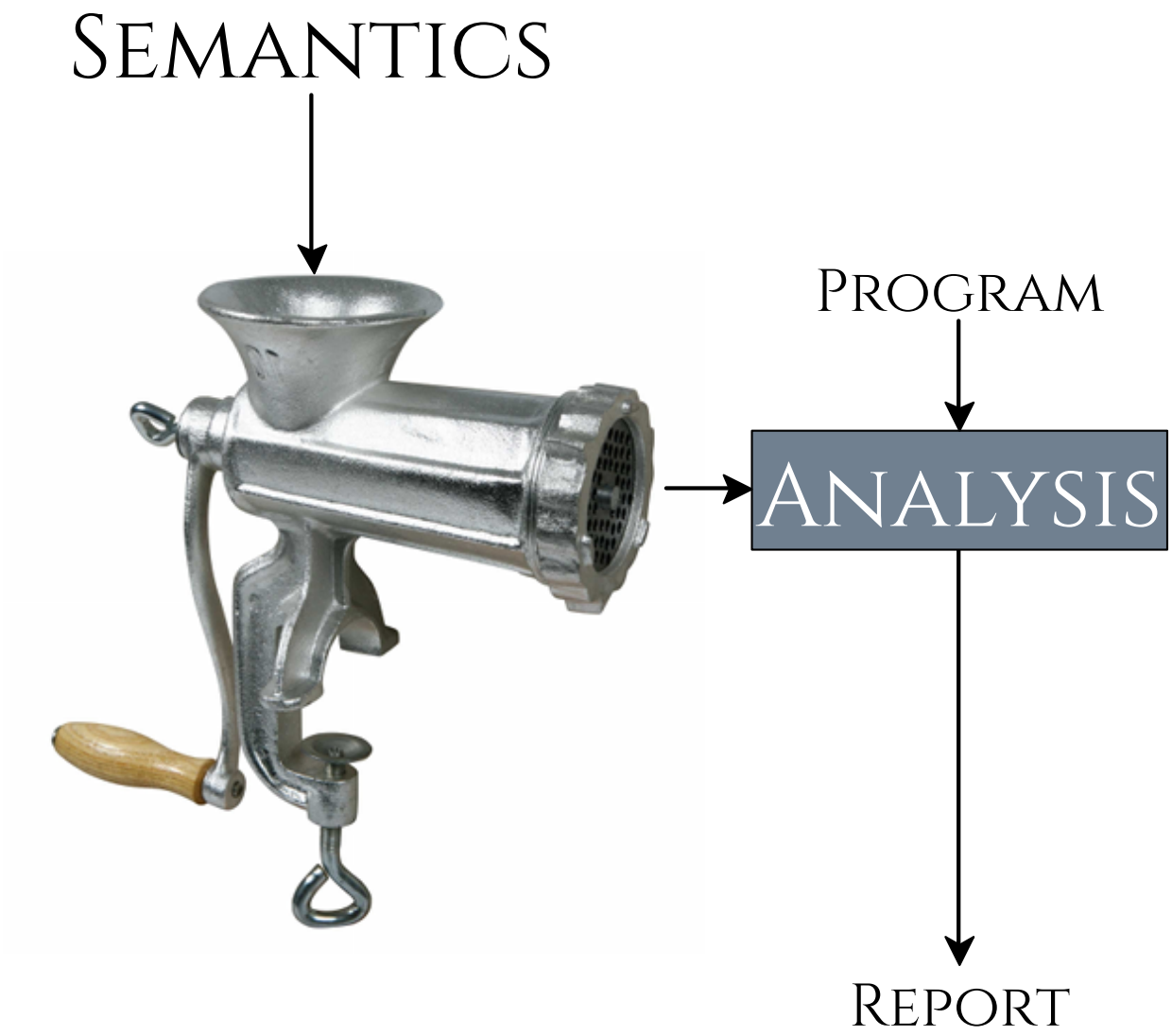

Semantics
Program
Analysis
Report

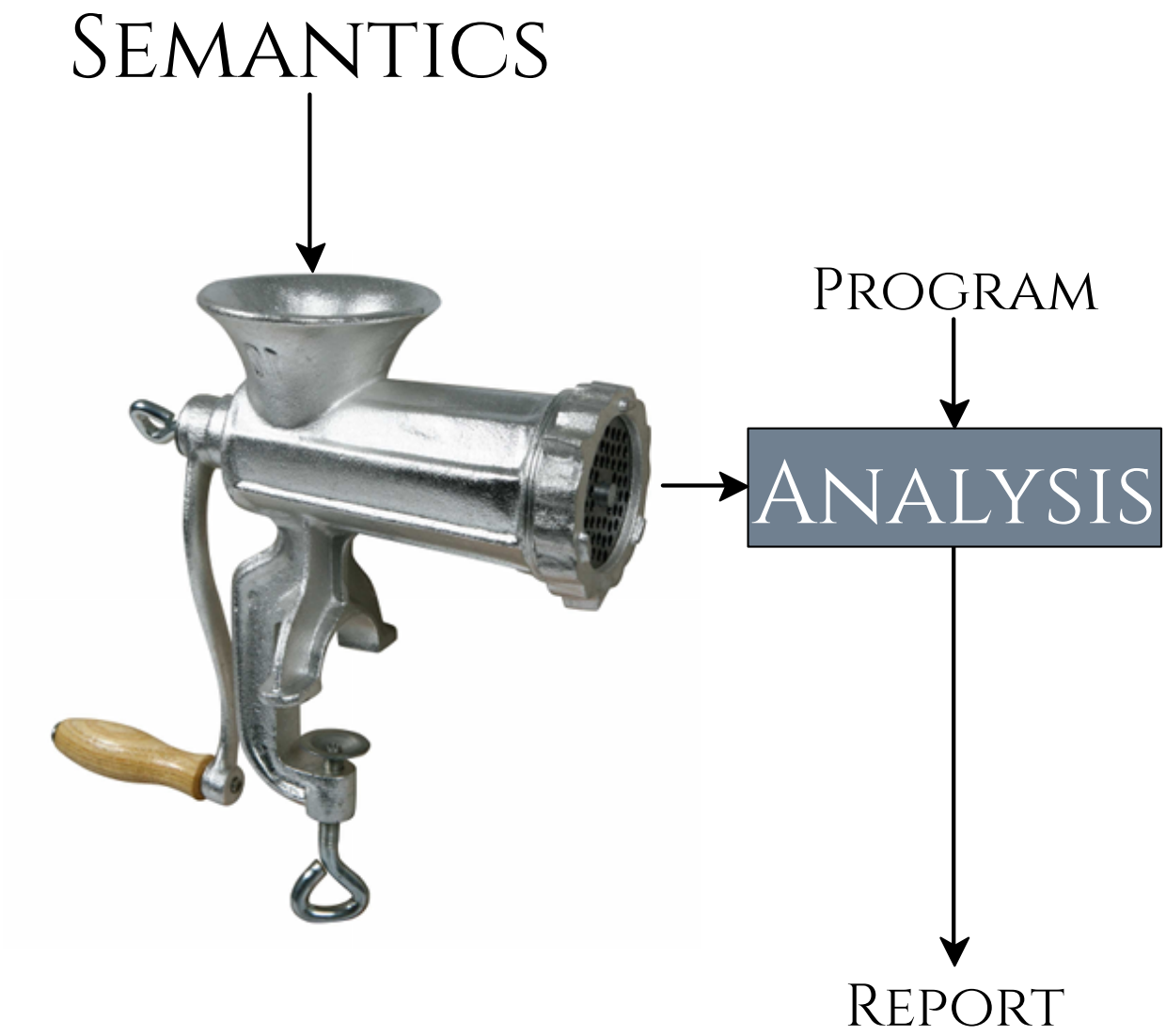

Semantics
Program
Analysis
Report

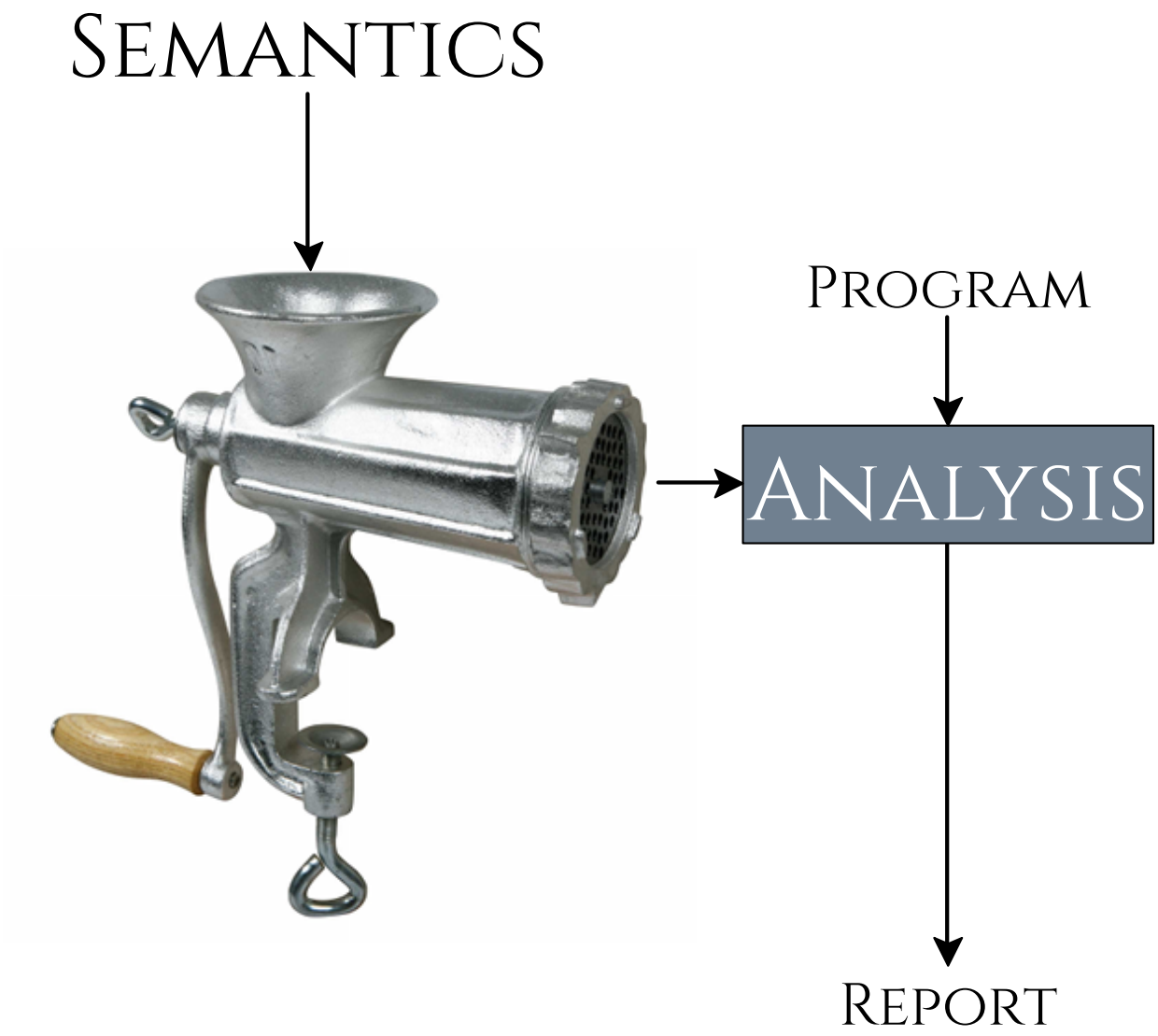

Semantics
Program
Analysis
Report

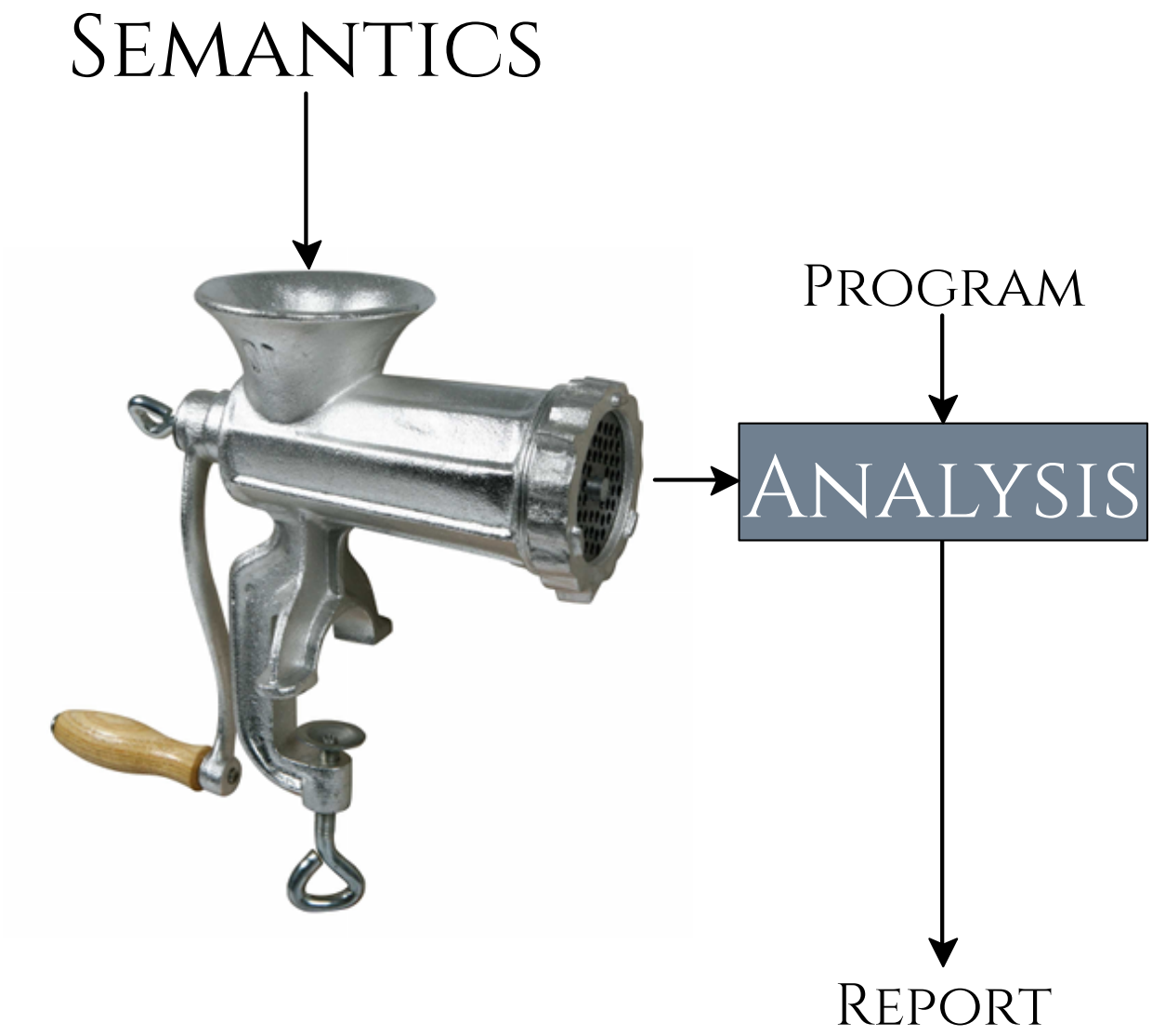

Semantics
Program
Analysis
Report

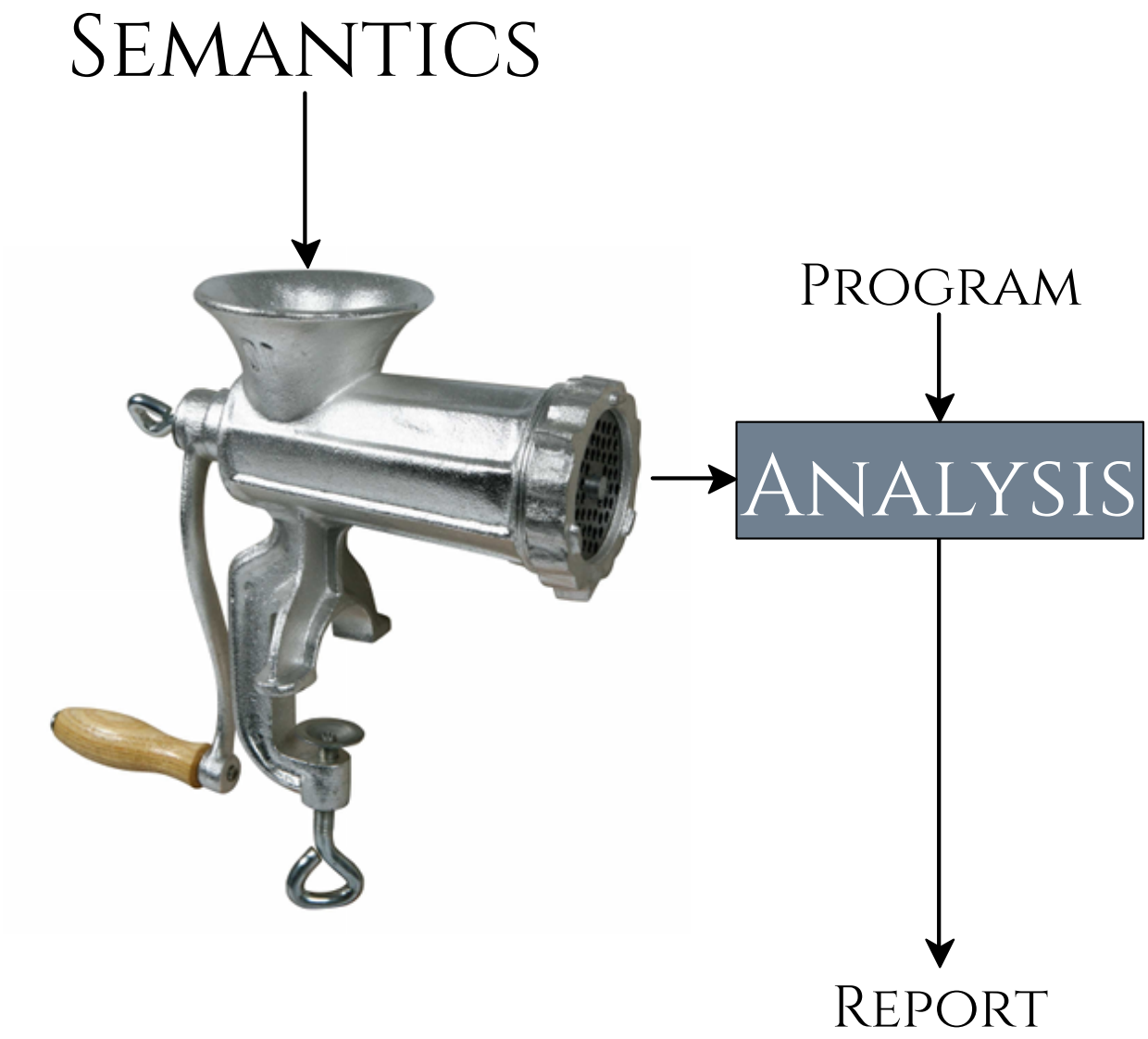

Semantics
Program
Analysis
Report

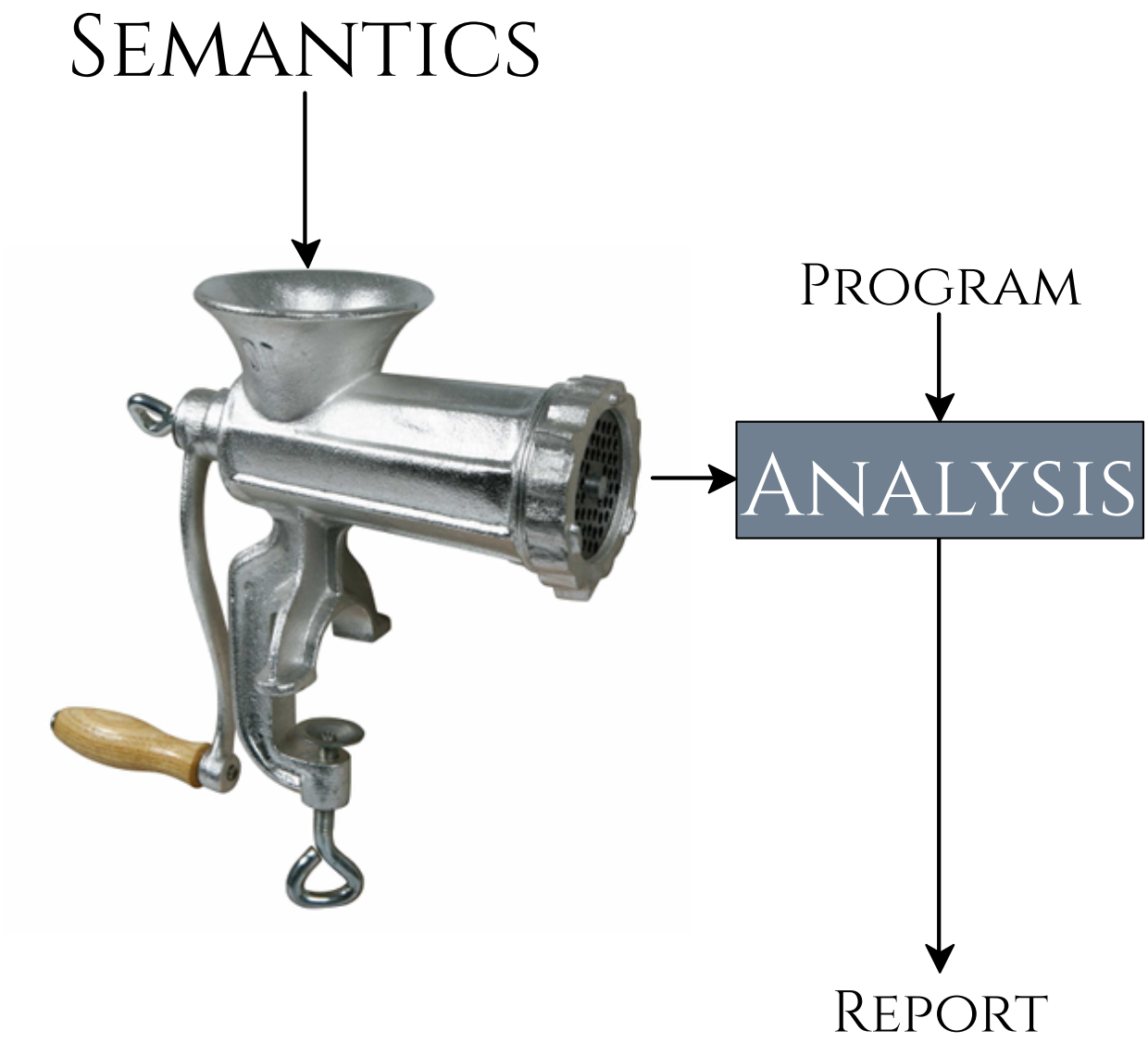

Semantics
Program
Analysis
Report

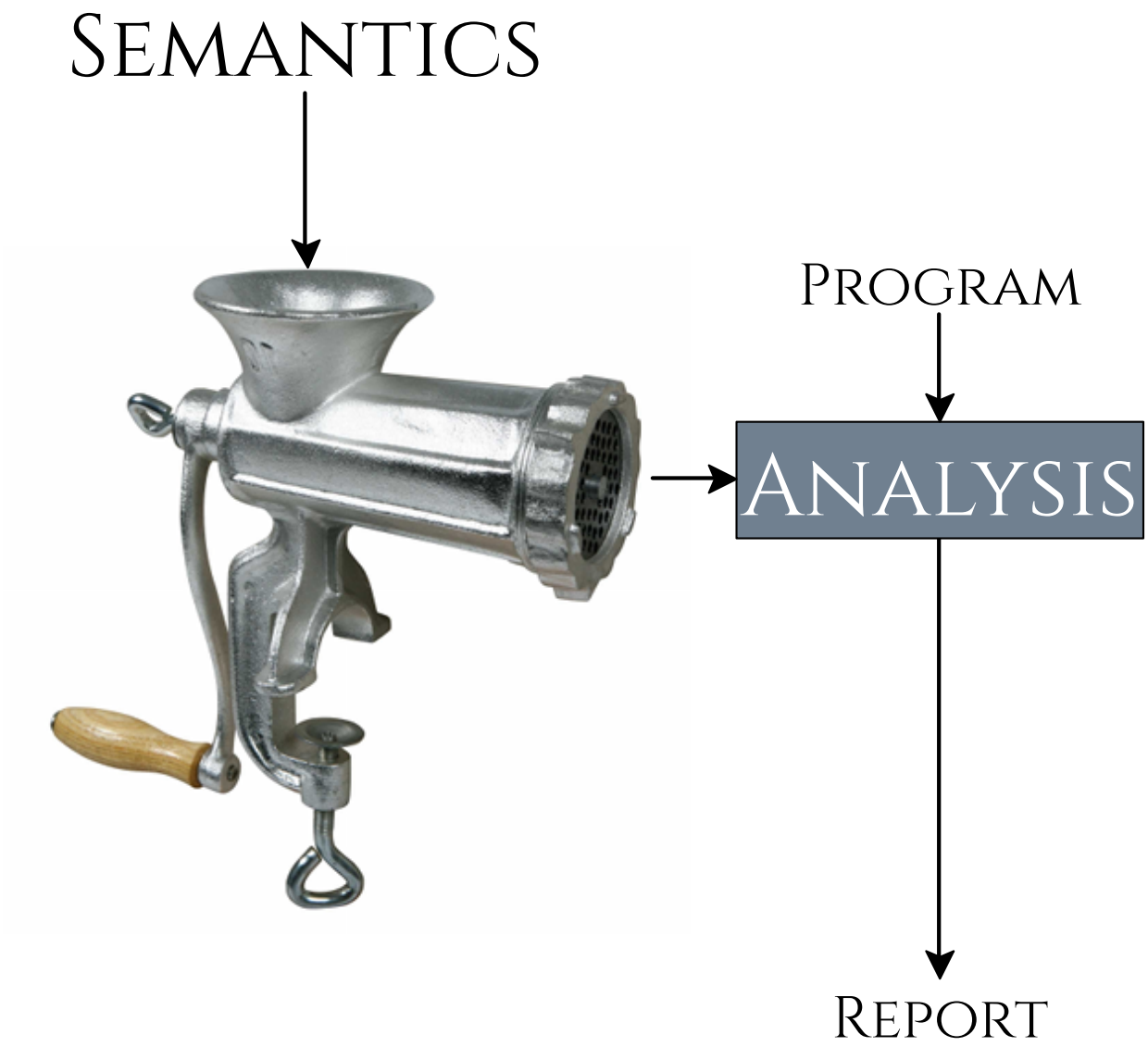

Semantics
Program
Analysis
Report

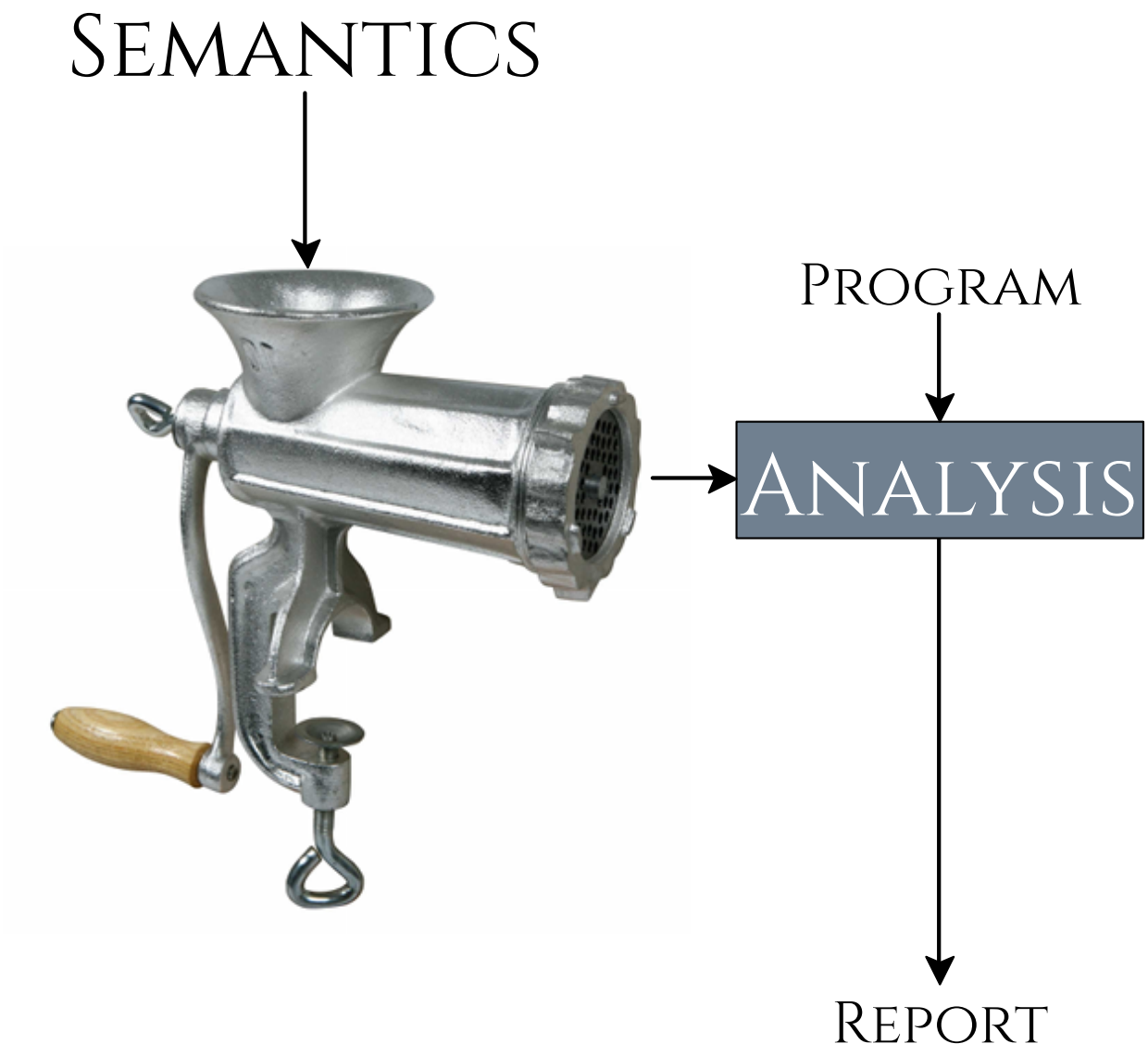

Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report

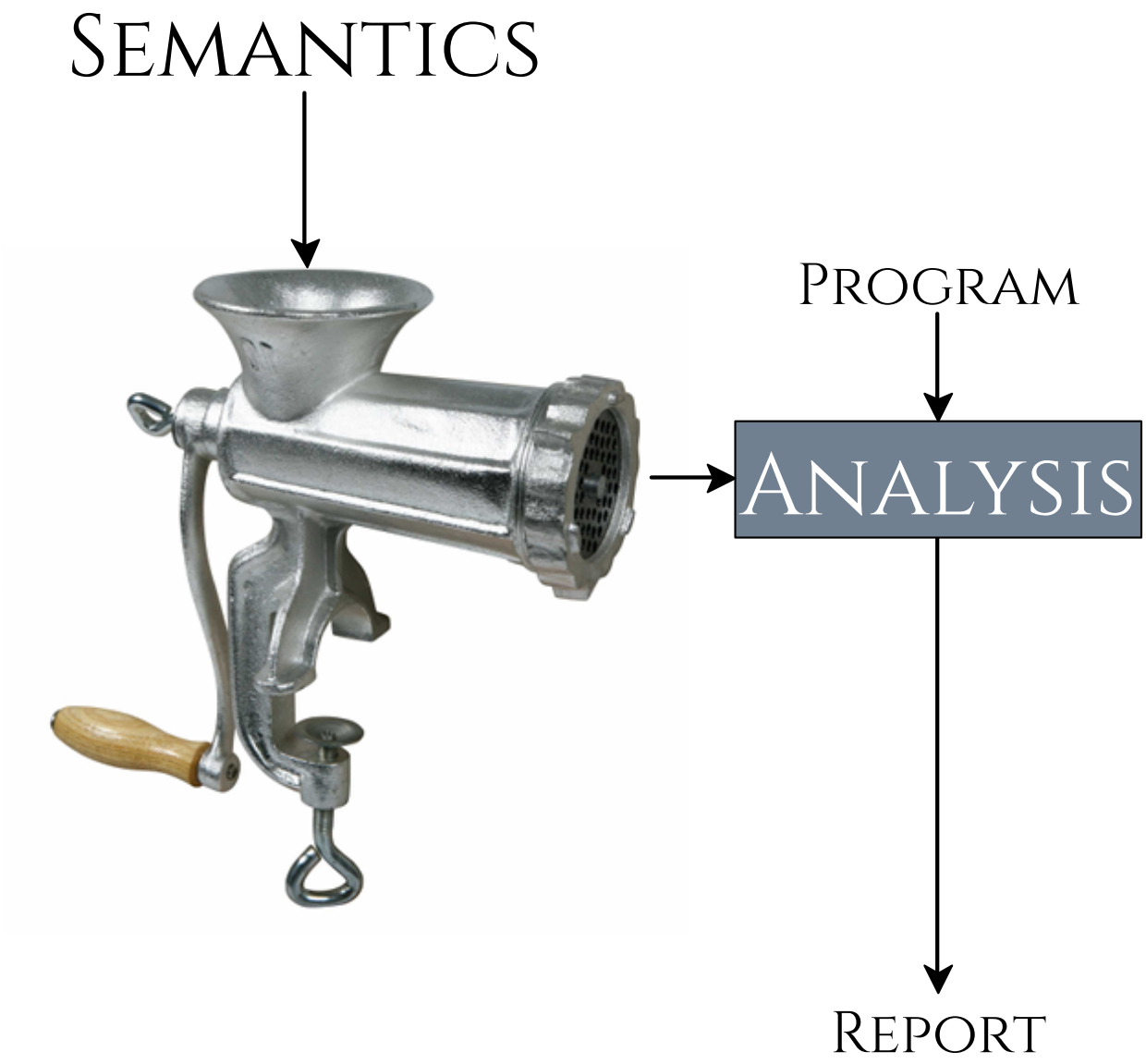

Semantics
Program
Analysis
Report

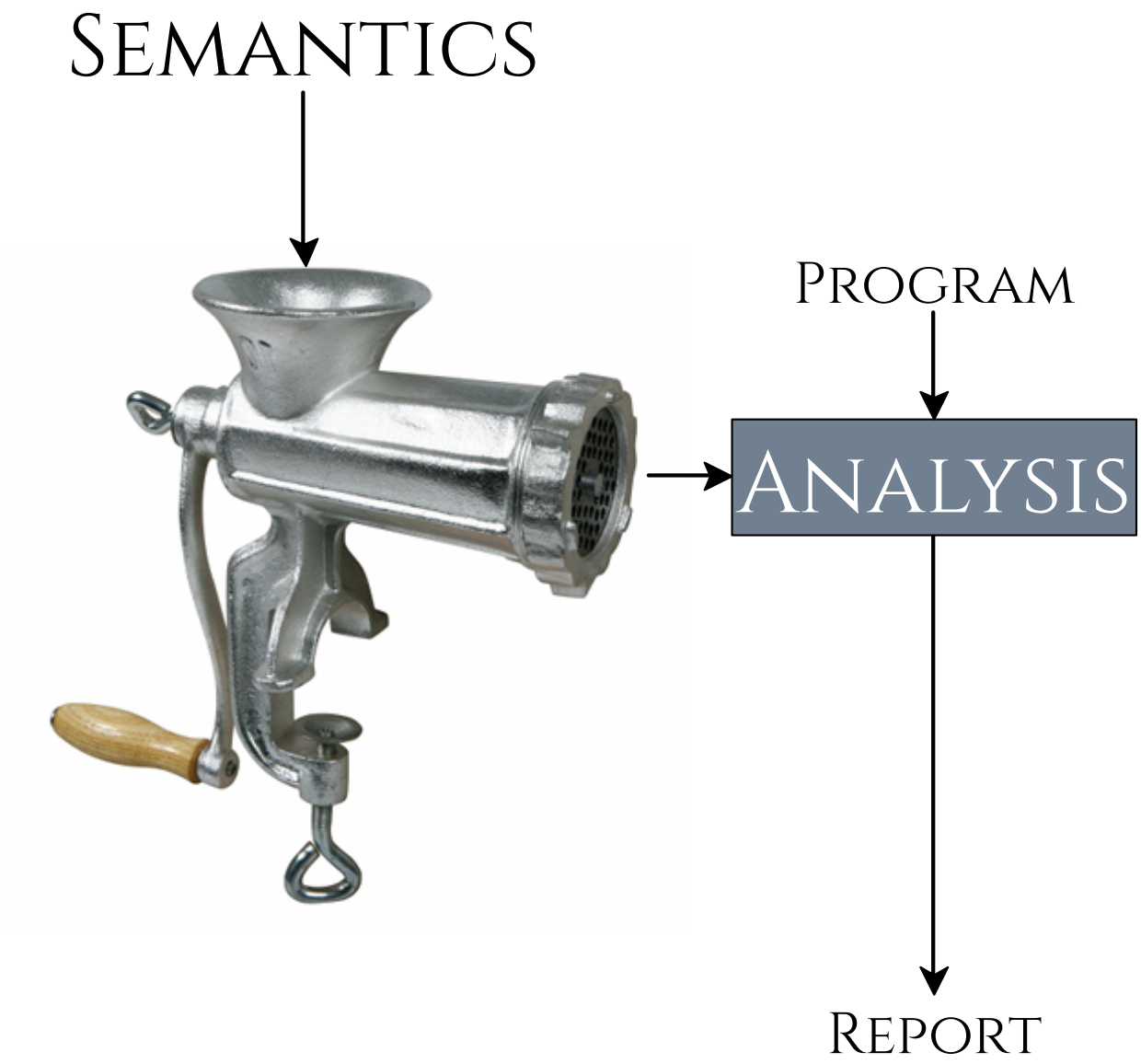
Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

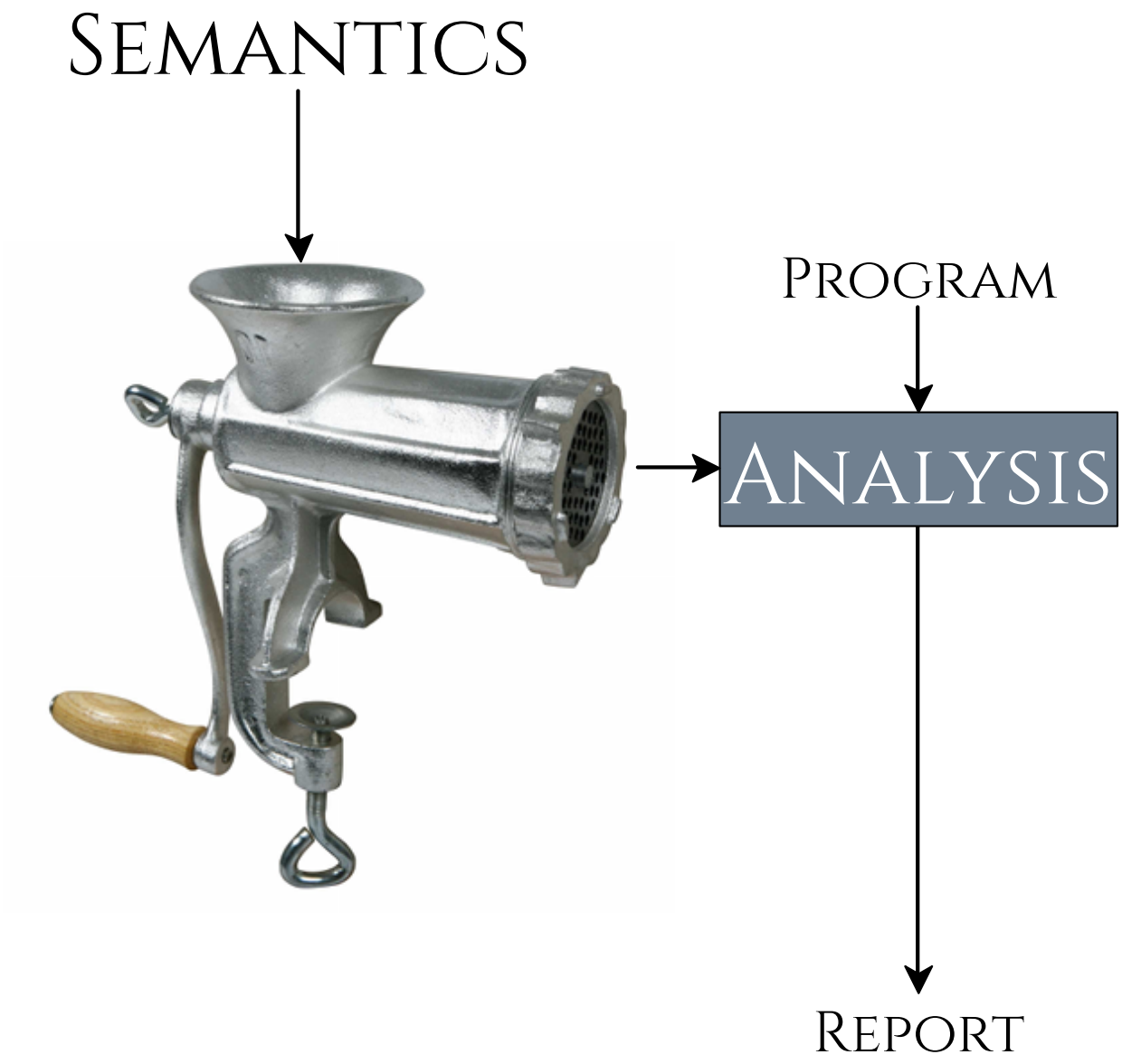
Report

Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

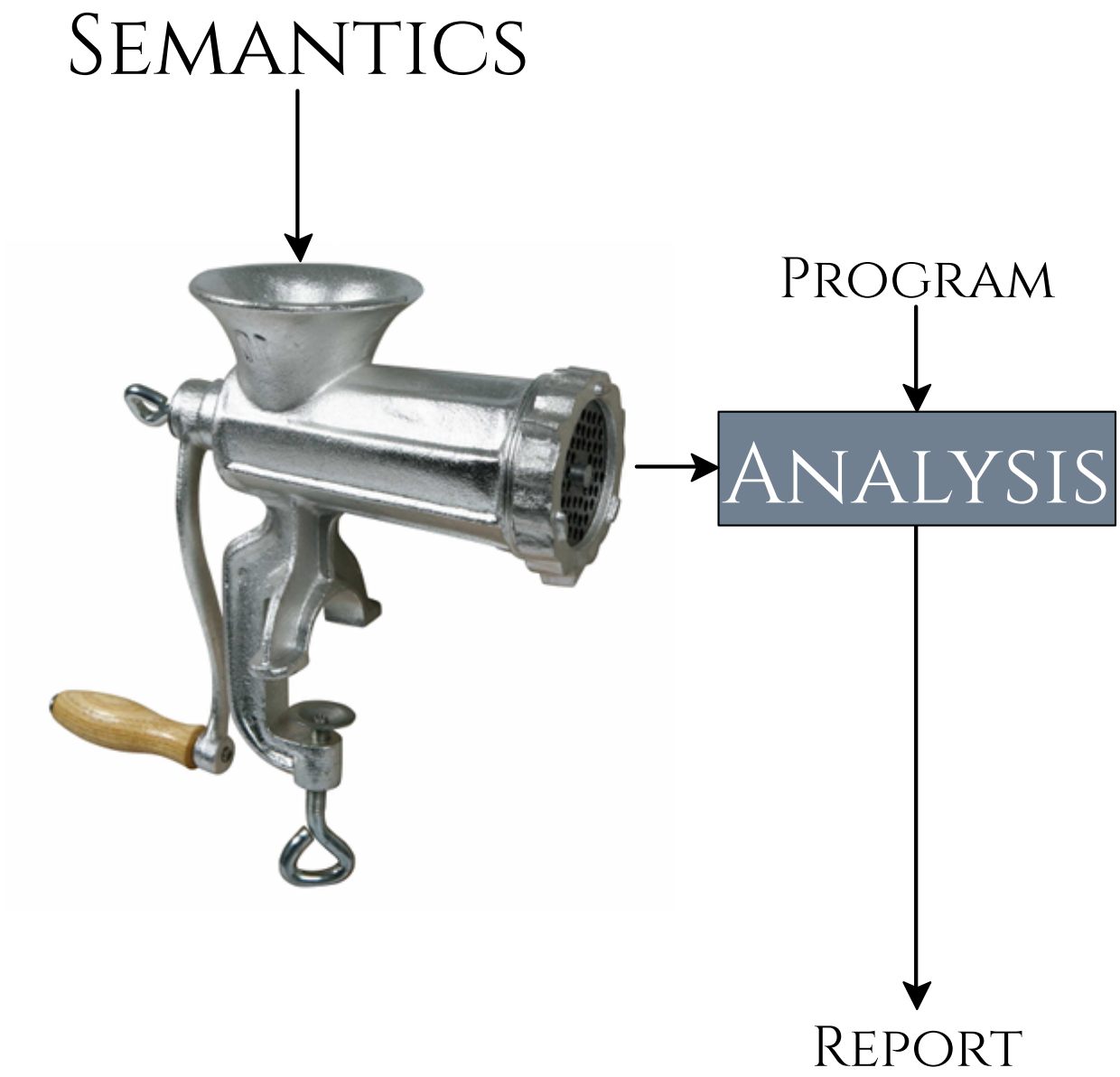
Report

Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

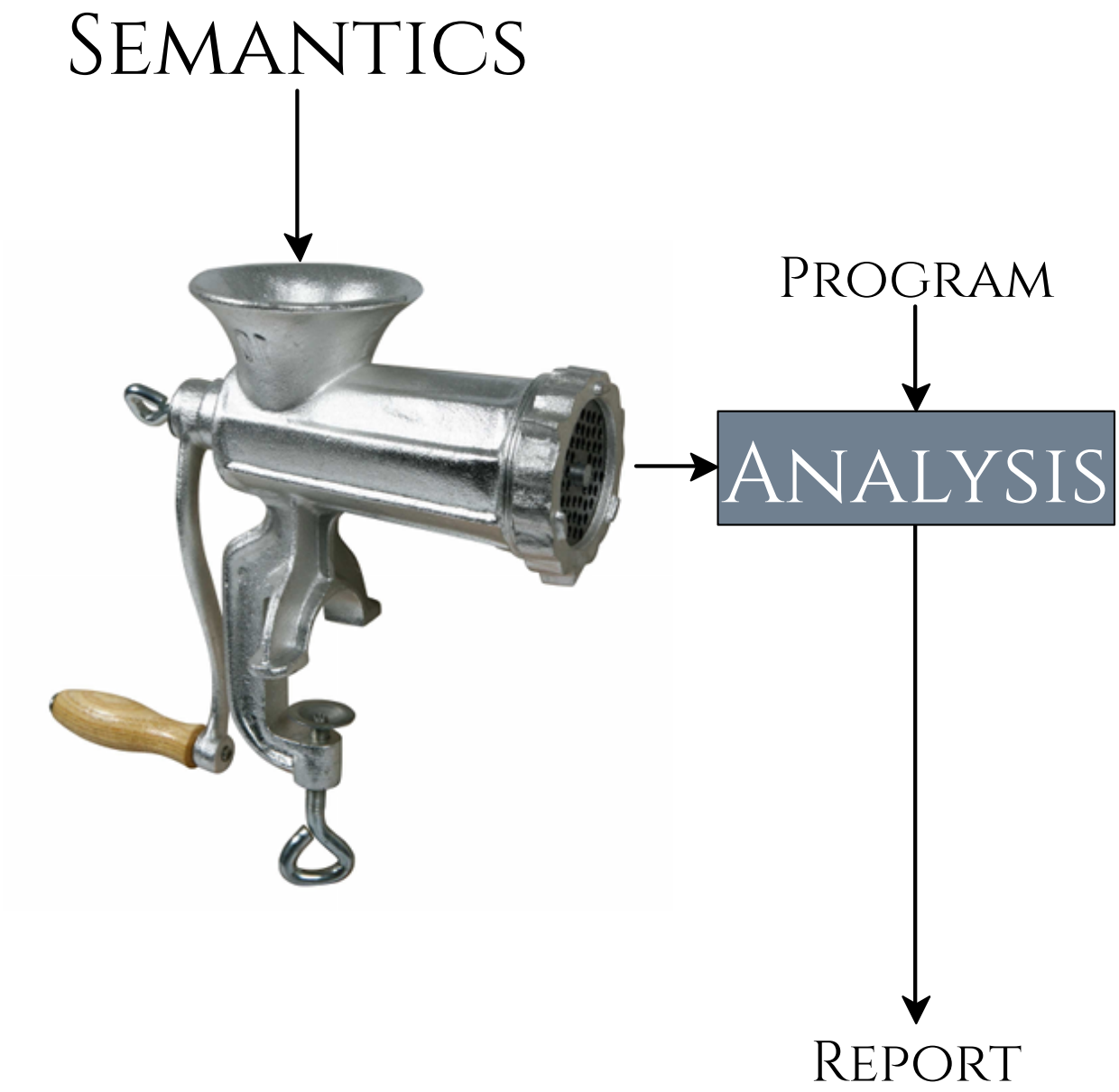
Report

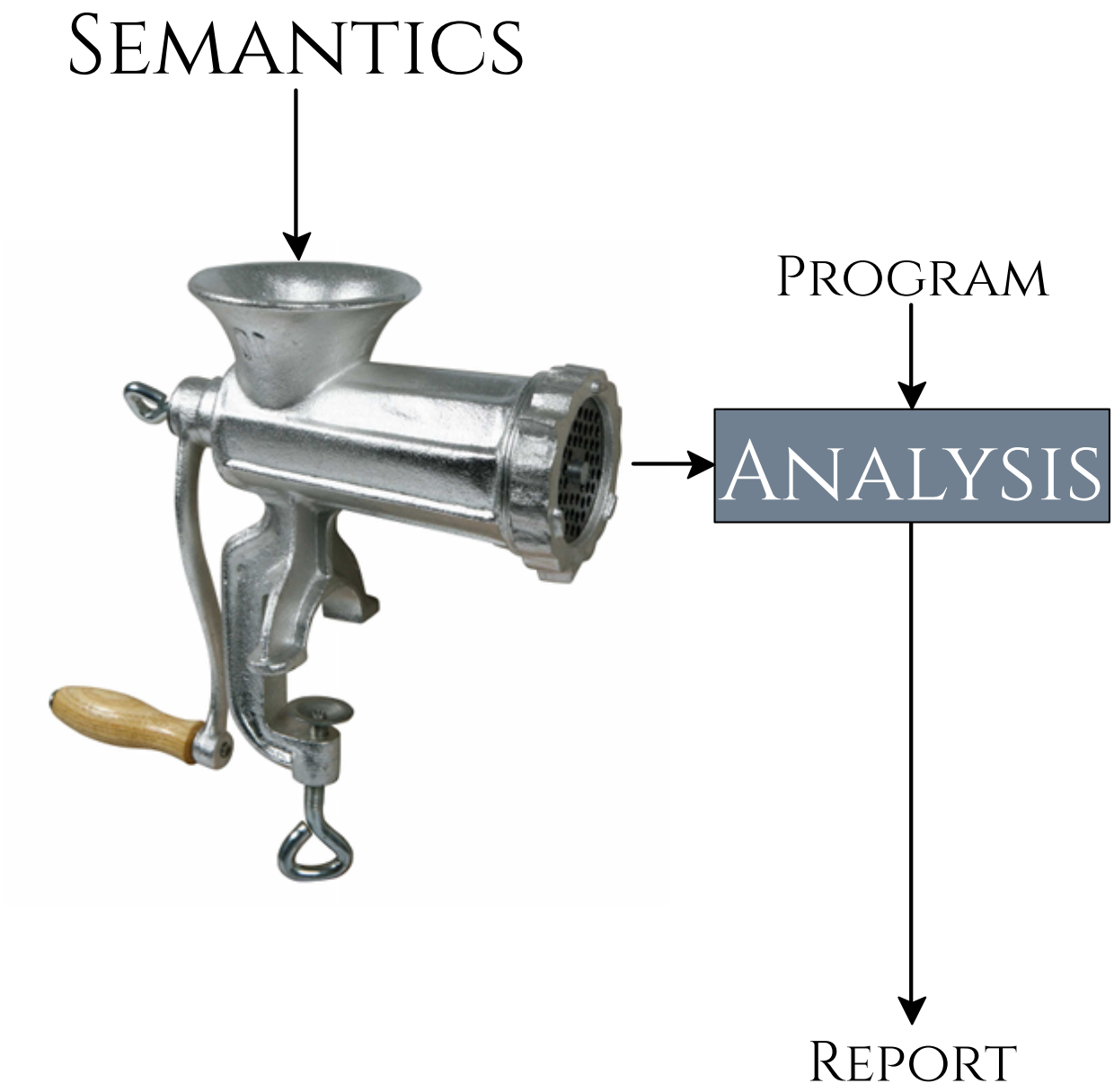
Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report

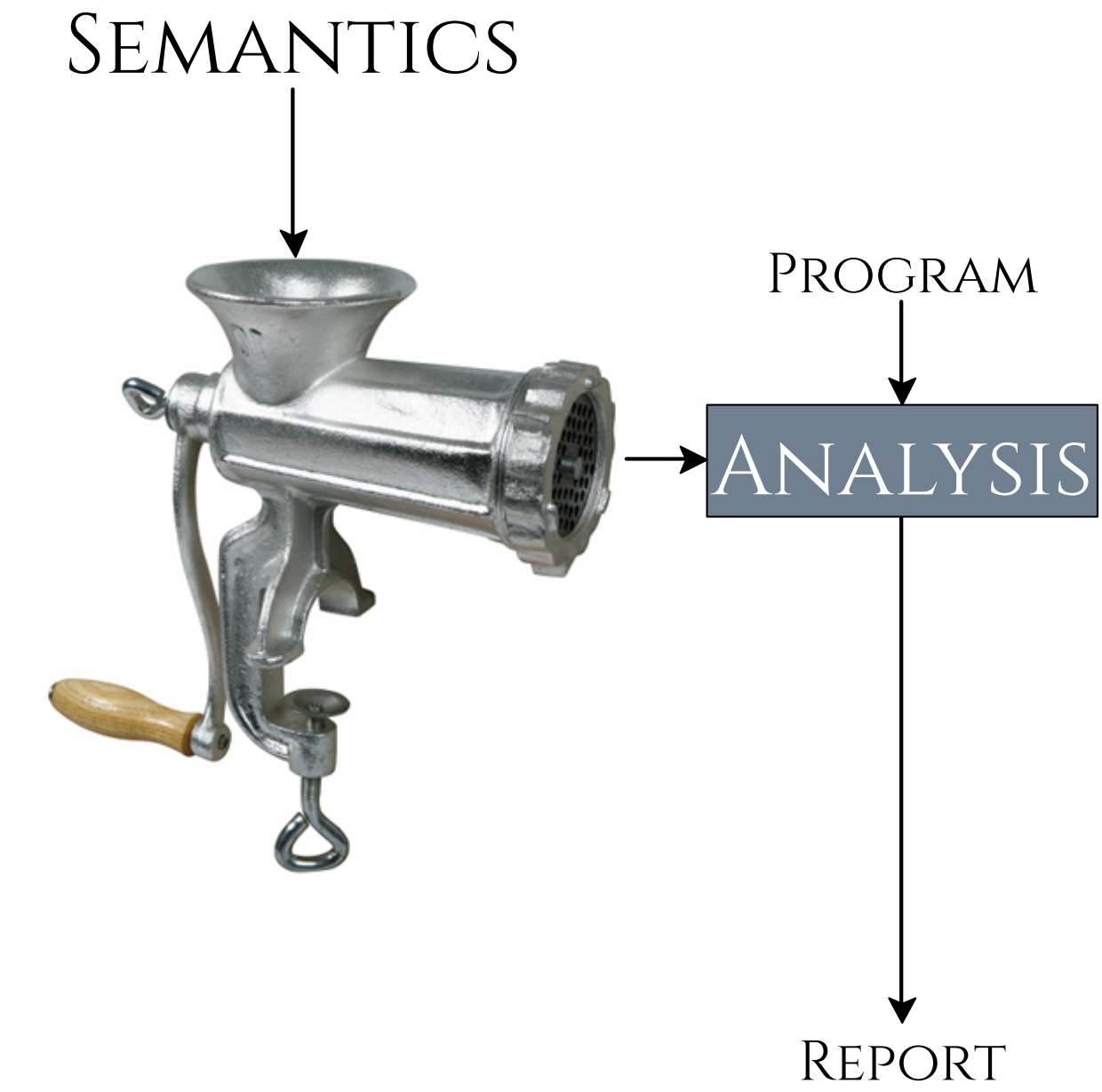

Semantics
Program
Analysis
Report

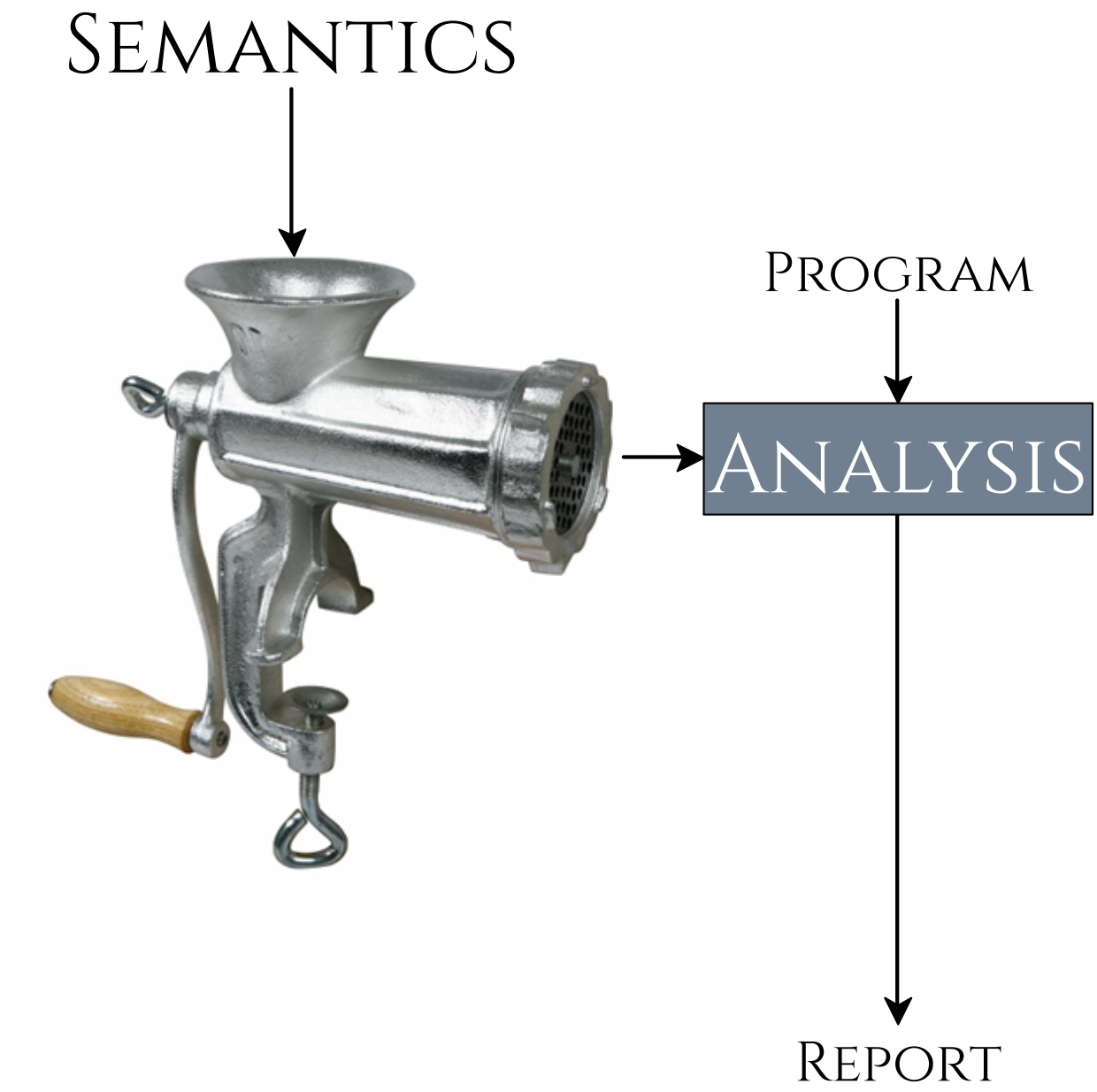
Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

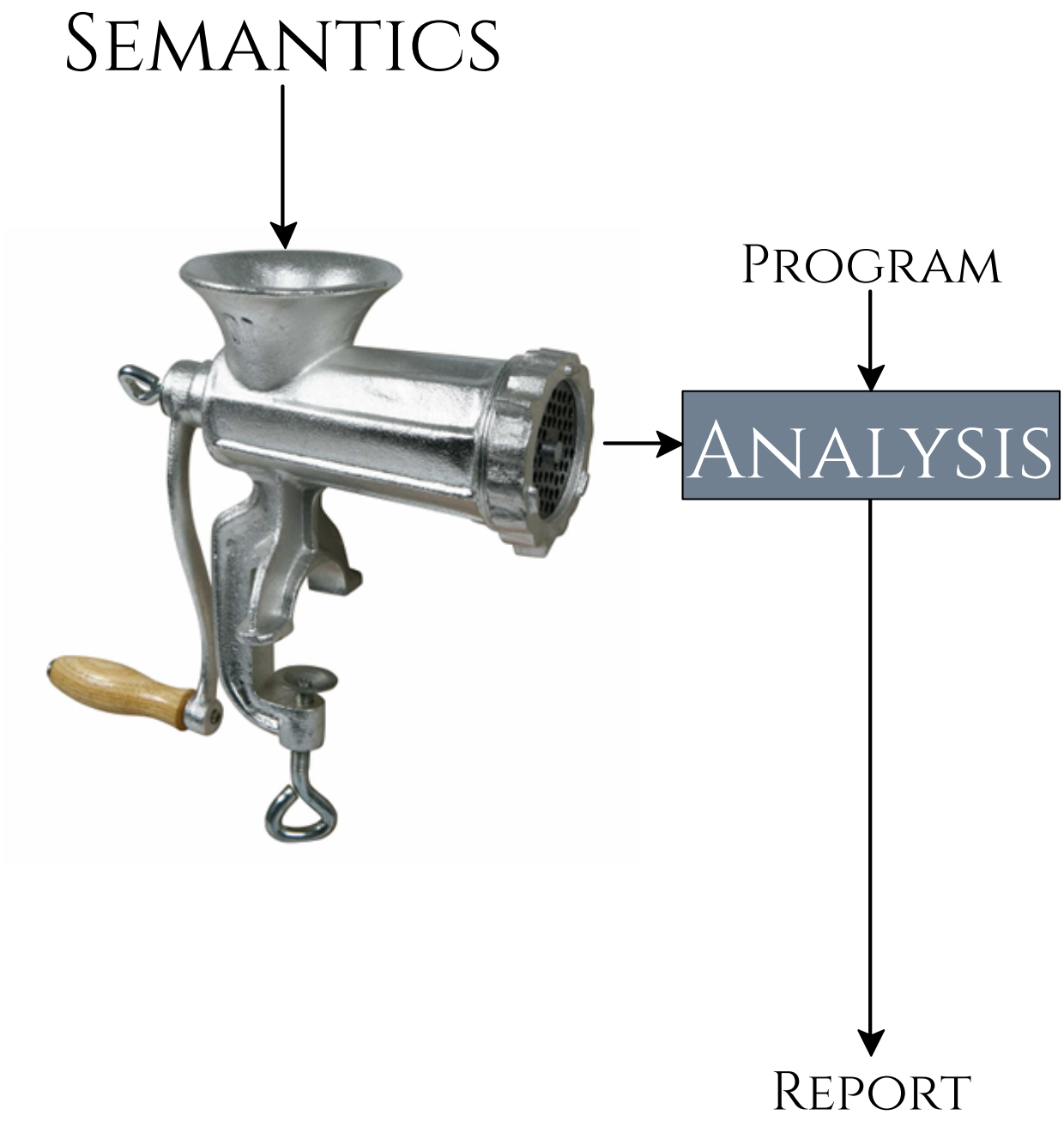
Report

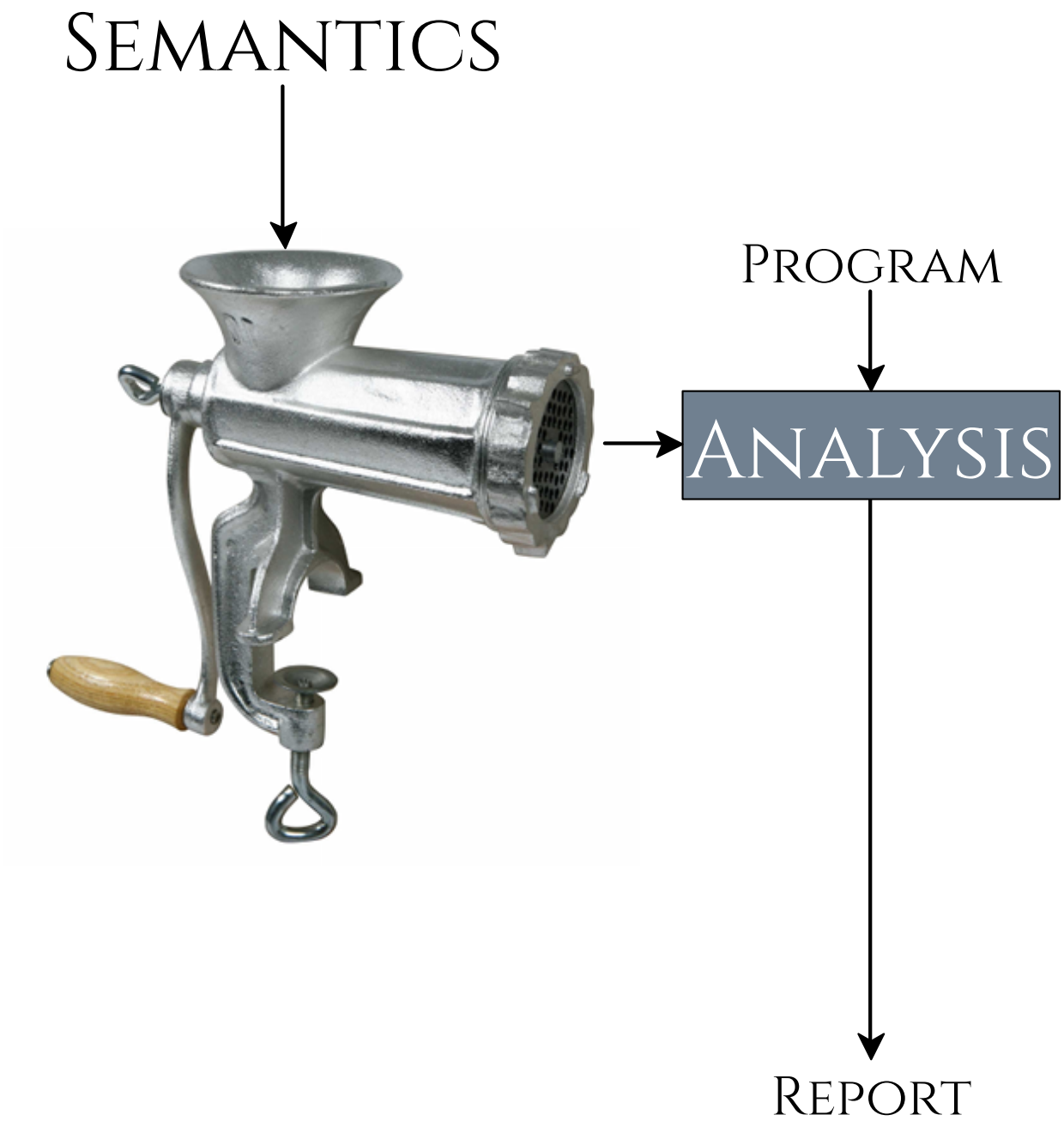
Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

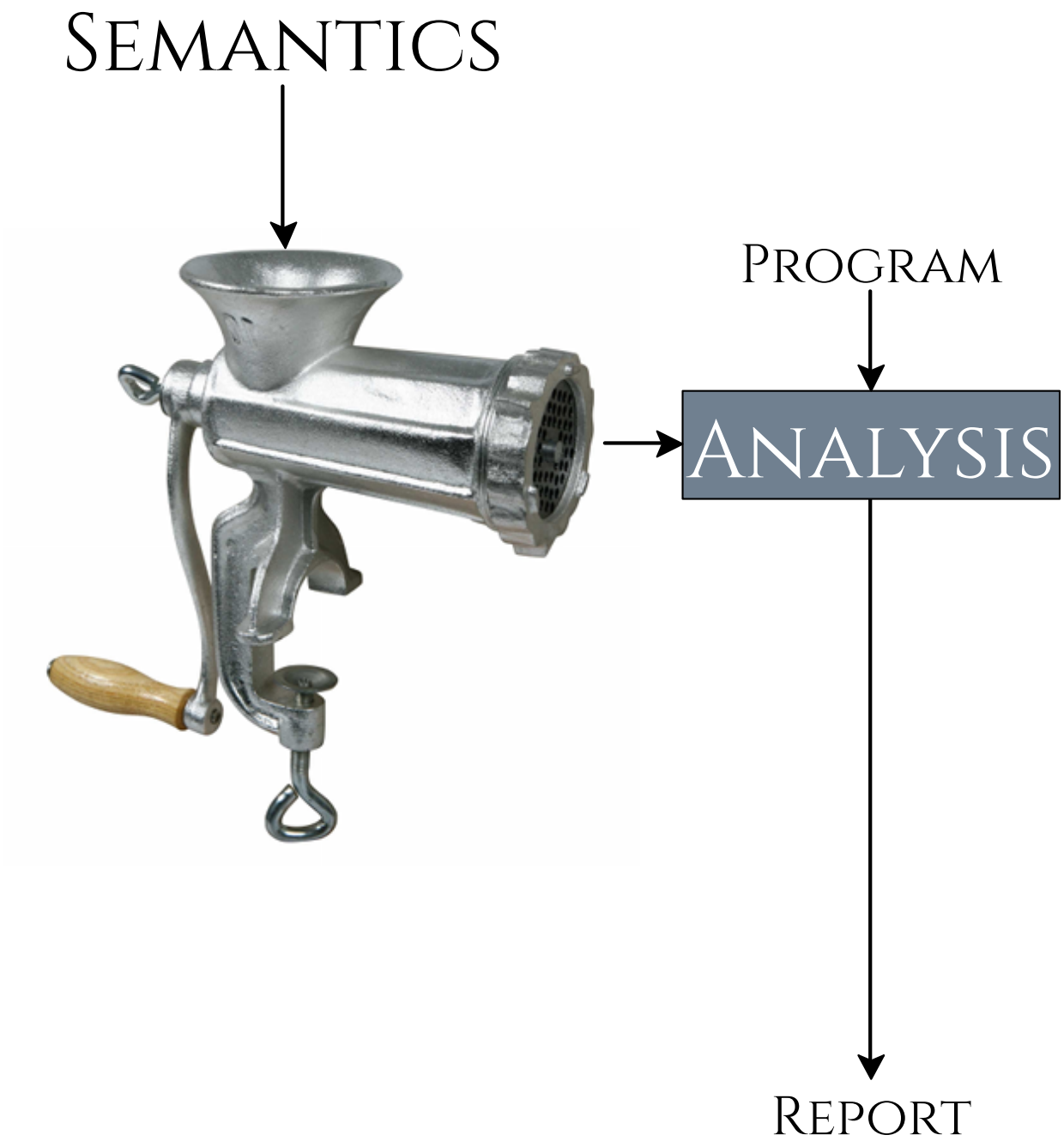
Report

Semantics

Program

Analysis

Report


Semantics
Program
Analysis
Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Report

Semantics

Program

Analysis

Maybe

Semantics

Program

Analysis

Maybe

Semantics

Program

Analysis

Maybe

Semantics

Program

Analysis

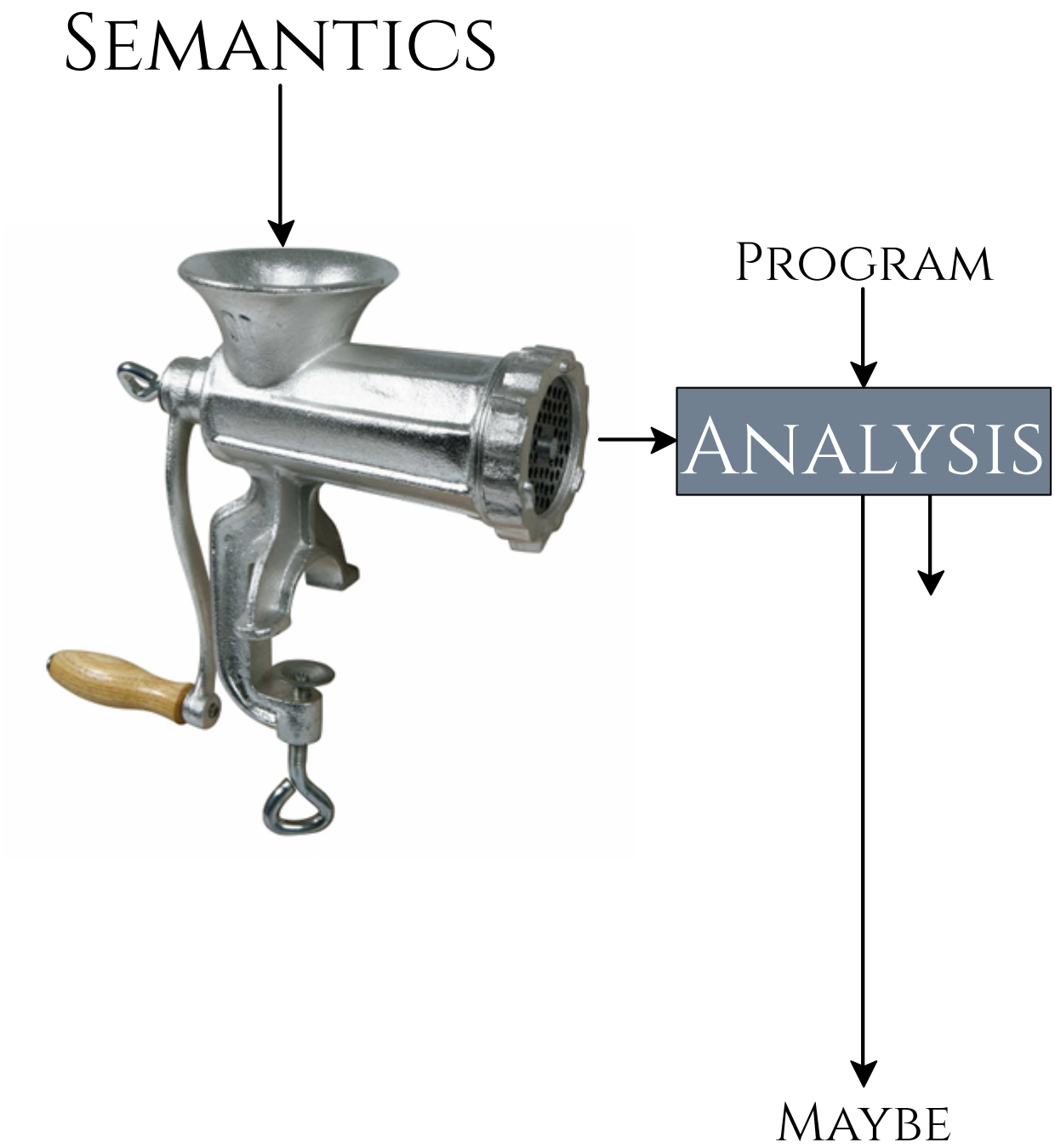
Maybe

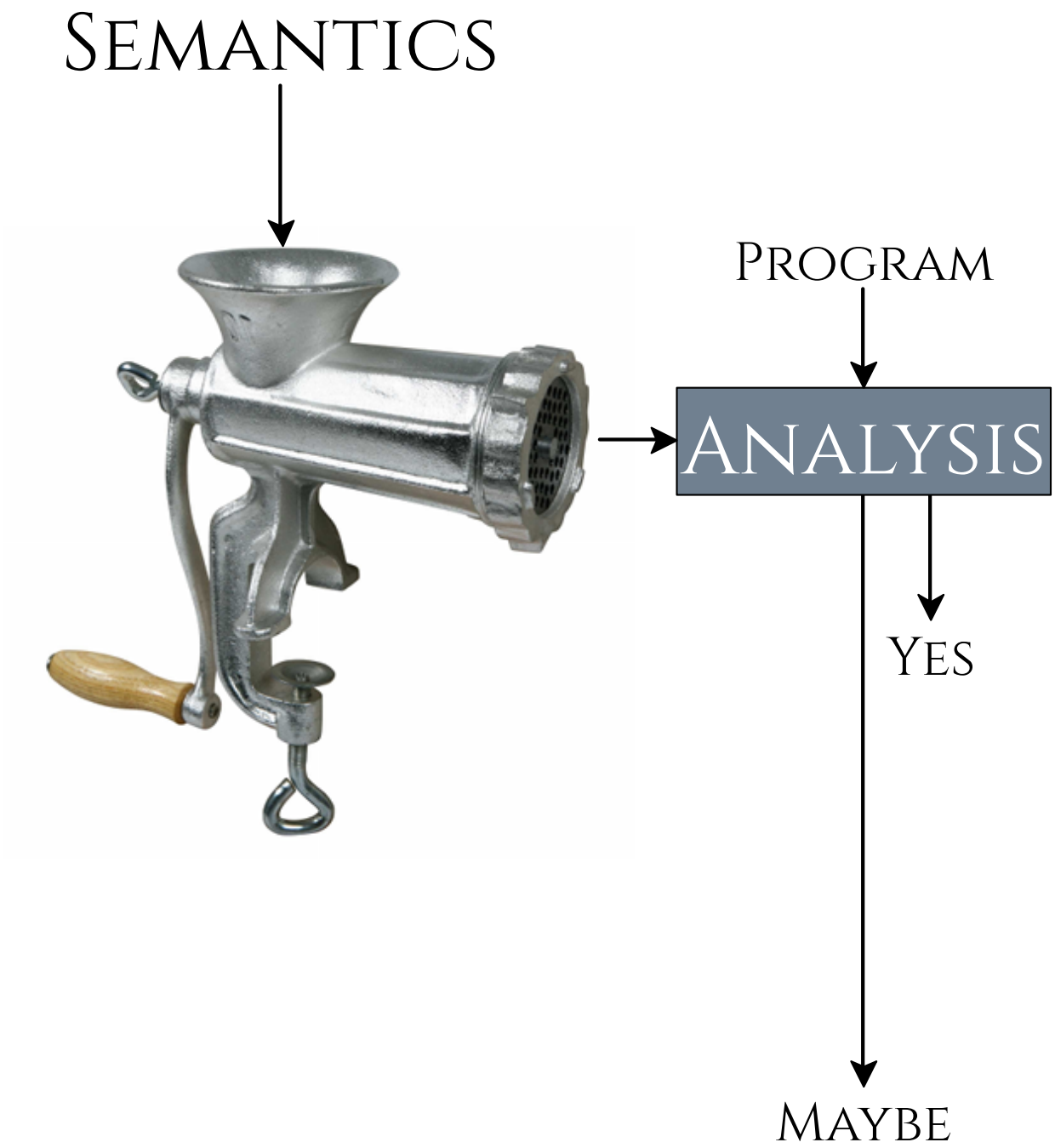
Semantics

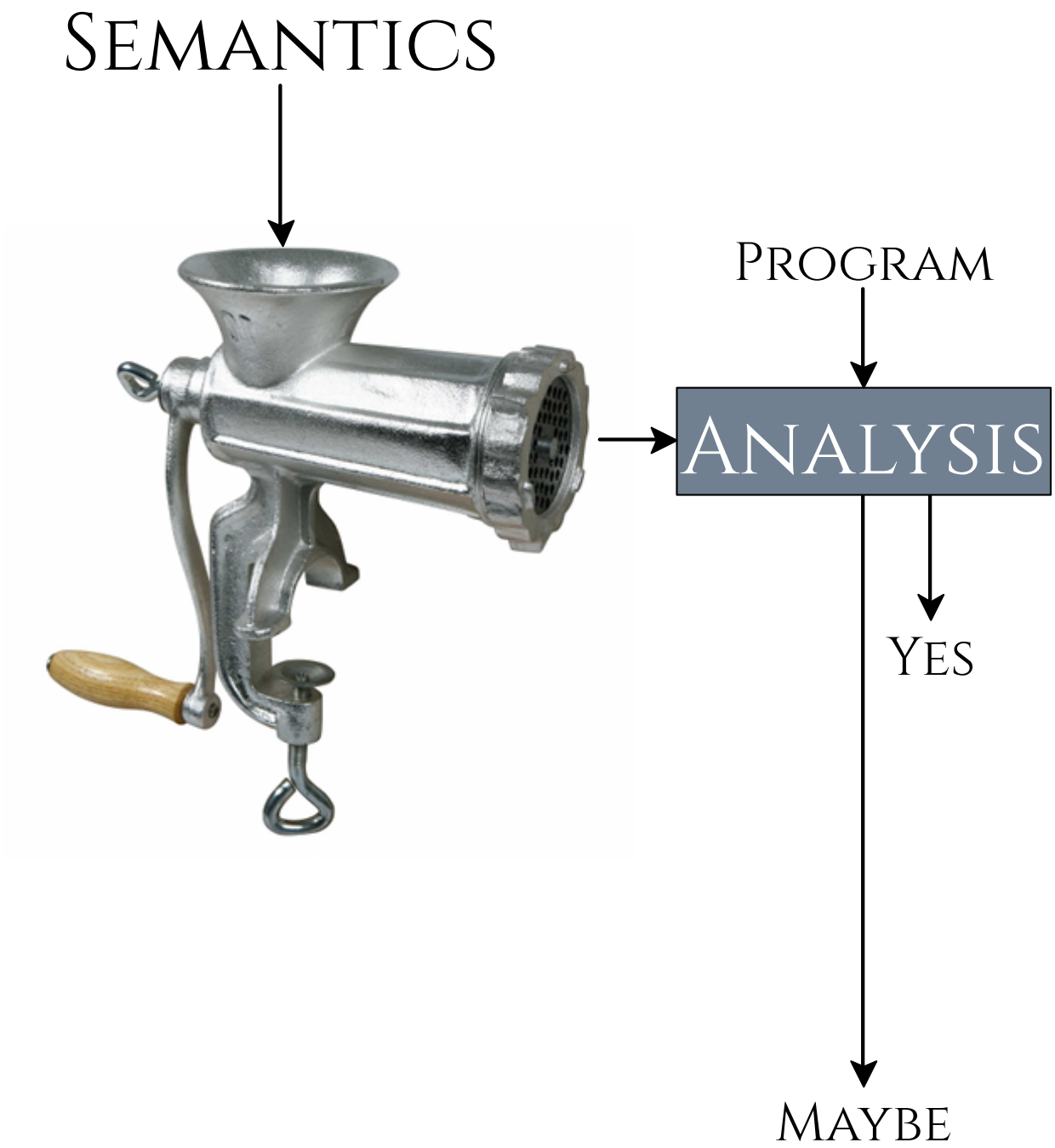
Program

Analysis

Maybe


Semantics
Program
Analysis
Maybe

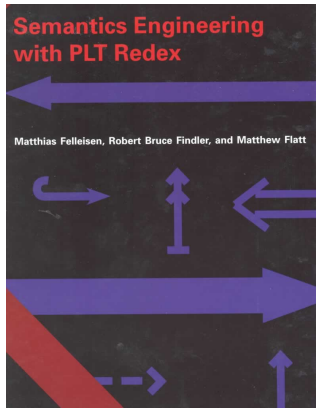

Semantics
Program
Analysis
Yes
Maybe


Semantics
Program
Analysis
Yes
Maybe

Start with CESK machine

Control

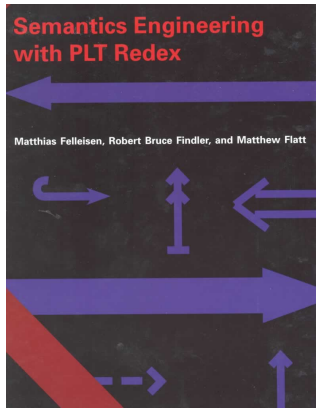
$\langle e, \rho, \sigma, \kappa \rangle$



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

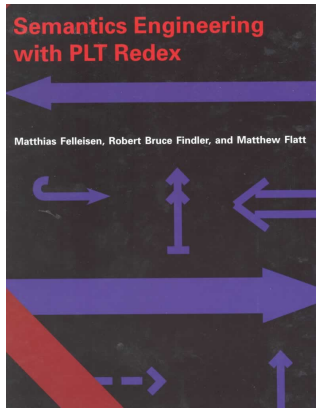
Environment



Start with CESK machine

Store

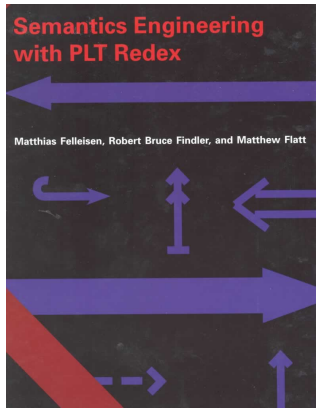
$\langle e, \rho, \sigma, \kappa \rangle$



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

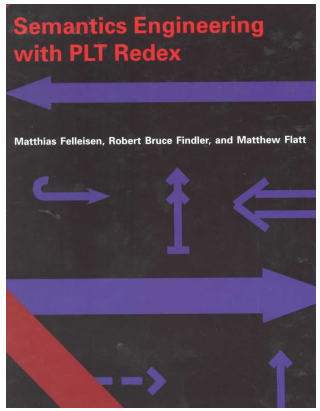
Kontinuation



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

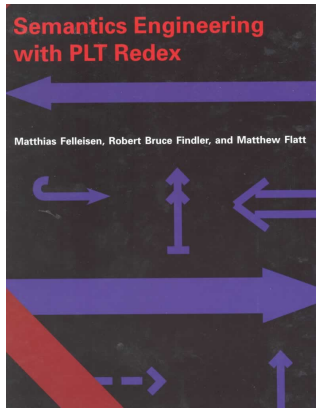
Kontinuation



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

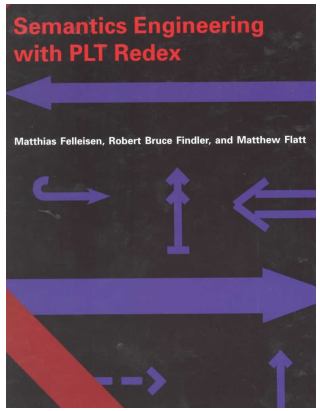
Kontinuation



Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

Kontinuation

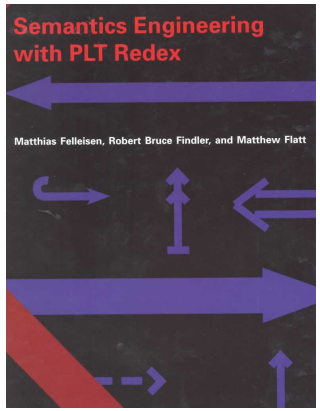


Start with CESK machine

$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$

↑

Kontinuation

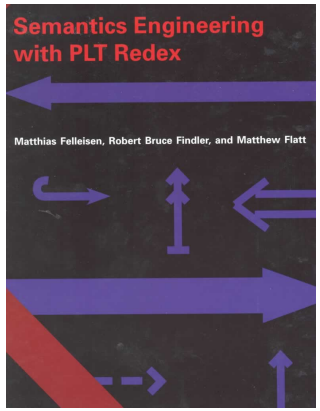


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Kontinuation

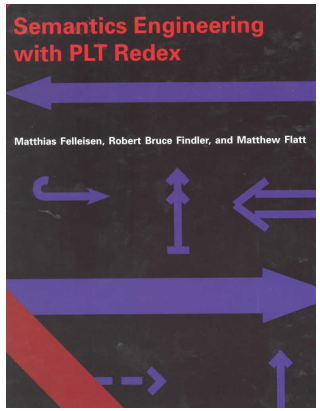


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Continuation

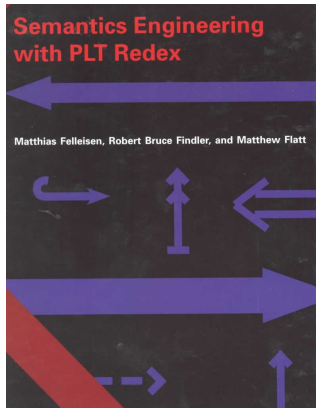


Start with CESK machine

$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$

$\uparrow$

Continuation

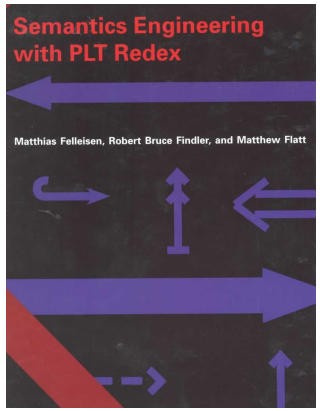


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Continuation

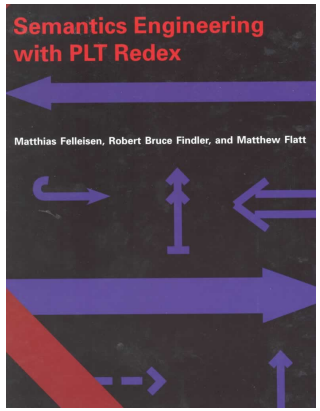


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

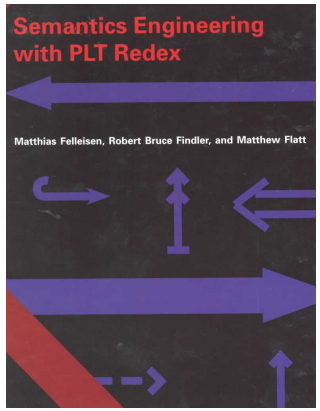


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

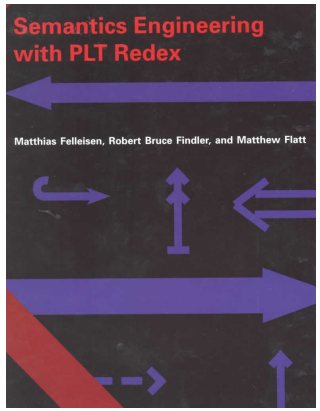


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

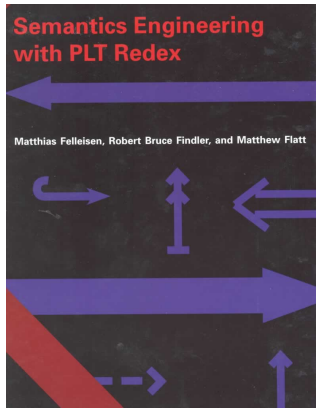


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

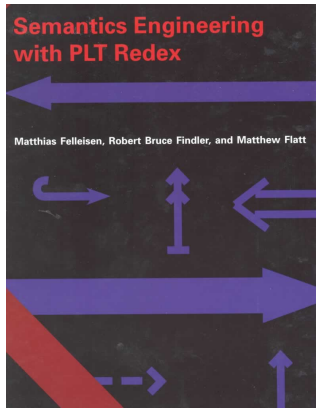


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

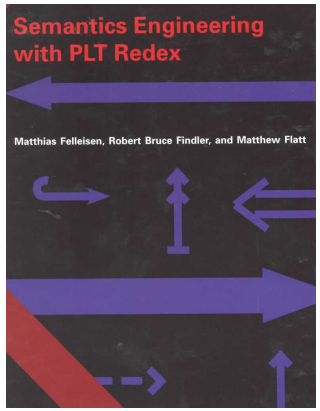


Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

↑

Continuation

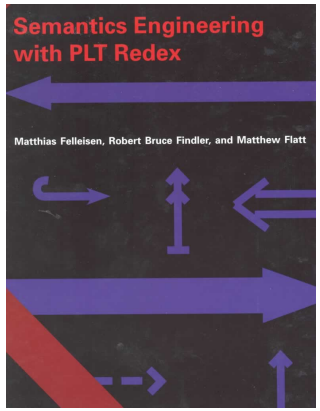


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Continuation

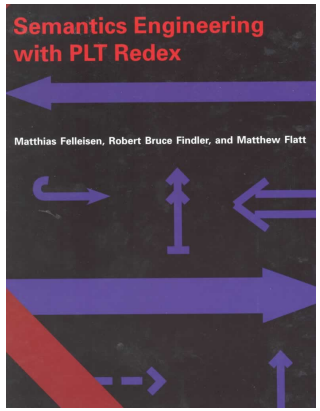


Start with CESK machine

$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$

$\uparrow$

Continuation

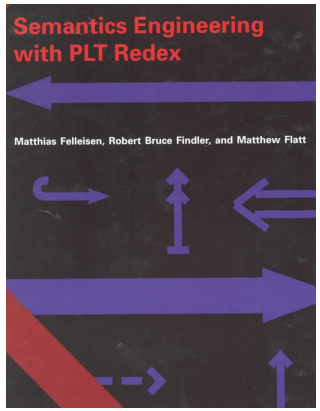


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Continuation

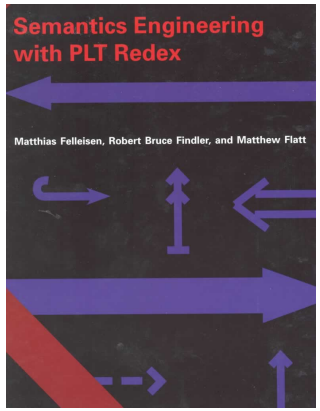


Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

$\uparrow$

Kontinuation

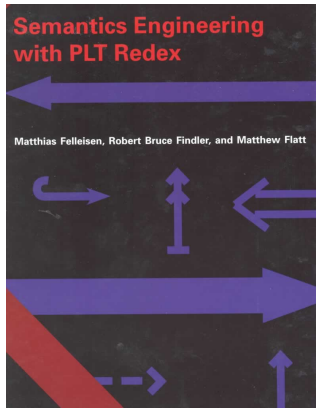


Start with CESK machine

$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$

↑

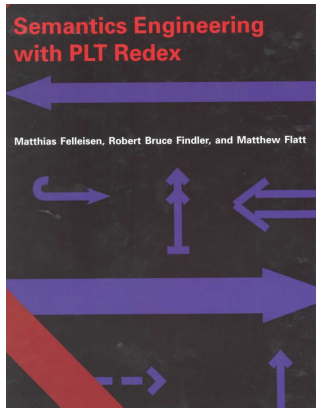
Kontinuation



Start with CESK machine

$\langle e, \rho, \sigma, \kappa \rangle$

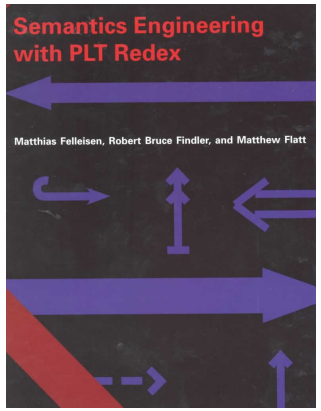
Kontinuation



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

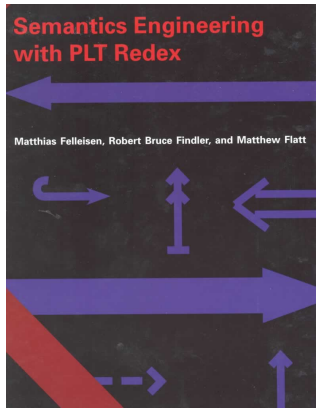
Kontinuation



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

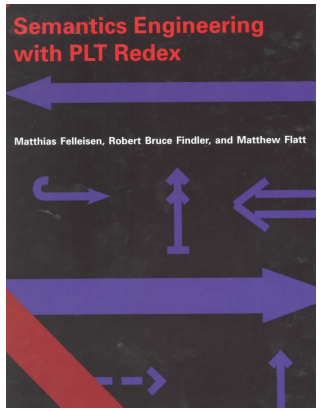
Kontinuation



Start with CESK machine

$$\langle e, \rho, \sigma, \kappa \rangle$$

Kontinuation

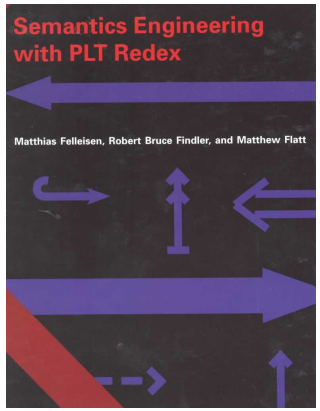


Start with CESK machine

$$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$$

↑

Kontinuation

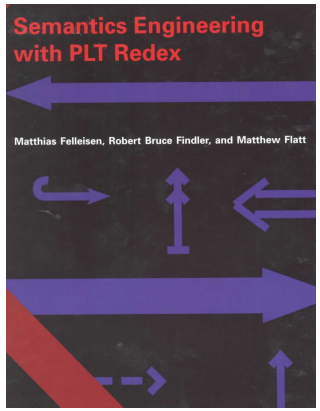


Start with CESK machine

$$\langle e \cdot \rho \cdot \sigma \cdot \kappa \rangle$$

↑

Kontinuation



Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathsf{appL}\ U\ [])] $$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathsf{appL}\ U\ [])] \rangle$$

$$\mapsto \langle U\ []\ []\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathsf{appL}\ U\ [])] \rangle$$

$$\mapsto \langle U\ []\ []\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$$

$$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$(U\ U) \mapsto_\beta (U\ U)$

$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathsf{appL}\ U\ [])] \rangle$

$\mapsto \langle U\ []\ []\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$

$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathsf{appL}\ x\ [x \mapsto a])] \rangle$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathrm{appL}\ U\ [])] \rangle$$

$$\mapsto \langle U\ []\ []\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

$$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$$

$$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

$$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathrm{appL}\ U\ [])] $$

$$\mapsto \langle U\ []\ []\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

$$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$$

$$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

$$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

$$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$$(U\ U) \mapsto_\beta (U\ U)$$

$$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\mathrm{appL}\ U\ [])] $$

$$\mapsto \langle U\ []\ []\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

$$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$$

$$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

$$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appL}\ x\ [x \mapsto a])] \rangle$$

$$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

$$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathrm{appR}\ \langle U, [] \rangle)] \rangle$$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$(U\ U) \mapsto_\beta (U\ U)$

$\langle (U\ U)\ []\ []\ [] \rangle \mapsto \langle U\ []\ []\ [(\text{appL}\ U\ [])] \rangle$

$\mapsto \langle U\ []\ []\ [(\text{appR}\ \langle U, [] \rangle)] \rangle$

$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\text{appL}\ x\ [x \mapsto a])] \rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\text{appL}\ x\ [x \mapsto a])] \rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\text{appR}\ \langle U, [] \rangle)] \rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\text{appR}\ \langle U, [] \rangle)] \rangle$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$(U\ U)$

$\langle (U\ U$ $)]$

Finitely many addresses ⇒

Finitely many states

$\mapsto \langle U\ []\ []\ [(\text{appR}\ \langle U,[]\rangle)]\rangle$

$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U,[]\rangle]\ []\rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U,[]\rangle]\ [(\text{appL}\ x\ [x \mapsto a])]\rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U,[]\rangle]\ [(\text{appL}\ x\ [x \mapsto a])]\rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U,[]\rangle]\ [(\text{appR}\ \langle U,[]\rangle)]\rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U,[]\rangle]\ [(\text{appR}\ \langle U,[]\rangle)]\rangle$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

$(U\ U)$

$\langle (U\ U$ $)]$

Finitely many addresses ⇒

Finitely many states?

$\mapsto \langle U\ []\ []\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$

$\mapsto \langle (x\ x)\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [] \rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathsf{appL}\ x\ [x \mapsto a])] \rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathsf{appL}\ x\ [x \mapsto a])] \rangle$

$\mapsto \langle x\ [x \mapsto a]\ [a \mapsto \langle U, [] \rangle]\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$

$\mapsto \langle U\ []\ [a \mapsto \langle U, [] \rangle]\ [(\mathsf{appR}\ \langle U, [] \rangle)] \rangle$

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

(U U)

⟨(U U

Finitely many addresses ⇒

Finitely many states?

)]

↦ ⟨U [] [] [(appR ⟨U,[]⟩)]⟩

↦ ⟨(

Problem: implicit allocation!

↦ ⟨x

a])]⟩

↦ ⟨U [] [a ↦ ⟨U,[]⟩] [(appL x [x ↦ a])]⟩

↦ ⟨x [x ↦ a] [a ↦ ⟨U,[]⟩] [(appR ⟨U,[]⟩)]⟩

↦ ⟨U [] [a ↦ ⟨U,[]⟩] [(appR ⟨U,[]⟩)]⟩

Let's evaluate $\Omega = ((\lambda\ x\ (x\ x))\ (\lambda\ x\ (x\ x)))$

$$U = (\lambda\ x\ (x\ x))$$

(U U)

⟨(U U

)]

Finitely many addresses ⇒

Finitely many states?

↦ ⟨U [] [] [(appR ⟨U,[]⟩)]⟩

↦ ⟨(

Problem: implicit allocation!

The stack can grow unbounded

↦ ⟨x

a])]⟩

↦ ⟨U [] [a ↦ ⟨U,[]⟩] [(appL x [x ↦ a])]⟩

↦ ⟨x [x ↦ a] [a ↦ ⟨U,[]⟩] [(appR ⟨U,[]⟩)]⟩

↦ ⟨U [] [a ↦ ⟨U,[]⟩] [(appR ⟨U,[]⟩)]⟩

What does it mean to finitize allocation?

What does it mean to finitize allocation?

Some allocations will reuse addresses.

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

$[a \mapsto v_o]$

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

$[a \mapsto v_0]$ $[a \mapsto v_1]$

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

$[a \mapsto v_0]$ $[a \mapsto v_1]$

$[a \mapsto \{v_0, v_1\}]$

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

$[a \mapsto v_0]$ $[a \mapsto v_1]$

$[a \mapsto \{v_0, v_1\}]$

Lookup a means *either* v_0 *or* v_1

What does it mean to finitize allocation?

Some allocations will reuse addresses.

Old usage not dead!

$[a \mapsto v_0]$ $[a \mapsto v_1]$

$[a \mapsto \{v_0, v_1\}]$

Lookup a means *either* v_0 *or* v_1 (or both)

Question: when do we lookup?

Question: when do we lookup?

AAM: whenever a rule says.

Question: when do we lookup?

AAM: whenever a rule says.

$\langle x\ \rho\ \sigma\ \kappa\rangle \mapsto \langle v\ \rho'\ \sigma\ \kappa\rangle$ where $\langle v,\rho'\rangle \in \sigma(\rho(x))$

Question: when do we lookup?

AAM: whenever a rule says.

$\langle x\ \rho\ \sigma\ \kappa\rangle \mapsto \langle v\ \rho'\ \sigma\ \kappa\rangle$ where $\langle v,\rho'\rangle \in \sigma(\rho(x))$

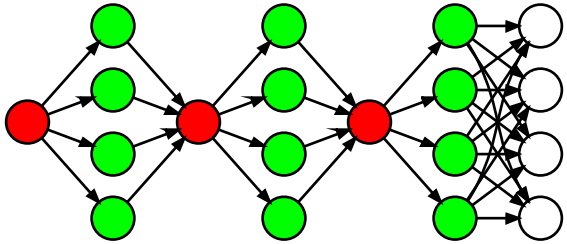
(f x y)

Question: when do we lookup?

AAM: whenever a rule says.

$\langle x\ \rho\ \sigma\ \kappa\rangle \mapsto \langle v\ \rho'\ \sigma\ \kappa\rangle$ where $\langle v,\rho'\rangle \in \sigma(\rho(x))$

(f x y)

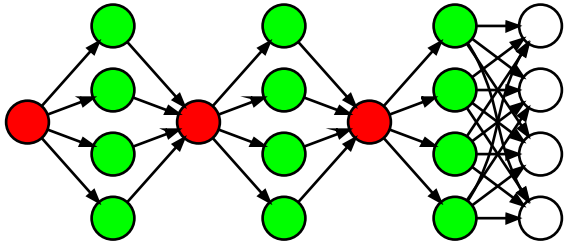


Question: when do we lookup?

AAM: whenever a rule says.

$\langle x\ \rho\ \sigma\ \kappa\rangle \mapsto \langle v\ \rho'\ \sigma\ \kappa\rangle$ where $\langle v,\rho'\rangle \in \sigma(\rho(x))$

(f x y)



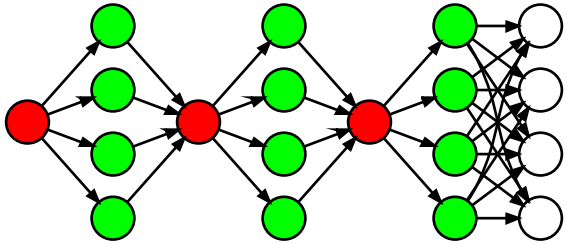
Me: whenever a looked up value is *inspected.*

Question: when do we lookup?

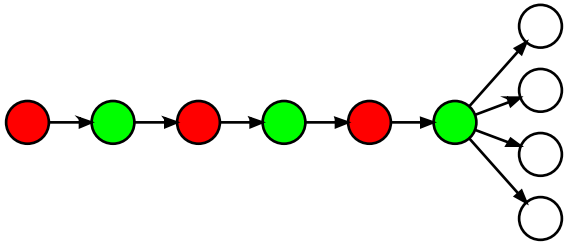
AAM: whenever a rule says.

$\langle x\ \rho\ \sigma\ \kappa\rangle \mapsto \langle v\ \rho'\ \sigma\ \kappa\rangle$ where $\langle v,\rho'\rangle \in \sigma(\rho(x))$

(f x y)

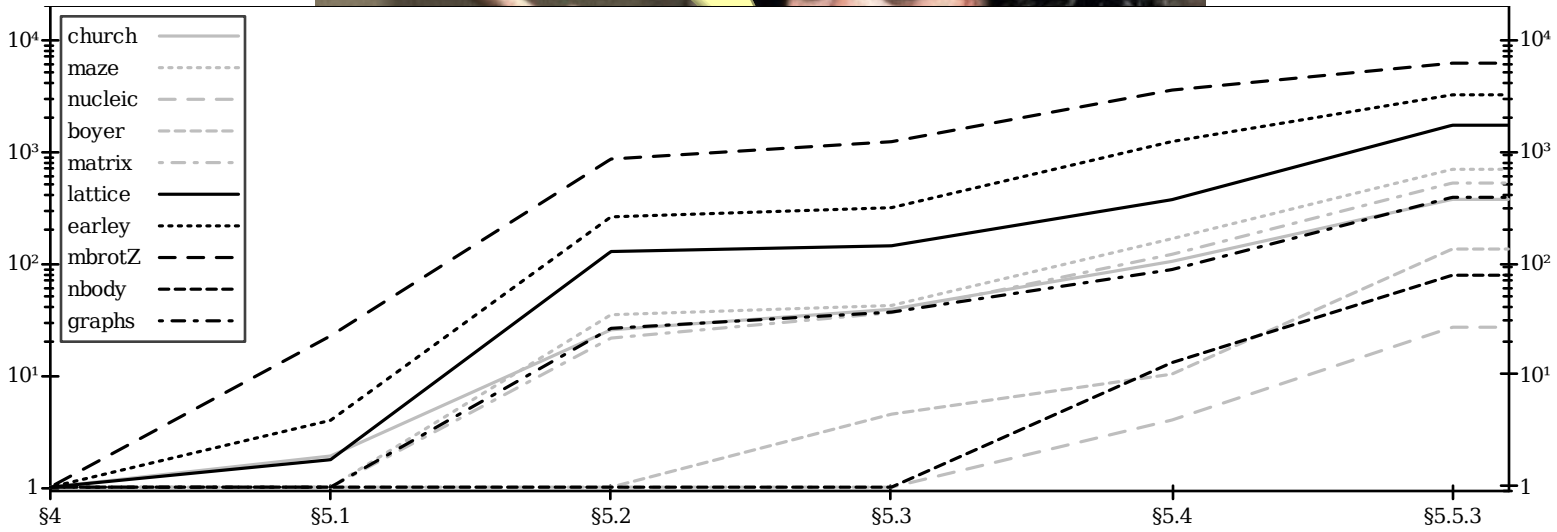


Me: whenever a looked up value is *inspected.*




Yada
yada
yada...

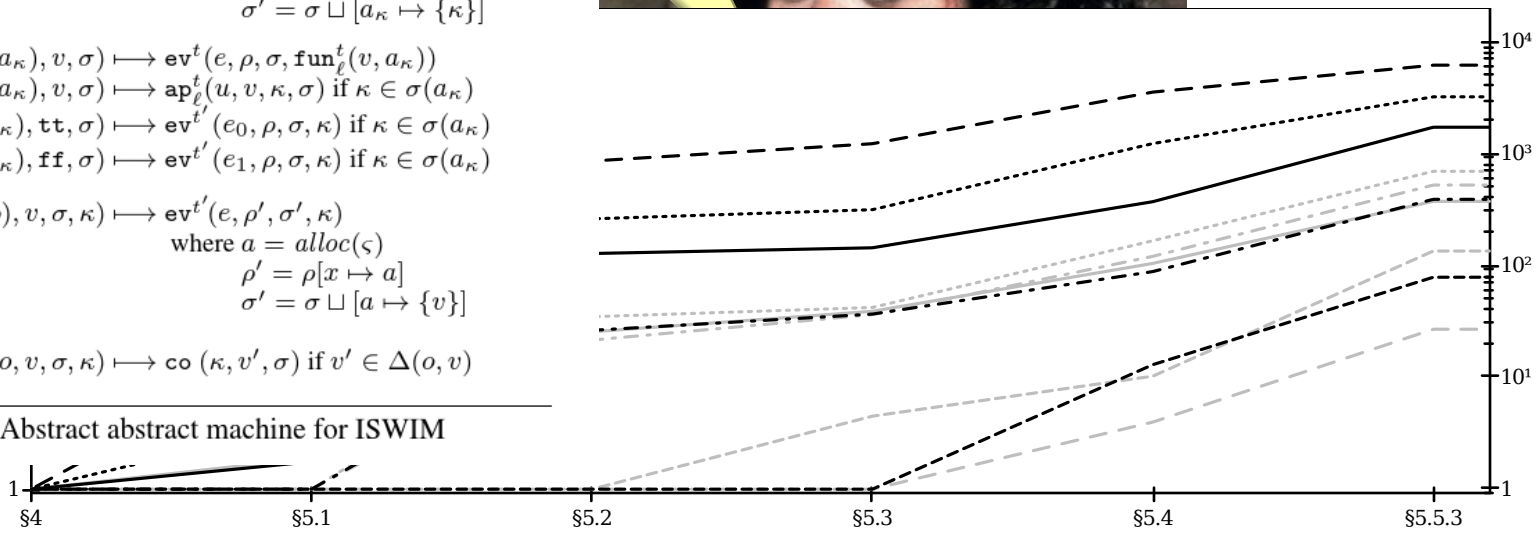

Yada
yada
yada...


church
maze
nucleic
boyer
matrix
lattice
earley
mbrotZ
nbody
graphs
10⁴
10³
10²
10¹
1
§4
§5.1
§5.2
§5.3
§5.4
§5.5.3

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\begin{aligned}
\varsigma \longmapsto \varsigma' &\text{ defined to be the following} \\
&\text{let } t' = tick(\varsigma) \\
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_1, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e, \rho', \sigma', \kappa) \\
&\quad \text{where } a = alloc(\varsigma) \\
&\quad\quad \rho' = \rho[x \mapsto a] \\
&\quad\quad \sigma' = \sigma \sqcup [a \mapsto \{v\}] \\
\\
\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)
\end{aligned}$$

Figure 5. Abstract abstract machine for ISWIM



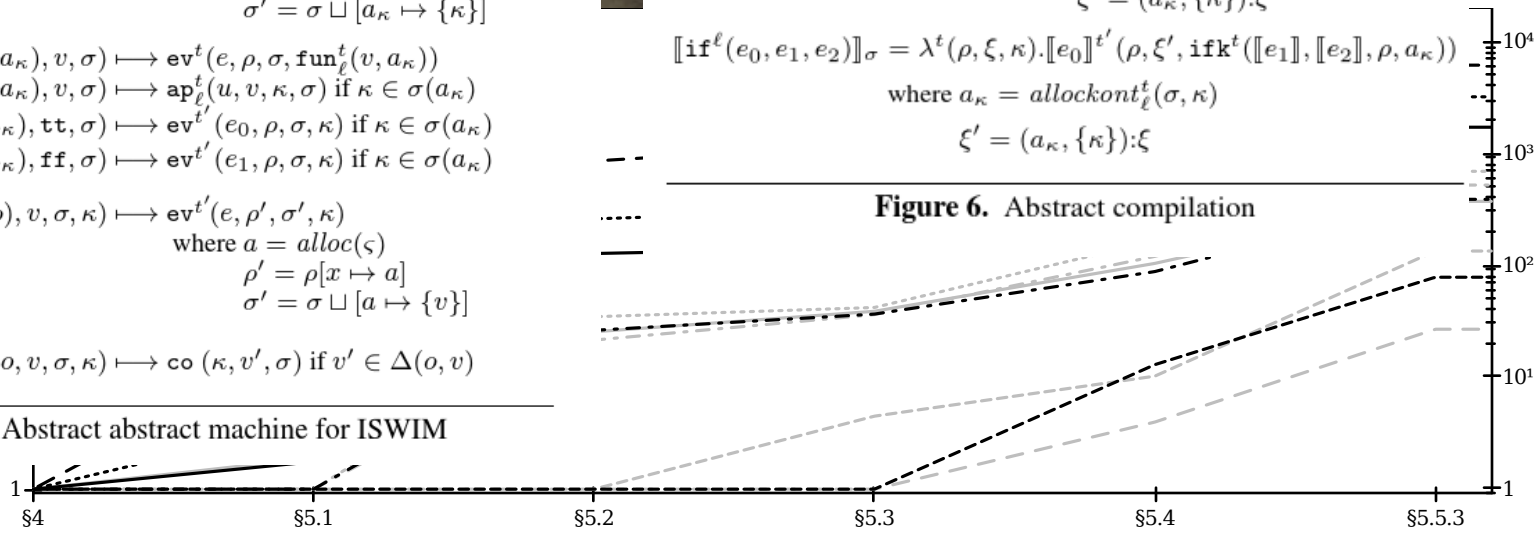
$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\begin{aligned}
\varsigma \longmapsto \varsigma' &\text{ defined to be the following} \\
&\text{let } t' = tick(\varsigma) \\
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\qquad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\qquad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_1, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e, \rho', \sigma', \kappa) \\
&\quad \text{where } a = alloc(\varsigma) \\
&\qquad \rho' = \rho[x \mapsto a] \\
&\qquad \sigma' = \sigma \sqcup [a \mapsto \{v\}] \\
\\
\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)
\end{aligned}$$

Figure 5. Abstract abstract machine for ISWIM

$$\begin{aligned}
[\![_]\!] : Expr &\to Store \\
&\to Env \times Store\Delta \times Kont \times Time \\
&\to State \\
t' &= tick(\ell, \rho, \sigma, t) \\
[\![\mathtt{var}\ (x)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, \mathtt{addr}\ (\rho(x))), \xi \\
[\![\mathtt{lit}\ (l)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, l), \xi \\
[\![\mathtt{lam}\ (x, e)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, \mathtt{clos}\ (x, [\![e]\!], \rho)), \xi \\
[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).[\![e_0]\!]^{t'}(\rho, \xi', \mathtt{arg}^t_\ell([\![e_1]\!], \rho, a_\kappa)) \\
&\text{where} \quad a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\qquad\quad \xi' = (a_\kappa, \{\kappa\}){:}\xi \\
[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).[\![e_0]\!]^{t'}(\rho, \xi', \mathtt{ifk}^t([\![e_1]\!], [\![e_2]\!], \rho, a_\kappa)) \\
&\text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad \xi' = (a_\kappa, \{\kappa\}){:}\xi
\end{aligned}$$

Figure 6. Abstract compilation

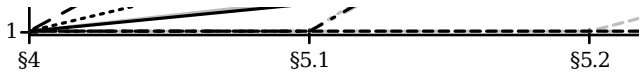


$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$\varsigma \longmapsto \varsigma'$ defined to be the following
let $t' = tick(\varsigma)$

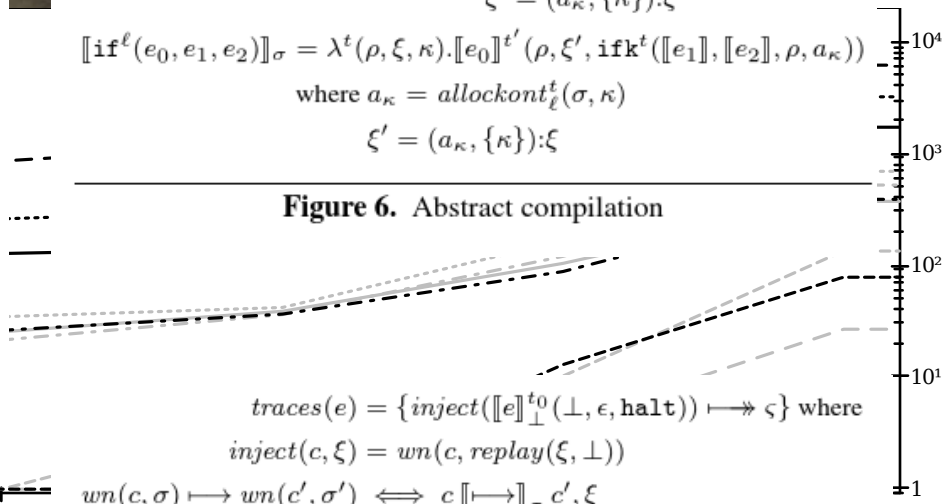
$$\begin{aligned}
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_1, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e, \rho', \sigma', \kappa) \\
&\quad \text{where } a = alloc(\varsigma) \\
&\quad\quad \rho' = \rho[x \mapsto a] \\
&\quad\quad \sigma' = \sigma \sqcup [a \mapsto \{v\}] \\
\\
\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)
\end{aligned}$$

Figure 5. Abstract abstract machine for ISWIM



$$\begin{aligned}
[\![_]\!] &: Expr \to Store \\
&\to Env \times Store\Delta \times Kont \times Time \\
&\to State \\
t' &= tick(\ell, \rho, \sigma, t) \\
[\![\mathtt{var}\ (x)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, \mathtt{addr}\ (\rho(x))), \xi \\
[\![\mathtt{lit}\ (l)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, l), \xi \\
[\![\mathtt{lam}\ (x, e)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).\mathtt{co}\ (\kappa, \mathtt{clos}\ (x, [\![e]\!], \rho)), \xi \\
[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).[\![e_0]\!]^{t'}(\rho, \xi', \mathtt{arg}^t_\ell([\![e_1]\!], \rho, a_\kappa)) \\
&\quad \text{where} \quad a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad\quad\quad \xi' = (a_\kappa, \{\kappa\}){:}\xi \\
[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma &= \lambda^t(\rho, \xi, \kappa).[\![e_0]\!]^{t'}(\rho, \xi', \mathtt{ifk}^t([\![e_1]\!], [\![e_2]\!], \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \xi' = (a_\kappa, \{\kappa\}){:}\xi
\end{aligned}$$

Figure 6. Abstract compilation



$$\begin{aligned}
traces(e) &= \{inject([\![e]\!]^{t_0}_\perp(\perp, \epsilon, \mathtt{halt})) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where} \\
inject(c, \xi) &= wn(c, replay(\xi, \perp)) \\
wn(c, \sigma) \longmapsto wn(c', \sigma') &\iff c\ [\![\longmapsto]\!]_\sigma\ c', \xi \\
&\xi \text{ is such that } replay(\xi, \sigma) = \sigma'
\end{aligned}$$

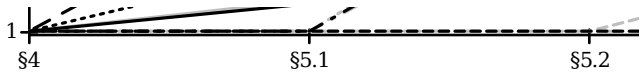
$$\begin{aligned}
\mathtt{co}\ (\mathtt{arg}^t_\ell(k, \rho, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ k^t(\sigma)(\rho, \xi, \mathtt{fun}^t_\ell(a_f, a_\kappa)) \\
\text{where } a_f = alloc(\varsigma) \\
\xi = (a_f, force(\sigma, v)){:}\epsilon \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(a_f, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ \mathtt{ap}^t_\ell(u, a, \kappa), (a, force(\sigma, v)){:}\epsilon \\
\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ [\![\longmapsto]\!]_\sigma\ k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ [\![\longmapsto]\!]_\sigma\ k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, k, \rho), a, \kappa)\ [\![\longmapsto]\!]_\sigma\ k^{t'}(\sigma)(\rho', \xi, \kappa) \\
\text{where } \rho' = \rho[x \mapsto a] \\
\xi = (a, \sigma(a)){:}\epsilon
\end{aligned}$$

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$\varsigma \longmapsto \varsigma'$ defined to be the following
let $t' = tick(\varsigma)$

$$\begin{aligned}
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_1, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e, \rho', \sigma', \kappa) \\
&\quad \text{where } a = alloc(\varsigma) \\
&\quad\quad \rho' = \rho[x \mapsto a] \\
&\quad\quad \sigma' = \sigma \sqcup [a \mapsto \{v\}] \\
\\
\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)
\end{aligned}$$

Figure 5. Abstract abstract machine for ISWIM



$$\begin{aligned}
[\![_]\!] &: Ex \\
t' &= ti \\
[\![\mathtt{var}\ (x)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{lit}\ (l)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{lam}\ (x, e)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma &= \lambda^t \\
&\text{wh} \\
[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma &= \lambda^t \\
&\text{wher}
\end{aligned}$$

Figu

$$\begin{aligned}
traces(e \\
inject(c, \xi \\
wn(c, \sigma) &\longmapsto wn(c', \sigma' \\
\xi &\text{ is such that} \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(k, \rho, & \\
&\text{wh}
\end{aligned}$$

$$lookup(V, t) = \begin{cases} vs & \text{if } V = (t', vs){:}V', t' \leq t \\ vs' & \text{if } V = (t', vs){:}(t'', vs'){:}V', t' \end{cases}$$

$$\begin{aligned}
\sigma \sqcup_t [a \mapsto vs] &= \sigma[a \mapsto V], \Delta? \\
\text{where } (V, \Delta?) &= \sigma(a) \sqcup_t vs \\
\epsilon \sqcup_t vs &= (t, vs), \mathtt{tt} \\
(t', vs){:}V \sqcup_t vs' &= (t', vs \sqcup vs'){:}V, \delta?(vs, vs') \text{ if } t' \\
V \sqcup_t vs &= (t+1, vs^*){:}V, \mathtt{tt} \text{ if } vs_t \neq vs^* \\
\text{where } vs_t &= lookup(\sigma(a), t) \\
vs^* &= vs \sqcup vs_t \\
V \sqcup_t vs &= V, \mathtt{ff} \text{ otherwise}
\end{aligned}$$

$$\begin{aligned}
System &= (\widehat{State} \to \mathbb{N}^*) \times \wp(\widehat{State}) \times Store \times \\
\mathcal{F} &: System \to System \\
\mathcal{F}(\hat{S}, F, \sigma, t) &= (\hat{S}', F', \sigma', t') \\
\text{where } I &= \{(c', \xi) \mid c \in F, c\,[\![\longmapsto]\!]_{\sigma^*}\, c', \xi\} \\
\sigma^* &= \lambda a.lookup(\sigma(a), t) \\
(\sigma', \Delta?) &= replay(appendall(\{\xi \mid (_, \xi) \in I\}), \sigma) \\
t' &= \begin{cases} t+1 & \text{if } \Delta? \\ t & \text{otherwise} \end{cases} \\
F' &= \{c \mid (c, _) \in I, \Delta? \vee \hat{S}(c) \neq t{:}_\} \\
\hat{S}' &= \lambda c. \begin{cases} t{:}\hat{S}(c) & \text{if } c \in F' \\ \hat{S}(c) & \text{otherwise} \end{cases}
\end{aligned}$$

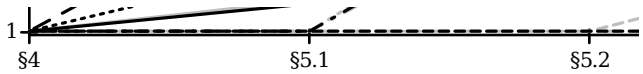
$$\begin{aligned}
\varsigma &= (a_f, force(\sigma, v)):\epsilon \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(a_f, a_\kappa), v)\ &[\![\longmapsto]\!]_\sigma\ \mathtt{ap}^t_\ell(u, a, \kappa), (a, force(\sigma, v)){:}\epsilon \\
&\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ &[\![\longmapsto]\!]_\sigma\ k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ &[\![\longmapsto]\!]_\sigma\ k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, k, \rho), a, \kappa)\ &[\![\longmapsto]\!]_\sigma\ k^{t'}(\sigma)(\rho', \xi, \kappa) \\
&\text{where } \rho' = \rho[x \mapsto a] \\
&\quad \xi = (a, \sigma(a)){:}\epsilon
\end{aligned}$$

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$\varsigma \longmapsto \varsigma'$ defined to be the following
let $t' = tick(\varsigma)$

$$\begin{aligned}
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_1, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e, \rho', \sigma', \kappa) \\
&\quad \text{where } a = alloc(\varsigma) \\
&\quad\quad \rho' = \rho[x \mapsto a] \\
&\quad\quad \sigma' = \sigma \sqcup [a \mapsto \{v\}] \\
\\
\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)
\end{aligned}$$

Figure 5. Abstract abstract machine for ISWIM



$$\begin{aligned}
[\![_]\!] &: Ex \\
t' &= ti \\
[\![\mathtt{var}\ (x)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{lit}\ (l)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{lam}\ (x, e)]\!]_\sigma &= \lambda^t \\
[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma &= \lambda^t \\
&\text{wh} \\
[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma &= \lambda^t \\
&\text{wher}
\end{aligned}$$

Figu

$$\begin{aligned}
traces(e \\
inject(c, \xi \\
wn(c, \sigma) &\longmapsto wn(c', \sigma' \\
\xi &\text{ is such that} \\
\\
\mathtt{co}\ (\mathtt{arg}^t_\ell(k, \rho, & \\
&\text{wh}
\end{aligned}$$

$$lookup(V, t) = \begin{cases} vs & \text{if } V = (t', vs){:}V', t' \leq t \\ vs' & \text{if } V = (t', vs){:}(t'', vs'){:}V', t' \end{cases}$$

$$\begin{aligned}
\sigma \sqcup_t [a \mapsto vs] &= \sigma[a \mapsto V], \Delta? \\
&\text{where } (V, \Delta?) = \sigma(a) \sqcup_t vs \\
\epsilon \sqcup_t vs &= (t, vs), \mathtt{tt} \\
(t', vs){:}V \sqcup_t vs' &= (t', vs \sqcup vs'){:}V, \delta?(vs, vs') \text{ if } t' \\
V \sqcup_t vs &= (t+1, vs^*){:}V, \mathtt{tt} \text{ if } vs_t \neq vs^* \\
&\text{where } vs_t = lookup(\sigma(a), t) \\
vs^* &= vs \sqcup vs_t \\
V \sqcup_t vs &= V, \mathtt{ff} \text{ otherwise}
\end{aligned}$$

$$System = (\widehat{State} \to \mathbb{N}^*) \times \wp(\widehat{State}) \times Store \times$$

$$\begin{aligned}
replay([(a_i, vs_i), \ldots], \sigma) &= (\sigma', \delta?(vs_i, \sigma(a_i)) \vee \ldots) \\
&\text{where } \sigma' = \sigma \sqcup [a_i \mapsto vs_i] \sqcup \ldots \\
\delta?(vs, vs') &= vs' \stackrel{?}{=} vs \sqcup vs' \\
\sigma' &= \lambda a.lookup(\sigma(a), t)
\end{aligned}$$

$$(\sigma', \Delta?) = replay(appendall(\{\xi \mid (\_, \xi) \in I\}), \sigma)$$

$$t' = \begin{cases} t+1 & \text{if } \Delta? \\ t & \text{otherwise} \end{cases}$$

$$F' = \{c \mid (c, \_) \in I, \Delta? \vee \hat{S}(c) \neq t{:}\_\}$$

$$\hat{S}' = \lambda c. \begin{cases} t'{:}\hat{S}(c) & \text{if } c \in F' \\ \hat{S}(c) & \text{otherwise} \end{cases}$$

$$\varsigma = (a_f, force(\sigma, v)){:}\epsilon$$

$$\begin{aligned}
\mathtt{co}\ (\mathtt{fun}^t_\ell(a_f, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ & \mathtt{ap}^t_\ell(u, a, \kappa), (a, force(\sigma, v)){:}\epsilon \\
&\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ [\![\longmapsto]\!]_\sigma\ & k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ [\![\longmapsto]\!]_\sigma\ & k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\\
\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, k, \rho), a, \kappa)\ [\![\longmapsto]\!]_\sigma\ & k^{t'}(\sigma)(\rho', \xi, \kappa) \\
&\text{where } \rho' = \rho[x \mapsto a] \\
&\quad \xi = (a, \sigma(a)){:}\epsilon
\end{aligned}$$

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

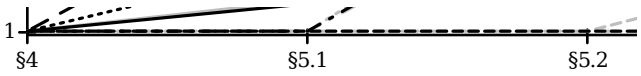
$\varsigma \longmapsto \varsigma'$ defined to be the following
let $t' = tick(\varsigma)$

$$\begin{aligned}
\mathtt{ev}\ (\mathtt{var}\ (x), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, v, \sigma) \text{ if } v \in \sigma(\rho(x)) \\
\mathtt{ev}\ (\mathtt{lit}\ (l), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, l, \sigma) \\
\mathtt{ev}^t(\mathtt{lam}\ (x, e), \rho, \sigma, \kappa) &\longmapsto \mathtt{co}\ (\kappa, \mathtt{clos}\ (x, e, \rho), \sigma) \\
\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa)) \\
&\quad \text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa) \\
&\quad\quad \sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}] \\
\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) &\longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa)) \\
\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) &\longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) &\longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa) \\
\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) &\longmapsto \mathtt{ev}^{t'}(
\end{aligned}$$

$$\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(e$$
where

$$\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) \longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)$$

Figure 5. Abstract abstract machine for ISWIM



$$[\![\_]\!] : Ex$$

$$t' = ti$$

$$[\![\mathtt{var}\ (x)]\!]_\sigma = \lambda^t$$

$$[\![\mathtt{lit}\ (l)]\!]_\sigma = \lambda^t$$

$$[\![\mathtt{lam}\ (x, e)]\!]_\sigma = \lambda^t$$

$$[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma = \lambda^t$$

wh

$$[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma = \lambda^t$$

wher

$$traces(e$$

$$inject(c, \xi$$

$$wn(c, \sigma) \longmapsto wn(c', \sigma'$$

ξ is such that

$$\mathtt{co}\ (\mathtt{arg}^t_\ell(k, \rho, c$$

wh

$$lookup(V, t) = \begin{cases} vs & \text{if } V = (t', vs):V', t' \leq t \\ vs' & \text{if } V = (t', vs):(t'', vs'):V', t' \end{cases}$$

$$\sigma \sqcup_t [a \mapsto vs] = \sigma[a \mapsto V], \Delta?$$
$$\text{where } (V, \Delta?) = \sigma(a) \sqcup_t vs$$
$$\epsilon \sqcup_t vs = (t, vs), \mathtt{tt}$$
$$(t', vs):V \sqcup_t vs' = (t', vs \sqcup vs'):V, \delta?(vs, vs') \text{ if } t'$$
$$V \sqcup_t vs = (t + 1, vs^*):V, \mathtt{tt} \text{ if } vs_t \neq vs^*$$
$$\text{where } vs_t = lookup(\sigma(a), t)$$
$$vs^* = vs \sqcup vs_t$$
$$V \sqcup_t vs = V, \mathtt{ff} \text{ otherwise}$$

$$\widehat{State} \to \mathbb{N}^*) \times \wp(\widehat{State}) \times Store \times$$

$$.], \sigma) = (\sigma', \delta?(vs_i, \sigma(a_i)) \vee \ldots)$$

$$\text{ere } \sigma' = \sigma \sqcup [a_i \mapsto vs_i] \sqcup \ldots$$

$$, vs') = vs' \stackrel{?}{=} vs \sqcup vs'$$

$$\sigma' = \lambda a.lookup(\sigma(a), t)$$

$$(\sigma', \Delta?) = replay(appendall(\{\xi \mid (\_, \xi) \in I\}), \sigma)$$

$$t' = \begin{cases} t + 1 & \text{if } \Delta? \\ t & \text{otherwise} \end{cases}$$

$$F' = \{c \mid (c, \_) \in I, \Delta? \vee \hat{S}(c) \neq t:\_\}$$

$$\hat{S}' = \lambda c. \begin{cases} t':\hat{S}(c) & \text{if } c \in F' \\ \hat{S}(c) & \text{otherwise} \end{cases}$$

$$\varsigma = (a_f, force(\sigma, v)):\epsilon$$

$$\mathtt{co}\ (\mathtt{fun}^t_\ell(a_f, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ \mathtt{ap}^t_\ell(u, a, \kappa), (a, force(\sigma, v)):\epsilon$$
$$\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ [\![\longmapsto]\!]_\sigma\ k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ [\![\longmapsto]\!]_\sigma\ k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$

$$\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, k, \rho), a, \kappa)\ [\![\longmapsto]\!]_\sigma\ k^{t'}(\sigma)(\rho', \xi, \kappa)$$
$$\text{where } \rho' = \rho[x \mapsto a]$$
$$\xi = (a, \sigma(a)):\epsilon$$

Simple transform

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\varsigma \longmapsto \varsigma' \text{ defined}$$

$$\mathtt{ev}^t(\mathtt{app}^{\ell}(e_0, e_1), \rho, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_{\ell}(e_1, \rho, a_\kappa))$$
$$\text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa)$$
$$\sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}]$$

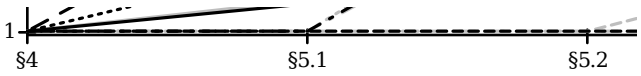
$$\mathtt{ev}^t(\mathtt{if}^{\ell}(e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa))$$
$$\text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa)$$
$$\sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}]$$

$$\mathtt{co}\,(\mathtt{arg}^t_{\ell}(e, \rho, a_\kappa), v, \sigma) \longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_{\ell}(v, a_\kappa))$$
$$\mathtt{co}\,(\mathtt{fun}^t_{\ell}(u, a_\kappa), v, \sigma) \longmapsto \mathtt{ap}^t_{\ell}(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\,(\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\,(\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) \longmapsto \mathtt{ev}^{t'}(\ldots$$

$$\mathtt{ap}^t_{\ell}(\mathtt{clos}\,(x, e, \rho), v, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(\ldots$$
$$\text{where}\ldots$$

$$\mathtt{ap}^t_{\ell}(o, v, \sigma, \kappa) \longmapsto \mathtt{co}\,(\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)$$

Figure 5. Abstract abstract machine for ISWIM



$$[\![\_]\!] : Ex$$

$$t' = ti$$
$$[\![\mathtt{var}\,(x)]\!]_\sigma = \lambda$$
$$[\![\mathtt{lit}\,(l)]\!]_\sigma = \lambda$$
$$[\![\mathtt{lam}\,(x, e)]\!]_\sigma = \lambda$$
$$[\![\mathtt{app}^{\ell}(e_0, e_1)]\!]_\sigma = \lambda$$
$$\text{wh}$$

$$[\![\mathtt{if}^{\ell}(e_0, e_1, e_2)]\!]_\sigma = \lambda$$
$$\text{wher}$$

$$traces(e$$
$$inject(c, \xi$$
$$wn(c, \sigma) \longmapsto wn(c', \sigma'$$
$$\xi \text{ is such that}$$

$$\mathtt{co}\,(\mathtt{arg}^t_{\ell}(k, \rho, \ldots$$
$$\text{wh}$$

$$lookup(V, t) = \begin{cases} vs & \text{if } V = (t', vs):V', t' \leq t \\ vs' & \text{if } V = (t', vs):(t'', vs'):V', t' \end{cases}$$

$$\sigma \sqcup_t [a \mapsto vs] = \sigma[a \mapsto V], \Delta?$$
$$\text{where } (V, \Delta?) = \sigma(a) \sqcup_t vs$$
$$\epsilon \sqcup_t vs = (t, vs), \mathtt{tt}$$
$$(t', vs):V \sqcup_t vs' = (t', vs \sqcup vs'):V, \delta?(vs, vs') \text{ if } t'$$
$$V \sqcup_t vs = (t+1, vs^*):V, \mathtt{tt} \text{ if } vs_t \neq vs^*$$
$$\text{where } vs_t = lookup(\sigma(a), t)$$
$$vs^* = vs \sqcup vs_t$$
$$V \sqcup_t vs = V, \mathtt{ff} \text{ otherwise}$$

$$\widehat{State} \to \mathbb{N}^*) \times \wp(\widehat{State}) \times Store \times$$
$$\ldots], \sigma) = (\sigma', \delta?(vs_i, \sigma(a_i)) \vee \ldots)$$
$$\text{ere } \sigma' = \sigma \sqcup [a_i \mapsto vs_i] \sqcup \ldots$$
$$\ldots, vs') = vs' \stackrel{?}{=} vs \sqcup vs'$$
$$\sigma' = \lambda a.lookup(\sigma(a), t)$$
$$(\sigma', \Delta?) = replay(appendall(\{\xi \mid (\_, \xi) \in I\}), \sigma)$$
$$t' = \begin{cases} t+1 & \text{if } \Delta? \\ t & \text{otherwise} \end{cases}$$
$$F' = \{c \mid (c, \_) \in I, \Delta? \vee \hat{S}(c) \neq t{:}\_\}$$
$$\hat{S}' = \lambda c. \begin{cases} t{:}\hat{S}(c) & \text{if } c \in F' \\ \hat{S}(c) & \text{otherwise} \end{cases}$$

$$\varsigma = (a_f, force(\sigma, v)):\epsilon$$
$$\mathtt{co}\,(\mathtt{fun}^t_{\ell}(a_f, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ \mathtt{ap}^t_{\ell}(u, a, \kappa), (a, force(\sigma, v)):\epsilon$$
$$\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\,(\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ [\![\longmapsto]\!]_\sigma\ k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\,(\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ [\![\longmapsto]\!]_\sigma\ k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$

$$\mathtt{ap}^t_{\ell}(\mathtt{clos}\,(x, k, \rho), a, \kappa)\ [\![\longmapsto]\!]_\sigma\ k^{t'}(\sigma)(\rho', \xi, \kappa)$$
$$\text{where } \rho' = \rho[x \mapsto a]$$
$$\xi = (a, \sigma(a)):\epsilon$$

$$traces(e) = \{\mathtt{ev}^{t_0}(e, \varnothing, \varnothing, \mathtt{halt}) \longmapsto\!\!\!\!\!\rightarrow \varsigma\} \text{ where}$$

$$\varsigma \longmapsto \varsigma' \text{ defined}$$

$$\mathtt{ev}^t(\mathtt{app}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{arg}^t_\ell(e_1, \rho, a_\kappa))$$
$$\text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa)$$
$$\sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}]$$

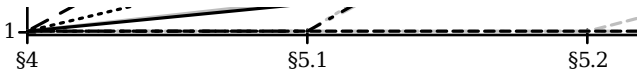
$$\mathtt{ev}^t(\mathtt{if}^\ell(e_0, e_1, e_2), \rho, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma', \mathtt{ifk}^t(e_1, e_2, \rho, a_\kappa))$$
$$\text{where } a_\kappa = allockont^t_\ell(\sigma, \kappa)$$
$$\sigma' = \sigma \sqcup [a_\kappa \mapsto \{\kappa\}]$$

$$\mathtt{co}\ (\mathtt{arg}^t_\ell(e, \rho, a_\kappa), v, \sigma) \longmapsto \mathtt{ev}^t(e, \rho, \sigma, \mathtt{fun}^t_\ell(v, a_\kappa))$$
$$\mathtt{co}\ (\mathtt{fun}^t_\ell(u, a_\kappa), v, \sigma) \longmapsto \mathtt{ap}^t_\ell(u, v, \kappa, \sigma) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{tt}, \sigma) \longmapsto \mathtt{ev}^{t'}(e_0, \rho, \sigma, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(e_0, e_1, \rho, a_\kappa), \mathtt{ff}, \sigma) \longmapsto \mathtt{ev}^{t'}(\ldots$$

$$\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, e, \rho), v, \sigma, \kappa) \longmapsto \mathtt{ev}^{t'}(\ldots$$
$$\text{where}\ \ldots$$

$$\mathtt{ap}^t_\ell(o, v, \sigma, \kappa) \longmapsto \mathtt{co}\ (\kappa, v', \sigma) \text{ if } v' \in \Delta(o, v)$$

Figure 5. Abstract abstract machine for ISWIM



$$[\![\_]\!] : Ex$$

$$t' = ti$$
$$[\![\mathtt{var}\ (x)]\!]_\sigma = \lambda^t$$
$$[\![\mathtt{lit}\ (l)]\!]_\sigma = \lambda^t$$
$$[\![\mathtt{lam}\ (x, e)]\!]_\sigma = \lambda^t$$
$$[\![\mathtt{app}^\ell(e_0, e_1)]\!]_\sigma = \lambda^t$$
$$\text{wh}$$

$$[\![\mathtt{if}^\ell(e_0, e_1, e_2)]\!]_\sigma = \lambda^t$$
$$\text{wher}$$

$$traces(e$$
$$inject(c, \xi$$
$$wn(c, \sigma) \longmapsto wn(c', \sigma'$$
$$\xi \text{ is such that}$$

$$\mathtt{co}\ (\mathtt{arg}^t_\ell(k, \rho, \ldots$$
$$\text{wh}$$

$$lookup(V, t) = \begin{cases} vs & \text{if } V = (t', vs):V', t' \leq t \\ vs' & \text{if } V = (t', vs):(t'', vs'):V', t' \end{cases}$$

$$\sigma \sqcup \ldots \quad \ldots \Delta?$$
$$\text{wh}$$

$$(t', vs):V \sqcup_t vs' = (t', vs \sqcup \ldots) \text{ if } t'$$
$$V \sqcup_t vs = (t+1, vs^*):V, \mathtt{tt} \text{ if } vs_t \neq vs^*$$
$$\text{where } vs_t = lookup(\sigma(a), t)$$
$$vs^* = vs \sqcup vs_t$$
$$V \sqcup_t vs = V, \mathtt{ff} \text{ otherwise}$$

$$\ldots \longrightarrow \mathbb{N}^*) \times \wp(\widehat{State}) \times Store \times$$
$$\ldots], \sigma) = (\sigma', \delta?(vs_i, \sigma(a_i)) \vee \ldots)$$
$$\text{ere } \sigma' = \sigma \sqcup [a_i \mapsto vs_i] \sqcup \ldots$$
$$\ldots, vs') = vs' \stackrel{?}{=} vs \sqcup vs'$$
$$\sigma' = \lambda a.lookup(\sigma(a), t)$$
$$(\sigma', \Delta?) = replay(appendall(\{\xi \mid (\_, \xi) \in I\}), \sigma)$$
$$t' = \begin{cases} t+1 & \text{if } \Delta? \\ t & \text{otherwise} \end{cases}$$
$$F' = \{c \mid (c, \_) \in I, \Delta? \vee \hat{S}(c) \neq t:\_\}$$
$$\hat{S}' = \lambda c. \begin{cases} t':\hat{S}(c) & \text{if } c \in F' \\ \hat{S}(c) & \text{otherwise} \end{cases}$$

$$\mathtt{co}\ (\mathtt{fun}^t_\ell(a_f, a_\kappa), v)\ [\![\longmapsto]\!]_\sigma\ \mathtt{ap}^t_\ell(u, a, \kappa), (a, force(\sigma, v)):\epsilon$$
$$\text{if } u \in \sigma(a_f), \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{tt})\ [\![\longmapsto]\!]_\sigma\ k^t_0(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$
$$\mathtt{co}\ (\mathtt{ifk}^t(k_0, k_1, \rho, a_\kappa), \mathtt{ff})\ [\![\longmapsto]\!]_\sigma\ k^t_1(\sigma)(\rho, \epsilon, \kappa) \text{ if } \kappa \in \sigma(a_\kappa)$$

$$\mathtt{ap}^t_\ell(\mathtt{clos}\ (x, k, \rho), a, \kappa)\ [\![\longmapsto]\!]_\sigma\ k^{t'}(\sigma)(\rho', \xi, \kappa)$$
$$\text{where } \rho' = \rho[x \mapsto a]$$
$$\xi = (a, \sigma(a)):\epsilon$$

Simple transform

Simple transform

Simple transform

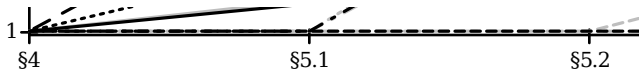
Ugly mess

Simple transform

Simple transform

Simple transform

Figure 5. Abstract abstract machine for ISWIM



Simple transform

Simple transform

Ugly mess

Simple transform

Human compiler

"What we hope ever to do with ease,

we must first learn to do with diligence."

~Samuel Johnson

I made a language

I made a language

Pattern ↦ *Expression [Side-conditions]*

I made a language

Pattern ↦ *Expression [Side-conditions]*

Equality is built-in

I made a language

Pattern ↦ *Expression [Side-conditions]*

Equality is built-in

First-order metafunctions

I made a language

Pattern ↦ *Expression [Side-conditions]*

Equality is built-in

First-order metafunctions

All allocation is mediated

I made a language

Pattern ↦ *Expression [Side-conditions]*

Equality is built-in

First-order metafunctions

All allocation is mediated

Any allocation strategy is sound

I made a language

Pattern ↦ *Expression [Side-conditions]*

Equality is built-in

First-order metafunctions

All allocation is mediated

Any allocation strategy is sound

Fresh allocation ensures completeness

My language in a slide

My language in a slide

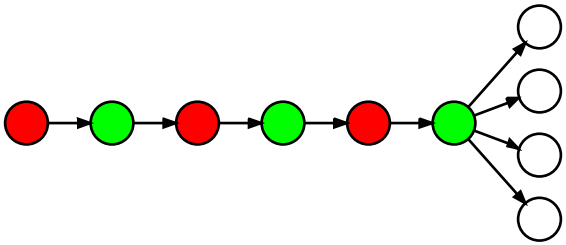
My language in a slide

My language in a slide

My language in a slide

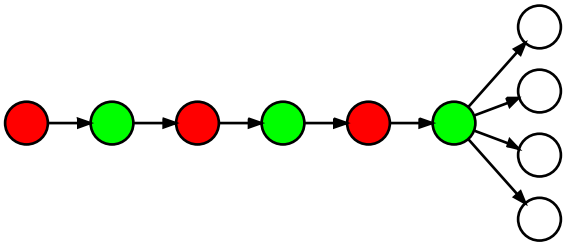
My language in a slide

Delay

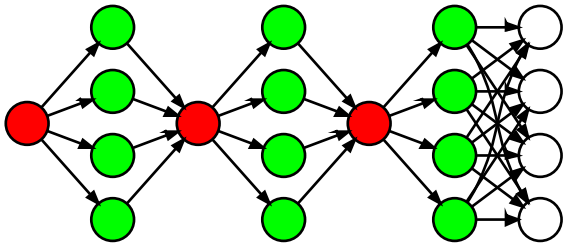


My language in a slide

Delay

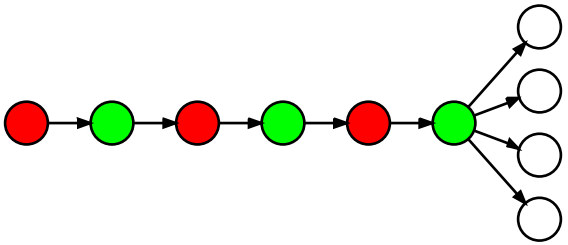


Resolve

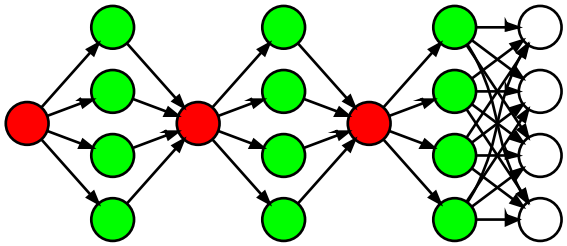


My language in a slide

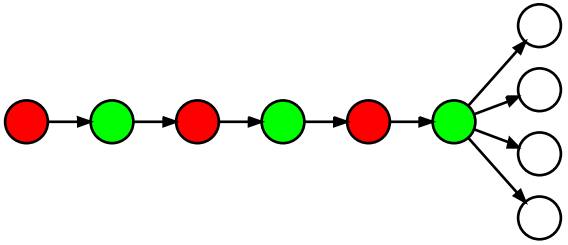
Delay



Resolve

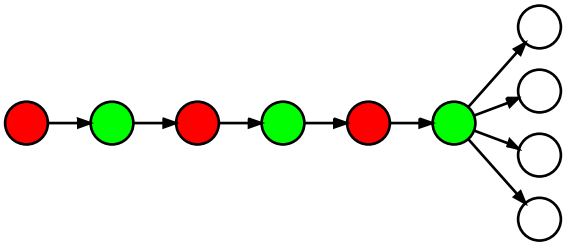


Deref

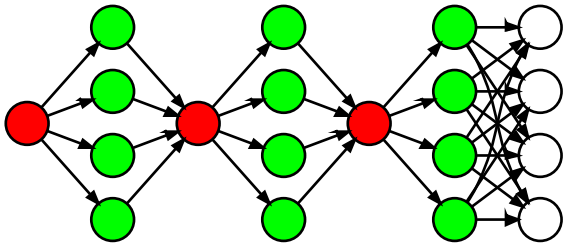


My language in a slide

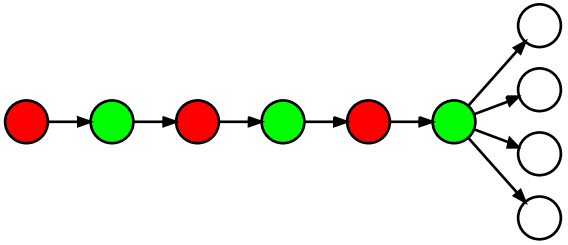
Delay



Resolve



Deref



Different behavior for fresh addresses

Definition: An abstract machine is

A finite set of rules

An address allocation function

A compound term allocation function

A collection of metafunctions

A function

An initial value

Todo:

Language design

Todo:

Related work

Todo:

Conclusion/future work

Todo:

What was proposed, what did