Galois Theory of Algorithms

Noson S. Yanofsky
Brooklyn College, The Graduate Center, CUNY

The Kolchin Seminar in Differential Algebra
The Graduate Center, CUNY
September 3, 2010

Motivation: What is an Algorithm?

In Corman, Leiserson, Rivest, and Stein's *Introduction to Algorithms, 2nd Ed.* Section 1.1: a definition of an algorithm:

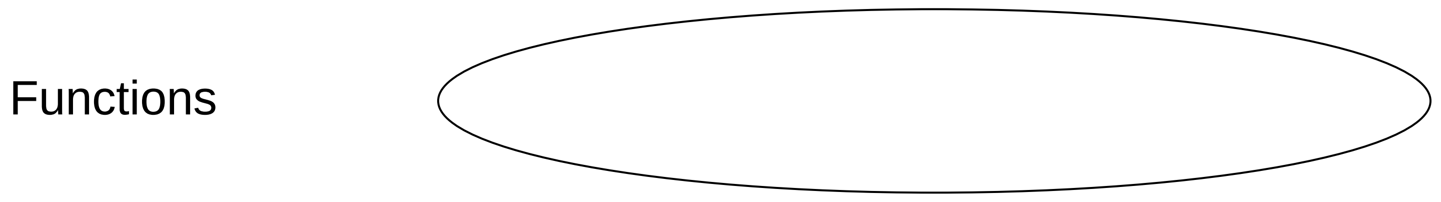
Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

- "Informally"?!?
- "well-defined"?
- "procedure"?

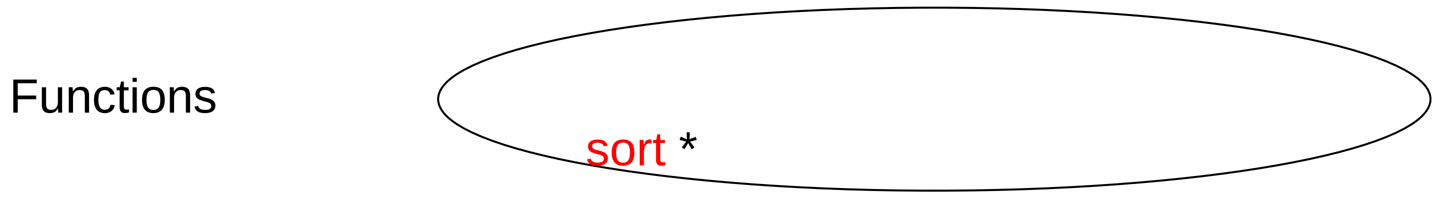
Motivation: Other Peoples Definitions

- Turing, Gurevich and others say “An algorithm is a program in this language/system/machine.”
- This is not the way we use the word. A program is an implementation or realization of an algorithm.
- A professor teaches an algorithm to a class and assigns them to program it. They all come back with programs. They are not all the same program. They are all the same algorithm.

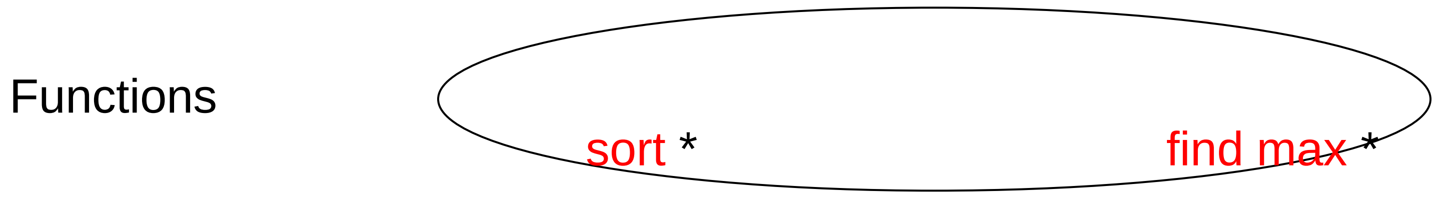
Definition of an Algorithm



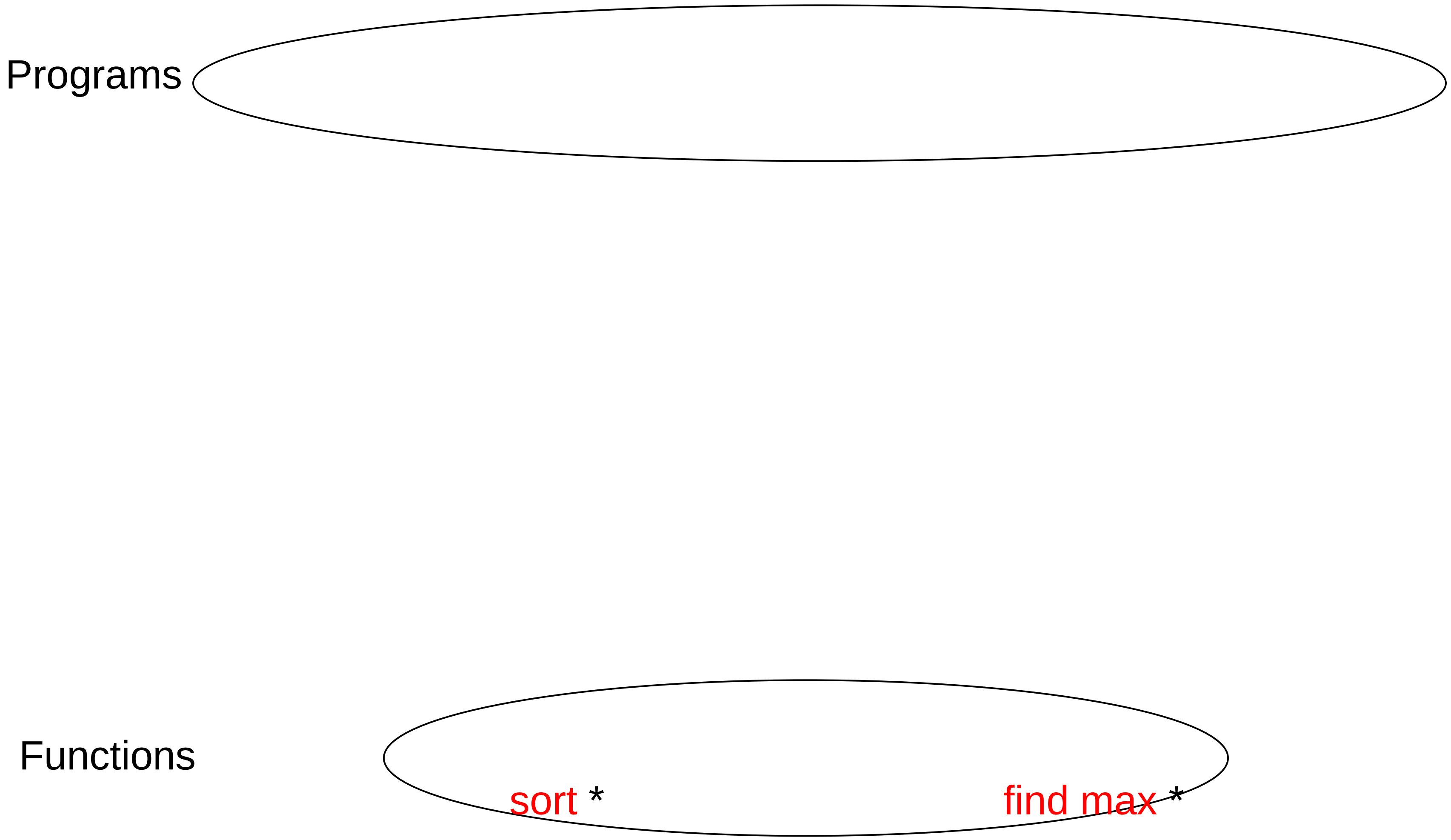
Definition of an Algorithm



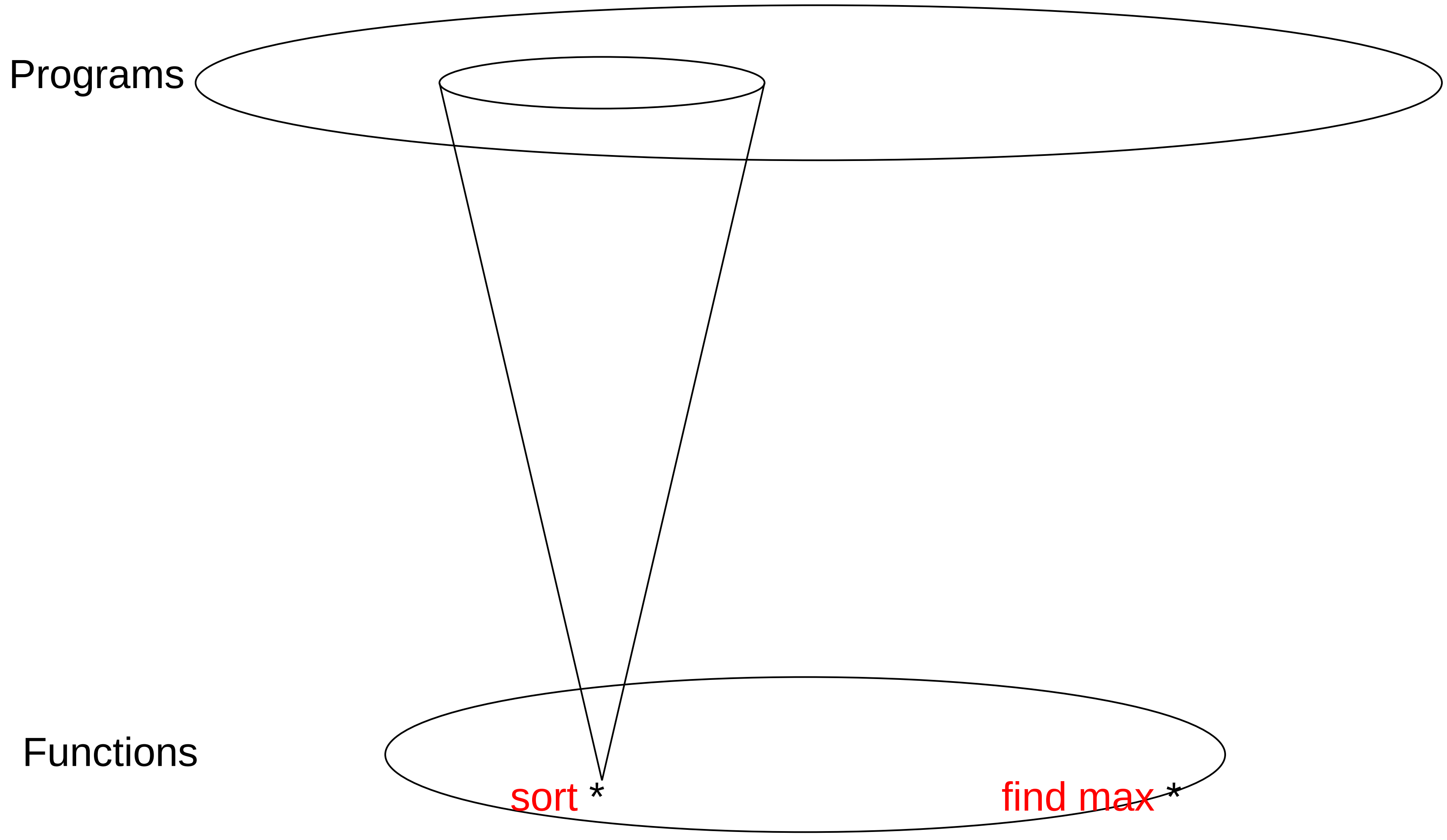
Definition of an Algorithm



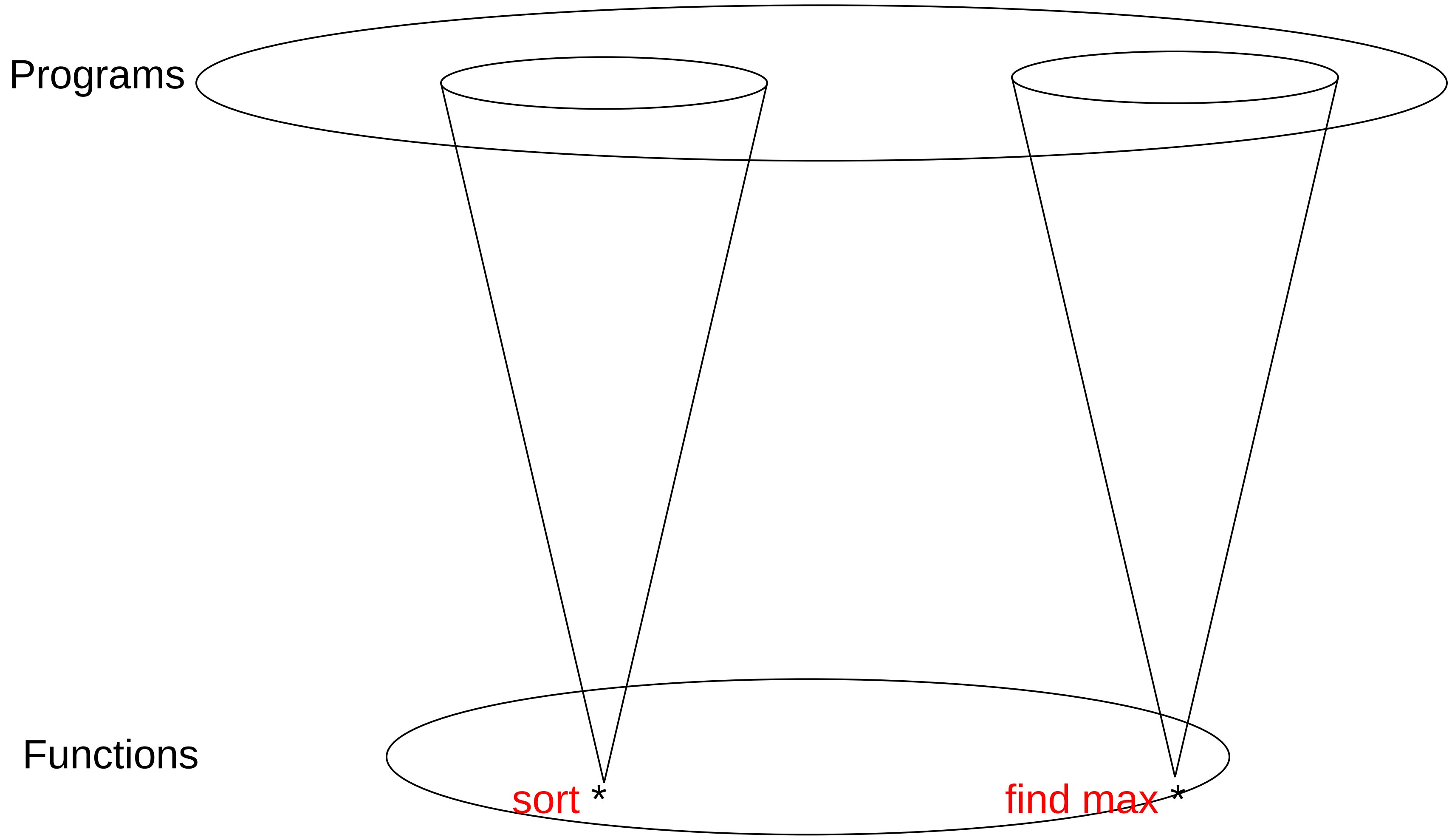
Definition of an Algorithm



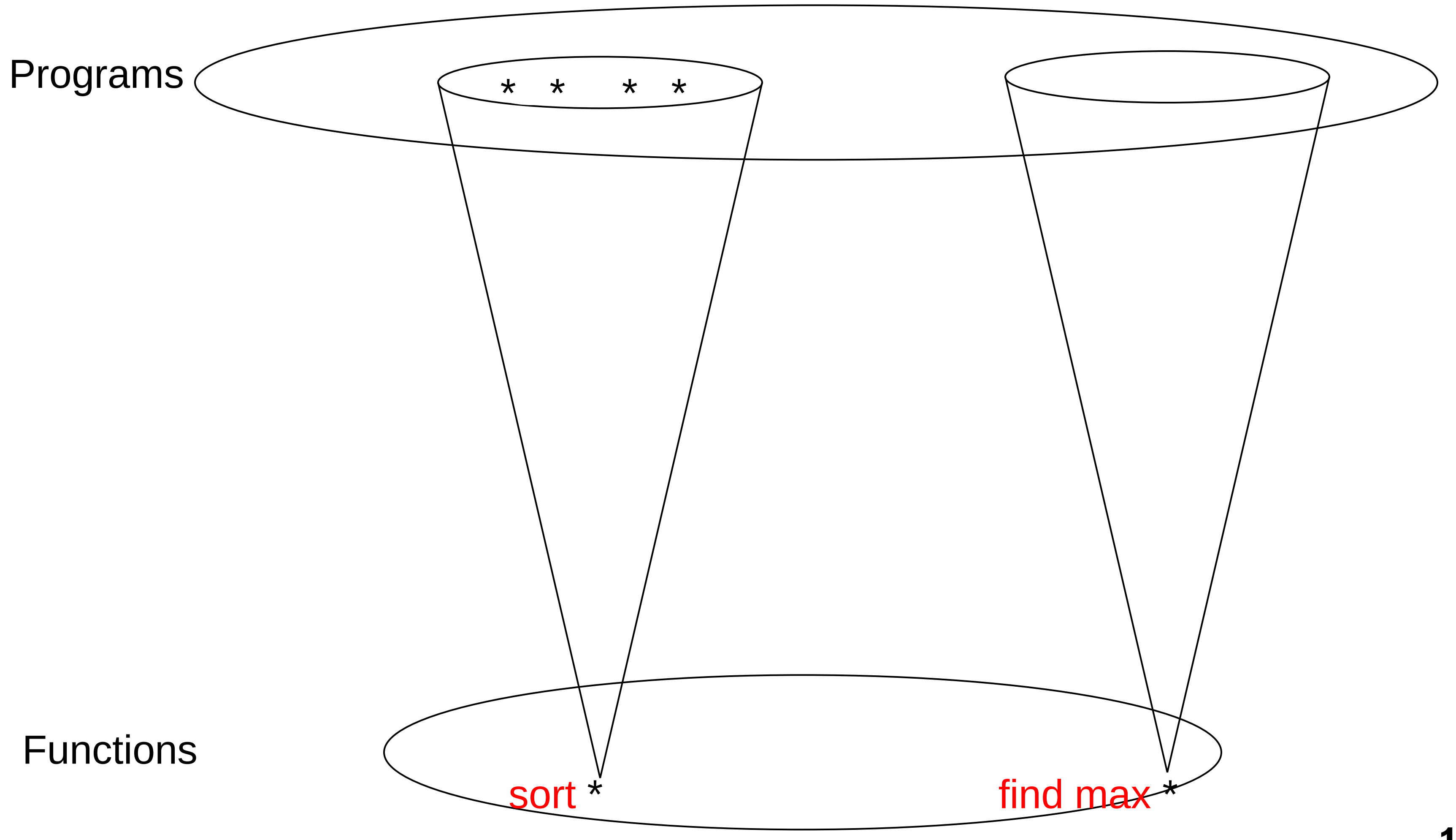
Definition of an Algorithm



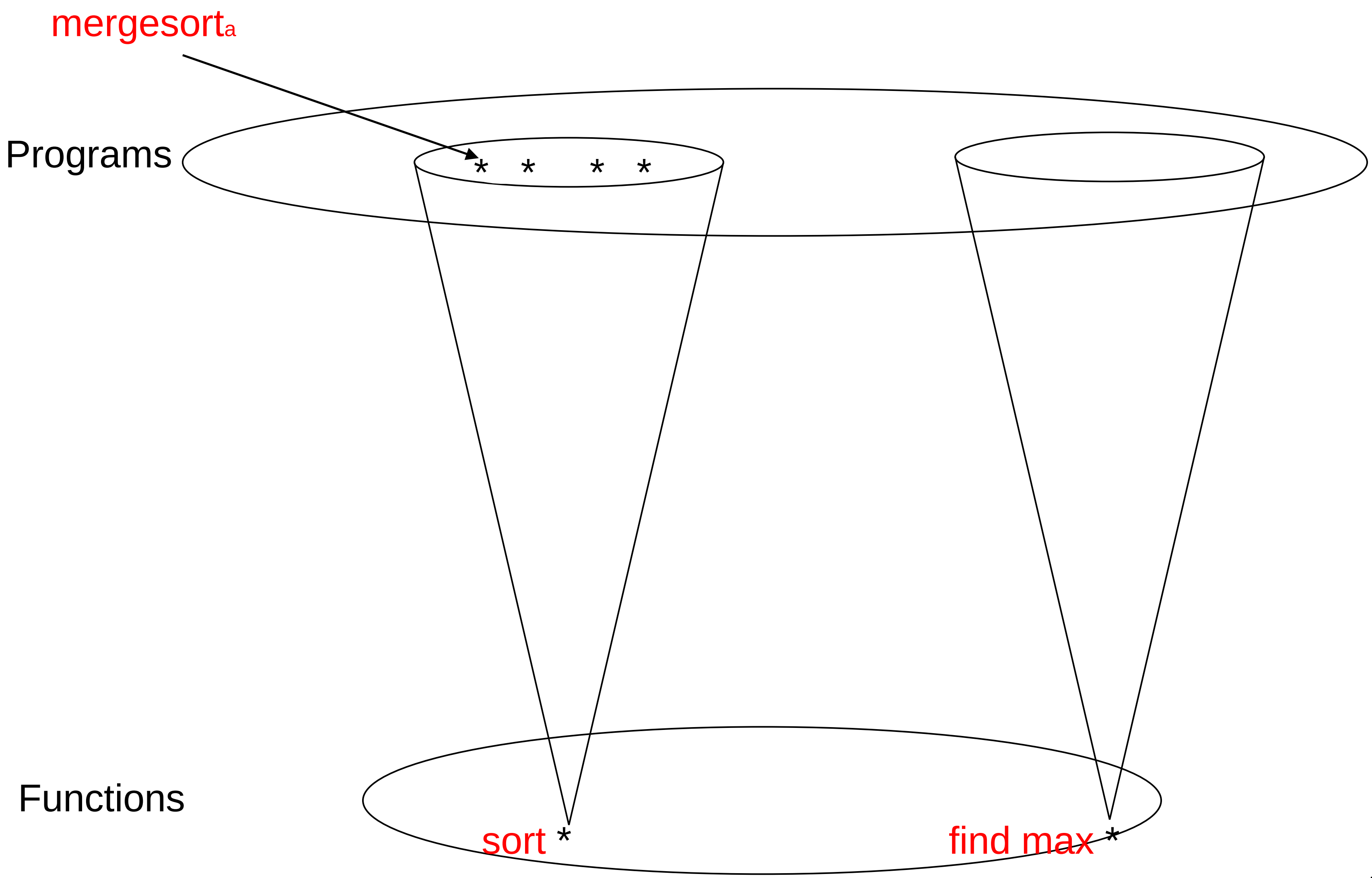
Definition of an Algorithm



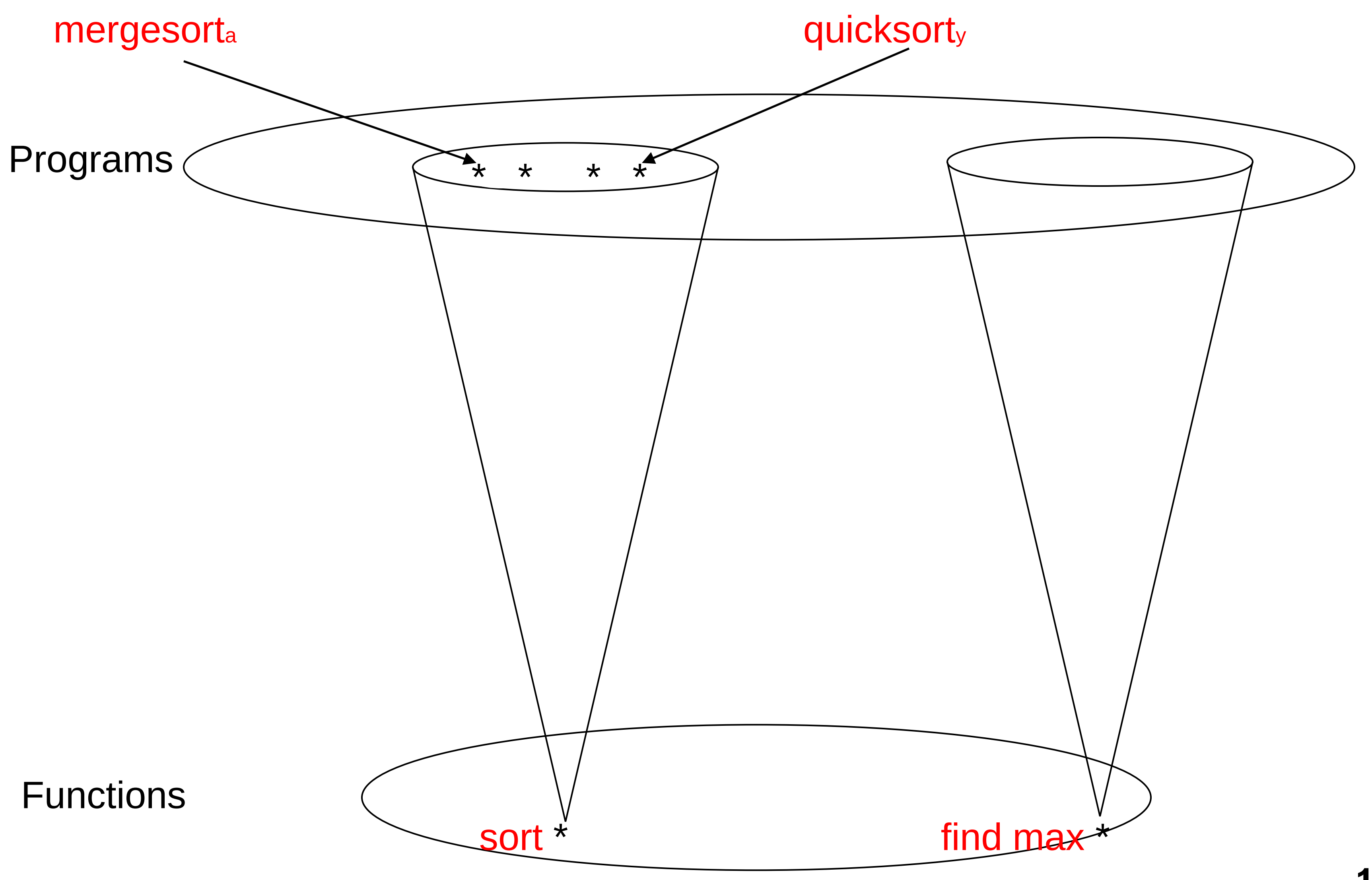
Definition of an Algorithm



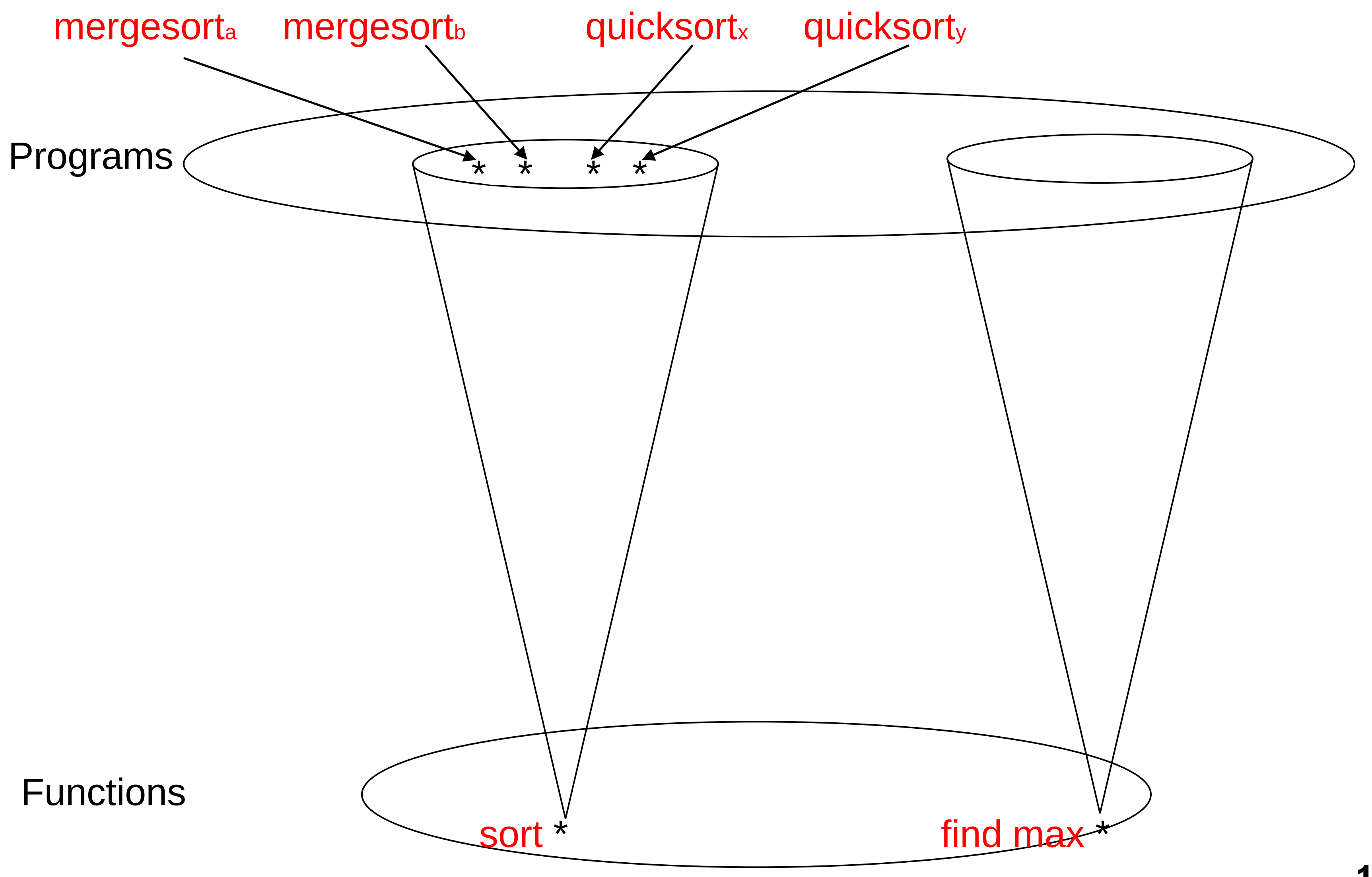
Definition of an Algorithm



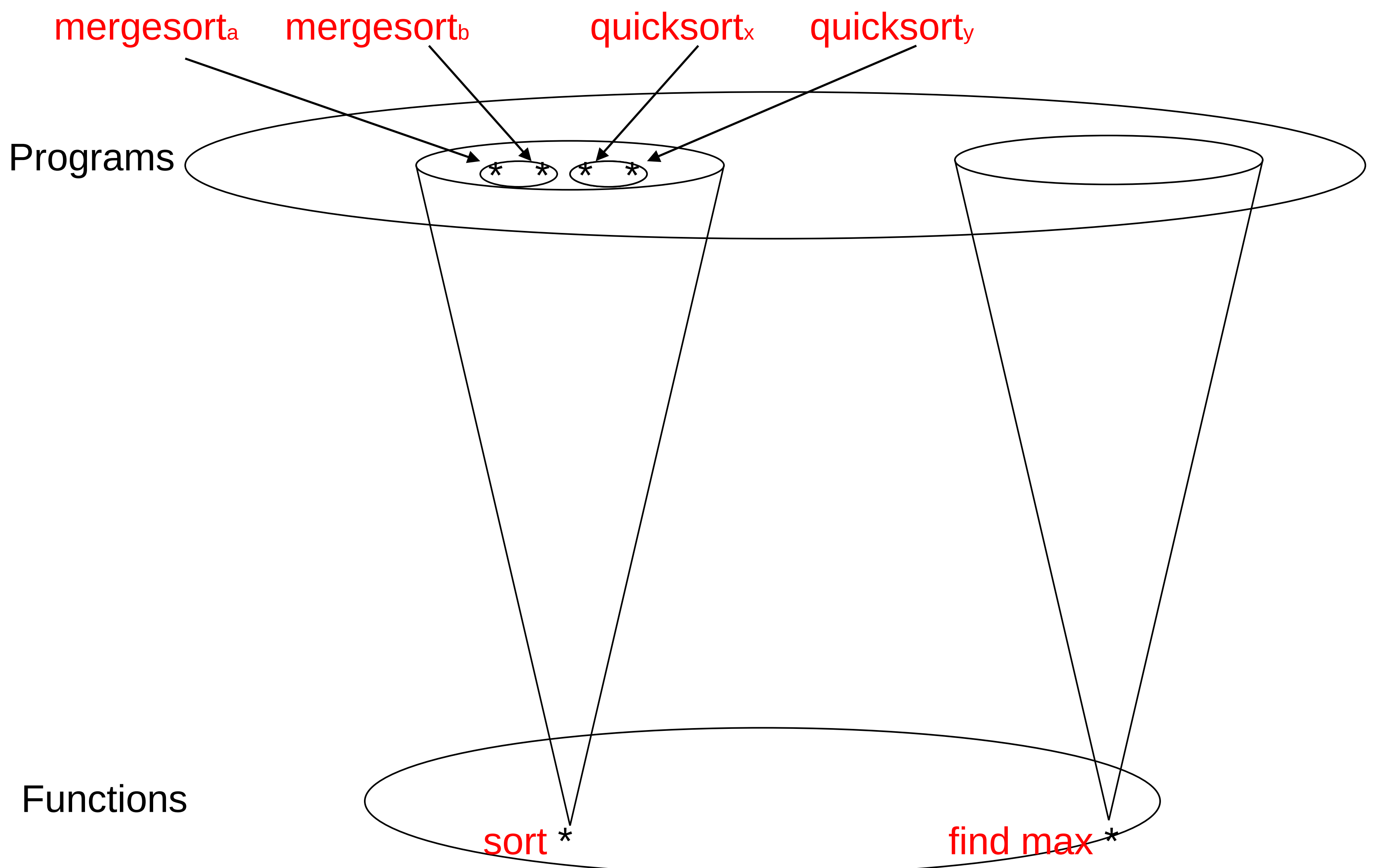
Definition of an Algorithm



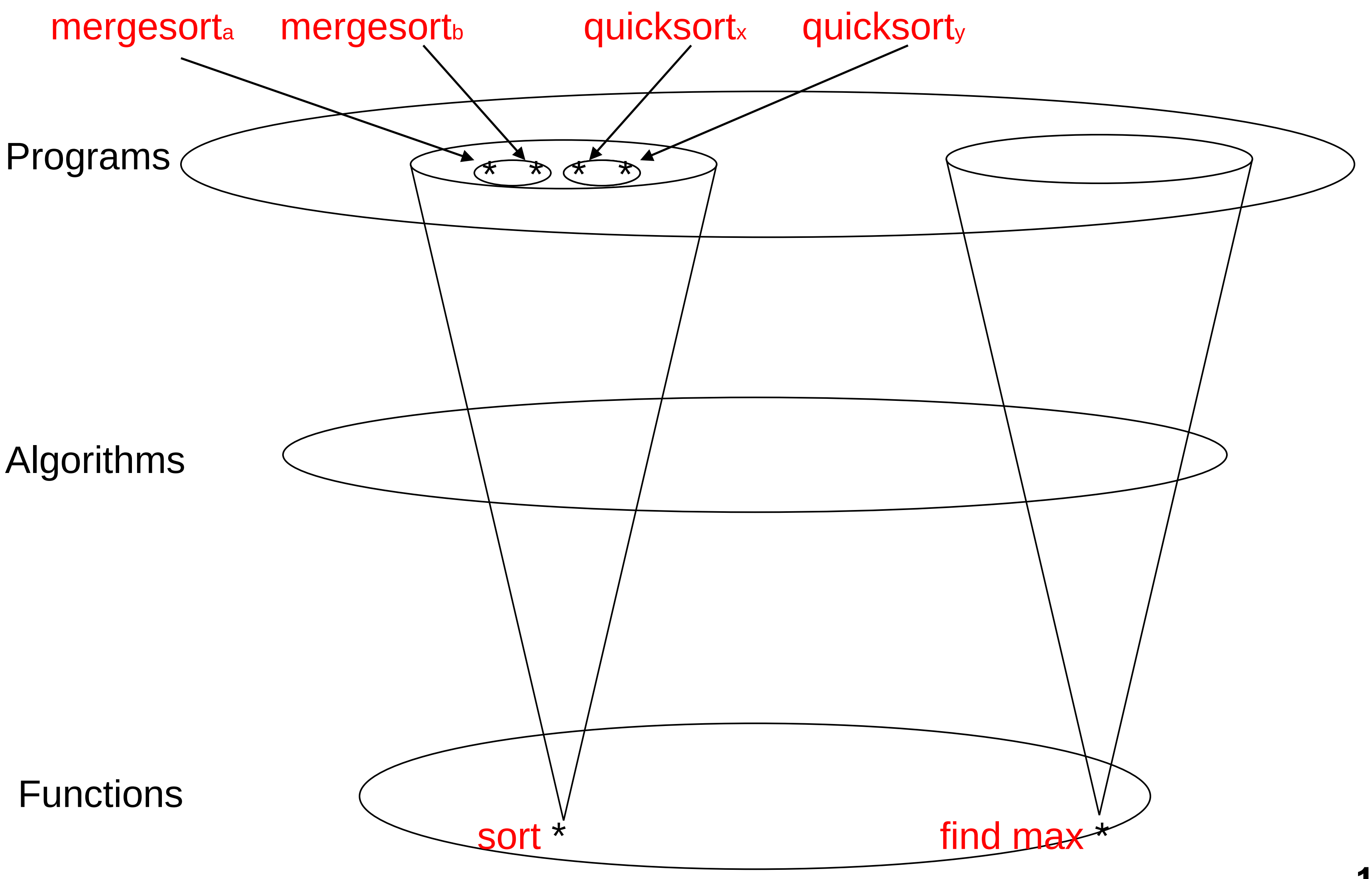
Definition of an Algorithm



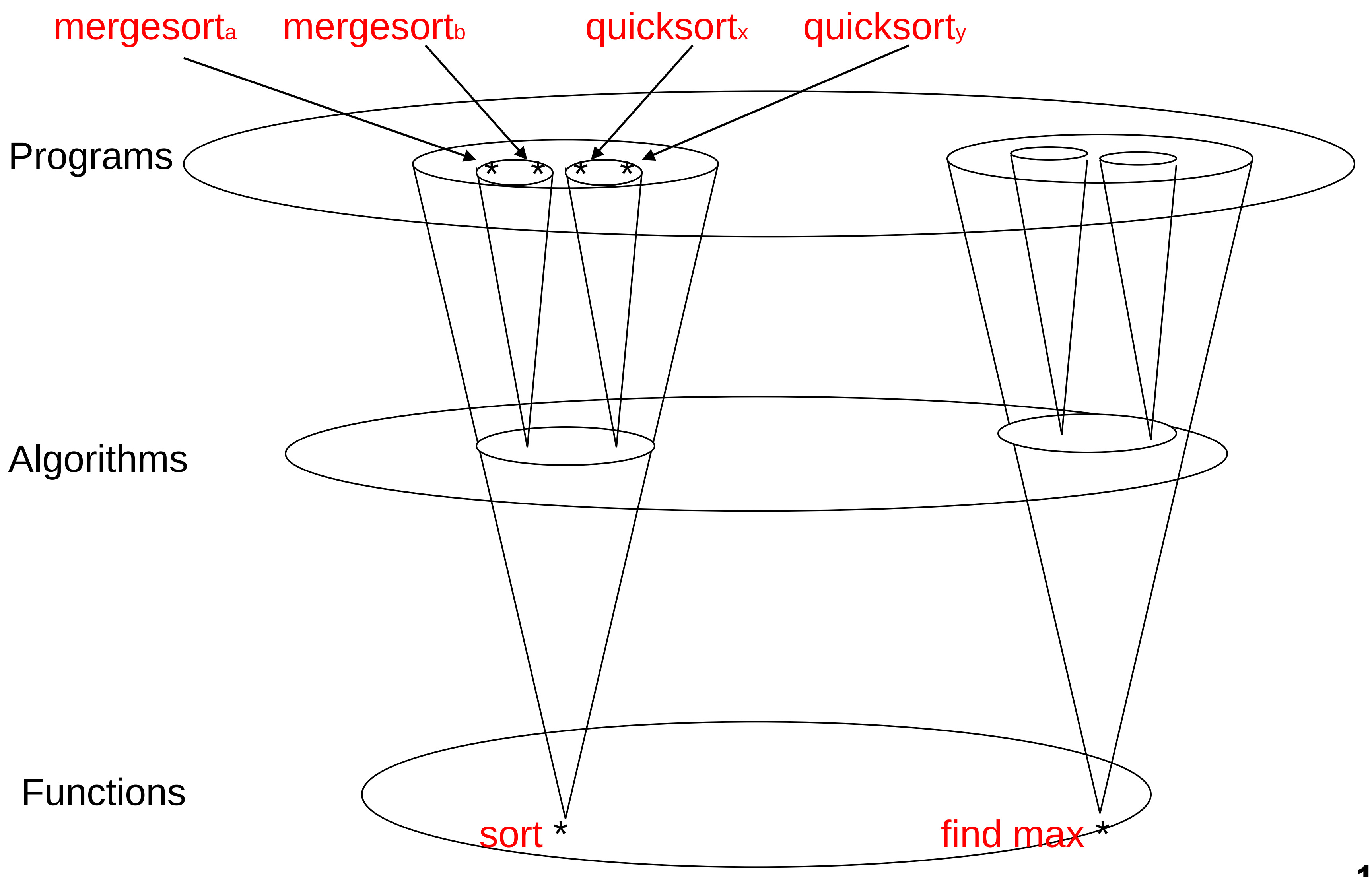
Definition of an Algorithm



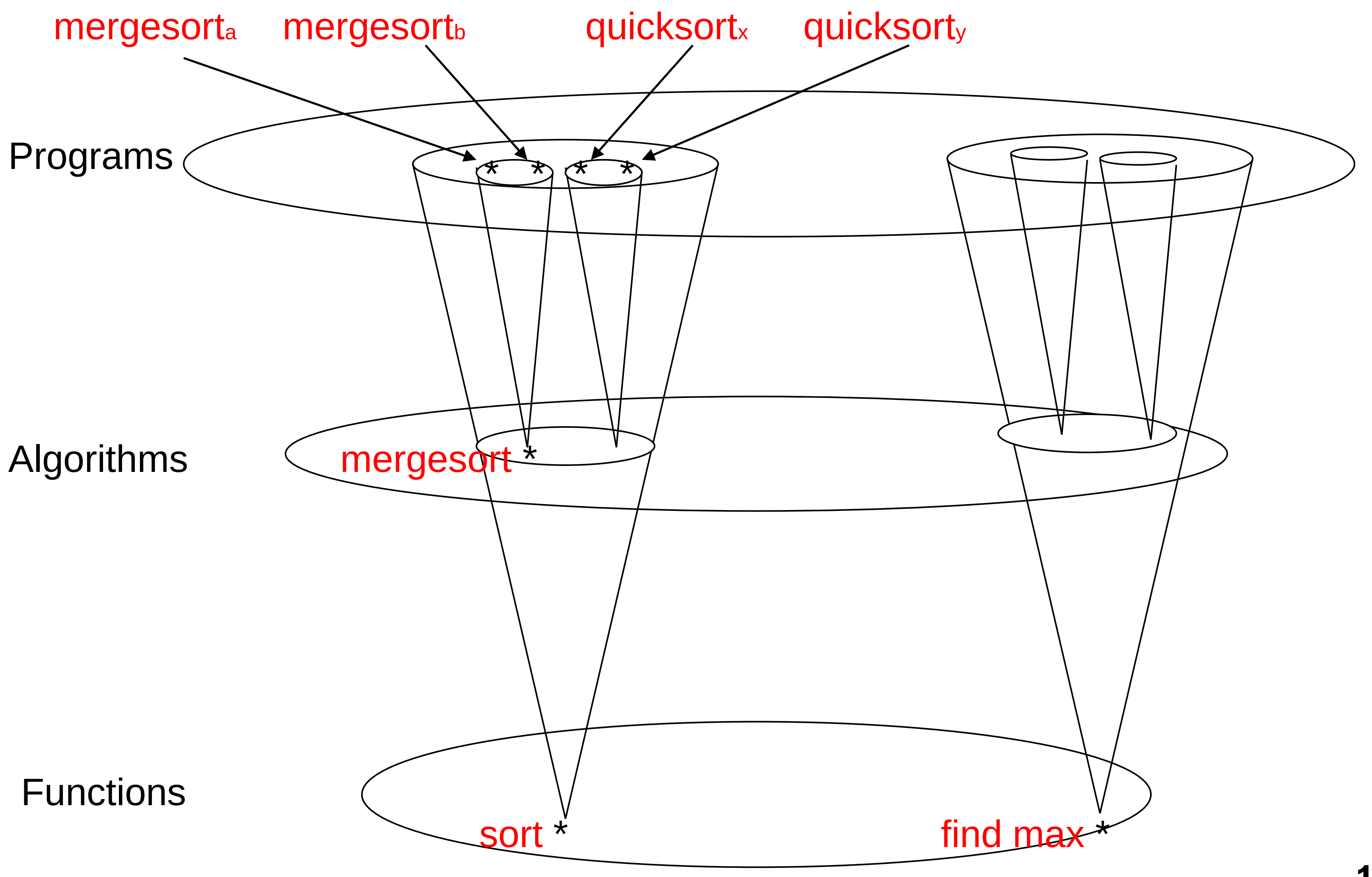
Definition of an Algorithm



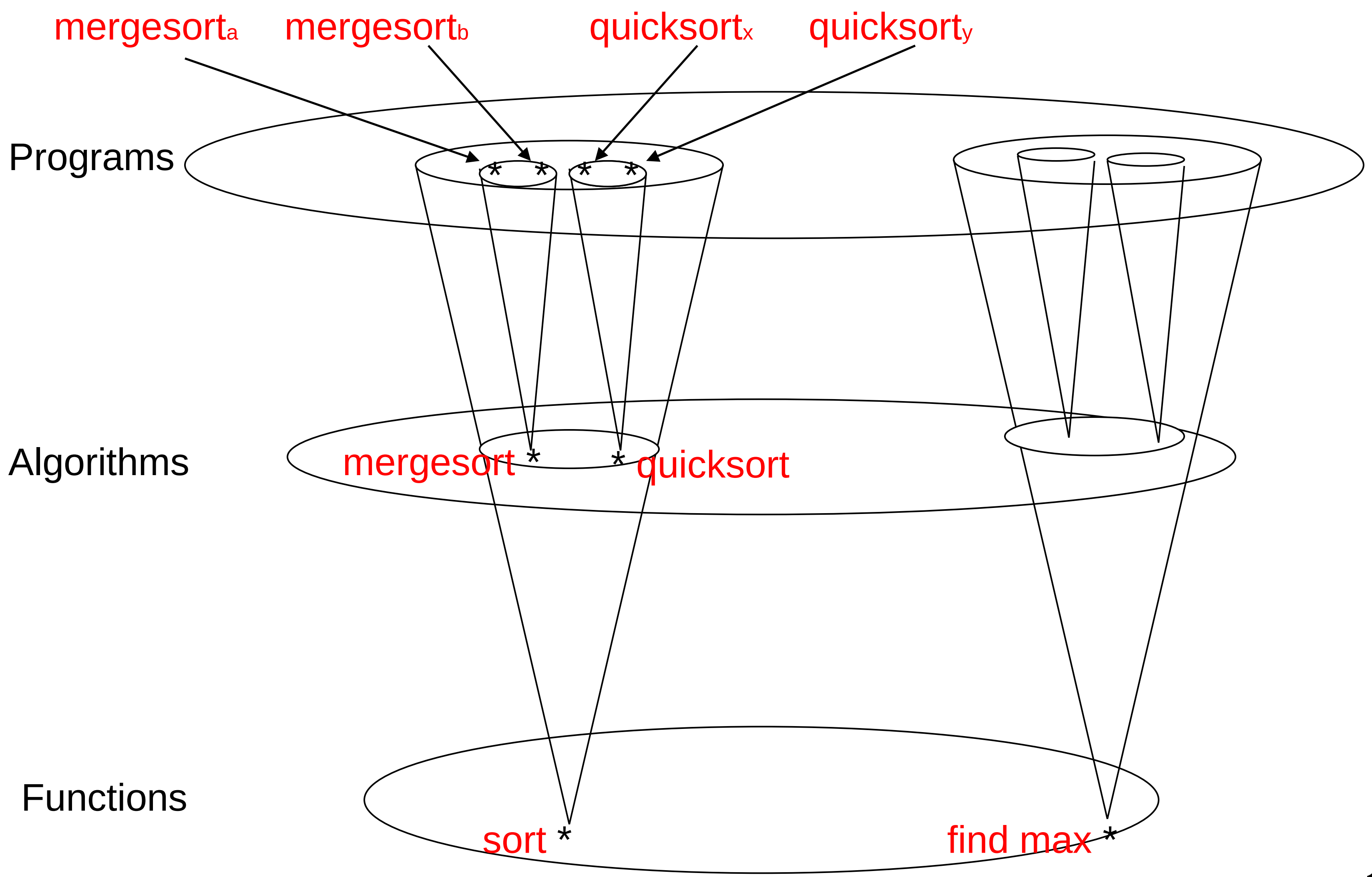
Definition of an Algorithm



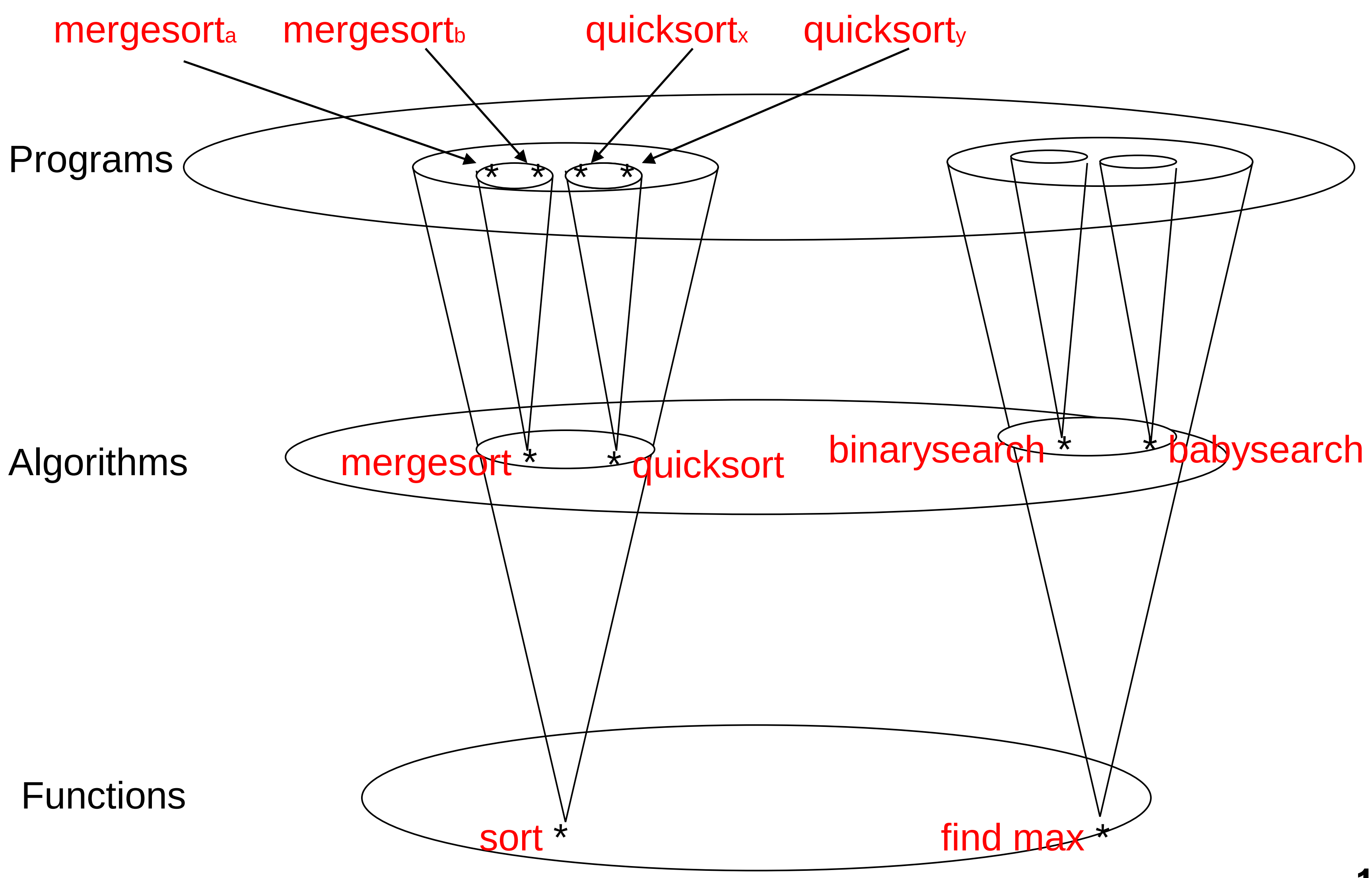
Definition of an Algorithm



Definition of an Algorithm



Definition of an Algorithm



Definition of an Algorithm

An algorithm is defined to be an equivalence class of programs. It is a set of programs that implement that algorithm.

We look at the set of all programs and partition them. Two programs are deemed equivalent if they are “essentially” the same. A program is a representative of an algorithm, i.e. a representative of the equivalence class.

Definition of an Algorithm: Analogies

- Frege: The number 42 is an equivalence class of all finite sets that have 42 elements in it. Look at all finite sets and say two are equivalent if there exists a one-to-one correspondence between them. Every set with 42 elements is an “implementation” or “realization” of 42.

- The rational number 3/5 is an equivalence class of fractions like 3/5. Look at all pairs of numbers (x,y) and make the following equivalence relation.

  (x,y) ~ (x’,y’) If and only if xy’=yx’.

  6/10 and 60/100 are other “implementation” or “realizations” of 3/5.

- “An object is the sum of all its descriptions.”

Big Picture

$$\begin{tikzcd}
\text{Programs} \arrow[d] \\
\text{Algorithms} \arrow[d] \\
\text{Functions}
\end{tikzcd}$$

The Programs

Descriptions of Primitive Recursive functions. $f : N^k \to N$

- Initial functions:
  - null n(x)=0
  - successor s(x)=x+1
  - projections $\pi_i^n(x_1, x_2, ..., x_n) = x_i$

The Programs

Descriptions of Primitive Recursive functions. $f : N^k \to N$

- Composition. h(x)=(f o g)(x)=f(g(x))
- Bracket. Given $f : N^m \to N^a$ and $g : N^m \to N^b$

  $h = \langle f, g \rangle : N^m \to N^{a+b}$....$h(x) = \langle f(x), g(x) \rangle$
- Recursion. Given $f : N^n \to N^a$ and

  $g : N^{n+a+1} \to N^a$

We get $h : N^{n+1} \to N^a$

The Programs --- Recursion

$$h(x_1, x_2, \ldots, x_n, 0) = f(x_1, x_2, \ldots, x_n)$$

$$h(x_1, x_2, \ldots, x_n, t+1) = g(x_1, x_2, \ldots, x_n, h(x_1, x_2, \ldots, x_n, t), t)$$

The Programs --- Examples.

Addition:
$$m + 0 = m$$
$$m + (n+1) = s(m+n)$$

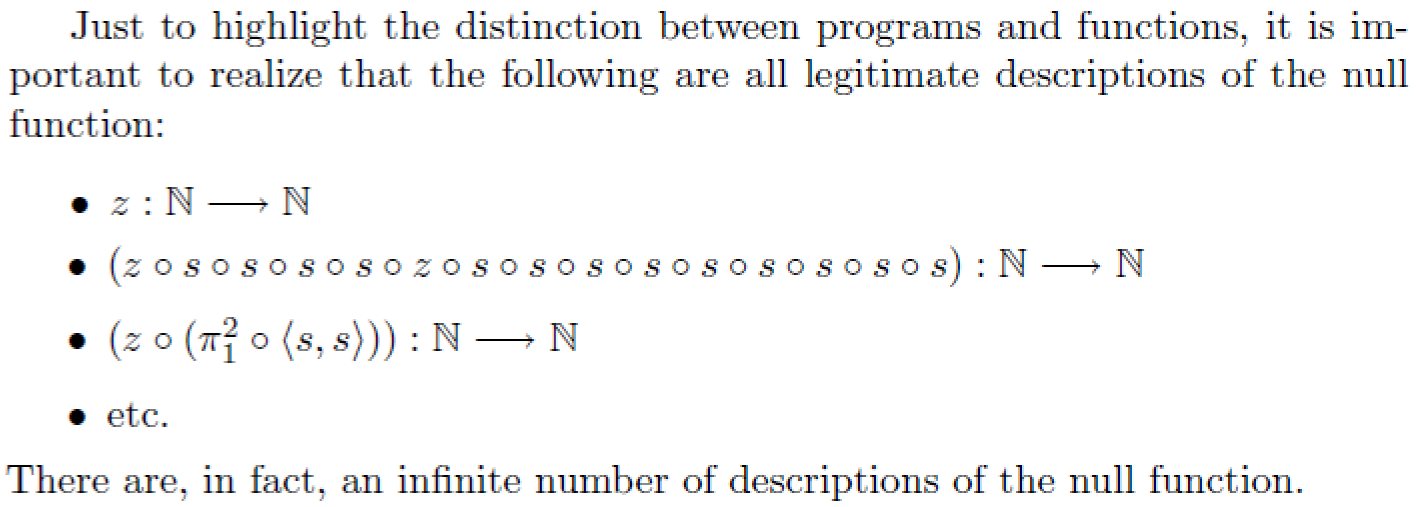
Multiplication:
$$m * 0 = 0$$
$$m * (n+1) = m * n + m$$

Predecessor:
$$P(0) = 0$$
$$P(t+1) = t$$

Subtraction: m-0=m

m-(n+1)=P(m-n)

The Programs --- Examples

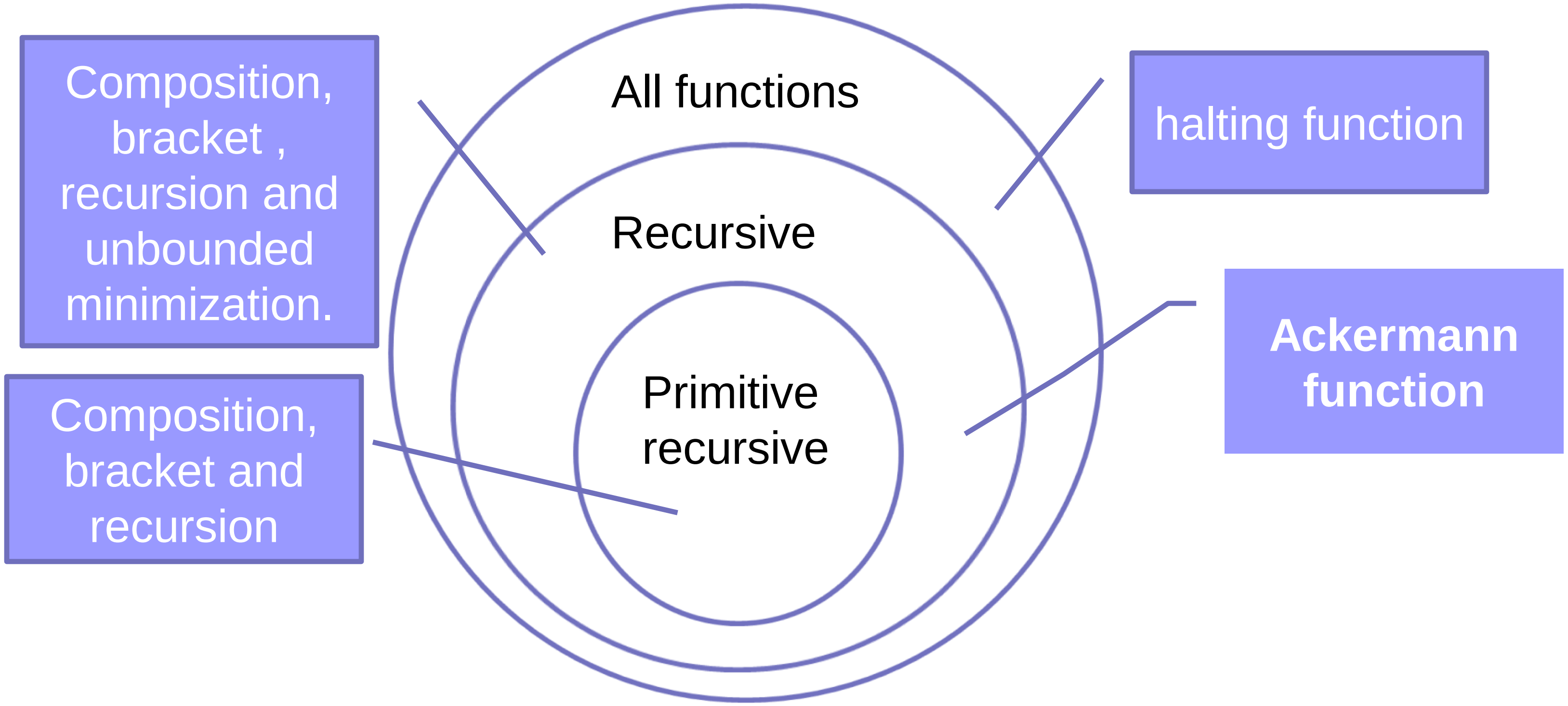


Just to highlight the distinction between programs and functions, it is important to realize that the following are all legitimate descriptions of the null function:

- $z : \mathbb{N} \longrightarrow \mathbb{N}$
- $(z \circ s \circ s \circ s \circ s \circ z \circ s \circ s \circ s \circ s \circ s \circ s \circ s \circ s \circ s) : \mathbb{N} \longrightarrow \mathbb{N}$
- $(z \circ (\pi_1^2 \circ \langle s, s \rangle)) : \mathbb{N} \longrightarrow \mathbb{N}$
- etc.

There are, in fact, an infinite number of descriptions of the null function.

The Programs --- The context


Composition, bracket , recursion and unbounded minimization.
All functions
halting function
Recursive
Ackermann function
Primitive recursive
Composition, bracket and recursion

The Programs

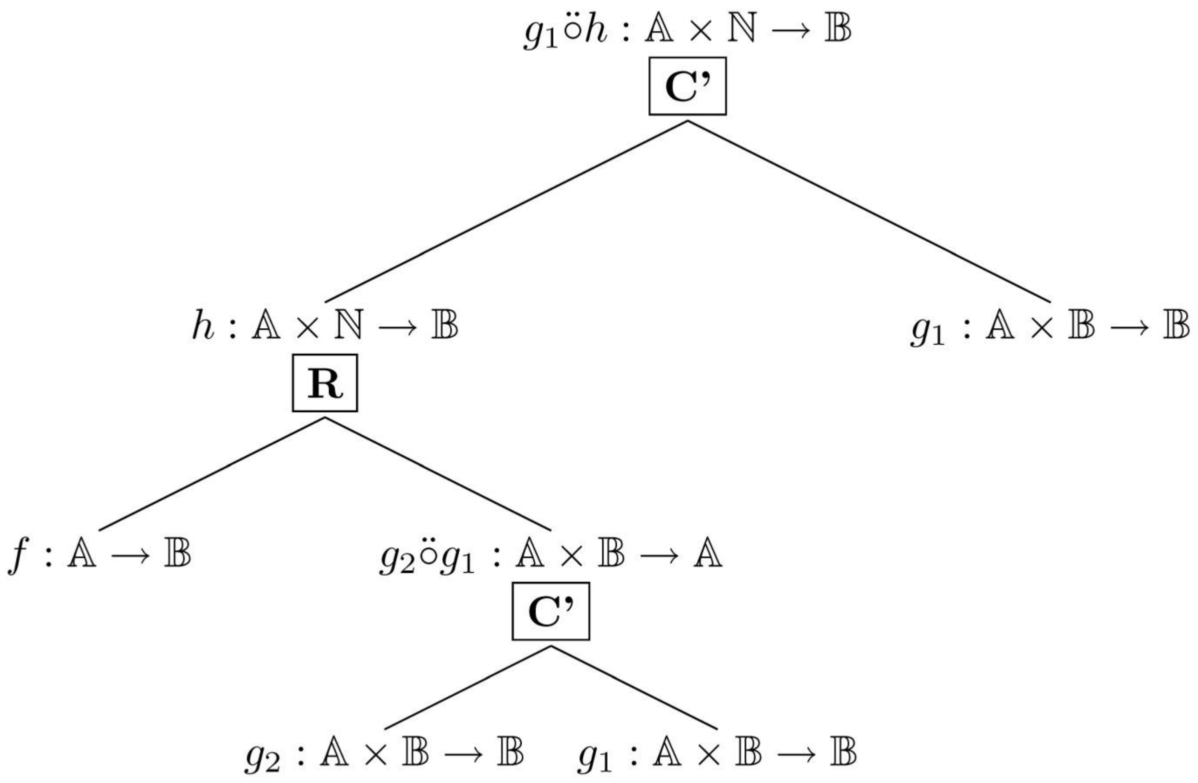
Descriptions are trees whose leaves are decorated by initial functions and whose internal nodes are colored by C, R, B.

$$\begin{tikzcd}
& g \circ f : \mathbb{A} \to \mathbb{C} & \\
& \boxed{\mathbf{C}} \arrow[dl, no head] \arrow[dr, no head] & \\
f : \mathbb{A} \to \mathbb{B} & & g : \mathbb{B} \to \mathbb{C}
\end{tikzcd}$$

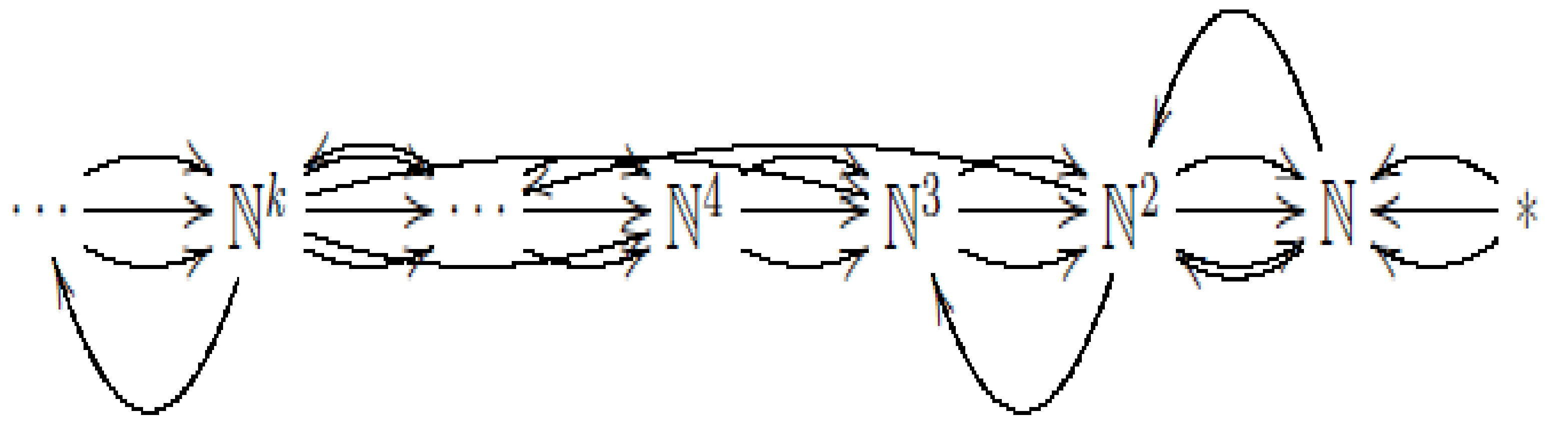
$$\begin{tikzcd}
& h = f \sharp g : \mathbb{A} \times \mathbb{N} \to \mathbb{B} & \\
& \boxed{\mathbf{R}} \arrow[dl, no head] \arrow[dr, no head] & \\
f : \mathbb{A} \to \mathbb{B} & & g : \mathbb{A} \times \mathbb{B} \to \mathbb{B}
\end{tikzcd}$$

$$\begin{tikzcd}
& \langle f, g \rangle : \mathbb{A} \to \mathbb{B} \times \mathbb{C} & \\
& \boxed{\mathbf{B}} \arrow[dl, no head] \arrow[dr, no head] & \\
f : \mathbb{A} \to \mathbb{B} & & g : \mathbb{A} \to \mathbb{C}
\end{tikzcd}$$

The Programs: An Example


$g_1 \ddot{\circ} h : \mathbb{A} \times \mathbb{N} \to \mathbb{B}$
C'
$h : \mathbb{A} \times \mathbb{N} \to \mathbb{B}$
R
$g_1 : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$
$f : \mathbb{A} \to \mathbb{B}$
$g_2 \ddot{\circ} g_1 : \mathbb{A} \times \mathbb{B} \to \mathbb{A}$
C'
$g_2 : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$
$g_1 : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$

The Programs as a graph



Structures: P.R. Programs

- The set of P.R. programs form a directed graph with extra structure.
  - Vertices are powers of natural numbers: N, N^2, N^3,…
  - Edges are P.R. programs from N^m to N^n.
  - There is a composition of edges: compose one P.R. program with another. Not associative.
  - For every N^n there is an identity edge. But it does not act like a unit.
  - There is a product function (not a functor): f and g go to <f,g>. (Bracket)
  - There is a recursion function (not a functor): f and g go to f#g. (Recursion.)

The Equivalences

- Composition is Associative: (PoQ)oR ~ Po(QoR)
- Identity programs as Units: Id o P ~ P ~ P o Id
- For unrelated processes:

```
      Process1          ~          Process2
      Process2                     Process1
```

The Equivalences

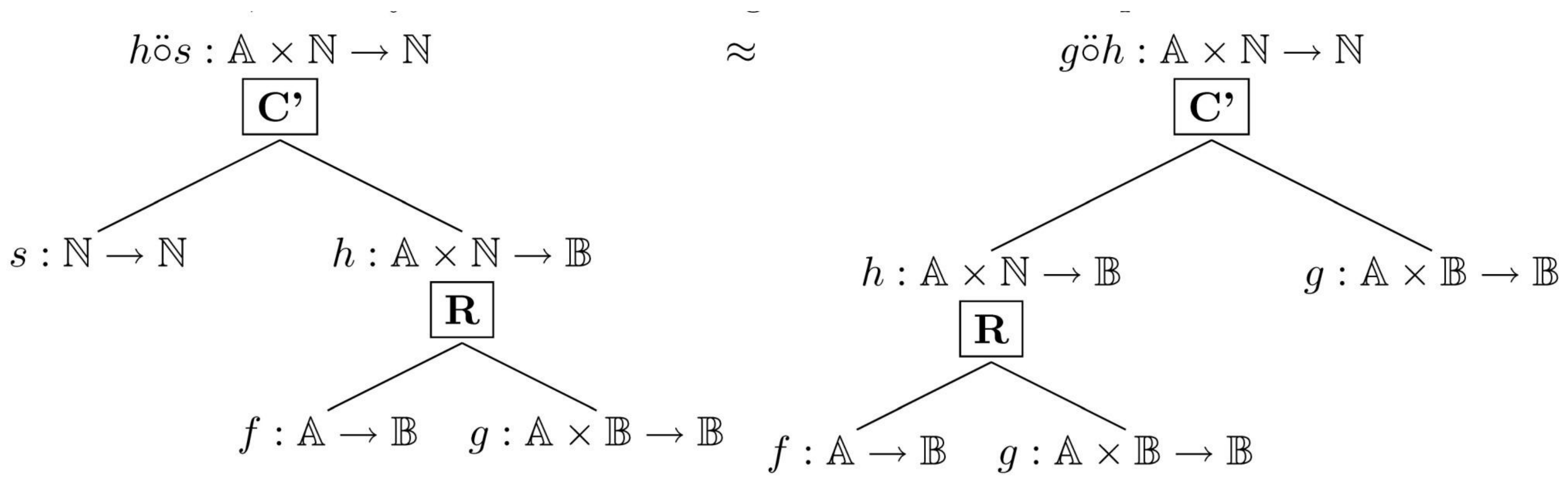
- For unrelated processes:

```
For i=1 to n                         For i=1 to n
        Process1              ~              Process1
        Process2                     For i=1 to n
                                             Process2
```

-
  ```
  For i=1 to n                         For i=1 to n-1
          Process1              ~              Process1
                                       Process1
  ```

- Others

The Equivalences: An Example



$h \circ s : \mathbb{A} \times \mathbb{N} \to \mathbb{N}$
$\approx$
$g \circ h : \mathbb{A} \times \mathbb{N} \to \mathbb{N}$
C'
$s : \mathbb{N} \to \mathbb{N}$
$h : \mathbb{A} \times \mathbb{N} \to \mathbb{B}$
R
$f : \mathbb{A} \to \mathbb{B}$
$g : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$
C'
$h : \mathbb{A} \times \mathbb{N} \to \mathbb{B}$
$g : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$
R
$f : \mathbb{A} \to \mathbb{B}$
$g : \mathbb{A} \times \mathbb{B} \to \mathbb{B}$

Structures: P.R. Algorithms

- The set of P.R. algorithms form a category with extra structure.
  - Objects are powers of natural numbers: N, N^2, N^3,…
  - Morphisms are algorithms from N^m to N^n.
  - Composition of edges: compose algorithms. Associative.
  - For every N^n there is an identity morphism. It acts like a unit.
  - There is a product bifunctor: f and g go to <f,g>. (Bracket)
  - The category has a weak natural number object. (Recursion.)
- Main Theorem: The category of P.R. algorithms is the initial object in the 2-category of categories with products and weak natural number objects.
- (The other categories with such structure correspond to algorithms with oracles. Other information is added.)

Natural Number Object

$$\begin{tikzcd} * \arrow[r, "0"] & \mathbb{N} \arrow[r, "s"] & \mathbb{N} \end{tikzcd}$$

$$\begin{tikzcd} * \arrow[r, "0"] \arrow[dr, "k"'] & \mathbb{N} \arrow[r, "s"] \arrow[d, "h"'] & \mathbb{N} \arrow[d, "h"] \\ & \mathbb{N} \arrow[r, "g"'] & \mathbb{N} \end{tikzcd}$$

Structures: P.R. Functions

- The set of P.R. functions form a category with extra structure.
  - Objects are powers of natural numbers: N, N^2, N^3,…
  - Morphisms are functions from N^m to N^n.
  - Composition of edges: compose functions. Associative.
  - For every N^n there is an identity map. It acts like a unit.
  - There is a product bifunctor: f and g go to <f,g>. (Bracket)
  - The category has a STRONG natural number object. (Recursion.)
- Theorem: The category of P.R. functions is the initial object in the 2-category of categories with products and STRONG natural number objects.
- (The other categories with such structure correspond to primitive recursive functions with oracles. Other information is added.)

Analogies to Other Areas

$$\begin{tikzcd}
\textbf{Syntax} \arrow[d] & \text{Programs} \arrow[d] & \text{Braid Diagrams} \arrow[d] \\
\textbf{In Between} \arrow[d] & \text{Algorithms} \arrow[d] & \text{Braid Groups} \arrow[d] \\
\textbf{Semantics} & \text{Functions} & \text{Symmetric Groups}
\end{tikzcd}$$

Analogies to Other Areas

$$\begin{tikzcd}
\text{Braid Diagrams} \arrow[d] \\
\text{Braid Groups} \arrow[d] \\
\text{Symmetric Groups}
\end{tikzcd}$$

- Just as we can only represent an algorithm by giving a program, so too, the only way to represent a braid is by giving a braid diagram.

- Just as our set of Programs does not have enough structure to form a category, so too, the set of Braid Diagrams does not have a worthwhile structure. One can compose braid diagrams sequentially and parallel. But there is no associativity.

- Just as we can get the category of algorithms by looking at equivalence classes of programs, so too, we can get braids by looking at equivalence classes of braid diagrams. With braid diagrams we look at Reidermeister moves to determine when two braid diagrams are really the same. Here we look at relations to tell when two programs are the same.

Analogies to Other Areas

$$\begin{tikzcd}
\text{Braid Diagrams} \arrow[d] \\
\text{Braid Groups} \arrow[d] \\
\text{Symmetric Groups}
\end{tikzcd}$$

• Just as we are not giving the final word about what relations to use, so too, there is no final word about which Reidermeister moves to use. Depending on your choice, you will get braids, ribbons, oriented ribbons etc.

• Just as our category of Algorithms is the free category with products and weak natural number objects generated by the empty category, so too, the category of Braids is the free braided monoidal category generated by one object.

• Just as we can go down to the level of functions by making two algorithms that perform the same function equivalent, so to, we can add a relation that two strings can cross each other and get Symmetric Groups.

• Just as the main focus of computer scientists are algorithms and not programs, so to, the main focus of topologists is braids and not braid diagrams.

Possible Graphs of Algorithms

Associative Composition: (fog)oh ~ fo(goh)

Unit of Composition: f o id ~ f ~ id o f

Associative Bracket: <<f,g>,h> ~ <f,<g,h>>

Comp dist over brac: <f,g>oh ~ <foh,goh>

Brak is almost commut: <f,g> ~ tw o <g,f>

Twist is idempotent: tw o tw ~ id

Twist is coherent

Axioms of a natural number object

NNO and product respect each other

$$\begin{tikzcd}
& \text{PRdesc} \arrow[dl] \arrow[dr] & \\
\text{PRalgC} \arrow[dr] & & \text{PRalgI} \arrow[dl] \\
& \text{PRalgCat} \arrow[dl] \arrow[dr] & \\
\text{PRalgCatX} \arrow[dr] & & \text{PRalgCatN} \arrow[dl] \\
& \text{PRalgCatXN} \arrow[d] & \\
& \text{PRfunc} &
\end{tikzcd}$$

Galois Theory of Algorithms

$$\begin{tikzcd}\text{Programs} \arrow[r] & \text{Programs}\end{tikzcd}$$

In order to study the possible quotients of programs we look at automorphisms of programs. This forms a group.

Galois Theory of Algorithms

$$\begin{tikzcd}
\text{Programs} \arrow[rr] \arrow[dr] & & \text{Programs} \arrow[dl] \\
& \text{Functions} &
\end{tikzcd}$$

We don't just want to mix up programs. The automorphism must preserve functionality. Φ(mergesort$_x$)=quicksort$_a$.

Aut(P/F)

Philosophical Sidebar

Galois theory is the study of how a subfield sits inside a larger field.

or more generally:

Galois theory is the study of how a sub-object sits inside a larger object.

Co-Galois theory is the study of how a quotient object is inside an object.

Galois Theory of Algorithms

$$\begin{tikzcd}
\text{Programs} \arrow[rr] \arrow[dr] \arrow[ddr] & & \text{Programs} \arrow[dl] \arrow[ddl] \\
& \text{Algorithms} \arrow[d] & \\
& \text{Functions} &
\end{tikzcd}$$

Look at the automorphisms of programs that preserve algorithms. $\Phi(\text{mergesort}_x)=\text{mergesort}_z$. This is a group Aut(P/A) and is a subgroup of Aut(P/F).

Galois Theory of Algorithms

$$\begin{tikzcd}
\text{Programs} \arrow[rrrr] \arrow[dr] \arrow[ddrr] & & & & \text{Programs} \arrow[dlll] \arrow[ddll] \\
& \text{Algorithms} \arrow[rr] \arrow[dr] & & \text{Algorithms} \arrow[dl] & \\
& & \text{Functions} & &
\end{tikzcd}$$

There is no reason to think that it is a normal subgroup… but we can still get the quotient set:

$* \rightarrow \mathrm{Aut}(P/A) \rightarrow \mathrm{Aut}(P/F) \rightarrow \mathrm{Aut}(A/F) \rightarrow *$

Galois Theory of Algorithms

Intermediate algorithmic universes | **Automorphism Groups**

$$\begin{tikzcd}
\text{Programs} \arrow[d] & \text{Aut}(P/P) = \{\text{Id}\} \arrow[d, hook] \\
\text{Algorithms} \arrow[d] & \text{Aut}(P/A) \arrow[d, hook] \\
\text{Algorithms'} \arrow[d] & \text{Aut}(P/A') \arrow[d, hook] \\
\text{Functions} & \text{Aut}(P/F)
\end{tikzcd}$$

Fundamental Theorem of Galois Theory

The lattice of subgroups of Aut(P/F)

is equivalent to

The dual lattice of intermediate algorithmic universes.

Intuition:

Two programs will be exchangeable

if and only if

they are considered the same algorithm.

Galois Theory of Algorithms – Future Directions

Classical Galois Theory

- Zassenhaus lemma for algorithms.
- Schreier refinement theorem for algorithms.
- Jordan-Holder theorem for algorithms.
- Krull-Schmidt theorem for algorithms.

Applications

- Extend to all computable functions.
- Calculate some groups.
- Morphisms between algorithms: Compilers.
- Impossibility results.

References

- Yu. I. Manin *A Course in Mathematical Logic for Mathematicians 2nd ed.* Springer (2010) (Chapter IX).
- Yu. I. Manin "Renormalization and computation I: motivation and background". http://arxiv.org/abs/0904.4921
- Yu. I. Manin "Renormalization and Computation II: Time Cut-off and the Halting Problem". http://arxiv.org/abs/0908.3430
- Yu. I. Manin and N. Yanofsky "Enriched Programming Methods with Unrestricted Parallelism". Work in Progress.
- N. Yanofsky "Galois Theory of Algorithms". Almost ready.
- N. Yanofsky "Towards a Definition of an Algorithm". http://arxiv.org/abs/math/0602053

Thank You!