Convergent Global Optimisation via Constructive Real Numbers

Todd Waugh Ambridge
(University of Birmingham)

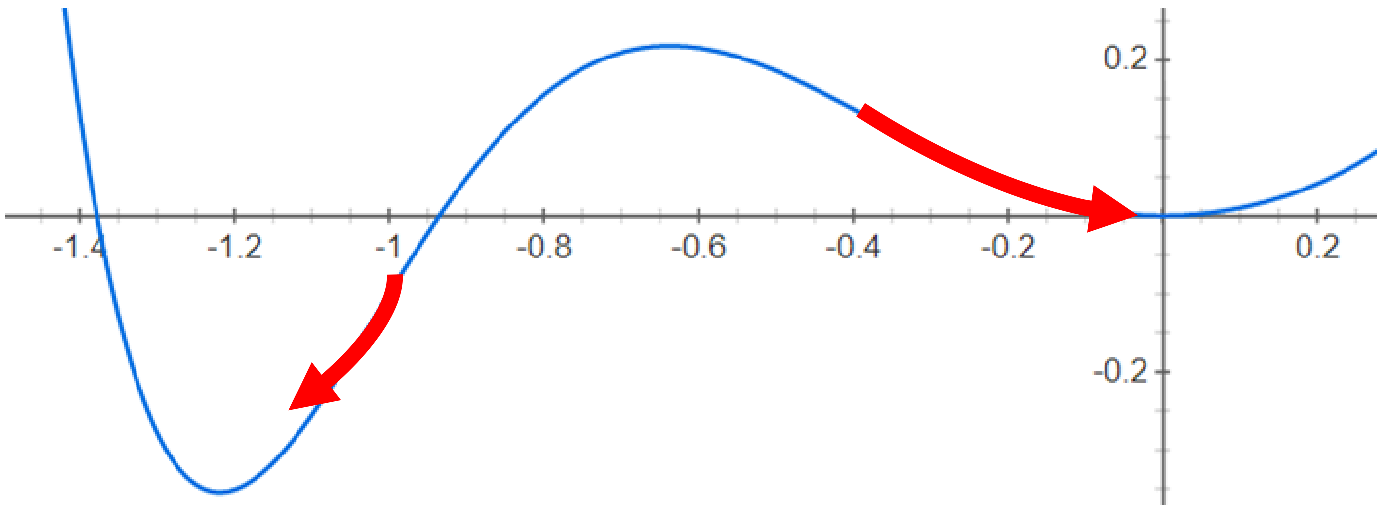
11th March 2021

In this talk I will…

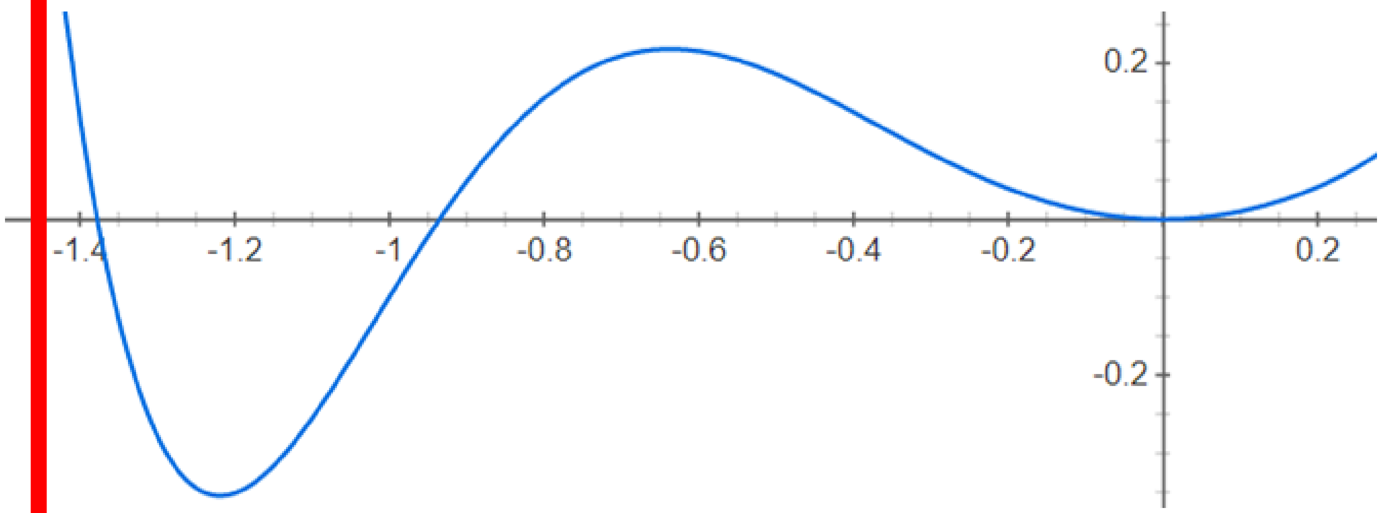
- Introduce **convergent global optimisation**
- Discuss how **floating-point real numbers** are an inappropriate data type for convergent global optimisation
- Introduce two types for **arbitrary-precision real numbers**
- Show how **global optimisation converges** on one of these types
- *Apply this framework to machine learning to exhibit* ***convergence properties for regression*** *on arbitrary 'searchable' data types*

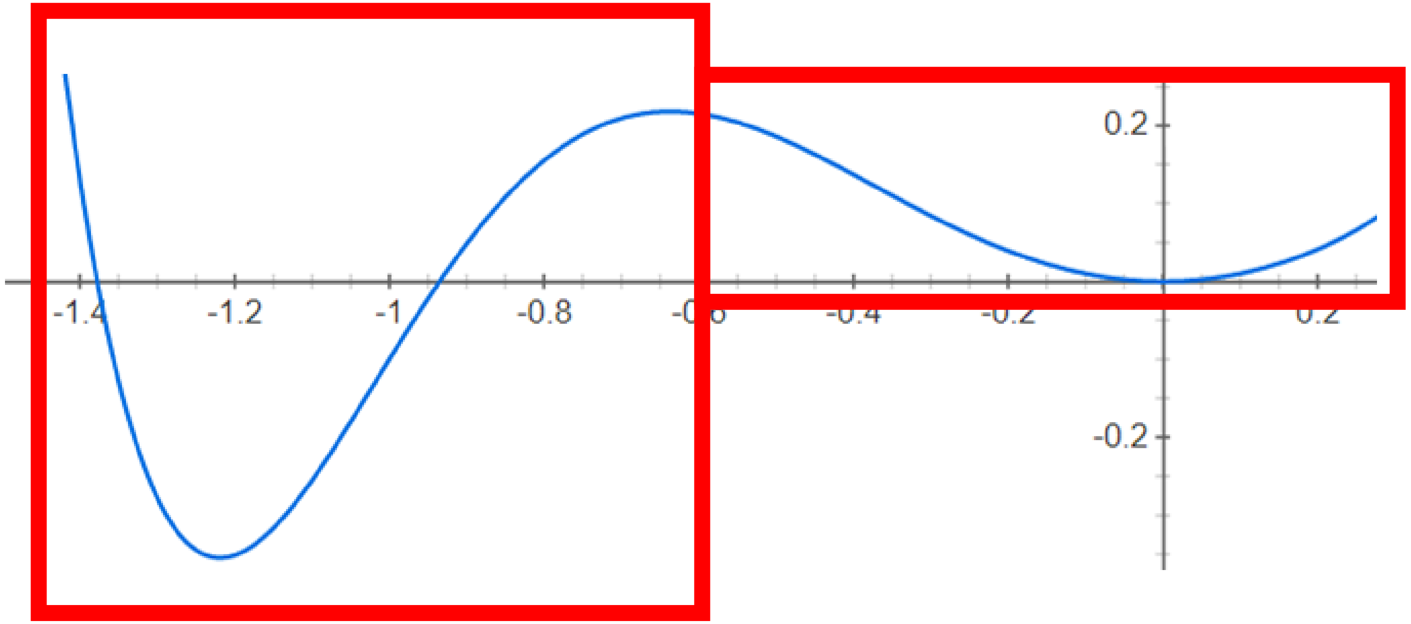
Optimisation: Efficiency vs. Correctness

- Optimisation is a core component of supervised machine learning.
- It has broad applications to function approximation algorithms, such as **interpolation** and **regression**.
- Efficient **local optimisation** algorithms, e.g. **gradient descent**, have been studied extensively – *and applied to deep learning!*
- Convergent **global optimisation** algorithms, e.g**. branch-and-bound**, are much less practically available – *but never yield an incorrect result!*

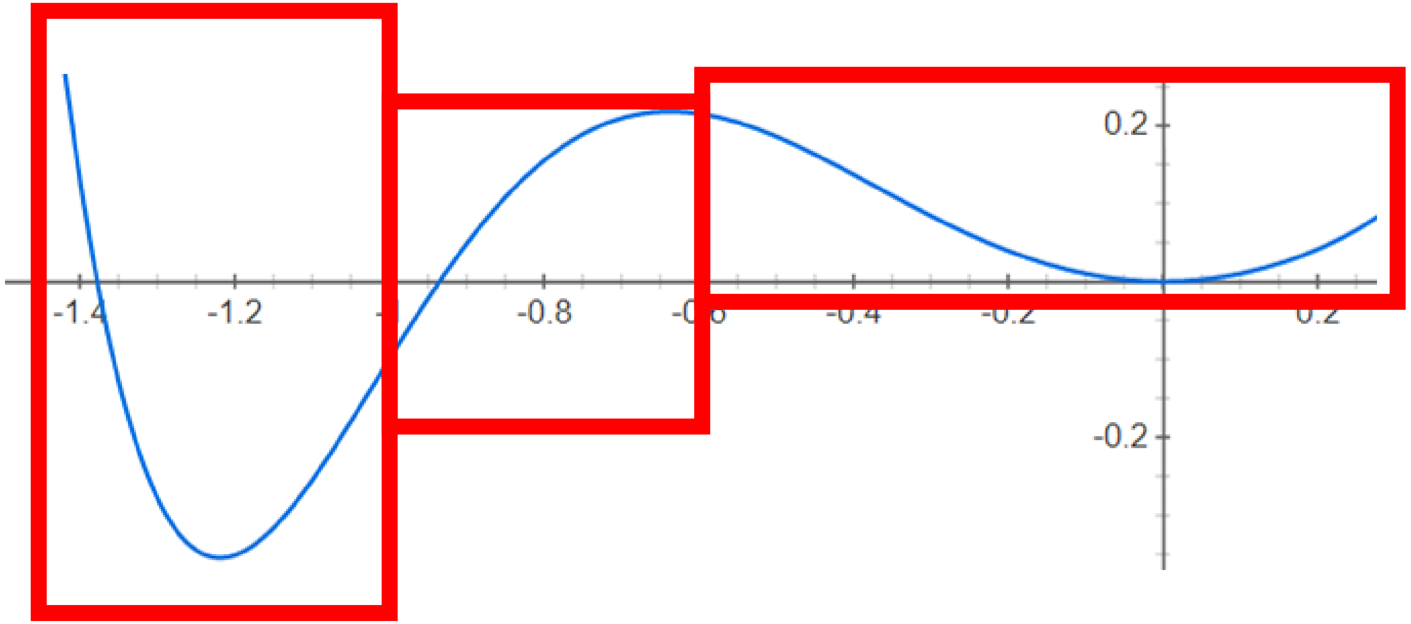


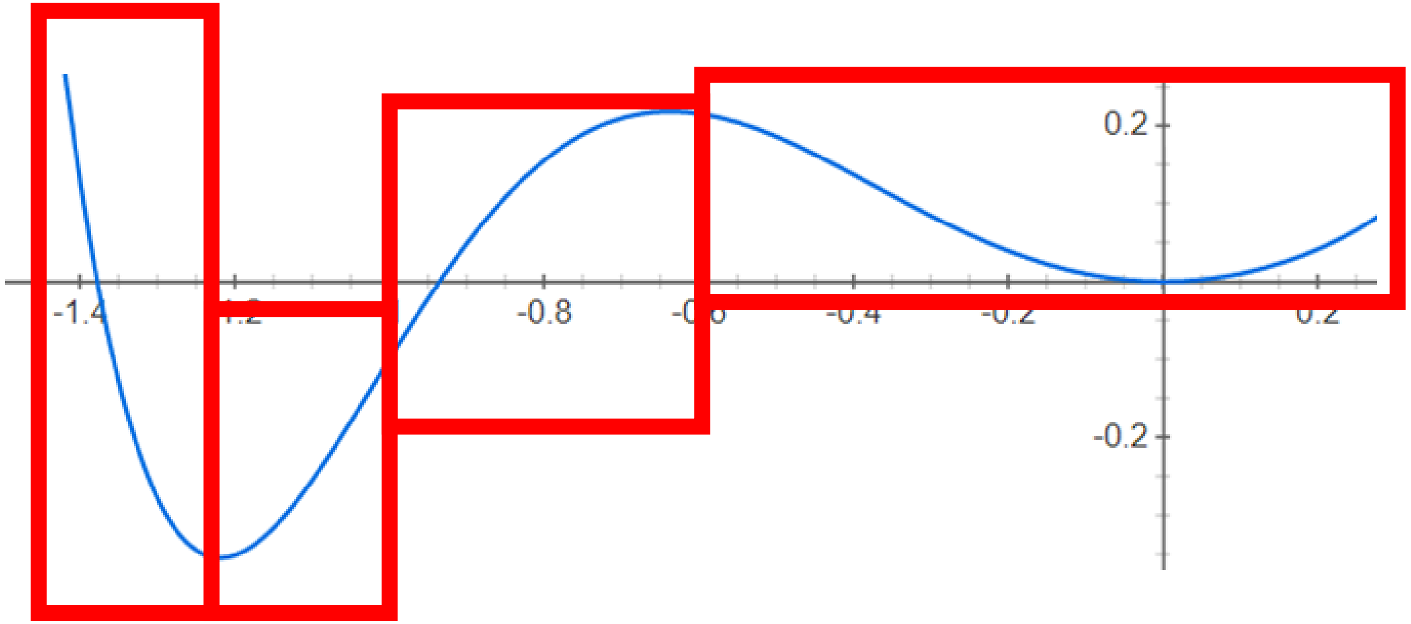
0.2
-0.2
-1.4
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0.2


0.2
-1.4
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0.2
-0.2

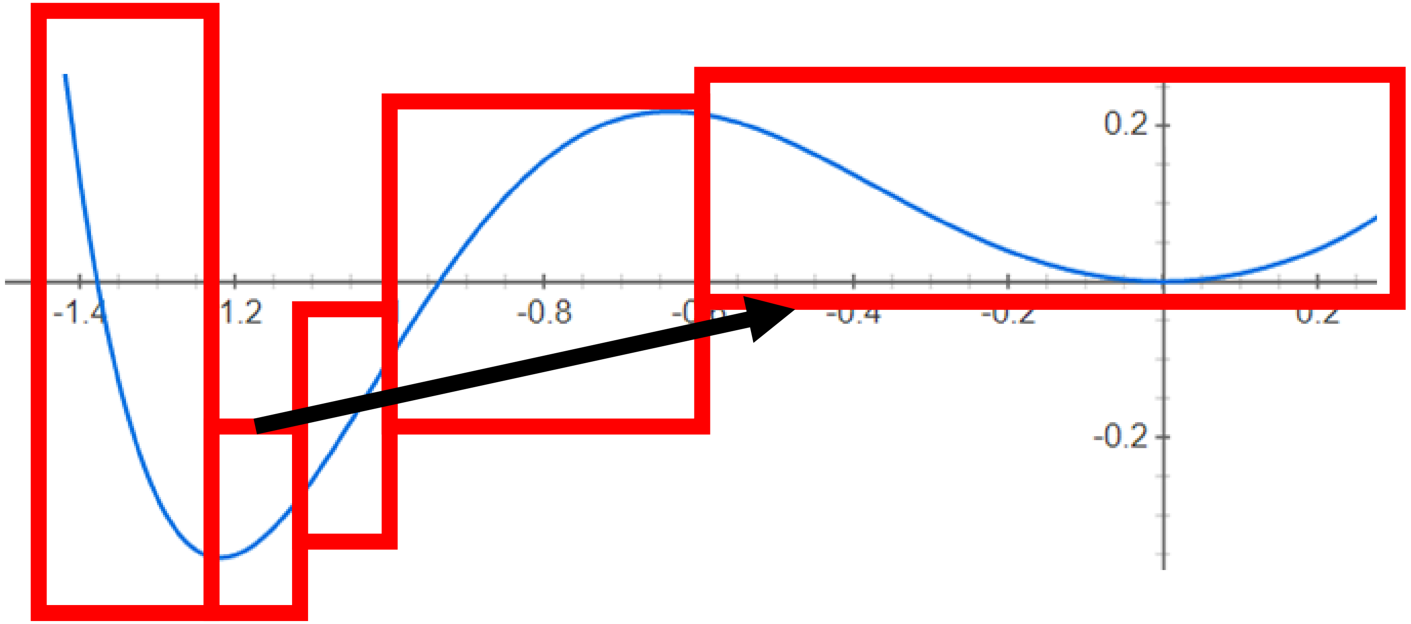


0.2
-0.2
-1.4
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0.2

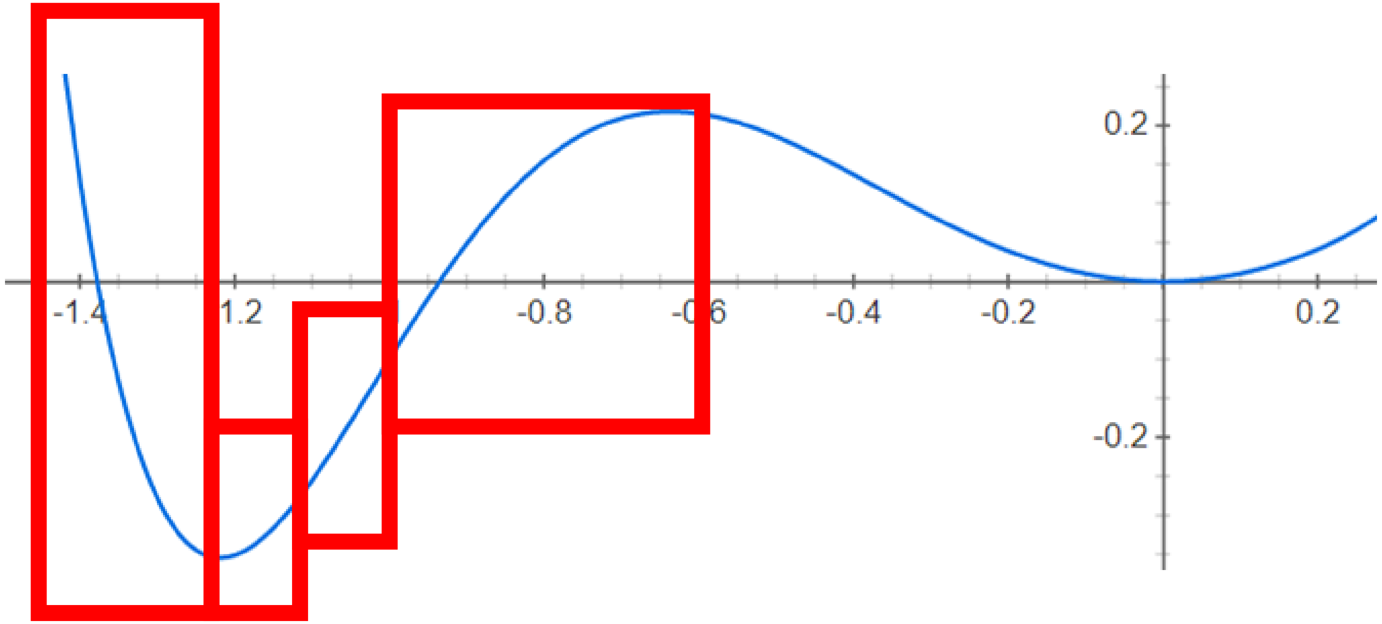

0.2
-0.2
-1.4
-1.2
-0.8
-0.6
-0.4
-0.2
0.2



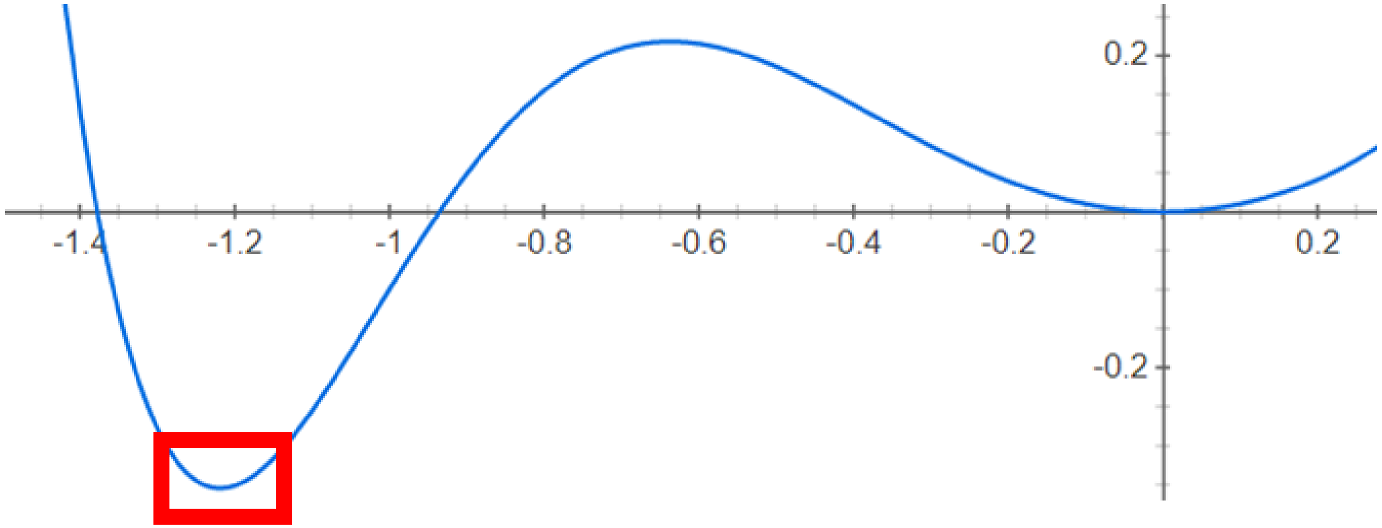
0.2
-0.2
-1.4
-0.8
-0.4
-0.2
0.2



0.2
-0.2
-1.4
1.2
-0.8
-0.4
-0.2
0.2

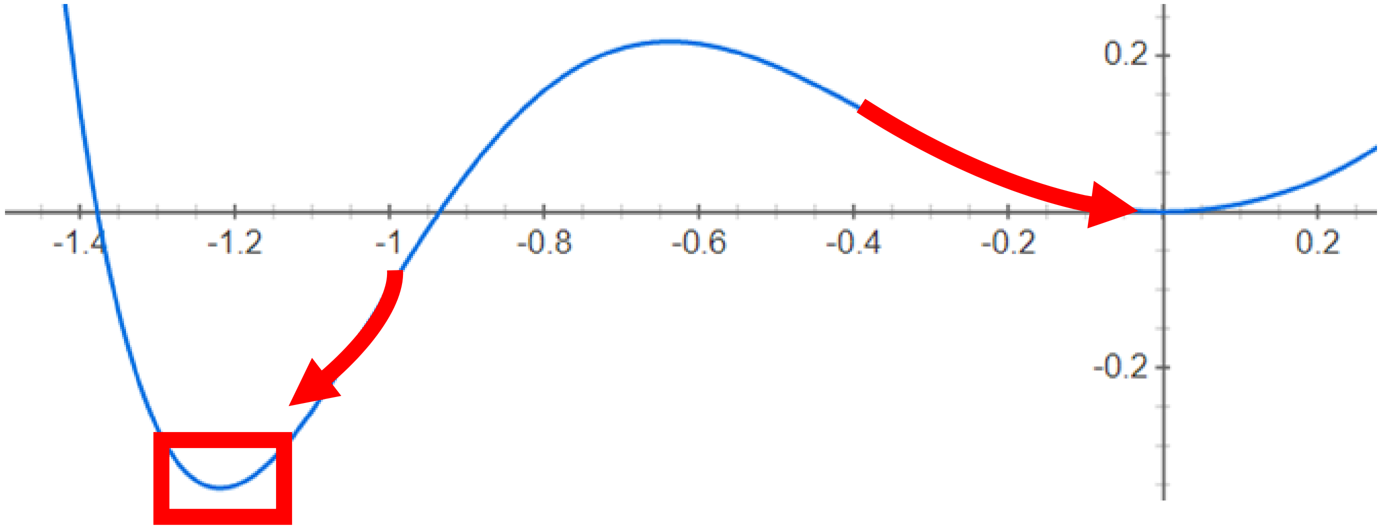


0.2
-0.2
-1.4
-1.2
-0.8
-0.6
-0.4
-0.2
0.2



0.2
-0.2
-1.4
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0.2

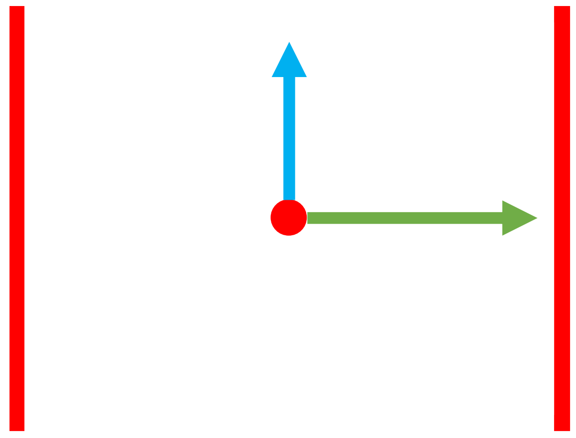
- Gradient descent: *Local* minima via *derivative* of function
- Branch-and-bound: *Global* minima via *continuity* of function



Convergence of Global Optimisation

- A function $f : \mathbb{R} \to \mathbb{R}$ is *continuous* if it comes equipped with a *modulus of continuity* function $m : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that,

$$m(x, |x - y|)\ .$$



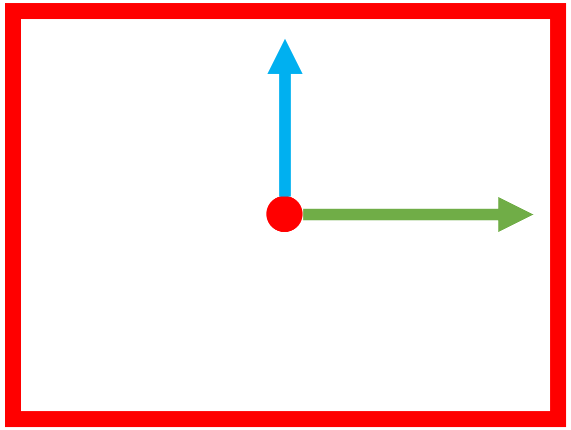
L. H. de Figueiredo, R. J. V. Iwaarden, and J. Stolfi. Fast interval branch-and-bound methods for unconstrained global optimization with affine arithmetic. 1997.

Convergence of Global Optimisation

- A function $f : \mathbb{R} \to \mathbb{R}$ is *continuous* if it comes equipped with a *modulus of continuity* function $m : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ such that,

$$|f(x) - f(y)| \leq m(x, |x - y|)\ .$$

- A global optimisation algorithm is *convergent* if, for any $\epsilon : \mathbb{R}$, we can compute $x_0 : \mathbb{R}$ such that $|f(x_0) - f(x)| < \epsilon$.
- A **branch-and-bound algorithm** converges if:
  1. The function is continuous,
  2. The *branching* procedure ensures the width of the widest box tends to 0,
  3. The *bounding* procedure ensures that, as the width of a box decreases, so too does its height.



Global Optimisation using Floating-point Reals

- When computer scientists talk about "real numbers", they often mean "floating-point numbers" – *for obvious reasons!*
- However, floats are an inappropriate data type for performing global optimisation…

```cpp
int main()
{
    float x = 0.0;
    float y = 0.0;
    for (int i = 0; i < 10; i++) {
        x += 0.1;
        for (int j = 0; j < 10; j++) {
            y += 0.01;
        }
    }
    std::cout << x << std::endl;
    std::cout << y << std::endl;
}
```

```
1
0.999999
```

Exploiting Verified Neural Networks via Floating Point Numerical Error

Technical Report

Kai Jia
MIT CSAIL
jiakai@mit.edu

Martin Rinard
MIT CSAIL
rinard@csail.mit.edu

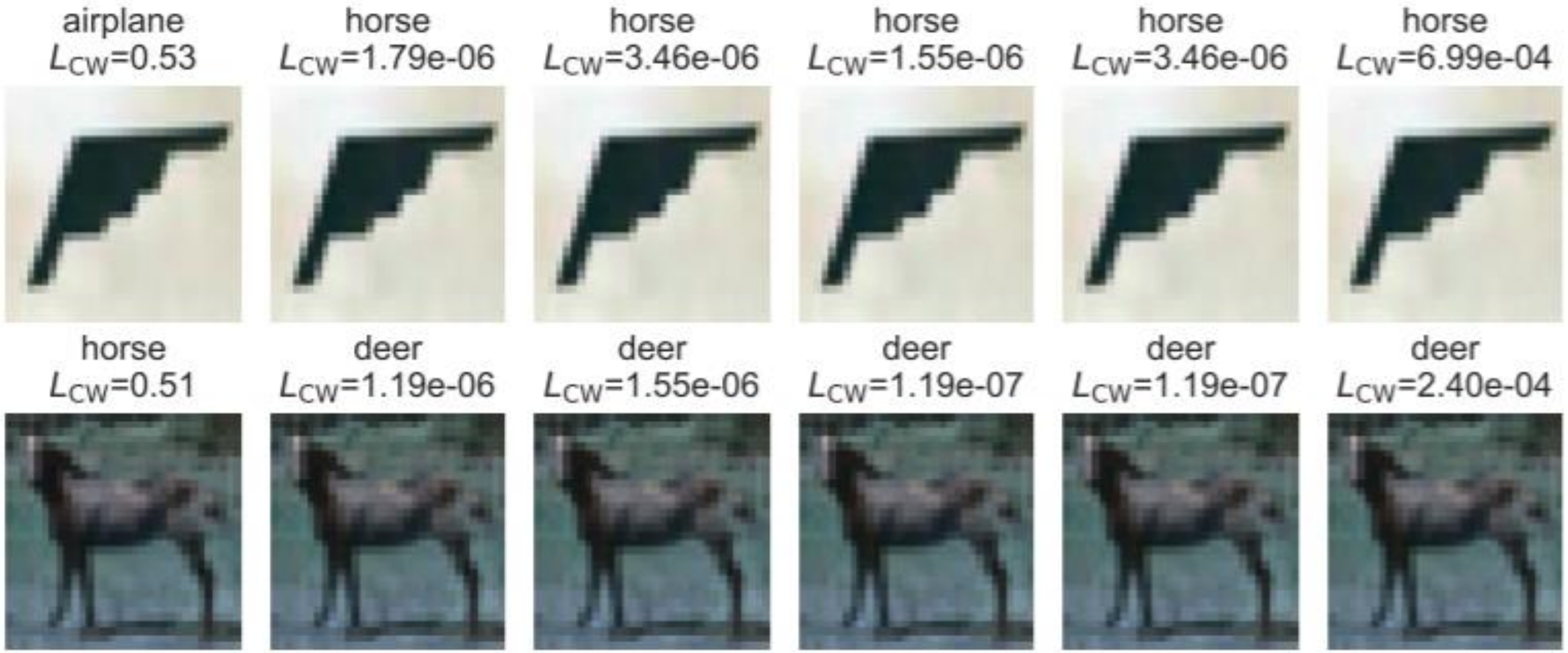
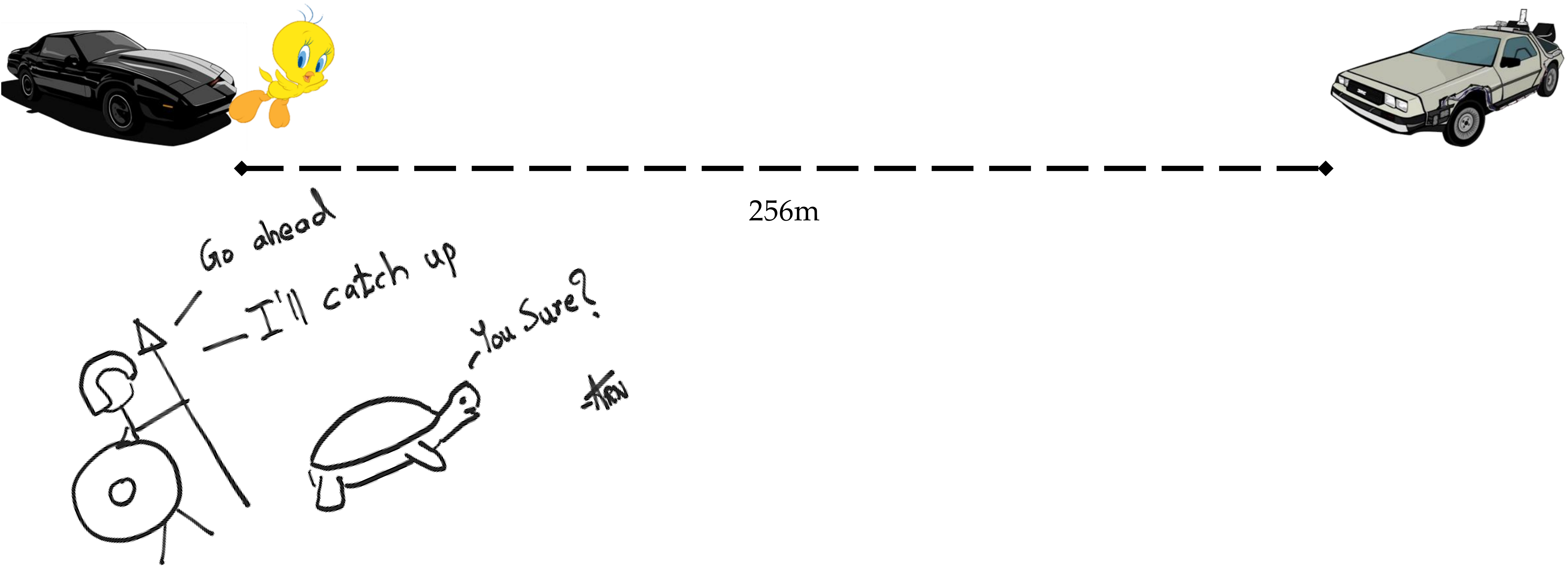
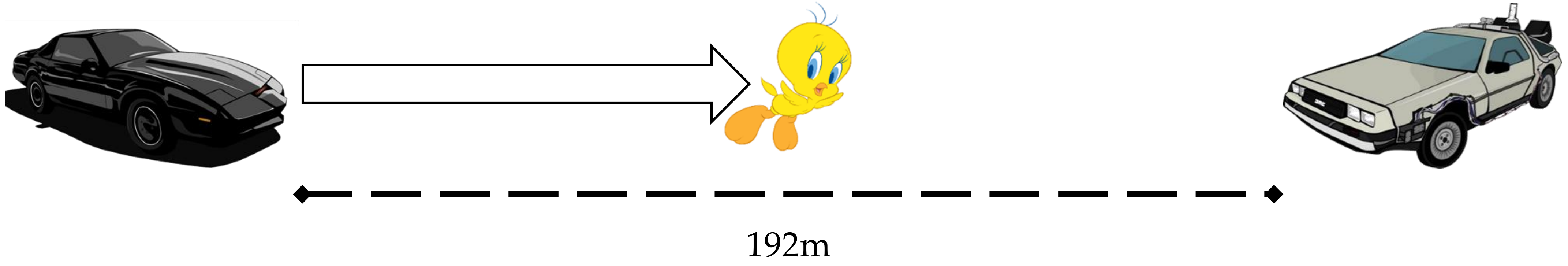
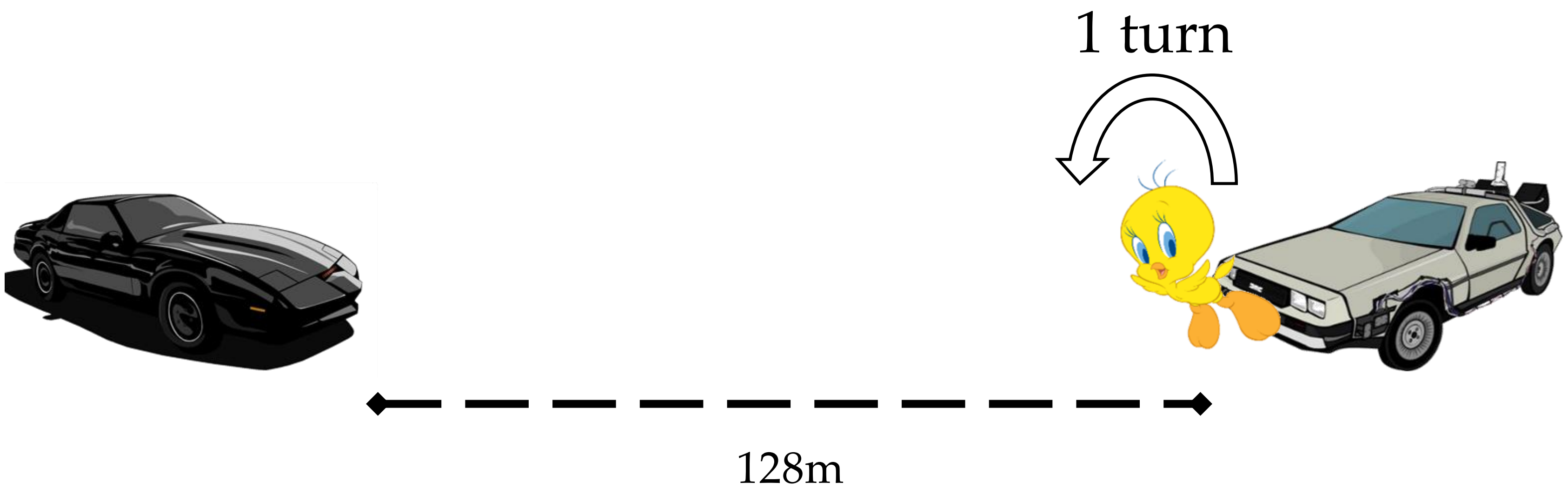
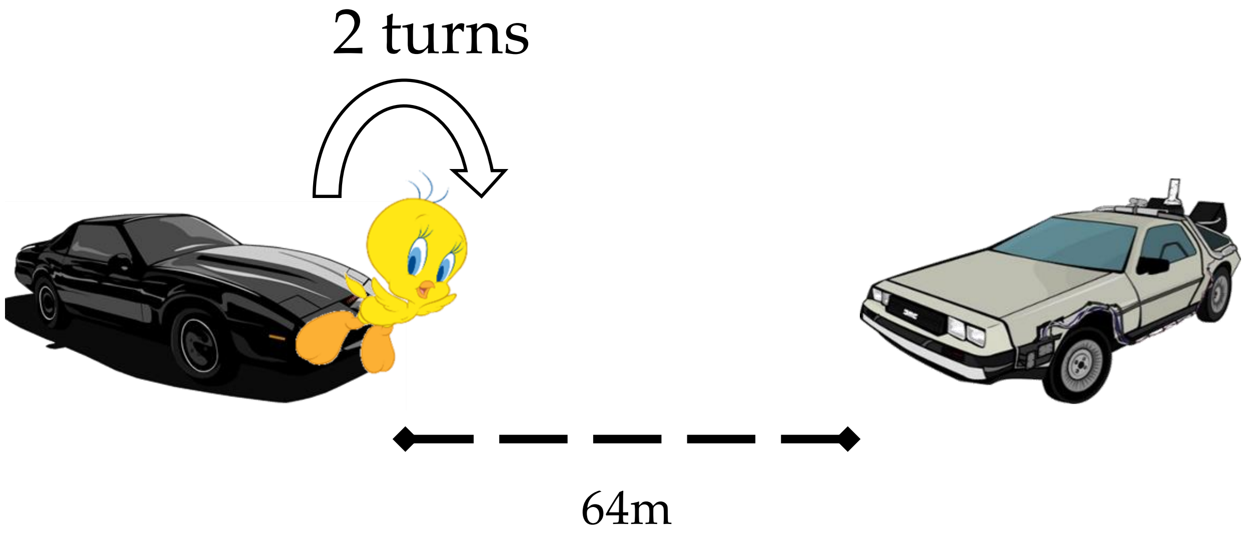


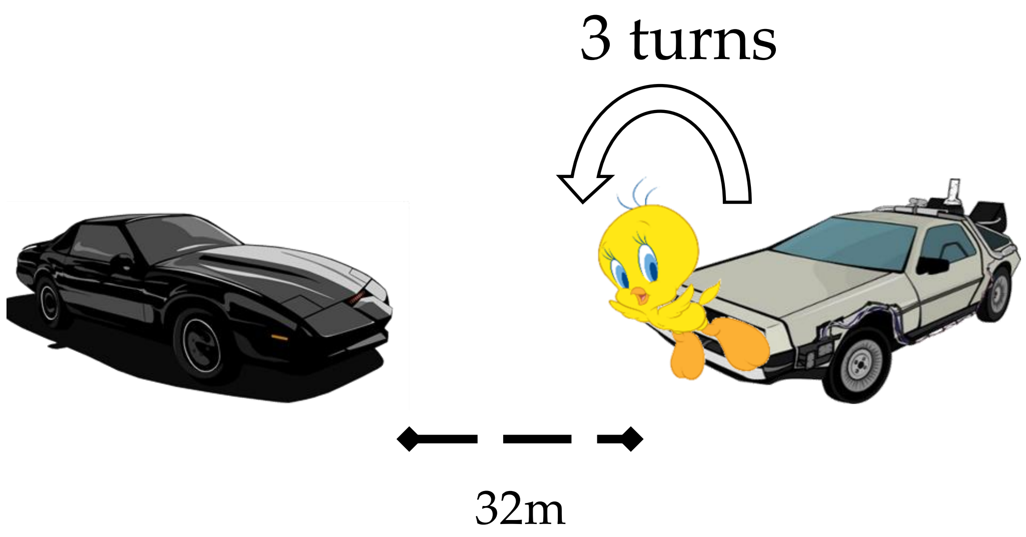
Figure 3: Adversarial Images Found by Our Method

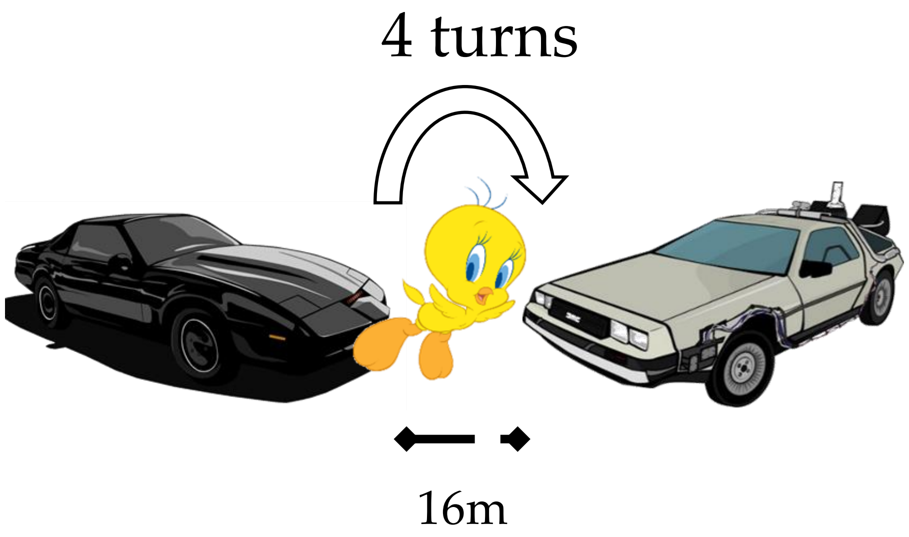

256m
Go ahead
I'll catch up
You Sure?

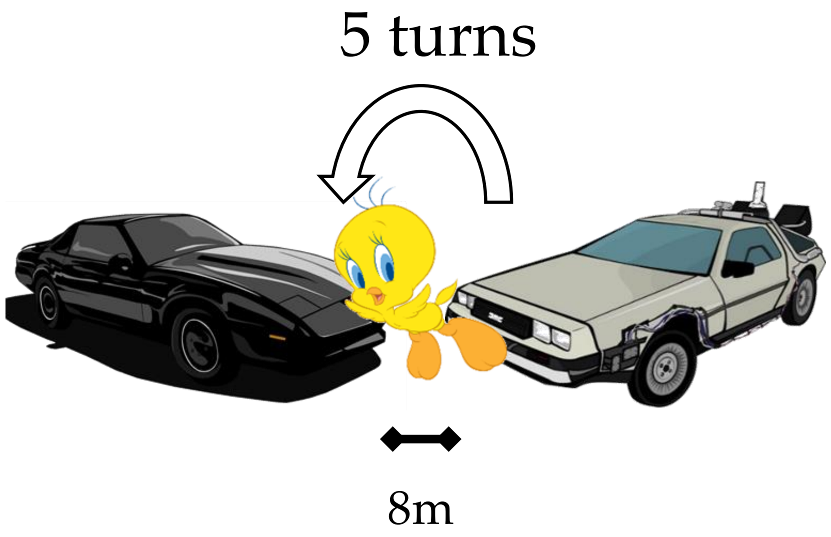

192m

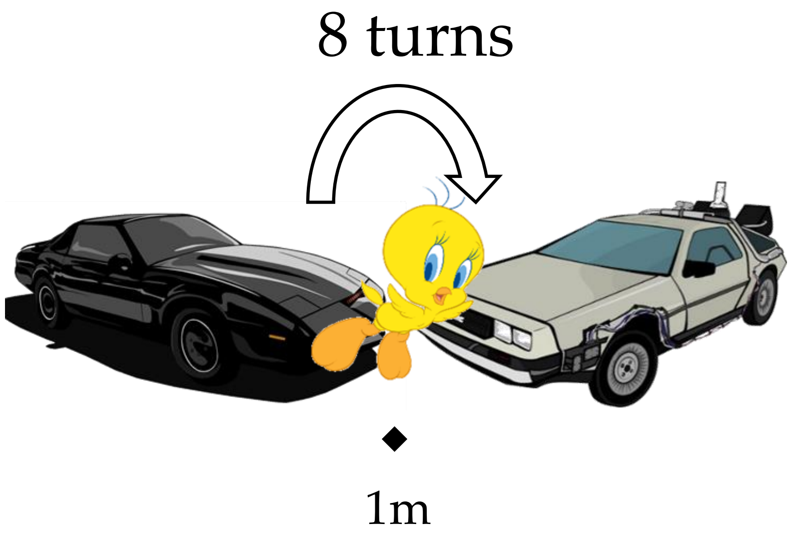

1 turn
128m

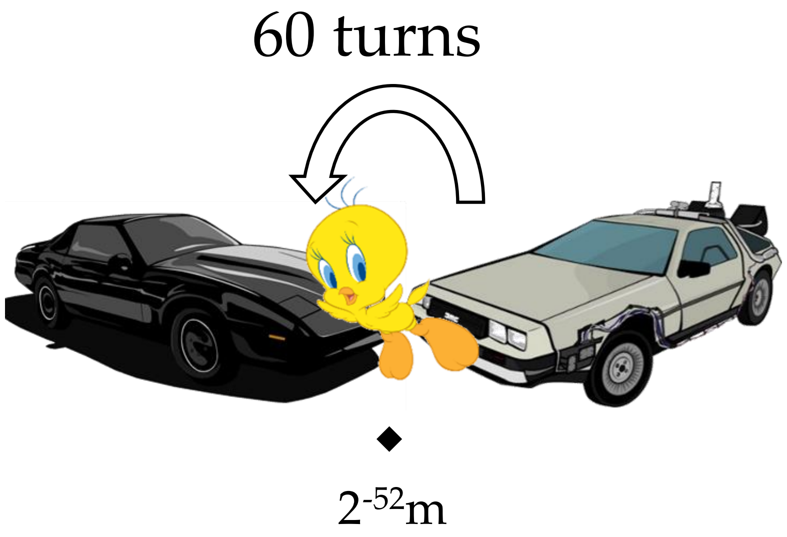

2 turns
64m


3 turns
32m


4 turns
16m

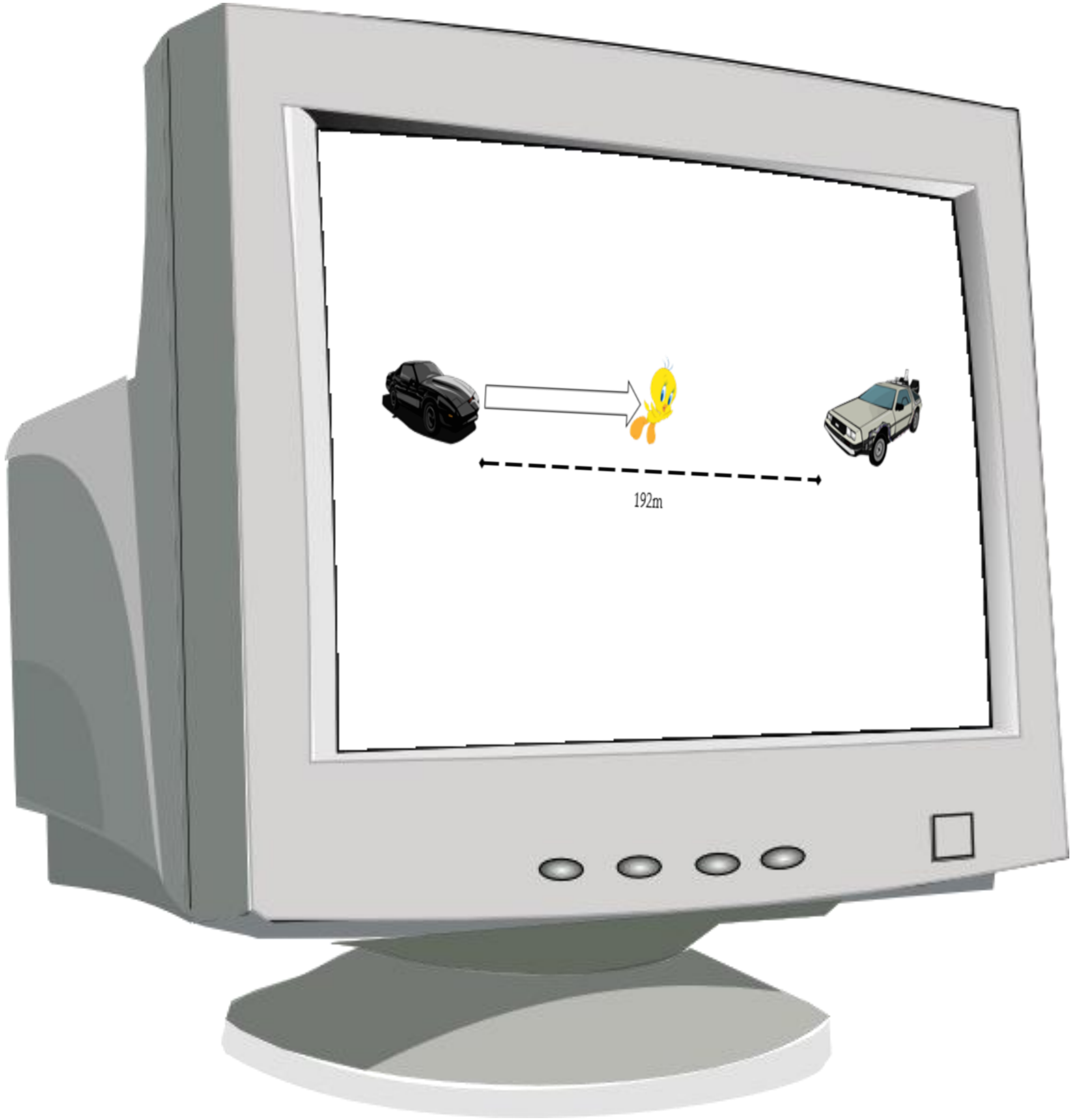

5 turns
8m

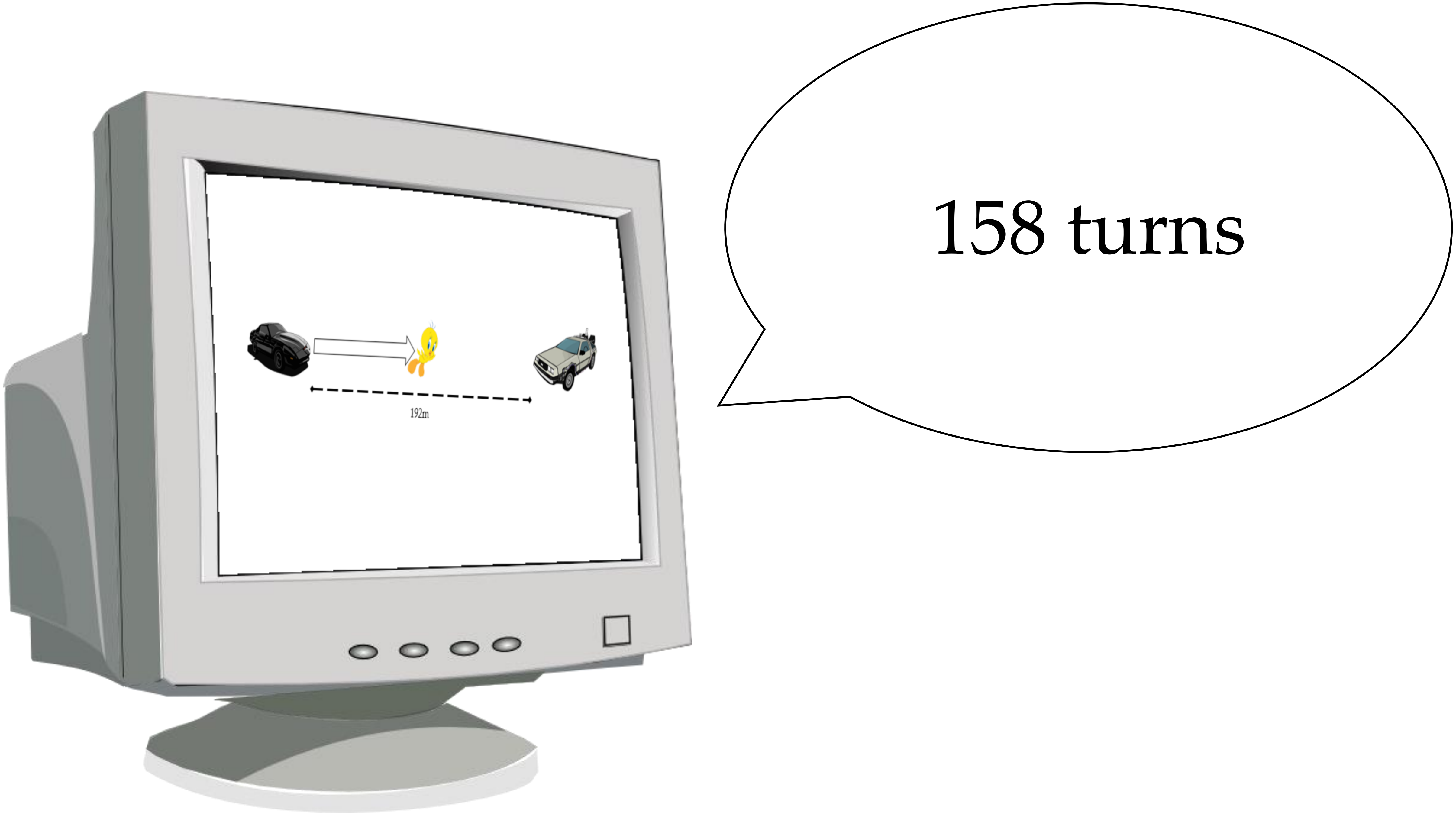

8 turns
1m

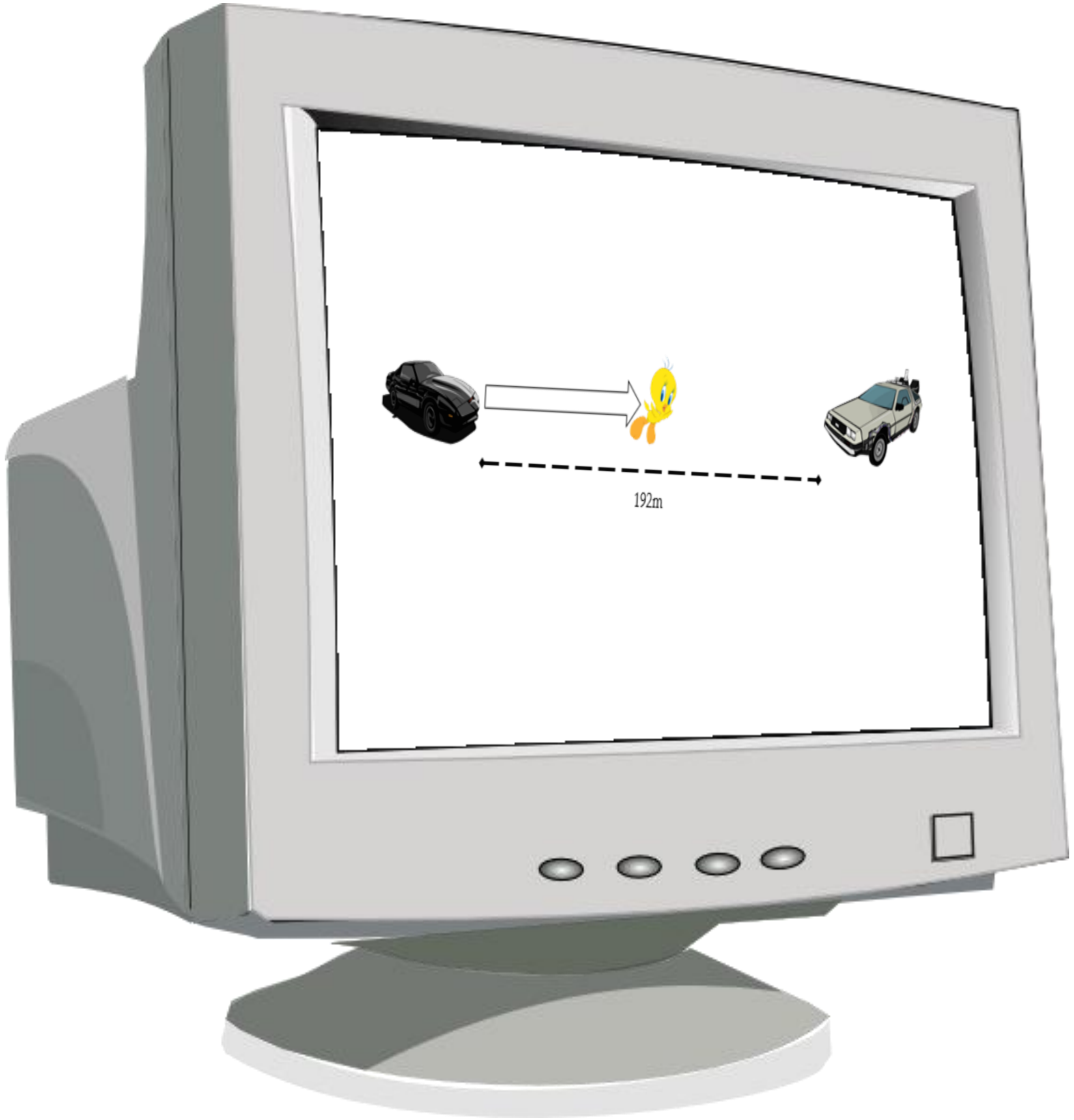

60 turns
2-52m

∞ turns

0m


192m

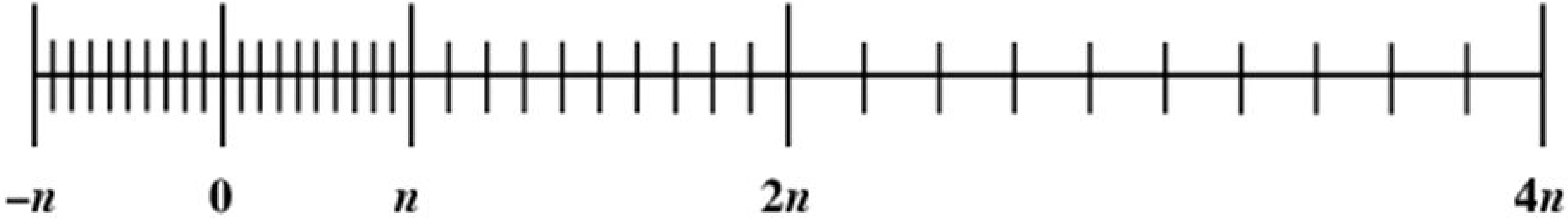

192m
158 turns



Is 158 a good simulation of infinity?

Global Optimisation using Floating-point Reals

- These errors are, in practice, often unimportant – but sometimes they are *crucial*.
- Floating-point has a very high level of precision – but this *granularity* is *fixed*.
  - Floating-point is a *'discrete'* data type.
  - This affects both continuity and convergence.
- Global optimisation convergence cannot be guaranteed.
- We thus require a '*continuous*' data type for *arbitrary-precision* real numbers…



A. B. Booij, “Analysis in univalent type theory,” Ph.D. dissertation, University of Birmingham, 2020.

What are Constructive Reals?

- **Constructive reals** are those real numbers $x : R$ that can be constructively *located*: either $p < x$ or $x < q$ for any $p, q : Q$.
- **Constructive reals** are those real numbers $x : R$ that can be reconstructed by an algorithm to *any degree of precision*.
  - A pair (i, T) where $i = floor(x)$ and $T: N^+ \to \{1 \ldots 10\}$ where $T(n)$ is the nth decimal digit of x.
  - A function $f: Z \to Z$ such that $| x - 2^n * f(n) | \leq 2^{n-1}$.
- **Constructive reals** are a data type where the *granularity* of the real line is *dynamic* – and converges to the real line itself.

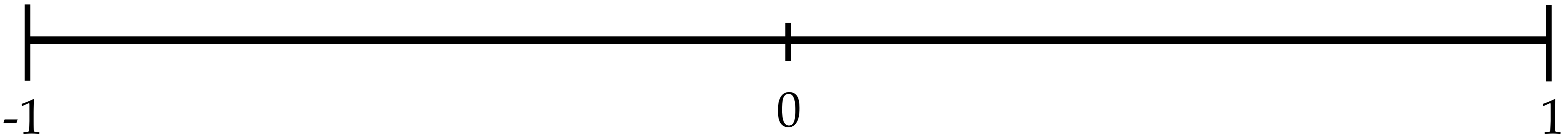
P. Di Gianantonio, “A functional approach to computability on real numbers,”
European Association For Theoretical Computer Science, vol. 50, pp. 518–518, 1993.

Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

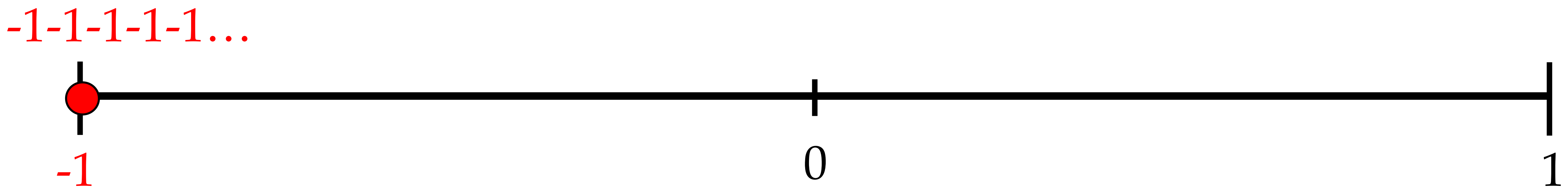


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

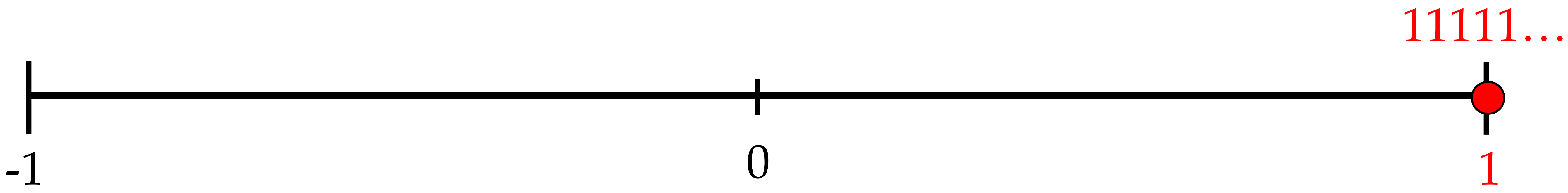


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

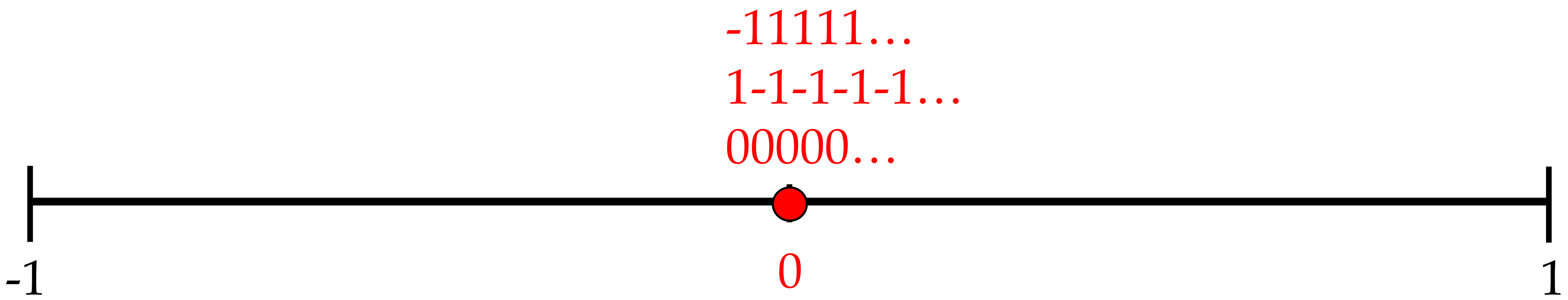


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] \coloneqq \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

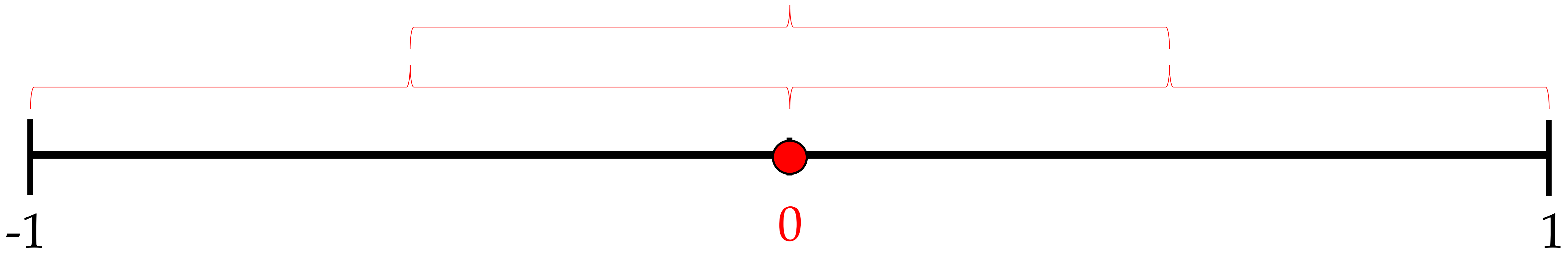


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

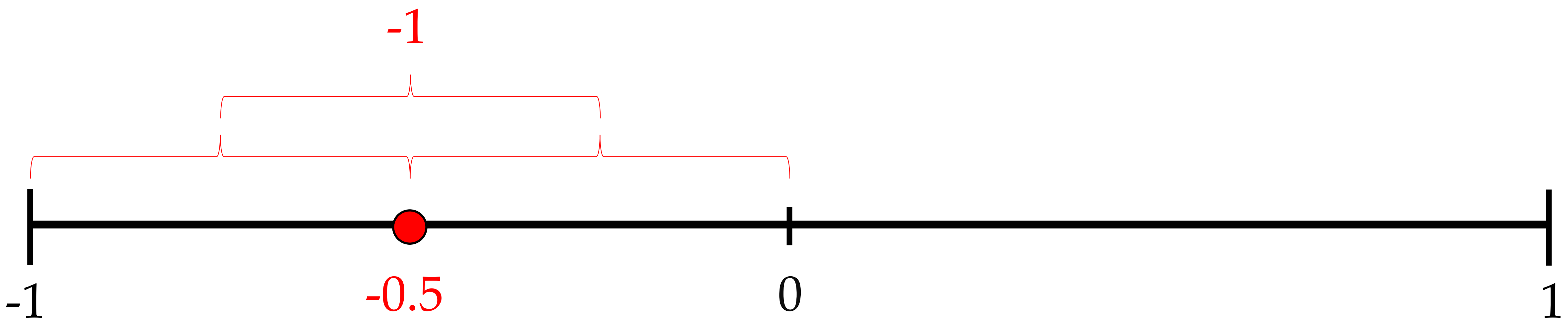


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

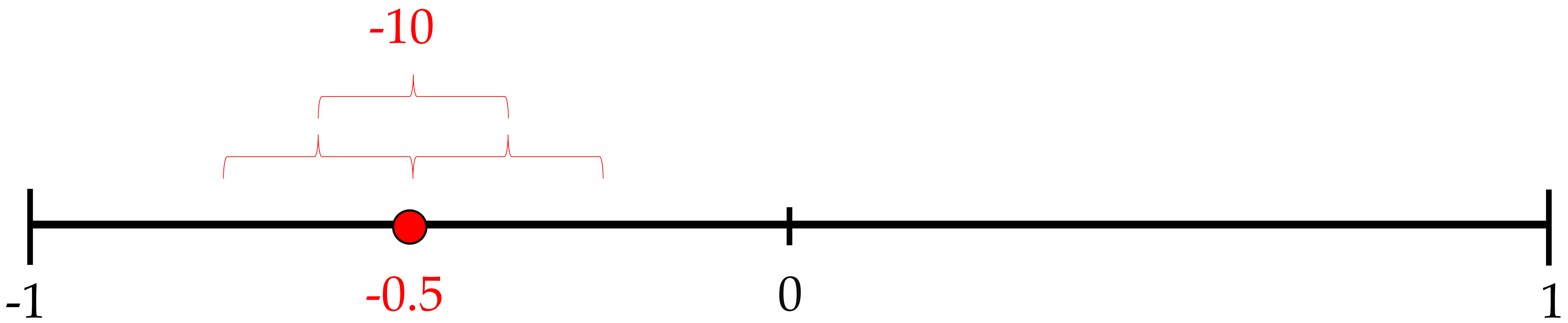


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \rightarrow \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \rightarrow \{-1,0,1\}$.

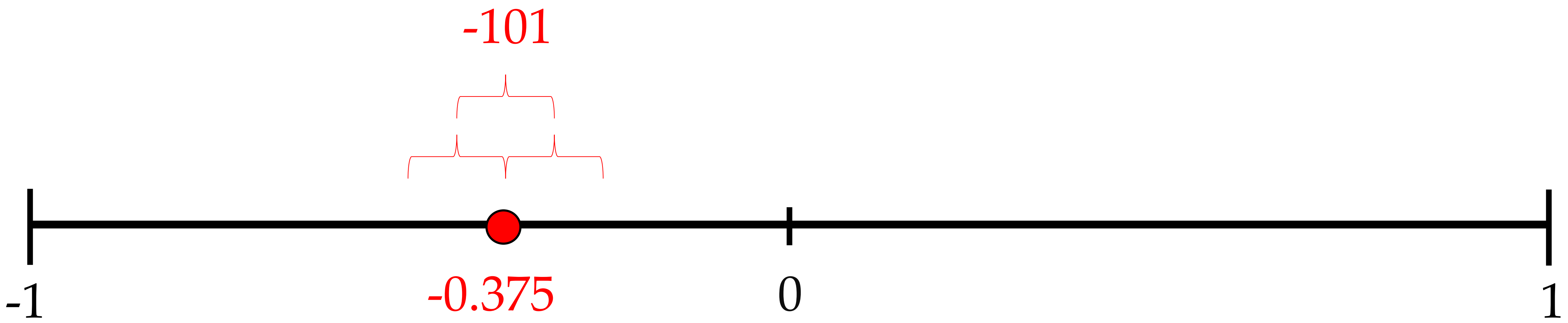


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.

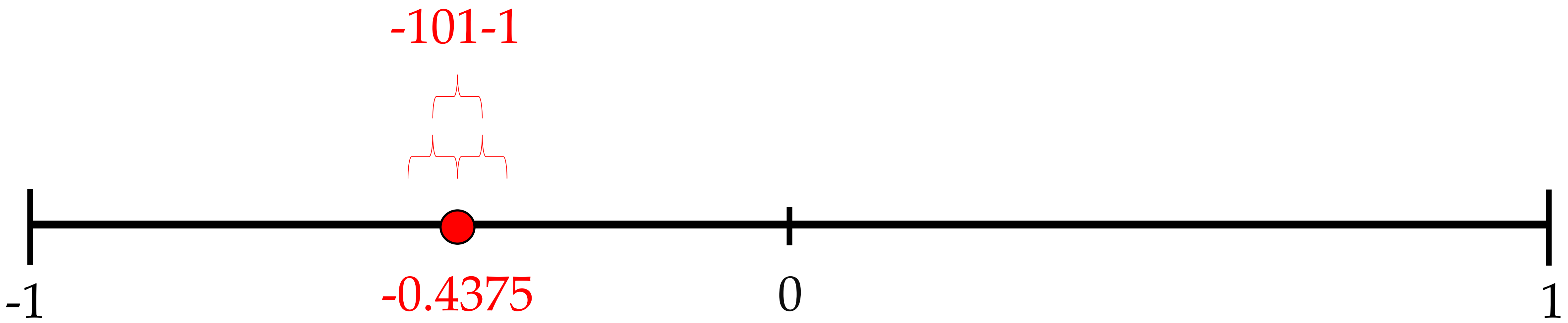


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \rightarrow \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \rightarrow \{-1,0,1\}$.

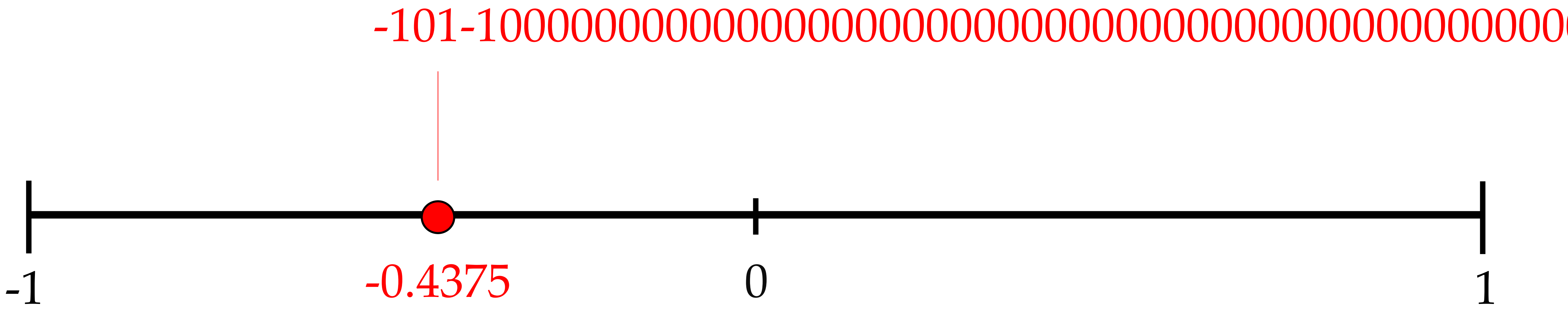


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \rightarrow \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \rightarrow \{-1,0,1\}$.

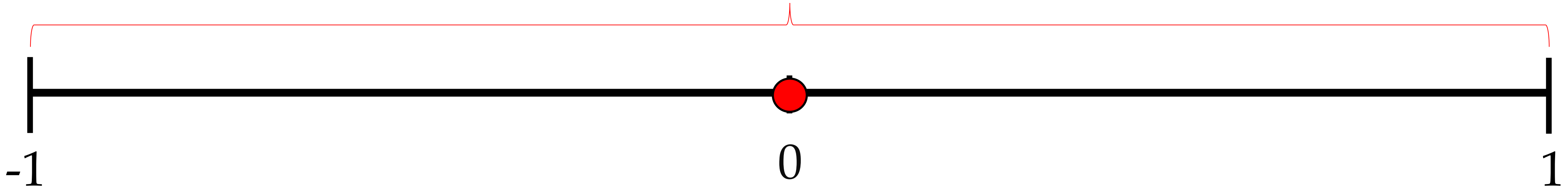


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] \coloneqq \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!

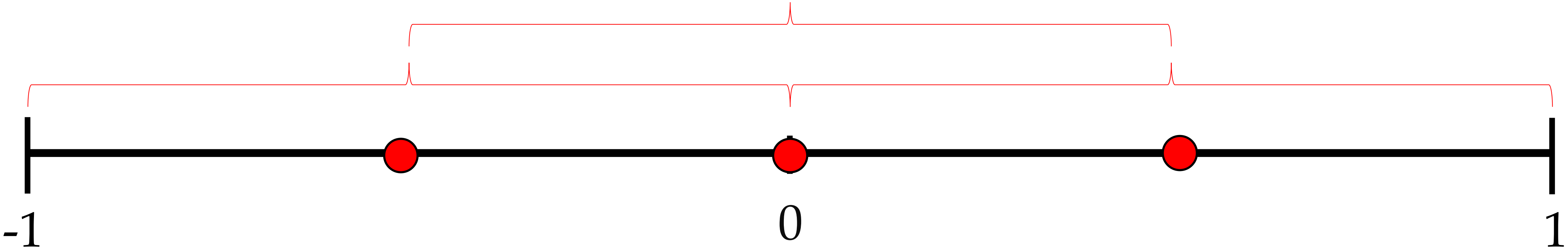


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!

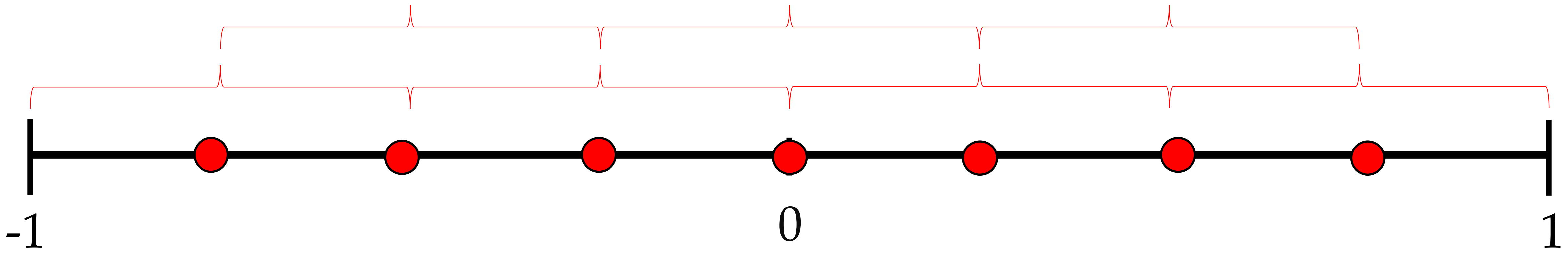


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!

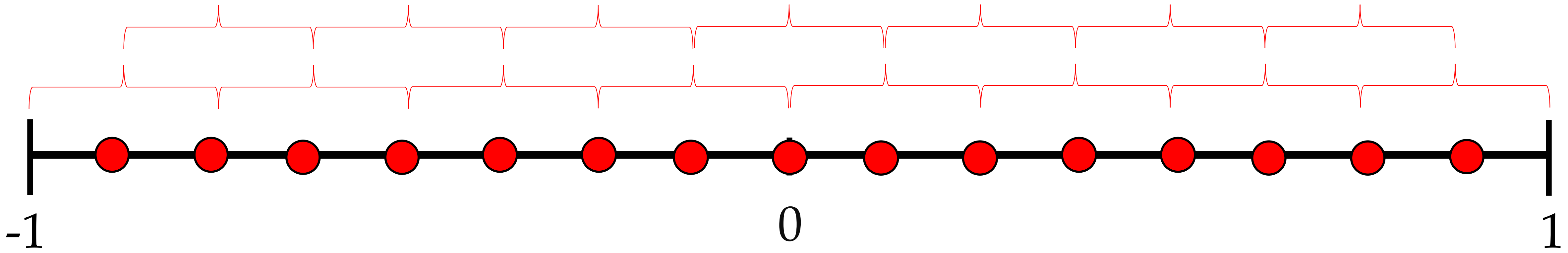


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!

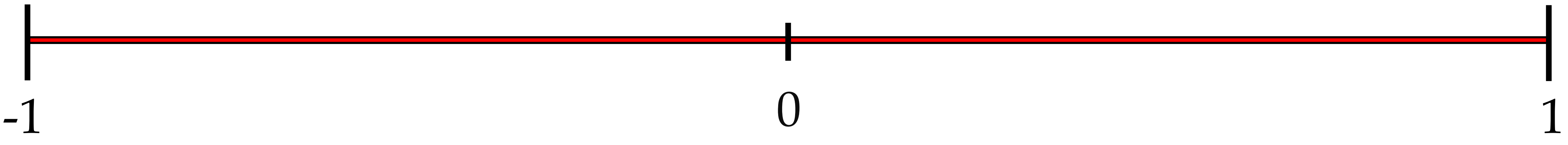


Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \ldots, n\}$ such that,

$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!



M. Escardo, “Real number computation in Haskell with real numbers ´ represented as infinite sequences of digits,” 2011.

Implementations of Constructive Reals

- **Signed-digit representation:** Numbers in $[-n, n]$ can be represented as infinitary sequences $\alpha : N \to \{-n, \dots, n\}$ such that,

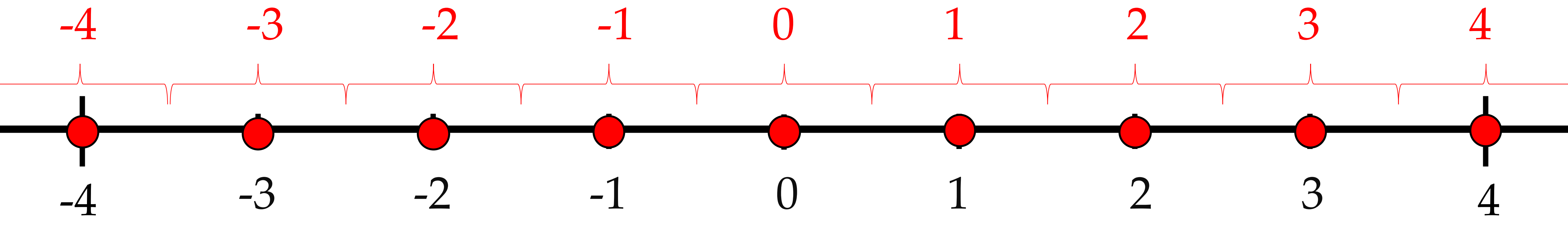
$$[\![\alpha]\!] := \sum_{(n=0)}^{\infty} \frac{\alpha_n}{2^{n+1}}.$$

- For example, we represent $[-1,1]$ by streams of type $N \to \{-1,0,1\}$.
- Truncation gives us *dynamic granularity*!
- There are defined *continuous* functions for negation, midpoint, infinitary midpoint, truncated addition and multiplication.
- The *modulus of continuity* for these functions tells us how many digits of input we require for each digit of output.

H.-J. Boehm, "Small-data computing: correct calculator arithmetic," *Communications of the ACM*, vol. 60, no. 8, pp. 44–49, 2017.

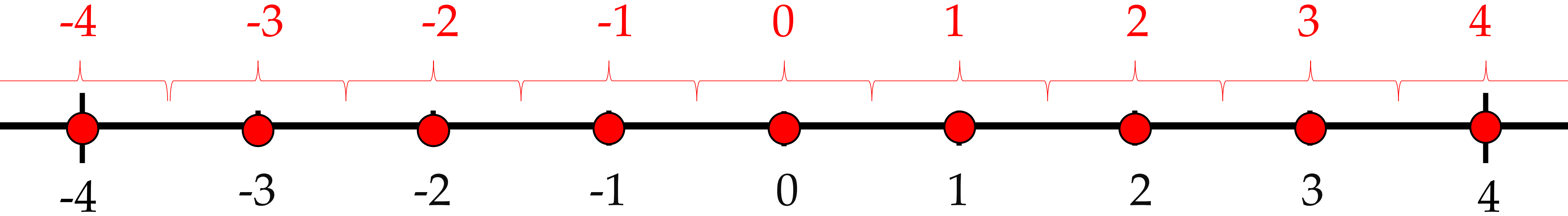
Implementations of Constructive Reals

- **Boehm encodings:** Real numbers are represented as Java objects x of the class CR, which has the method $BigInteger\ approx(int\ n)$ satisfying $|[\![x]\!] - 2^n * x.approx(n)| \leq 2^{n-1}$.
- For example, $PI.approx(-1) = 6$ and $PI.approx(-5) = 101$
  - …but also $THREE.approx(-1) = 6$.
- The precision-level n is used to *dynamically* specify the *granularity*!
- At level n the width of each representational interval is 2^n.



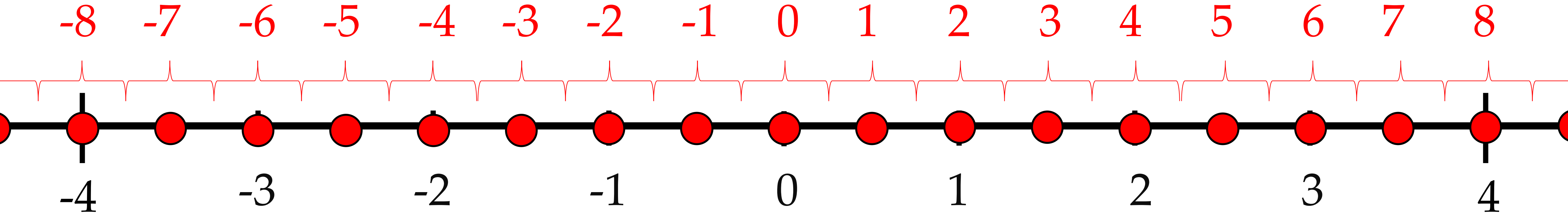
Implementations of Constructive Reals

- **Boehm encodings:** Real numbers are represented as Java objects x of the class CR, which has the method $BigInteger\ approx(int\ n)$ satisfying $|[\![x]\!] - 2^n * x.approx(n)| \leq 2^{n-1}$.
- For example, $PI.approx(-1) = 6$ and $PI.approx(-5) = 101$
  - …but also $THREE.approx(-1) = 6$.
- The precision-level n is used to *dynamically* specify the *granularity*!
- At level 0 the width of each representational interval is 1.



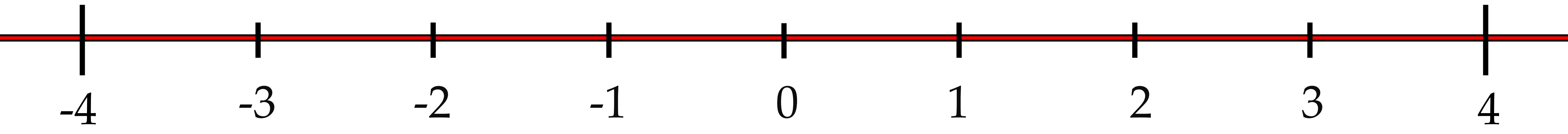
Implementations of Constructive Reals

- **Boehm encodings:** Real numbers are represented as Java objects x of the class CR, which has the method $BigInteger\ approx(int\ n)$ satisfying $|\llbracket x \rrbracket - 2^n * x.approx(n)| \leq 2^{n-1}$.
- For example, $PI.approx(-1) = 6$ and $PI.approx(-5) = 101$
  - …but also $THREE.approx(-1) = 6$.
- The precision-level n is used to *dynamically* specify the *granularity*!
- At level -1 the width of each representational interval is 0.5.



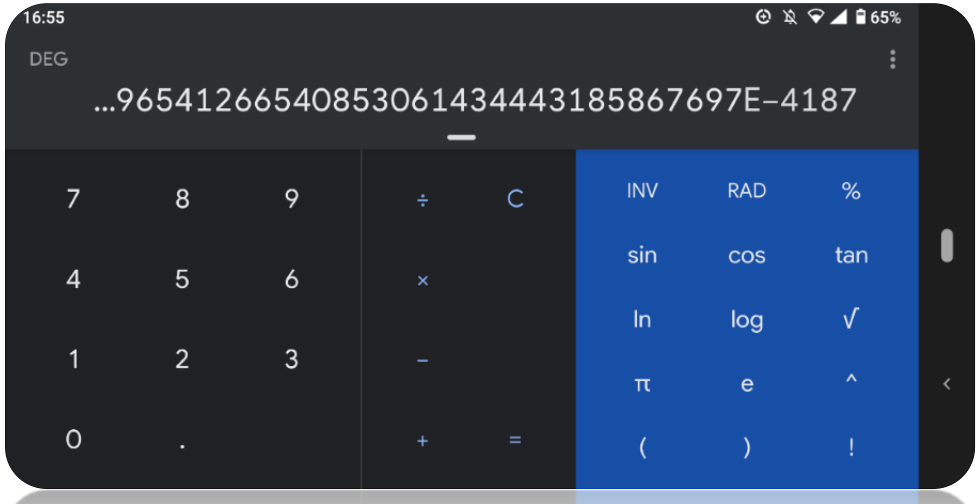
Implementations of Constructive Reals

- **Boehm encodings:** Real numbers are represented as Java objects x of the class CR, which has the method $BigInteger\ approx(int\ n)$ satisfying $|\llbracket x \rrbracket - 2^n * x.approx(n)| \leq 2^{n-1}$.
- For example, $PI.approx(-1) = 6$ and $PI.approx(-5) = 101$
  - …but also $THREE.approx(-1) = 6$.
- The precision-level n is used to *dynamically* specify the *granularity*!
- As the level decreases, the width converges to 0.



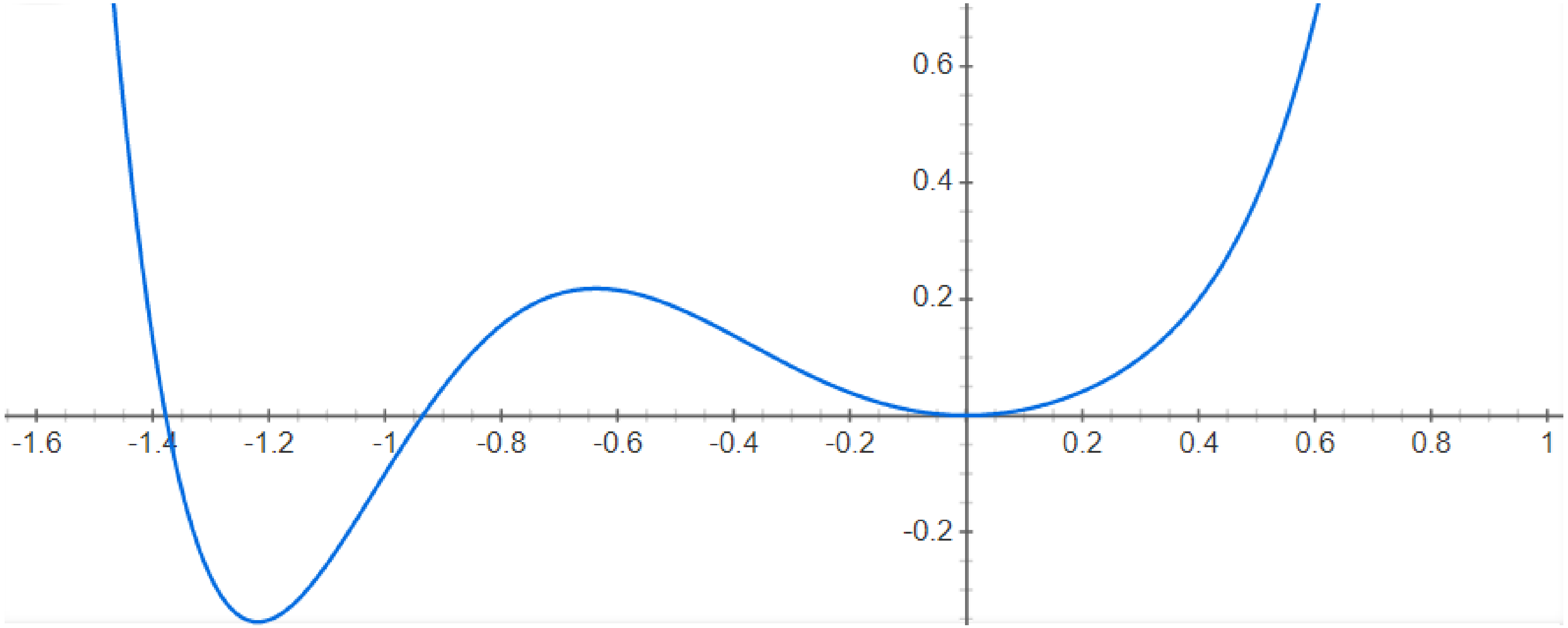
Implementations of Constructive Reals

- **Boehm encodings:** Real numbers are represented as Java objects x of the class CR, which has the method $BigInteger\ approx(int\ n)$ satisfying $|[\![x]\!] - 2^n * x.approx(n)| \leq 2^{n-1}$.
- For example, $PI.approx(-1) = 6$ and $PI.approx(-5) = 101$
  - …but also $THREE.approx(-1) = 6$.
- The precision-level n is used to *dynamically* specify the *granularity*!
- There are defined *continuous* functions for all operations one would expect for a mathematical calculator.
  - We can easily define *modulus of continuity* functions for each of these operations, also on Boehm encodings.



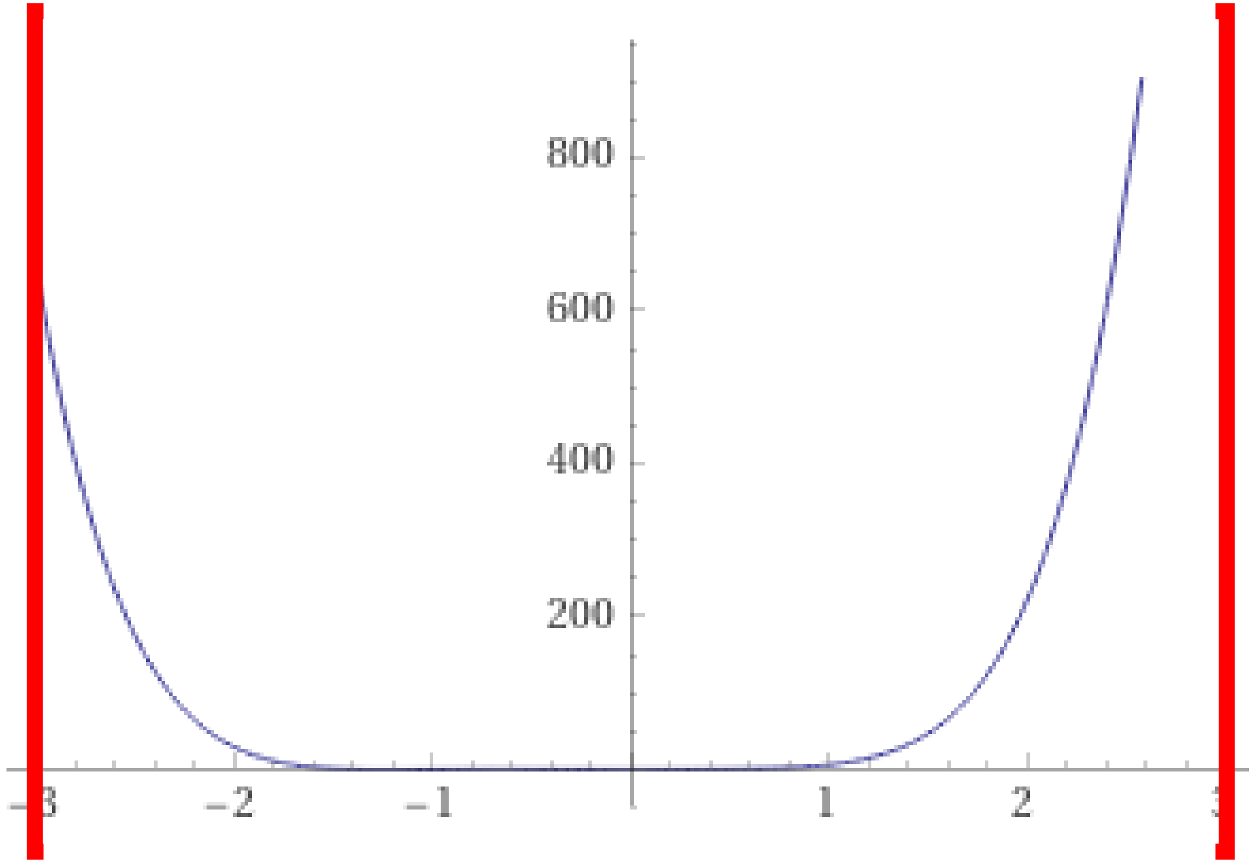
Global Optimisation via Boehm Encodings

$$f = 1.9x^6 + 3x^5 + x^2 \qquad \epsilon = 0.01$$



Global Optimisation via Boehm Encodings

$f = 1.9x^6 + 3x^5 + x^2 \qquad \epsilon = 0.01$

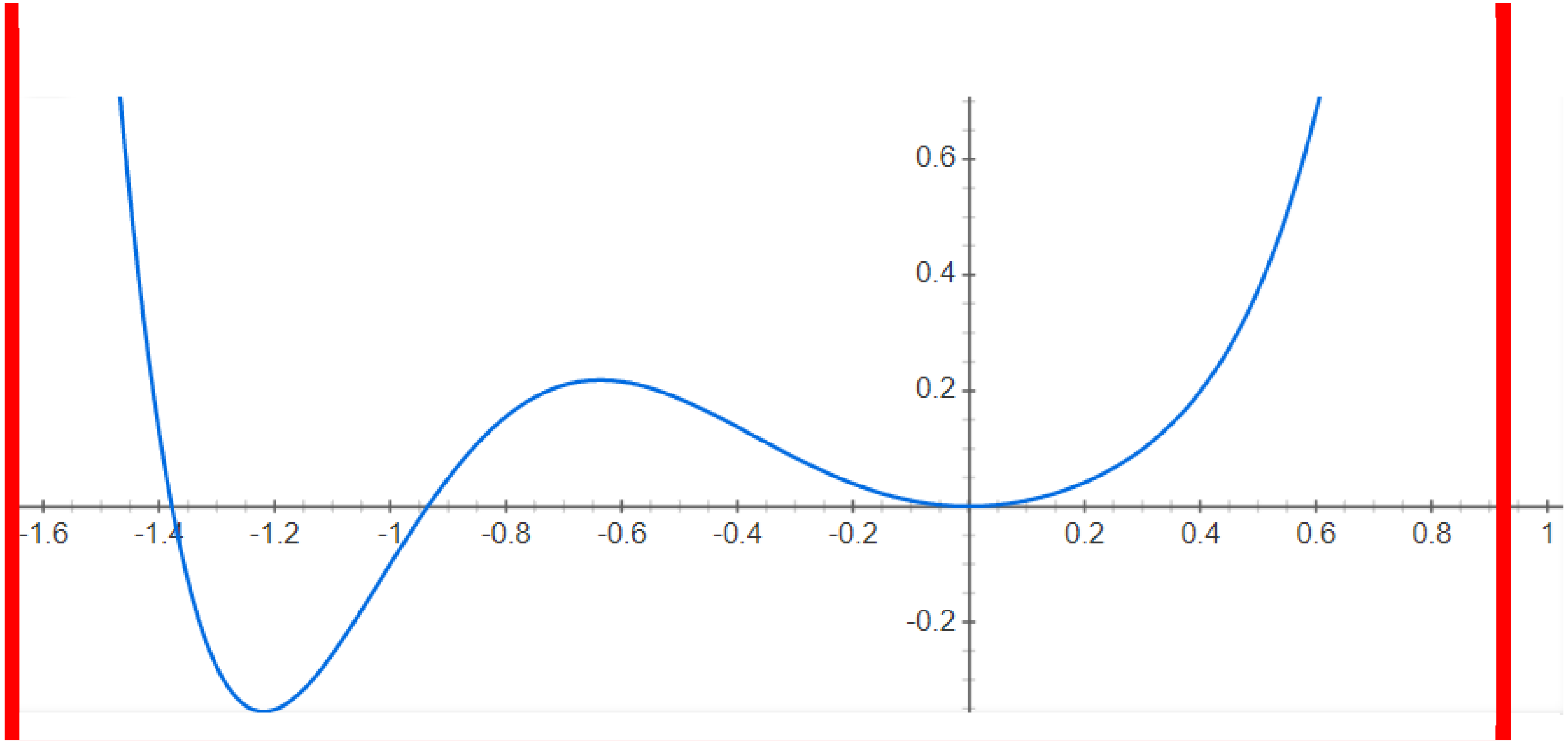


3 candidate intervals

$f([-3,3]) \Rightarrow [-16384,16384]$

Global Optimisation via Boehm Encodings

$f = 1.9x^6 + 3x^5 + x^2 \quad \epsilon = 0.01$

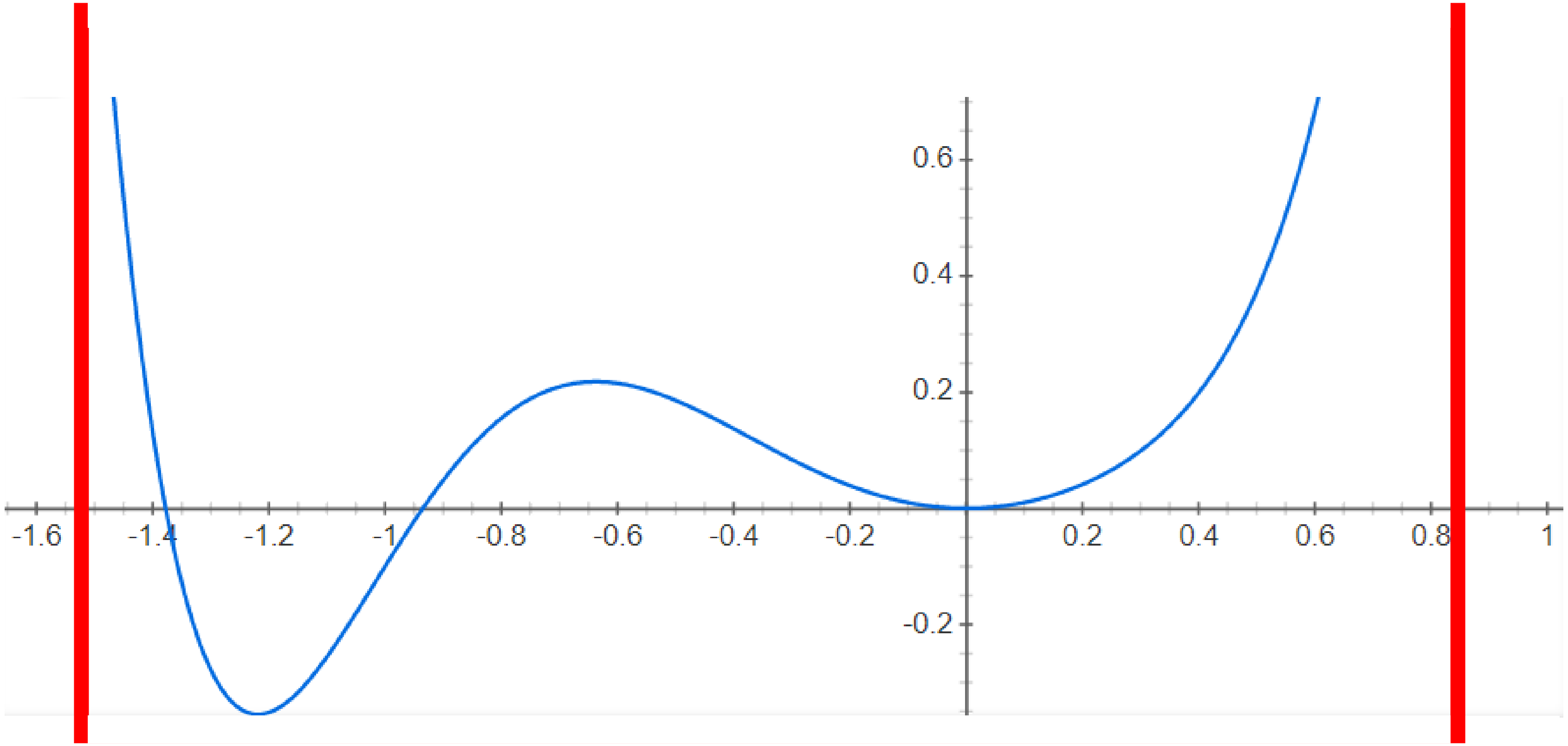


38 candidate intervals

$f([-1.6328125, 0.9375]) \Rightarrow [-4, 4]$

Global Optimisation via Boehm Encodings

$$f = 1.9x^6 + 3x^5 + x^2 \qquad \epsilon = 0.01$$

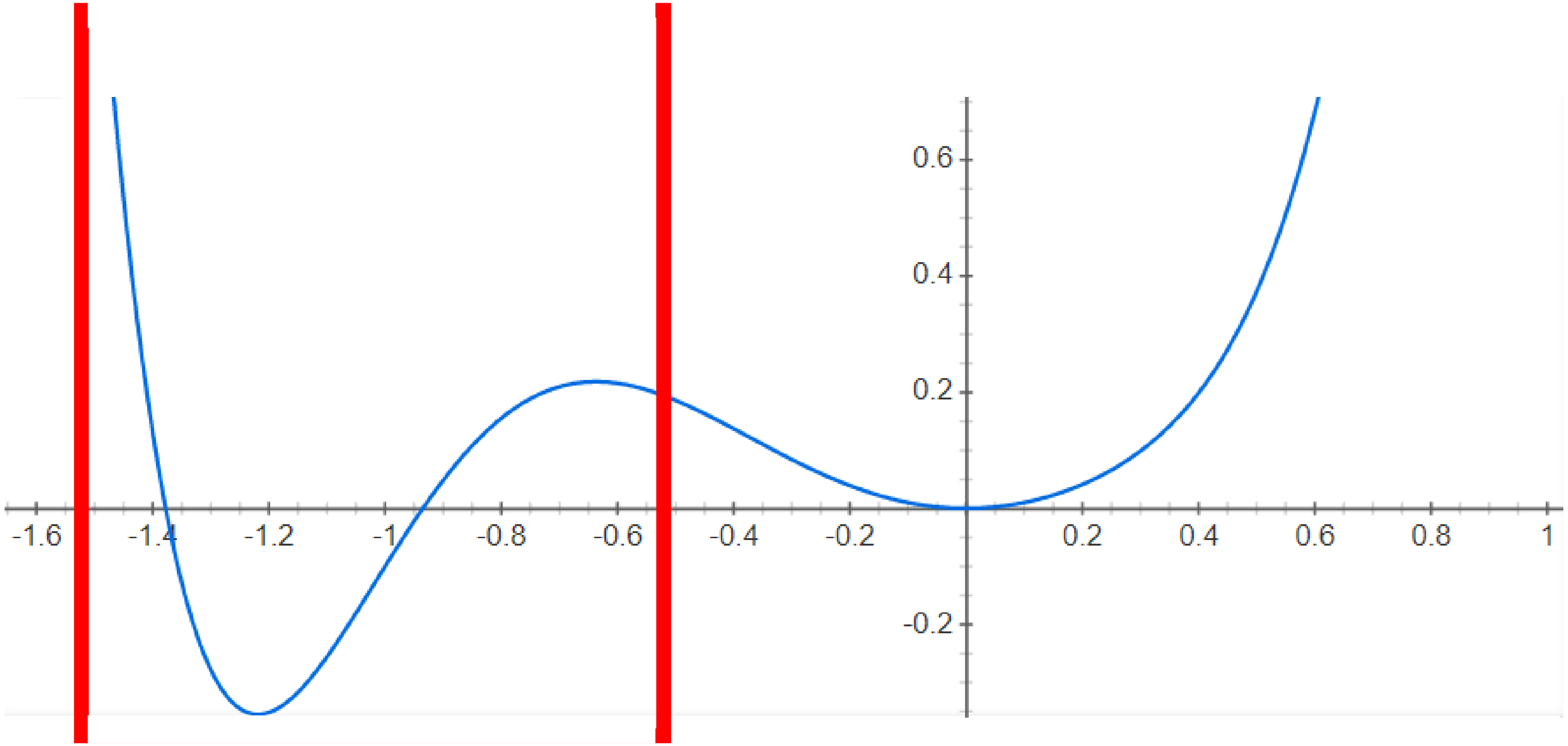


51 candidate intervals

$$f([-1.55078125, 0.8125]) \Rightarrow [-2, 2]$$

Global Optimisation via Boehm Encodings

$$f = 1.9x^6 + 3x^5 + x^2 \quad \epsilon = 0.01$$

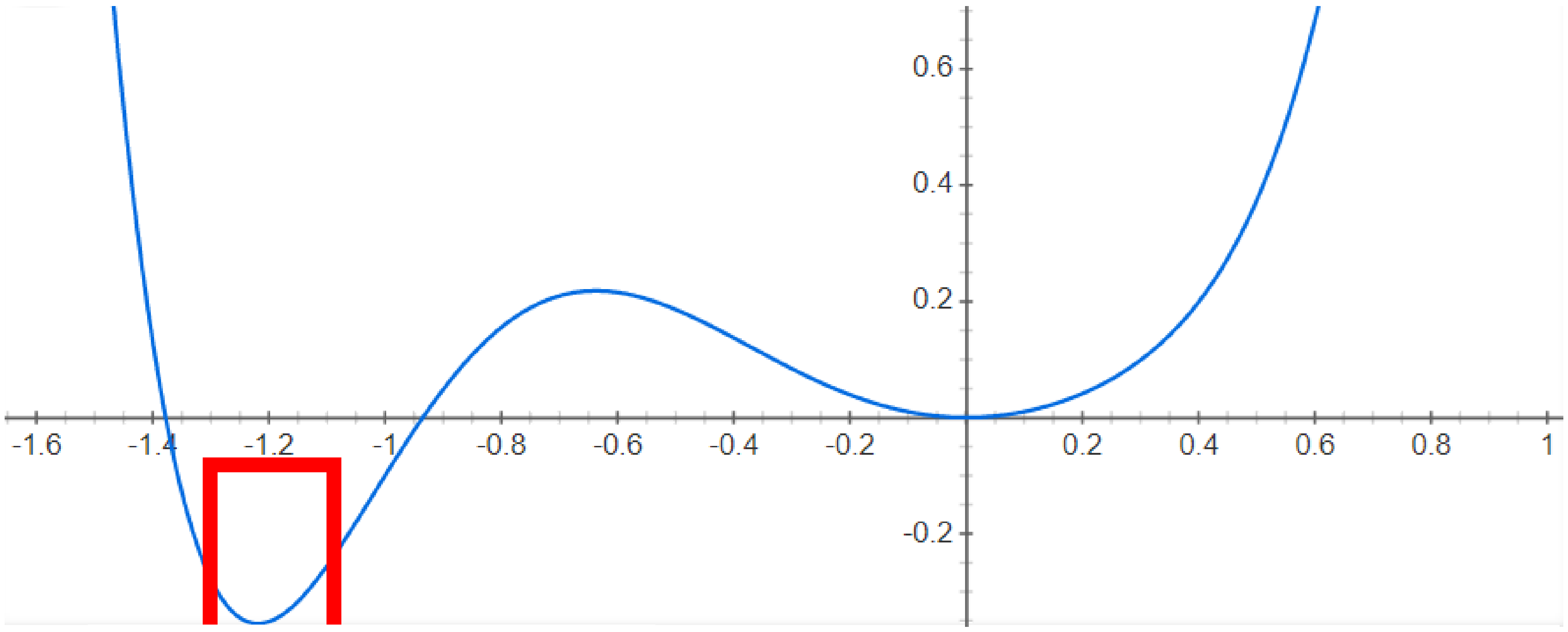


62 candidate intervals

$$f([-1.55078125, -0.5625]) \Rightarrow [-2, 2]$$

Global Optimisation via Boehm Encodings

$$f = 1.9x^6 + 3x^5 + x^2 \qquad \epsilon = 0.01$$

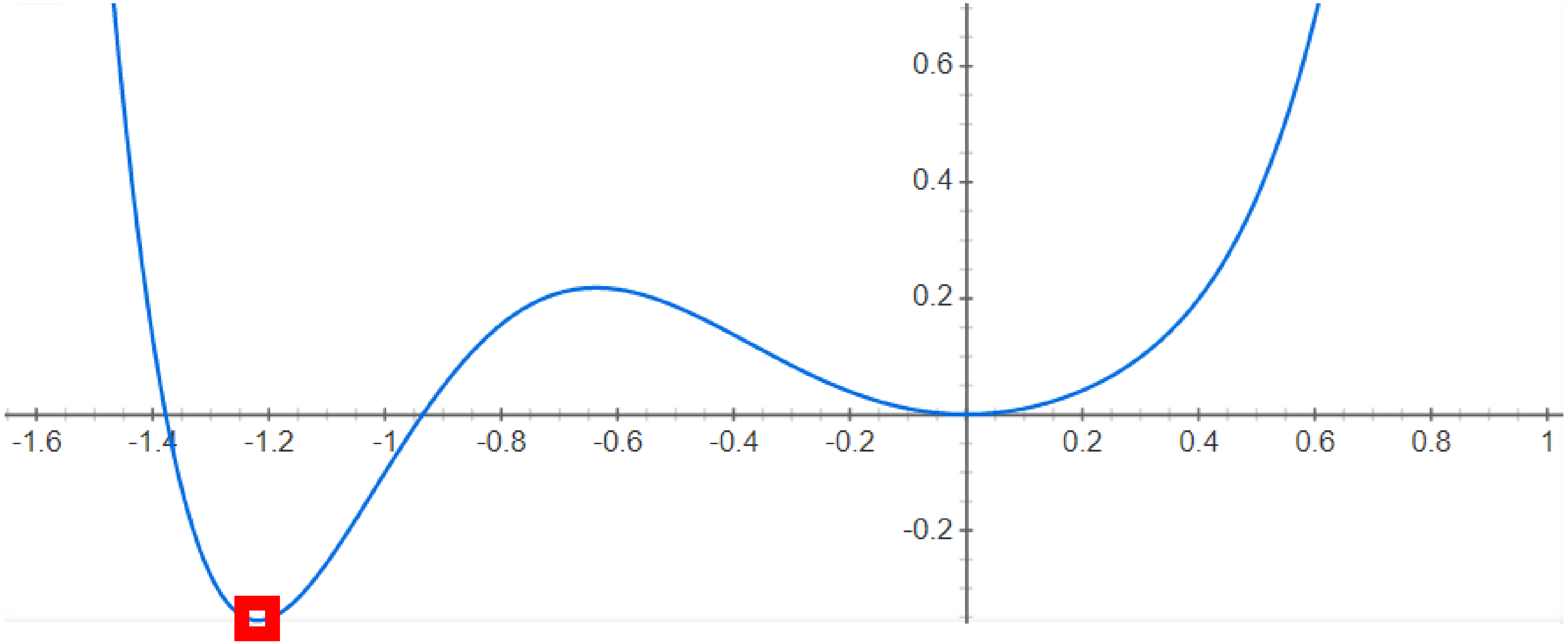


24 candidate intervals

$$f([-1.30859375, -1.1015625]) \Rightarrow [-0.75, -0.125]$$

Global Optimisation via Boehm Encodings

$$f = 1.9x^6 + 3x^5 + x^2 \qquad \epsilon = 0.01$$



315 candidate intervals

$$f([-1.23699951171875, -1.1998291015625]) \Rightarrow [-0.35546875, -0.34765625]$$

M. Escardo, “Infinite sets that admit fast exhaustive search,” in *22nd Annual IEEE Symposium on Logic in Computer Science* (LICS 2007), July 2007, pp. 443–452.

Application to Foundations of Regression

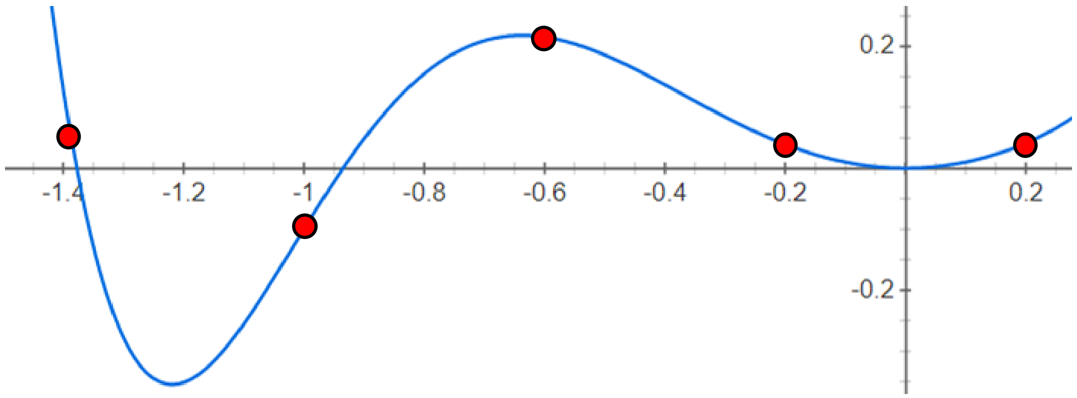
- A data type is called *searchable* if we can construct a *search algorithm* that, given a predicate, returns an element of that type that satisfies the predicate (if such an element exists).
- Every finite type is trivially searchable.
- Perhaps interestingly, some *‘dynamic’* infinite types are searchable on certain *‘continuous’* predicates.
  - Types such as the two we have shown for (compact intervals of) constructive real numbers!
- A predicate is *continuous* if it can be knowingly answered with a given limit on the *granularity* of these types.
  - This essentially allows the type to be searched as if it were finite!

Application to Foundations of Regression

- In function approximation, we wish to compute some **reconstructed function** $f: X \to Y$ via some *data observations* $(x_i, y_i): X \times Y$.
  - The data observations can be seen as coming from some **data oracle** $\Omega : X \to Y$ that may, or may not, be subject to *observation errors*.
- The goal in function approximation is to *minimise the loss*, measured by some **loss function** $L : (X \to Y) \to (X \to Y) \to R$, between the reconstructed function and the data oracle.
- A function approximation process is *convergent* if $\forall \epsilon : R. L(f, \Omega) < \epsilon$.

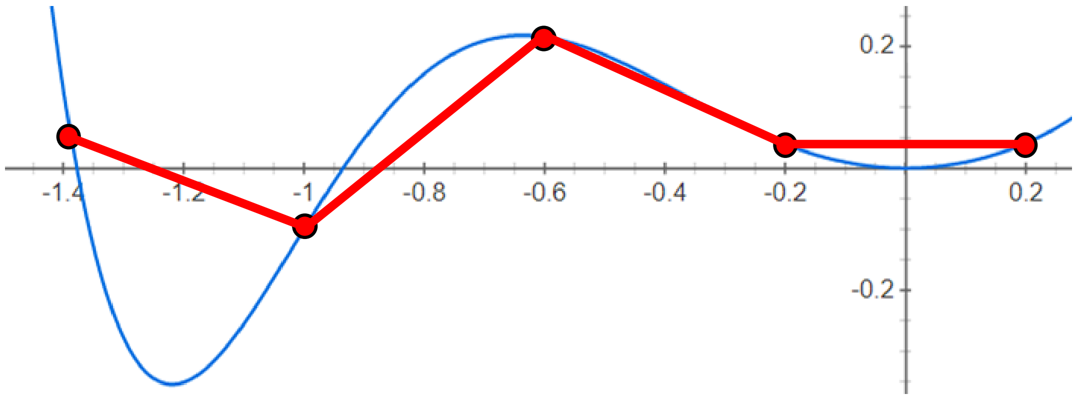
Application to Foundations of Regression

- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.



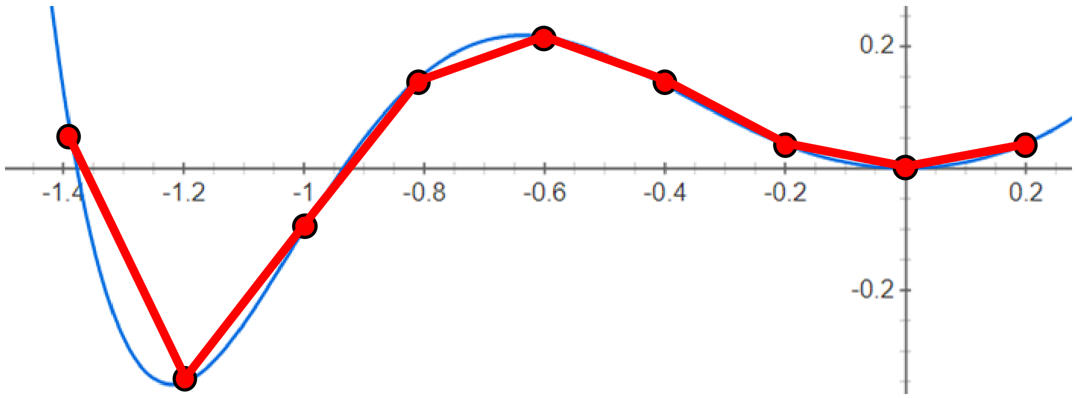
Application to Foundations of Regression

- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.



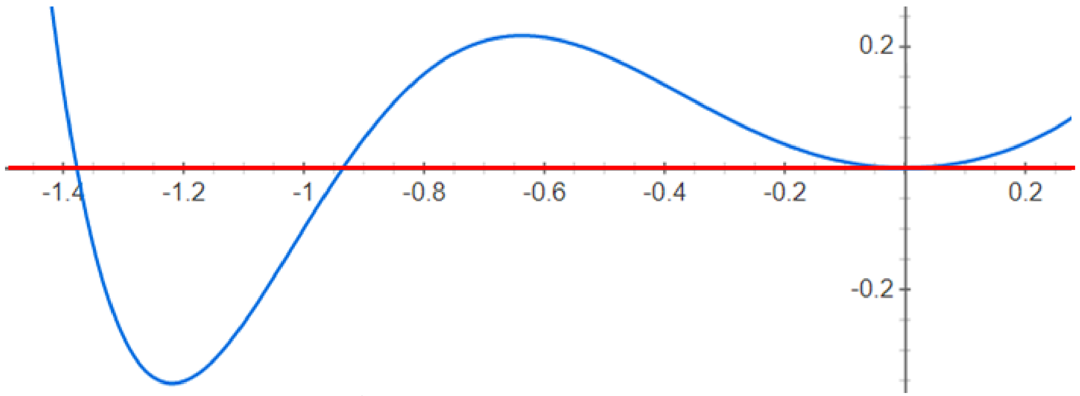
Application to Foundations of Regression

- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f: X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.



Application to Foundations of Regression

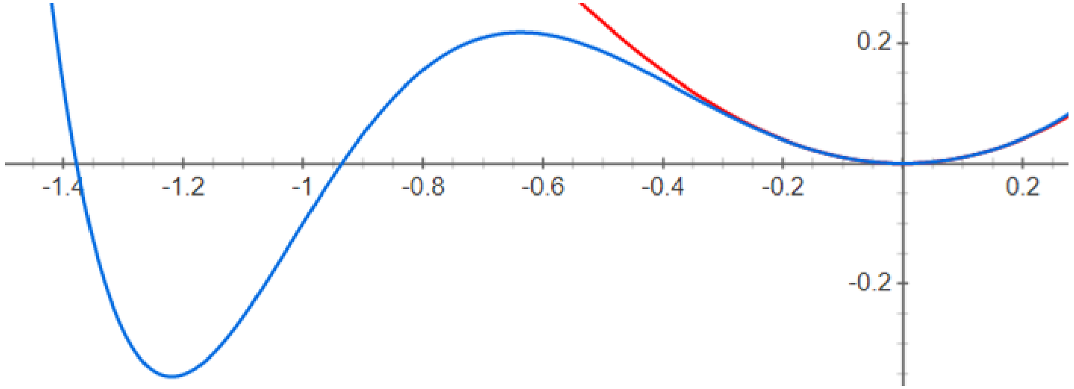
- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f\colon X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.
- Successful regression relies upon the choice of a particular *model function* $M\colon P \to (X \to Y)$.



$\lambda a.\, b.\, c.\, \lambda x.\, ax^6 + bx^5 + cx^2 : R^3 \to (R \to R)$ $0x^6 + 0x^5 + 0x^2$

Application to Foundations of Regression

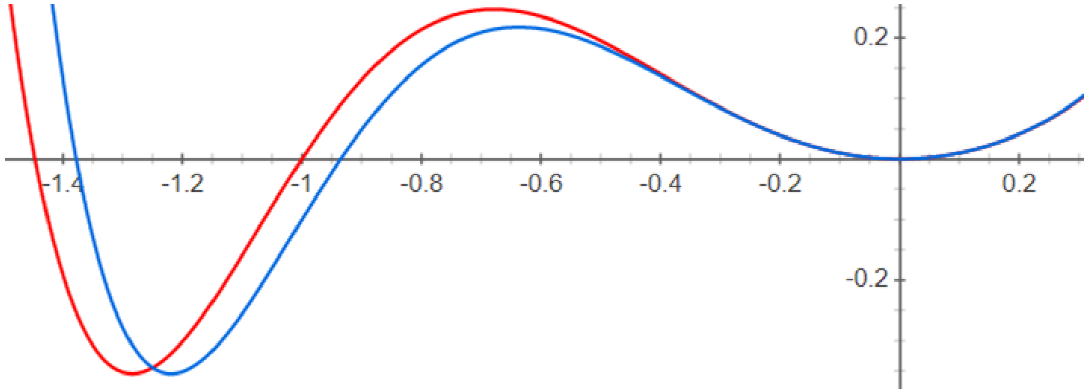
- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.
- Successful regression relies upon the choice of a particular *model function* $M : P \to (X \to Y)$.



$\lambda a.b.c.\lambda x.ax^6 + bx^5 + cx^2 : R^3 \to (R \to R)$ $\quad 2x^6 + 2x^5 + 1x^2$

Application to Foundations of Regression

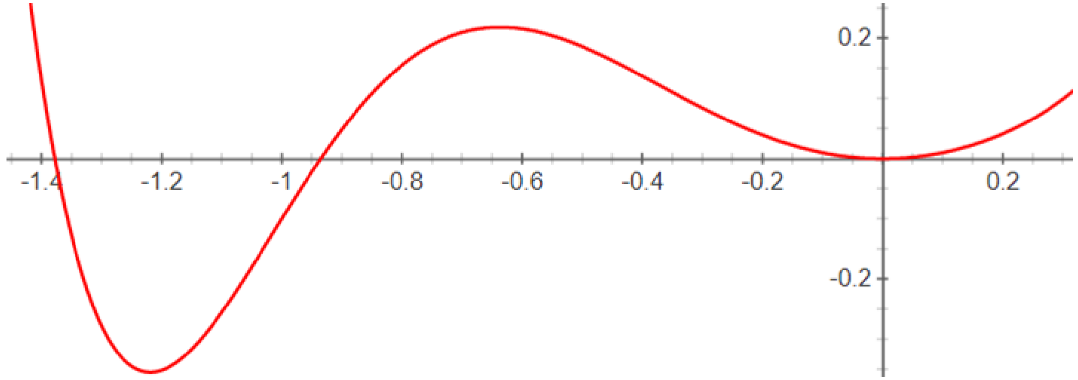
- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.
- Successful regression relies upon the choice of a particular *model function* $M : P \to (X \to Y)$.



$\lambda a.\, b.\, c.\, \lambda x.\, ax^6 + bx^5 + cx^2 : R^3 \to (R \to R)$ $1.5x^6 + 2.5x^5 + 1x^2$

Application to Foundations of Regression

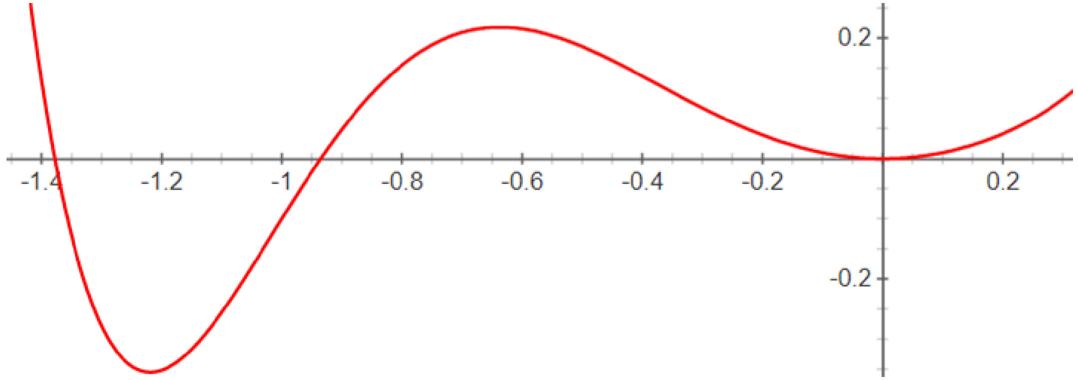
- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.
- Successful regression relies upon the choice of a particular *model function* $M : P \to (X \to Y)$.



$\lambda a.b.c.\lambda x.ax^6 + bx^5 + cx^2 : R^3 \to (R \to R)$ $1.9x^6 + 3x^5 + 1x^2$

Application to Foundations of Regression

- Function approximation is convergent if, for any $\epsilon : R$, the constructed $f : X \to Y$ minimises the loss, i.e. $L(f, \Omega) < \epsilon$.
- Two function approximation processes: **interpolation** vs. **regression**.
  - Convergence properties of interpolation are well-studied.
- Successful regression relies upon the choice of a particular *model function* $M : P \to (X \to Y)$.
- We are minimising the function



$$\lambda a, b, c.\, L\begin{pmatrix}\lambda x.\, ax^6 + bx^5 + cx^2, \\ \lambda x.\, 1.9x^6 + 3x^5 + x^2\end{pmatrix} : R^3 \to R$$

$1.9x^6 + 3x^5 + 1x^2$

- Convergent regression is convergent global optimisation!

Conclusions and Future Work

- Huge investment in local optimisation algorithms via gradient descent
  - Fantastic, efficient algorithms; as well as dedicated hardware
- But sometimes finding the best solution to a problem is important
  - Further improvements to local optimisation will not take us to global
- We have introduced a different line of work: **convergent global optimisation via constructive real numbers**
  - Floating-point numbers are unsuitable
- This line of work has promise
  - The theoretical guarantees can be established mathematically and applied to foundational questions, such as convergent regression
- The algorithms require a lot of work – but this work could be worthwhile