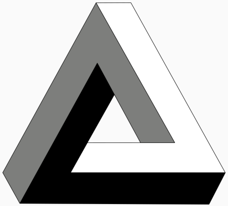
Stating the Obvious

boxy

September 2022

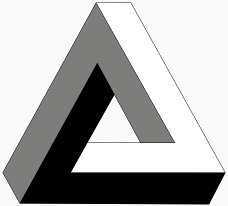
Paradoxes and Fallacies

- What is the smallest positive integer not definable in under hundred letters?



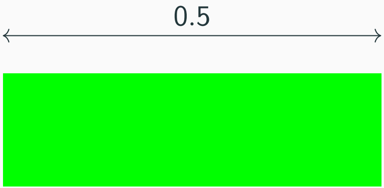
Paradoxes and Fallacies

- What is the smallest positive integer not definable in under hundred letters?
- It is impossible to run any distance



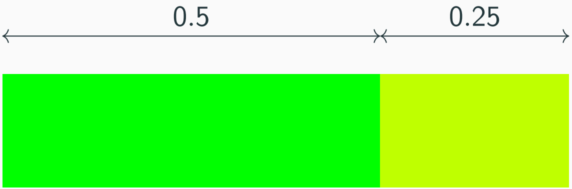
Zeno's Paradox

First run half a meter.



Zeno's Paradox

Then a quarter meter.



Zeno's Paradox

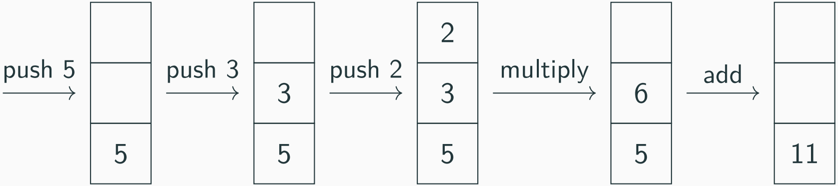
And so on.

Project Background

Programming Languages

Calculating $5 + 3 * 2$.

Calculating $5 + 3 * 2$.



Programming Languages

```
static void cmd_backup(struct userrec *u, int idx, char *par)
{
        putlog(LOG_CMDS, "*", "#%s# backup", dcc[idx].nick);
        dprintf(idx, "Backing up the channel & user files...\n");
        call_hook(HOOK_BACKUP);
}
```

```haskell
powerset = filterM (\_ -> [True, False])
```

The Obvious

> Some of our fellow sinners are among the most careful and competent logicians on the contemporary scene.
>
> Haskell B. Curry and Robert Feys

The Obvious

- “Roses are red and violets are blue” is logically equivalent to “Violets are blue and roses are red”

> Some of our fellow sinners are among the most careful and competent logicians on the contemporary scene.
>
> Haskell B. Curry and Robert Feys

The Obvious

- "Roses are red and violets are blue" is logically equivalent to "Violets are blue and roses are red"
- $2 + 3 = 5$

> Some of our fellow sinners are among the most careful and competent logicians on the contemporary scene.
>
> Haskell B. Curry and Robert Feys

The Obvious

- "Roses are red and violets are blue" is logically equivalent to "Violets are blue and roses are red"
- $2 + 3 = 5$
- $a + b = b + a$

> Some of our fellow sinners are among the most careful and competent logicians on the contemporary scene.
>
> Haskell B. Curry and Robert Feys

The Obvious

- "Roses are red and violets are blue" is logically equivalent to "Violets are blue and roses are red"
- $2 + 3 = 5$
- $a + b = b + a$
- $1 \neq 2$

> Some of our fellow sinners are among the most careful and competent logicians on the contemporary scene.
>
> Haskell B. Curry and Robert Feys

Applications

Applications

- New languages can reduce labour

Applications

- New languages can reduce labour
- New languages and theorem provers can eliminate bugs

Applications

- New languages can reduce labour
- New languages and theorem provers can eliminate bugs
  - \$59 billion loss per year in US

Applications

- New languages can reduce labour
- New languages and theorem provers can eliminate bugs
  - $59 billion loss per year in US
  - Death

Foundations of Mathematics

Meta-Languages

Meta-Languages

$$\frac{A \quad B}{A \wedge B}\ (\wedge I)$$

$$\frac{A \wedge B}{A}\ (\wedge E_1)$$

$$\frac{A \wedge B}{B}\ (\wedge E_2)$$

Meta-Languages

$$\frac{A \qquad B}{A \wedge B}\ (\wedge I)$$

$$\frac{A \wedge B}{A}\ (\wedge E_1)$$

$$\frac{A \wedge B}{B}\ (\wedge E_2)$$

$$\dfrac{\dfrac{A \wedge B}{B}\ (\wedge E_2) \qquad \dfrac{A \wedge B}{A}\ (\wedge E_2)}{B \wedge A}\ (\wedge I)$$

λ-Calculi

Proof languages are often based off the λ-calculus.

λ-Calculi

Proof languages are often based off the λ-calculus.

$$T = \underbrace{\lambda x.}_{\text{Abstraction}} \lambda y.x$$

$$F = \lambda x.\lambda y.y$$

$$N = \lambda b.\ \underbrace{b\ F\ T}_{\text{Application}}$$

λ-Calculi

Proof languages are often based off the λ-calculus.

$$
\begin{aligned}
& (\lambda b.b\ F\ T)(\lambda x.\lambda y.y) \\
\rightarrow\ & (\lambda x.\lambda y.y)\ F\ T \\
\rightarrow\ & T
\end{aligned}
$$

The Language

Formal Language Theory

Formal Language Theory

$$\langle \mathit{add} \rangle \rightarrow\ <\text{var}>\ \text{"+"}\ <\text{add}>\ |\ <\text{mult}>$$

$$\langle \mathit{mult} \rangle \rightarrow\ <\text{var}>\ \text{"*"}\ <\text{mult}>\ |\ \text{"("}\ <\text{add}>\ \text{")"}\ |\ <\text{var}>$$

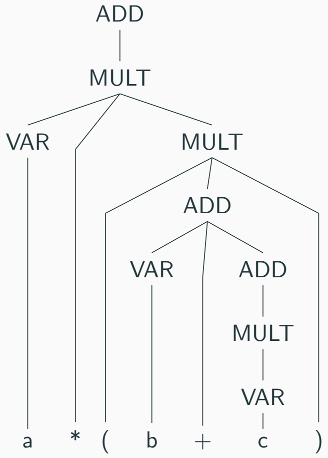
$$\langle \mathit{var} \rangle \rightarrow\ \text{"a"}\ |\ \text{"b"}\ |\ \text{"c"}$$

Formal Language Theory

⟨*add*⟩ → <var> "+" <add> | <mult>

⟨*mult*⟩ → <var> "*" <mult> | "(" <add> ")" | <var>

⟨*var*⟩ → "a" | "b" | "c"

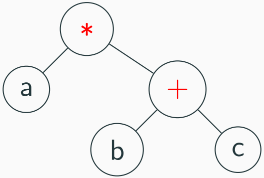


Formal Language Theory

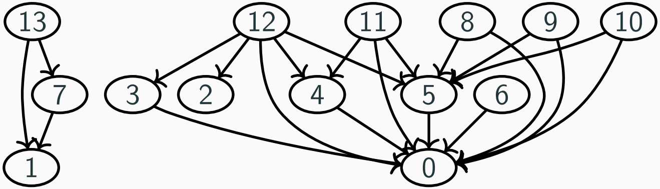
$$\langle \mathit{add} \rangle \rightarrow \text{<var> "+" <add> | <mult>}$$

$$\langle \mathit{mult} \rangle \rightarrow \text{<var> "*" <mult> | "(" <add> ")" | <var>}$$

$$\langle \mathit{var} \rangle \rightarrow \text{"a" | "b" | "c"}$$

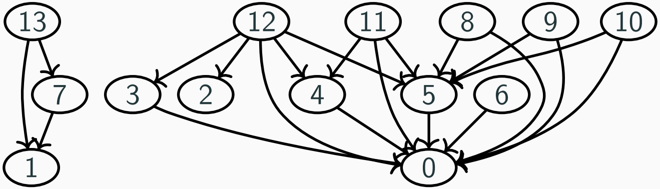


Implementation Details



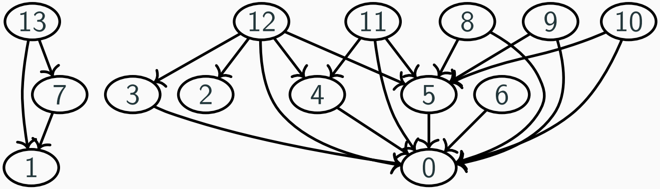
Implementation Details

- The language is parsed



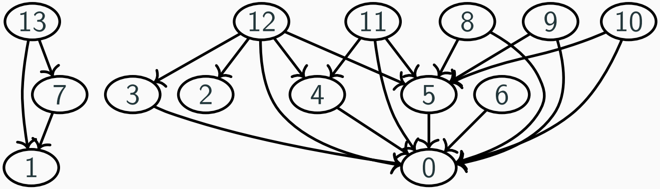
Implementation Details

- The language is parsed
- Definitions across files are resolved



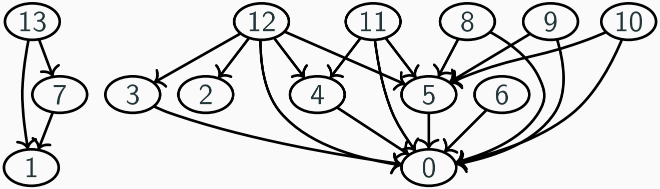
Implementation Details

- The language is parsed
- Definitions across files are resolved
- Definitions are topologically sorted



Implementation Details

- The language is parsed
- Definitions across files are resolved
- Definitions are topologically sorted
- Type-checking is done



One is Not Two

Logic definitions:

```
bottom = forall a : Prop, a;
not = fun a : Prop => forall b : a, bottom;
```

One is Not Two

Logic definitions:

```
bottom = forall a : Prop, a;
not = fun a : Prop => forall b : a, bottom;
```

Definition of natural numbers:

```
nat : Set;
0 : nat;
succ : forall a : nat, nat;
1 = succ 0; 2 = succ 1;
```

One is Not Two

Axioms:

```
equal : forall a : nat, forall b : nat, Prop;
lower : forall a : nat,
        forall b : nat,
        forall p : equal (succ a) (succ b), equal a b;
0_ne_succ : forall a : nat, not (equal 0 (succ a));
```

One is Not Two

The proof:

```
1_ne_2 :  not (equal 1 2)
       =  fun p : equal 1 2
       => let 0_eq_1 = lower 0 1 p
       in 0_ne_succ 0 0_eq_1;
```

Tada!

The point of philosophy is to start with something so simple as not to seem worth stating, and to end with something so paradoxical that no one will believe it.

Bertrand Russell