Data Types with Negation

Robert Atkey
University of Strathclyde
robert.atkey@strath.ac.uk

Fun in the REPL
1st November 2023

Funded by EPSRC: EP/T026960/1 *AISEC: AI Secure and Explainable by Construction*

```
data Nat : Set where
  zero : Nat
  succ : Nat → Nat
```

```
data Even : Nat → Set where
  zero    : Even zero
  succOdd : ∀{n} → Odd n → Even (succ n)

data Odd : Nat → Set where
  succEven : ∀{n} → Even n → Odd (succ n)
```

What if we could use the power of *negative thinking*?

```
data Even : Nat → Set where
  zero : Even zero
  succ : ∀{n} → not (Even n) → Even (succ n)
```

Error: Non strictly positive occurrence of "Even" in `"forall n : nat, not (Even n) -> Even (S n)"`.

Even is not strictly positive, because it occurs
to the left of an arrow
in the type of the constructor `succ`
in the definition of Even.

Making choices

A Context-Free Grammar:

$$S ::= aSa \mid aa$$

A Context-Free Grammar:

$$S ::= aSa \mid aa$$

What is the language $\mathcal{L}(S)$?

A Context-Free Grammar:

$$S ::= aSa \mid aa$$

What is the language $\mathcal{L}(S)$?

$$\mathcal{L}(S) = \{a^{2k} \mid k \geq 1\}$$

A Parsing Expression Grammar (PEG):

$$S ::= aSa \; / \; aa$$

A Parsing Expression Grammar (PEG):

$$S ::= aSa \ / \ aa$$

What is the language $\mathcal{L}(S)$?

A Parsing Expression Grammar (PEG):

$$S ::= aSa \;/\; aa$$

What is the language $\mathcal{L}(S)$?

$$\mathcal{L}(S) = \{a^{2^k} \mid k \geq 1\}$$

What is the parse tree of a PEG?

Unordered choice:

$$S ::= A \mid B$$

Parse trees:

$$
\begin{aligned}
&\textsf{data}\ \mathsf{S} : \mathsf{String} \times \mathsf{String} \to \mathsf{Set}\ \textsf{where}\\
&\quad \textsf{prod1} : \forall \{i\ o\} \to \mathsf{A}(i, o) \to \mathsf{S}(i, o)\\
&\quad \textsf{prod2} : \forall \{i\ o\} \to \mathsf{B}(i, o) \to \mathsf{S}(i, o)
\end{aligned}
$$

Ordered choice:

$$S ::= A \;/\; B$$

Parse trees:

$$\begin{array}{l}
\textsf{data}\ \mathsf{S} : \mathsf{String} \times \mathsf{String} \to \mathsf{Set}\ \textsf{where} \\
\quad \mathsf{prod1} : \forall \{i\ o\} \to \mathsf{A}(i, o) \to \mathsf{S}(i, o) \\
\quad \mathsf{prod2} : \forall \{i\ o\} \to \mathsf{not}\ (\exists o'.\, \mathsf{A}(i, o')) \to \mathsf{B}(i, o) \to \mathsf{S}(i, o)
\end{array}$$

Ordered choice:

$$S ::= A \;/\; B$$

Parse trees:

$$\begin{aligned}
&\mathsf{data}\ \mathrm{S} : \mathrm{String} \times \mathrm{String} \to \mathrm{Set}\ \mathsf{where}\\
&\quad \mathsf{prod1} : \forall\{i\ o\} \to \mathrm{A}(i, o) \to \mathrm{S}(i, o)\\
&\quad \mathsf{prod2} : \forall\{i\ o\} \to \mathrm{not}\ (\exists o'.\ \mathrm{A}(i, o')) \to \mathrm{B}(i, o) \to \mathrm{S}(i, o)
\end{aligned}$$

The prod2 constructor is only available if the first one didn't work.

Default reasoning

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{Guilty} : \mathrm{Person} \longrightarrow \mathrm{Set}\ \mathsf{where} \\
&\quad \mathsf{murder} : \mathrm{KilledSomeone}\ p \longrightarrow \mathrm{not}\ (\mathrm{StateApproved}\ p) \longrightarrow \mathrm{Guilty}\ p \\
&\quad \ldots
\end{aligned}
$$

```
data Guilty : Person → Set where
  murder : KilledSomeone p → not (StateApproved p) → Guilty p
  ...

data StateApproved : Person → Set where
  driver : CarDriver p → StateApproved p
  ...
```

```
data Liar : Set where
  liar : not Liar → Liar
```

```
data Liar : Set where
  liar : not Liar → Liar
```

?

Qn: *Can we give a "sensible" semantics to Data Types with Negation?*

Qn: *Can we give a "sensible" semantics to Data Types with Negation?*

Plan:

1. Study the semantics of (co)inductive data types
2. Work out what we mean by "not"
3. Put the two together (using some logic programming ideas)

Inductive Data Types

A Data Type:

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{D} : I \longrightarrow \mathrm{Set}\ \mathsf{where}\\
&\quad \mathsf{c}_1 : \forall \vec{x}.\ A_1 \longrightarrow D(\vec{e_1}) \longrightarrow D(e'_1)\\
&\qquad \ldots\\
&\quad \mathsf{c}_n : \forall \vec{x}.\ A_n \longrightarrow D(\vec{e_n}) \longrightarrow D(e'_n)
\end{aligned}
$$

is modelled as a functor $F_{\mathrm{D}} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$:

$$F_{\mathrm{D}}(X) = \lambda i. \coprod_j \left(\Sigma \vec{x}.\ A_j \times X(\vec{e_j}) \times [i = e'_j]\right)$$

With a functor

$$F_{\mathrm{D}} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$$

With a functor

$$F_{\mathrm{D}} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$$

We can ask for its *initial algebra*

$$in : F_{\mathrm{D}}(\mu F_{\mathrm{D}}) \longrightarrow \mu F_{\mathrm{D}}$$

With a functor

$$F_{\mathrm{D}} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$$

We can ask for its *initial algebra*

$$in : F_{\mathrm{D}}(\mu F_{\mathrm{D}}) \longrightarrow \mu F_{\mathrm{D}}$$

Which gives us:

1. $\mu F_{\mathrm{D}} : I \longrightarrow \mathrm{Set}$, a carrier
2. a recursion scheme for eliminating $x : \mu F_{\mathrm{D}}\ i$
3. induction principles
   with a bit of work: Hermida and Jacobs, Inf.&Comp. 1998

Construction:

$$0 \longrightarrow F_{\mathrm{D}}\ 0 \longrightarrow F_{\mathrm{D}}(F_{\mathrm{D}}\ 0) \longrightarrow F_{\mathrm{D}}(F_{\mathrm{D}}(F_{\mathrm{D}}\ 0)) \longrightarrow \ldots$$

$$\mu F_{\mathrm{D}} \approx \Sigma n.\ F_{\mathrm{D}}^{n}$$

...up to some quotienting.

* terms and conditions apply

Coinductive Data Types

Given a functor

$$F_{\mathrm{D}} : (I \to \mathrm{Set}) \to (I \to \mathrm{Set})$$

Given a functor

$$F_{\mathrm{D}} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$$

We can also ask for its *final coalgebra*

$$out : \nu F_{\mathrm{D}} \longrightarrow F_{\mathrm{D}}(\nu F_{\mathrm{D}})$$

Which gives us:

1. $\nu F_{\mathrm{D}} : I \longrightarrow \mathrm{Set}$, a carrier
2. a corecursion scheme, for building $x : \nu F_{\mathrm{D}}\ i$
3. coinduction / bisimulation principles
   with a bit of work: Hermida and Jacobs, Inf.&Comp. 1998

Construction:

$$1 \leftarrow F_{\mathrm{D}}\ 1 \leftarrow F_{\mathrm{D}}(F_{\mathrm{D}}\ 1) \leftarrow F_{\mathrm{D}}(F_{\mathrm{D}}(F_{\mathrm{D}}\ 1)) \leftarrow \ldots$$

$$\nu F_{\mathrm{D}} \approx \prod n.\ F_{\mathrm{D}}^{n}$$

...up to some naturality condition.

* terms and conditions apply

Negation

Negation, what is it?

$$\mathsf{not}\ A \stackrel{def}{=} A \longrightarrow \bot$$

$$\mathsf{not}\, A \stackrel{def}{=} A \to \bot$$

?

$$\mathsf{not}\ A \stackrel{def}{=} A \to \bot$$

?

Not covariant: don't get a functor!

Initial algebra and final coalgebra not well defined!

What do we mean when we say “not”?

1. “This leads to a contradiction”

$$\mathsf{not}\, A \overset{def}{=} A \longrightarrow \bot$$

1. “This leads to a contradiction”

$$\text{not } A \stackrel{\mathit{def}}{=} A \longrightarrow \perp$$

2. “I do not have any evidence to believe this”

1. “This leads to a contradiction”

$$\text{not}\,A \overset{def}{=} A \rightarrow \perp$$

2. “I do not have any evidence to believe this”

$$\begin{aligned}&\text{files}(X) \leftarrow \text{bird}(X), \text{not penguin}(X).\\&\text{bird}(\text{tweety}).\end{aligned}$$

1. “This leads to a contradiction”

$$\text{not}\, A \stackrel{def}{=} A \rightarrow \bot$$

2. “I do not have any evidence to believe this”

$$\begin{aligned}&\text{files}(X) \leftarrow \text{bird}(X), \text{not}\,\text{penguin}(X).\\&\text{bird}(\text{tweety}).\end{aligned}$$

Does “tweety” fly?

1. "This leads to a contradiction"

$$\text{not}\ A \overset{def}{=} A \rightarrow \bot$$

2. "I do not have any evidence to believe this"

$$\begin{aligned}&\text{files}(X) \leftarrow \text{bird}(X), \text{not}\ \text{penguin}(X).\\ &\text{bird}(\text{tweety}).\end{aligned}$$

Does "tweety" fly?

Classically (and intuitionistically): No.

1. "This leads to a contradiction"

$$\text{not}\, A \overset{def}{=} A \rightarrow \bot$$

2. "I do not have any evidence to believe this"

$$\begin{aligned}&\text{files}(X) \leftarrow \text{bird}(X), \text{not}\, \text{penguin}(X).\\&\text{bird}(\text{tweety}).\end{aligned}$$

Does "tweety" fly?

Classically (and intuitionistically): No. Intuitively – Yes?

1. “This leads to a contradiction”

$$\text{not } A \stackrel{def}{=} A \to \bot$$

2. “I do not have any evidence to believe this”

$$\begin{aligned}&\text{files}(X) \leftarrow \text{bird}(X), \text{not penguin}(X).\\&\text{bird}(\text{tweety}).\end{aligned}$$

Does “tweety” fly?

Classically (and intuitionistically): No. Intuitively – Yes?

Warning this kind of reasoning is *non monotonic*. If we later learn penguin(tweety), then we would have to retract.

A way to track evidence *for* and evidence *against* some data:

$$A = (A^+, A^-)$$

where A^+, A^- are (indexed) sets.

We could add condition $A^+ \to A^- \to \perp$, but we'll ignore this here.

Entailment flows forwards positively and backwards negatively:

$$(A^+, A^-) \Rightarrow (B^+, B^-) \stackrel{def}{=} (A^+ \longrightarrow B^+) \times (B^- \longrightarrow A^-)$$

call this category Chu and the morphisms truth morphisms.

Terminal Object ($\top$ / truth)

$$\top = (\top, \bot)$$

Initial Object ($\bot$ / falsity)

$$\bot = (\bot, \top)$$

Products (conjunction)

$$(A^+, A^-) \times (B^+, B^-) \overset{def}{=} (A^+ \times B^+, A^- + B^-)$$

Coproducts (disjunction)

$$(A^+, A^-) + (B^+, B^-) \overset{def}{=} (A^+ + B^+, A^- \times B^-)$$

Π-types (infinitary conjunction)

$$\Pi x{:}X.\ A[x] \stackrel{def}{=} (\Pi x : X.\ A^{+}[x], \Sigma x : X.\ A^{-}[x])$$

Σ-types (infinitary disjunction)

$$\Sigma x{:}X.\ A[x] \stackrel{def}{=} (\Sigma x : X.\ A^{+}[x], \Pi x : X.\ A^{-}[x])$$

Negation

$$\mathrm{not}\,(A^+, A^-) \stackrel{def}{=} (A^-, A^+)$$

Sets

$$[X] = (X, X \to \bot)$$

Properties of Chu and Negation:

1. Involutive: $\text{not}\ (\text{not}\ A) = A$
2. de Morgan: $\text{not}(A \times B) = (\text{not}\ A) + (\text{not}\ B)$
3. do not have excluded middle
4. Not Cartesian Closed

not a model of classical logic

is a model of classical linear logic

Initial Algebras in Chu

If we have a functor

$$F\colon (I \longrightarrow \mathrm{Chu}) \longrightarrow (I \longrightarrow \mathrm{Chu})$$

constructed from only $\times, +, \Pi, \Sigma, [-]$ then it can be separated:

$$F^{+} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set}) \qquad F^{-} : (I \longrightarrow \mathrm{Set}) \longrightarrow (I \longrightarrow \mathrm{Set})$$

and initial algebras in Chu can be constructed from those in Set:

$$\mu F = (\mu F^{+}, \nu F^{-})$$

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{Path} : \mathrm{Node} \times \mathrm{Node} \longrightarrow \mathrm{Chu}\ \mathsf{where} \\
&\quad \mathsf{stop} : \forall x \longrightarrow \mathrm{Path}(x, x) \\
&\quad \mathsf{step} : \forall x\, y\, z \longrightarrow \mathrm{Path}(x, y) \longrightarrow \mathrm{Step}(y, z) \longrightarrow \mathrm{Path}(x, z)
\end{aligned}
$$

$$\mathsf{data}\ \mathrm{Path} : \mathrm{Node} \times \mathrm{Node} \longrightarrow \mathrm{Chu}\ \mathsf{where}$$

$$\quad \mathsf{stop} : \forall x \longrightarrow \mathrm{Path}(x, x)$$

$$\quad \mathsf{step} : \forall x\, y\, z \longrightarrow \mathrm{Path}(x, y) \longrightarrow \mathrm{Step}(y, z) \longrightarrow \mathrm{Path}(x, z)$$

$$F_{\mathrm{Path}}(X)(x, z) = [x = z] + (\Sigma y.\, X(x, y) \times [\mathrm{Step}(y, z)])$$

$$\mathsf{data}\ \mathrm{Path} : \mathrm{Node} \times \mathrm{Node} \longrightarrow \mathrm{Chu}\ \mathsf{where}$$

$$\mathsf{stop} : \forall x \longrightarrow \mathrm{Path}(x, x)$$

$$\mathsf{step} : \forall x\, y\, z \longrightarrow \mathrm{Path}(x, y) \longrightarrow \mathrm{Step}(y, z) \longrightarrow \mathrm{Path}(x, z)$$

$$F_{\mathrm{Path}}(X)(x, z) = [x = z] + (\Sigma y.\, X(x, y) \times [\mathrm{Step}(y, z)])$$

$$F^{+}_{\mathrm{Path}}(X^{+})(x, z) = (x = z) + (\Sigma y.\, X^{+}(x, y) \times \mathrm{Step}(y, z))$$

$$F^{-}_{\mathrm{Path}}(X^{-})(x, z) = \neg(x = z) \times (\Pi y.\, X^{-}(x, y) + \neg\mathrm{Step}(y, z))$$

$$\mathsf{data}\ \mathrm{Path} : \mathrm{Node} \times \mathrm{Node} \longrightarrow \mathrm{Chu}\ \mathsf{where}$$

$$\quad \mathsf{stop} : \forall x \longrightarrow \mathrm{Path}(x, x)$$

$$\quad \mathsf{step} : \forall x\, y\, z \longrightarrow \mathrm{Path}(x, y) \longrightarrow \mathrm{Step}(y, z) \longrightarrow \mathrm{Path}(x, z)$$

$$F_{\mathrm{Path}}(X)(x, z) = [x = z] + (\Sigma y.\, X(x, y) \times [\mathrm{Step}(y, z)])$$

$$F^{+}_{\mathrm{Path}}(X^{+})(x, z) = (x = z) + (\Sigma y.\, X^{+}(x, y) \times \mathrm{Step}(y, z))$$

$$F^{-}_{\mathrm{Path}}(X^{-})(x, z) = \neg(x = z) \times (\Pi y.\, X^{-}(x, y) + \neg\mathrm{Step}(y, z))$$

$$\mathrm{Path} = (\mu F^{+}_{\mathrm{Path}}, \nu F^{-}_{\mathrm{Path}})$$

What about Data Types with Negation?

An obstacle:

$$\mathsf{not} : \mathsf{Chu}^{\mathsf{op}} \longrightarrow \mathsf{Chu}$$

An obstacle:

$$\mathsf{not} : \mathsf{Chu}^{\mathsf{op}} \longrightarrow \mathsf{Chu}$$

So a Data Type with Negation cannot yield a functor.

An obstacle:

$$\mathsf{not} : \mathsf{Chu}^{\mathsf{op}} \longrightarrow \mathsf{Chu}$$

So a Data Type with Negation cannot yield a functor.

Have we got anywhere?

Define another category Chu^i with the same objects but:

$$(A^+, A^-) \overset{i}{\Rightarrow} (B^+, B^-) = (A^+ \to B^+, A^- \to B^-)$$

Information morphisms:
parallel transformation of positive and negative information.

Negation is *covariant* in Chu^i

$$\mathrm{not} : \mathrm{Chu}^i \to \mathrm{Chu}^i$$

Negation is *covariant* in Chu^i

$$\mathsf{not} : \mathsf{Chu}^i \to \mathsf{Chu}^i$$

Initial F-Algebras in Chu^i:

$$\mu^i F = \mu(X^+, X^-).(F^+(X^+, X^-), F^-(X^+, X^-))$$

Negation is *covariant* in Chu^i

$$\mathsf{not} : \mathrm{Chu}^i \to \mathrm{Chu}^i$$

Initial F-Algebras in Chu^i:

$$\mu^i F = \mu(X^+, X^-).(F^+(X^+, X^-), F^-(X^+, X^-))$$

Works for all of our functors, but gives the “wrong” answer.

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{Path} : \mathrm{Node} \times \mathrm{Node} \longrightarrow \mathrm{Chu}\ \mathsf{where} \\
&\quad \mathsf{stop} : \forall x \longrightarrow \mathrm{Path}(x, x) \\
&\quad \mathsf{step} : \forall x\, y\, z \longrightarrow \mathrm{Path}(x, y) \longrightarrow \mathrm{Step}(y, z) \longrightarrow \mathrm{Path}(x, z)
\end{aligned}
$$

The $\mu^i F$ solution yields:

$$
(\mu^i F)^- = \mu X.\lambda(x, z).\neg[x = z] \times (\Pi y.\, X(x, y) + \neg\mathrm{Step}(y, z))
$$

i.e. all paths from x finitely never reach z.

Idea:

Refine the positive meaning of a data type with respect to information about non-provability.

Idea:
Refine the positive meaning of a data type with respect to information about non-provability.

The *reduct* of a logic program from **Stable Model Semantics** (Gelfond and Lifschitz, 1989).

Given $F\colon (I \longrightarrow \mathrm{Chu}) \longrightarrow (I \longrightarrow \mathrm{Chu})$

Given $F\colon (I \to \mathrm{Chu}) \to (I \to \mathrm{Chu})$ and $Y\colon I \to \mathrm{Chu}$, define

Given $F\colon (I \longrightarrow \mathrm{Chu}) \rightarrow (I \longrightarrow \mathrm{Chu})$ and $Y\colon I \longrightarrow \mathrm{Chu}$, define

$$F_{/Y}(X^+, X^-) = \lambda i.\,(F^+(X^+, Y^-)\,i, F^-(Y^+, X^-)\,i)$$

Y represents "a stage of knowledge".

Given $F\colon (I \longrightarrow \mathrm{Chu}) \to (I \longrightarrow \mathrm{Chu})$ and $Y\colon I \longrightarrow \mathrm{Chu}$, define

$$F_{/Y}(X^+, X^-) = \lambda i.\,(F^+(X^+, Y^-)\,i, F^-(Y^+, X^-)\,i)$$

Y represents "a stage of knowledge".

$F_{/Y}$ is separable, so we can get $\mu(F_{/Y}) : I \longrightarrow \mathrm{Chu}$.

Given $F\colon (I \to \mathrm{Chu}) \to (I \to \mathrm{Chu})$ and $Y : I \to \mathrm{Chu}$, define

$$F_{/Y}(X^+, X^-) = \lambda i.\,(F^+(X^+, Y^-)\,i, F^-(Y^+, X^-)\,i)$$

Y represents "a stage of knowledge".

$F_{/Y}$ is separable, so we can get $\mu(F_{/Y}) : I \to \mathrm{Chu}$.

Moreover, get a functor $\mu(F/-) : \mathrm{Chu}^I \to \mathrm{Chu}^I$.

A Semantics of Data Types with Negation

Given $F\colon (I \longrightarrow \mathrm{Chu}) \to (I \longrightarrow \mathrm{Chu})$ for a data type D, define

$$\begin{aligned} D &= \mu^i Y.\, \mu(F_{/Y}) \\ &= \mu(Y^+, Y^-).(\mu X^+.\, F^+(X^+, Y^-), \nu X^-.\, F^-(Y^+, X^-)) \end{aligned}$$

A Semantics of Data Types with Negation

Given $F\colon (I \longrightarrow \mathsf{Chu}) \longrightarrow (I \longrightarrow \mathsf{Chu})$ for a data type D, define

$$
\begin{aligned}
D &= \mu^{i} Y.\, \mu(F_{/Y}) \\
&= \mu(Y^{+}, Y^{-}).(\mu X^{+}.\, F^{+}(X^{+}, Y^{-}), \nu X^{-}.\, F^{-}(Y^{+}, X^{-}))
\end{aligned}
$$

If we replace Set by Bool, and add the consistency condition, then this coincides with the 3-valued stable model semantics (Przymusinski, 1990).

Examples

```
data Liar : Set where
    liar : not Liar → Liar
```

$$\mathsf{data}\ \mathsf{Liar} : \mathsf{Set}\ \mathsf{where}$$
$$\quad \mathsf{liar} : \mathsf{not}\ \mathsf{Liar} \longrightarrow \mathsf{Liar}$$

$$FX = \mathsf{not}X$$

$$\textsf{data Liar : Set where}$$
$$\quad \textsf{liar : not Liar} \to \textsf{Liar}$$

$$FX = \mathsf{not}X$$

$$F^{+}(X^{+}, X^{-}) = X^{-} \qquad F^{-}(X^{+}, X^{-}) = X^{+}$$

$$
\begin{aligned}
&\textsf{data}\ \mathsf{Liar} : \mathsf{Set}\ \textsf{where} \\
&\quad \mathsf{liar} : \mathsf{not}\ \mathsf{Liar} \longrightarrow \mathsf{Liar}
\end{aligned}
$$

$$FX = \mathsf{not} X$$

$$F^{+}(X^{+}, X^{-}) = X^{-} \qquad F^{-}(X^{+}, X^{-}) = X^{+}$$

$$
\begin{aligned}
\mathsf{Liar} &= \mu^{i}(Y^{+}, Y^{-}).(\mu X^{+}.\, F^{+}(X^{+}, Y^{-}), \nu X^{-}.\, F^{-}(Y^{+}, X^{-})) \\
&= \mu^{i}(Y^{+}, Y^{-}).(Y^{-}, Y^{+}) \\
&\cong (\bot, \bot)
\end{aligned}
$$

```
data NoBaseCase : Set where
  rec : NoBaseCase → NoBaseCase
```

```
data NoBaseCase : Set where
  rec : NoBaseCase → NoBaseCase
```

$$FX = X$$

```
data NoBaseCase : Set where
  rec : NoBaseCase → NoBaseCase
```

$$FX = X$$

$$\begin{aligned} \text{NoBaseCase} &= \mu(Y^+, Y^-).(\mu X^+.\, X^+, \nu X^-.\, X^-) \\ &\cong (\bot, \top) \end{aligned}$$

$$\mathsf{data}\ \mathrm{NoBaseCase} : \mathrm{Set}\ \mathsf{where}$$
$$\quad \mathsf{rec} : \mathrm{NoBaseCase} \longrightarrow \mathrm{NoBaseCase}$$

$$FX = X$$

$$\begin{aligned}\mathrm{NoBaseCase} &= \mu(Y^+, Y^-).(\mu X^+.\, X^+, \nu X^-.\, X^-) \\ &\cong (\bot, \top)\end{aligned}$$

No proofs, one (extensionally) refutation.

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{Even} : \mathrm{Nat} \longrightarrow \mathrm{Set}\ \mathsf{where}\\
&\quad \mathsf{zero}\ : \mathrm{Even}\ \mathsf{zero}\\
&\quad \mathsf{succ} : \forall \{n\} \longrightarrow \mathrm{not}\ (\mathrm{Even}\ n) \longrightarrow \mathrm{Even}\ (\mathsf{succ}\ n)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{data}\ \mathsf{Even} : \mathsf{Nat} \longrightarrow \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{zero}\ : \mathsf{Even}\ \mathsf{zero}\\
&\quad \mathsf{succ} : \forall\{n\} \longrightarrow \mathsf{not}\,(\mathsf{Even}\ n) \longrightarrow \mathsf{Even}\,(\mathsf{succ}\ n)
\end{aligned}
$$

$$
F X = \lambda\{\mathsf{zero} \longmapsto \top ; \mathsf{succ}\ n \longmapsto \mathsf{not}\,(X\,n)\}
$$

$$\mathsf{data}\ \mathsf{Even} : \mathsf{Nat} \to \mathsf{Set}\ \mathsf{where}$$
$$\quad \mathsf{zero} : \mathsf{Even}\ \mathsf{zero}$$
$$\quad \mathsf{succ} : \forall\{n\} \to \mathsf{not}\ (\mathsf{Even}\ n) \to \mathsf{Even}\ (\mathsf{succ}\ n)$$

$$FX = \lambda\{\mathsf{zero} \mapsto \top ; \mathsf{succ}\ n \mapsto \mathsf{not}\ (X\,n)\}$$

$$F^{+}(X^{+}, X^{-}) = \lambda\{\mathsf{zero} \mapsto \top ; \mathsf{succ}\ n \mapsto X^{-}\ n\}$$
$$F^{-}(X^{+}, X^{-}) = \lambda\{\mathsf{zero} \mapsto \bot ; \mathsf{succ}\ n \mapsto X^{+}\ n\}$$

$$
\begin{aligned}
&\mathsf{data}\ \mathsf{Even} : \mathsf{Nat} \longrightarrow \mathsf{Set}\ \mathsf{where} \\
&\quad \mathsf{zero} : \mathsf{Even}\ \mathsf{zero} \\
&\quad \mathsf{succ} : \forall\{n\} \longrightarrow \mathsf{not}\,(\mathsf{Even}\ n) \longrightarrow \mathsf{Even}\,(\mathsf{succ}\ n)
\end{aligned}
$$

$$
F X = \lambda\{\mathsf{zero} \mapsto \top; \mathsf{succ}\ n \mapsto \mathsf{not}\,(X\,n)\}
$$

$$
\begin{aligned}
F^{+}(X^{+}, X^{-}) &= \lambda\{\mathsf{zero} \mapsto \top; \mathsf{succ}\ n \mapsto X^{-}\ n\} \\
F^{-}(X^{+}, X^{-}) &= \lambda\{\mathsf{zero} \mapsto \bot; \mathsf{succ}\ n \mapsto X^{+}\ n\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Even} = \mu(Y^{+}, Y^{-}).\,(&\lambda\{\mathsf{zero} \mapsto \top; \mathsf{succ}\ n \mapsto Y^{-}\ n\}, \\
&\lambda\{\mathsf{zero} \mapsto \bot; \mathsf{succ}\ n \mapsto Y^{+}\ n\})
\end{aligned}
$$

which is $\cong$ to the mutually defined Even/Odd definition.

All research ends in ~~failure~~, the need for more research.

The two kinds of morphisms have an analogue in ωCPOs:

- truth morphisms $\approx$ continuous functions
- information morphisms $\approx$ embed/proj pairs

Arrange these into a double category.

The two kinds of morphisms have an analogue in ωCPOs:

- truth morphisms $\approx$ continuous functions
- information morphisms $\approx$ embed/proj pairs

Arrange these into a double category.

Split positive and negative uses into separate arguments:

$$F : \mathrm{Chu}^{\mathsf{hop}} \times \mathrm{Chu} \to \mathrm{Chu}$$

(a double functor)

The two kinds of morphisms have an analogue in ωCPOs:

- truth morphisms $\approx$ continuous functions
- information morphisms $\approx$ embed/proj pairs

Arrange these into a double category.

Split positive and negative uses into separate arguments:

$$F\colon \mathrm{Chu}^{\mathsf{hop}} \times \mathrm{Chu} \to \mathrm{Chu}$$

(a double functor)

Then see what carries over from solutions of domain equations.

- An obstacle: only isos are shared between truth and information morphisms, but ωCPO is a *framed bicategory*.

Conclusion

- Constructed a plausible semantics of data types with negation
- Generalisation of 3-valued stable model semantics
- Uses:
  - backtracking processes
  - default reasoning
  - error states, e.g. parse errors, ill-typed programs
- Can ASP be used to synthesise data types?

- Constructed a plausible semantics of data types with negation
- Generalisation of 3-valued stable model semantics
- Uses:
  - backtracking processes
  - default reasoning
  - error states, e.g. parse errors, ill-typed programs
- Can ASP be used to synthesise data types?

Related work:

- Weak negation / negation as failure: Clark (1978), Gelfond and Lifschitz (1988), Przymusinski (1989)
- Bilattices: Ginsberg (1986), Fitting (2020)
- "Anithesis translation"
  Affine logic for constructive mathematics Shulman (2018-22);

Remember to **think negatively**.

It may improve your expressiveness.