Data types with Negation

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

TYPES
23rd June 2022

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

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

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

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

What if we could use negation?

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

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

What if we could use negation?

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

Representing Parse Trees

A *Context-Free Grammar* (CFG):

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

Representation of the parse trees:

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

Representing Parse Trees

A *Parsing Expression Grammar* (PEG):

$$S ::= A \boxed{/} B$$

“Parse S as an A, or *if that fails*, as a B”

Representing Parse Trees

A *Parsing Expression Grammar* (PEG):

$$S ::= A \; \boxed{/} \; B$$

"Parse *S* as an *A*, or *if that fails*, as a *B*"

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

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

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

?

Semantics of Inductive Types

1. Construct a functor $F\colon (I \to \mathsf{Set}) \to (I \to \mathsf{Set})$ from the definition
   - Essentially a sum-of-products construction
2. The interpretation of the data type is the initial F-algebra
   - specifies a unique “fold”, derive induction principles

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

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

?

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

?

Not covariant: don't get a functor!

What do we mean when we say “not”?

1. “This leads to a contradiction”

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

What do we mean when we say “not”?

1. “This leads to a contradiction”

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

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

What do we mean when we say "not"?

1. "This leads to a contradiction"

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

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

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

What do we mean when we say "not"?

1. "This leads to a contradiction"

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

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

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

Does "tweety" fly?

What do we mean when we say "not"?

1. "This leads to a contradiction"

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

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

$$\text{files}(X) \leftarrow \text{bird}(X), \text{not flightless}(X).$$
$$\text{flightless}(X) \leftarrow \text{penguin}(X).$$
$$\text{bird}(\text{tweety}).$$

Does "tweety" fly?

Classically (and intuitionistically): no.

What do we mean when we say "not"?

1. "This leads to a contradiction"

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

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

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

Does "tweety" fly?

Classically (and intuitionistically): no. Intuitively – yes?

What do we mean when we say “not”?

1. “This leads to a contradiction”

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

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

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

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 our prior conclusion.

Constructive Falsity

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

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

Constructive Falsity

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

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

Entailment flows forwards positively and backwards negatively:

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

call this category Chu.

Aside: we could also add the condition $A^+ \times A^- \to \bot$, but will skip this for now.

Connectives

1. Conjunction

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

2. Disjunction

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

3. Infinitary Conjunction

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

4. Negation

$$\text{not}\,(A^+, A^-) \overset{def}{=} (A^-, A^+)$$

5. Sets

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

Initial Algebras in Chu

If we have

$$F : (I \to \mathsf{Chu}) \to (I \to \mathsf{Chu})$$

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

$$F^{+} : (I \to \mathsf{Set}) \to (I \to \mathsf{Set}) \qquad F^{-} : (I \to \mathsf{Set}) \to (I \to \mathsf{Set})$$

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

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

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

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

$$
F(X)(x, z) = [x = z] + (\Sigma y.\, X(x, y) \times [\mathsf{Step}(y, z)])
$$

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

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

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

$$\textsf{data Path} : \textsf{Node} \times \textsf{Node} \to \textsf{Set where}$$
$$\quad \textsf{stop} : \forall x \to \textsf{Path}(x, x)$$
$$\quad \textsf{step} : \forall x\, y\, z \to \textsf{Path}(x, y) \to \textsf{Step}(y, z) \to \textsf{Path}(x, z)$$

$$F(X)(x, z) = [x = z] + (\Sigma y.\, X(x, y) \times [\textsf{Step}(y, z)])$$

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

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

What do you want?

What about data types with negation?

Information!

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

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

Information morphisms – parallel transformation of the positive and negative.

Negation is *covariant* in Chu^i.

Information!

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 the positive and negative.

Negation is *covariant* in Chu^i.

Initial F-Algebras in Chu^i:

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

Information!

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

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

Information morphisms – parallel transformation of the positive and negative.

Negation is *covariant* in Chu^i.

Initial F-Algebras in Chu^i:

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

You won't get it! Works for all of our data types, but gives the "wrong" answer.

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

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

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

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

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

Given $F : (I \to \mathsf{Chu}) \to (I \to \mathsf{Chu})$

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

The *reduct* of a logic program from **Stable Model Semantics**

(Gelfond and Lifschitz, 1989).

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

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

The *reduct* of a logic program from **Stable Model Semantics**

(Gelfond and Lifschitz, 1989).

Given $F : (I \longrightarrow \text{Chu}) \longrightarrow (I \longrightarrow \text{Chu})$ and $Y : I \longrightarrow \text{Chu}$, define

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

Y represents “a stage of knowledge”.

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

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

Given $F : (I \to \text{Chu}) \to (I \to \text{Chu})$ and $Y : I \to \text{Chu}$, 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 $\mu(F/Y) : I \to \text{Chu}$.

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

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

Given $F : (I \to \mathrm{Chu}) \to (I \to \mathrm{Chu})$ 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 Sematics of Data types with Negation

Given $F : (I \to \mathrm{Chu}) \to (I \to \mathrm{Chu})$ 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 Sematics of Data types with Negation

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

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

If we replace Set by Bool, and add the $\mathrm{Chu}(\bot)$ constraint, then this coincides with the 3-valued stable model semantics (Przymusinski, 1990).

Example: Liar

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

$$FX = \mathrm{not}\, X$$

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

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

Example: Identity

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

$$FX = X$$

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

No proofs, one (extensionally) refutation.

Example: Even

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

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

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

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

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

Conclusion

- Constructed a plausible semantics of data types with negation
- Based on 3-valued stable model semantics of logic programming
- Useful for modelling backtracking processes, and default reasoning?

Conclusion

- Constructed a plausible semantics of data types with negation
- Based on 3-valued stable model semantics of logic programming
- Useful for modelling backtracking processes, and default reasoning?

Related work:

- Constructive Falsity: “anithesis translation” Shulman (2018)
- Weak negation / negation as failure: Clark (1978), Gelfond and Lifschitz (1988), Przymusinski (1989)
- Bilattices: Ginsberg (1986), Fitting (2020)

Conclusion

- Constructed a plausible semantics of data types with negation
- Based on 3-valued stable model semantics of logic programming
- Useful for modelling backtracking processes, and default reasoning?

Related work:

- Constructive Falsity: "anithesis translation" Shulman (2018)
- Weak negation / negation as failure: Clark (1978), Gelfond and Lifschitz (1988), Przymusinski (1989)
- Bilattices: Ginsberg (1986), Fitting (2020)

Future work:

- TODO: Implementation (underway)
- TODO: Reasoning Principles