LogicS

Lecture #7: DIY First-Order Logic

Carlos Areces and Patrick Blackburn

{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

NASSLLI 2010 - Bloomington - USA

The Story so Far

The Story so Far

- We saw how we can introduce different operators...
- ...and 'cook' our own logic.

The Story so Far

- We saw how we can introduce different operators. . .
- . . . and ‘cook’ our own logic.
- Now, we want to talk about First (and Higher) Order Logic tomorrow.

The Story so Far

- We saw how we can introduce different operators...
- ...and ‘cook’ our own logic.
- Now, we want to talk about First (and Higher) Order Logic tomorrow.
- What do you think? Can we mix the First Order Recipe?

What do we do in this Lecture

What do we do in this Lecture

- We'll cook First Order Logic à la NASSLLI10.

What do we do in this Lecture

- We'll cook First Order Logic à la NASSLLI10.
- We will see what we can reuse of what we already have. . .
- . . . and extend the language if necessary.

What do we do in this Lecture

- We'll cook First Order Logic à la NASSLLI10.
- We will see what we can reuse of what we already have. . .
- . . . and extend the language if necessary.
- We will then show that the language we obtain is actually equivalent to the 'classical' First Order Language.

Checking our Stock

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols
  - The boolean operators $\wedge$, $\neg$
  - The $\langle R \rangle$ operator
  - Constants
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols
  - The boolean operators $\wedge$, $\neg$
  - The $\langle R \rangle$ operator
  - Constants
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$
  - The $\langle R \rangle$ operator
  - Constants
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator
  - Constants
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The $:$ operator
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The $:$ operator $\surd$
  - The counting operators $\langle = n\ R \rangle$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The : operator $\surd$
  - The counting operators $\langle = n\ R \rangle$ $\times$
  - The universal operator $[U]$
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The : operator $\surd$
  - The counting operators $\langle = n\ R \rangle$ $\times$
  - The universal operator $[U]$ Close, but no cigar!
  - The reflexive and transitive closure operator $\langle R^* \rangle$
- Which one can we use?

Checking our Stock

- We want to define a language equivalent to First Order Logic (with equality and constants, but without function symbols).
- During the previous lectures we introduced a number of operators.
  - Propositional symbols $\surd$
  - The boolean operators $\wedge$, $\neg$ $\surd$
  - The $\langle R \rangle$ operator $\surd$
  - Constants $\surd$
  - The $:$ operator $\surd$
  - The counting operators $\langle = n\ R \rangle$ $\times$
  - The universal operator $[U]$ Close, but no cigar!
  - The reflexive and transitive closure operator $\langle R^* \rangle$ $\times$
- Which one can we use?

The $[U]$ operator is not enough

The $[U]$ operator is not enough

- Granted: we need universal quantification.

The $[U]$ operator is not enough

- Granted: we need universal quantification.
- But the $[U]$ operator is not expressive enough.

The $[U]$ operator is not enough

- Granted: we need universal quantification.
- But the $[U]$ operator is not expressive enough.
  - We won't prove it here (one way to do it, for example is noting that the language containing $[U]$ is still decidable, while full first order logic should be undecidable).

The $[U]$ operator is not enough

- Granted: we need universal quantification.
- But the $[U]$ operator is not expressive enough.
  - We won't prove it here (one way to do it, for example is noting that the language containing $[U]$ is still decidable, while full first order logic should be undecidable).
  - The universal operator is not fine grained enough:

    $[U]$ says for all

    and we need for all x

The $[U]$ operator is not enough

- Granted: we need universal quantification.
- But the $[U]$ operator is not expressive enough.
  - We won't prove it here (one way to do it, for example is noting that the language containing $[U]$ is still decidable, while full first order logic should be undecidable).
  - The universal operator is not fine grained enough:

    $[U]$ says for all

    and we need for all x
- First order quantification gives as a delicate control (via variables) of what we are quantifying on.

A Detour: ‘Repointing’ Names

A Detour: 'Repointing' Names

- I will define a litle piece of notation that I will need in the next slide.

A Detour: ‘Repointing’ Names

- I will define a litle piece of notation that I will need in the next slide.
- As I want it to be very clear, I’ll do it here and give an example.

A Detour: 'Repointing' Names

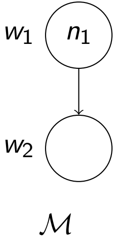
- I will define a litle piece of notation that I will need in the next slide.
- As I want it to be very clear, I'll do it here and give an example.
- Let
  - $\mathcal{M} = \langle D, \{R_1, \ldots, R_n\}, \{P_1, \ldots, P_m\}, \{N_1, \ldots, N_k\} \rangle$ be a model,
  - w an element in D ($w \in D$),
  - and n_i a name.

A Detour: 'Repointing' Names

- I will define a litle piece of notation that I will need in the next slide.
- As I want it to be very clear, I'll do it here and give an example.
- Let
  - $\mathcal{M} = \langle D, \{R_1, \ldots, R_n\}, \{P_1, \ldots, P_m\}, \{N_1, \ldots, N_k\}\rangle$ be a model,
  - w an element in D ($w \in D$),
  - and n_i a name.
- We write $\mathcal{M}\{n_i \mapsto w\}$ for the model obtained from $\mathcal{M}$ where the only change is that now n_i is interpreted as w (i.e., n_i points to w).

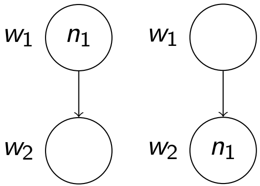
A Detour: 'Repointing' Names

- I will define a litle piece of notation that I will need in the next slide.
- As I want it to be very clear, I'll do it here and give an example.
- Let
  - $\mathcal{M} = \langle D, \{R_1, \ldots, R_n\}, \{P_1, \ldots, P_m\}, \{N_1, \ldots, N_k\}\rangle$ be a model,
  - w an element in D ($w \in D$),
  - and n_i a name.
- We write $\mathcal{M}\{n_i \mapsto w\}$ for the model obtained from $\mathcal{M}$ where the only change is that now n_i is interpreted as w (i.e., n_i points to w).



A Detour: 'Repointing' Names

- I will define a litle piece of notation that I will need in the next slide.
- As I want it to be very clear, I'll do it here and give an example.
- Let
  - $\mathcal{M} = \langle D, \{R_1, \ldots, R_n\}, \{P_1, \ldots, P_m\}, \{N_1, \ldots, N_k\}\rangle$ be a model,
  - w an element in D ($w \in D$),
  - and n_i a name.
- We write $\mathcal{M}\{n_i \mapsto w\}$ for the model obtained from $\mathcal{M}$ where the only change is that now n_i is interpreted as w (i.e., n_i points to w).



$\mathcal{M} \Rightarrow \mathcal{M}\{n_1 \mapsto w_2\}$

First Order Quantification

First Order Quantification

- We introduce the operator $\langle\!\langle n \rangle\!\rangle$ where n is a name (we will call the operator repoint n) as:
  $\mathcal{M}, w \models \langle\!\langle n \rangle\!\rangle\varphi$ iff for some w' $\mathcal{M}\{n \mapsto w'\}, w \models \varphi$

First Order Quantification

- We introduce the operator $\langle\!\langle n \rangle\!\rangle$ where n is a name (we will call the operator repoint n) as:
  $\mathcal{M}, w \models \langle\!\langle n \rangle\!\rangle\varphi$ iff for some w' $\mathcal{M}\{n \mapsto w'\}, w \models \varphi$
- Compare with
  $\mathcal{M}, w \models \langle R \rangle\varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.

First Order Quantification

- We introduce the operator $\langle\!\langle n \rangle\!\rangle$ where n is a name (we will call the operator repoint n) as:
  $\mathcal{M}, w \models \langle\!\langle n \rangle\!\rangle\varphi$ iff for some w' $\mathcal{M}\{n \mapsto w'\}, w \models \varphi$
- Compare with
  $\mathcal{M}, w \models \langle R \rangle\varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.
- Compare with $\langle U \rangle\varphi := \neg[U]\neg\varphi$
  $\mathcal{M}, w \models \langle U \rangle\varphi$ iff for some $w', \mathcal{M}, w' \models \varphi$

First Order Quantification

- We introduce the operator $\langle\!\langle n \rangle\!\rangle$ where n is a name (we will call the operator repoint n) as:
  $\mathcal{M}, w \models \langle\!\langle n \rangle\!\rangle\varphi$ iff for some w' $\mathcal{M}\{n \mapsto w'\}, w \models \varphi$
- Compare with
  $\mathcal{M}, w \models \langle R \rangle\varphi$ iff there is w' s.t. wRw' and $\mathcal{M}, w' \models \varphi$.
- Compare with $\langle U \rangle\varphi := \neg[U]\neg\varphi$
  $\mathcal{M}, w \models \langle U \rangle\varphi$ iff for some $w', \mathcal{M}, w' \models \varphi$
- Actually, using $\langle\!\langle n \rangle\!\rangle$ and : together we can define $[U]$:

$$[U]\varphi \text{ iff } \neg\langle\!\langle n \rangle\!\rangle(n{:}\neg\varphi)$$

First Order Quantification

- We introduce the operator $\langle\!\langle n \rangle\!\rangle$ where n is a name (we will call the operator repoint n) as:
  $$\mathcal{M}, w \models \langle\!\langle n \rangle\!\rangle\varphi \text{ iff for some } w' \; \mathcal{M}\{n \mapsto w'\}, w \models \varphi$$
- Compare with
  $$\mathcal{M}, w \models \langle R \rangle\varphi \text{ iff there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi.$$
- Compare with $\langle U \rangle\varphi := \neg[U]\neg\varphi$
  $$\mathcal{M}, w \models \langle U \rangle\varphi \text{ iff for some } w', \mathcal{M}, w' \models \varphi$$
- Actually, using $\langle\!\langle n \rangle\!\rangle$ and : together we can define $[U]$:

$$[U]\varphi \text{ iff } \neg\langle\!\langle n \rangle\!\rangle(n{:}\neg\varphi) \quad (\text{for } n \text{ not in } \varphi)$$

Capturing all First Order Logic

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\langle n \rangle\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$Tr(s = t) \quad = \quad s{:}t$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$
\begin{aligned}
Tr(s = t) &= s{:}t \\
Tr(P(s)) &= s{:}p
\end{aligned}
$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$\begin{aligned} Tr(s = t) &= s{:}t \\ Tr(P(s)) &= s{:}p \\ Tr(R(s,t)) &= s{:}\langle R \rangle t \end{aligned}$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$
\begin{aligned}
Tr(s = t) &= s{:}t \\
Tr(P(s)) &= s{:}p \\
Tr(R(s,t)) &= s{:}\langle R \rangle t \\
Tr(\neg\varphi) &= \neg Tr(\varphi)
\end{aligned}
$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$
\begin{aligned}
Tr(s = t) &= s{:}t \\
Tr(P(s)) &= s{:}p \\
Tr(R(s,t)) &= s{:}\langle R \rangle t \\
Tr(\neg\varphi) &= \neg Tr(\varphi) \\
Tr(\varphi \wedge \psi) &= Tr(\varphi) \wedge Tr(\psi)
\end{aligned}
$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R\rangle$ language, names, : and $\langle\!\langle n\rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$
\begin{array}{rcl}
Tr(s = t) & = & s{:}t \\
Tr(P(s)) & = & s{:}p \\
Tr(R(s,t)) & = & s{:}\langle R\rangle t \\
Tr(\neg\varphi) & = & \neg Tr(\varphi) \\
Tr(\varphi \wedge \psi) & = & Tr(\varphi) \wedge Tr(\psi) \\
Tr(\exists s.\varphi) & = & \langle\!\langle s\rangle\!\rangle Tr(\varphi)
\end{array}
$$

Capturing all First Order Logic

- We will now show that we can capture all First Order Logic using: the $\langle R \rangle$ language, names, : and $\langle\!\langle n \rangle\!\rangle$.
- We will (recursively) define a translation that will assign to each formula of the First Order Language, an equivalent formula in our language

$$
\begin{array}{rrcl}
 & Tr(s = t) & = & s{:}t \\
 & Tr(P(s)) & = & s{:}p \\
 & Tr(R(s,t)) & = & s{:}\langle R \rangle t \\
 & Tr(\neg\varphi) & = & \neg Tr(\varphi) \\
 & Tr(\varphi \wedge \psi) & = & Tr(\varphi) \wedge Tr(\psi) \\
 & Tr(\exists s.\varphi) & = & \langle\!\langle s \rangle\!\rangle Tr(\varphi) \\
(& Tr(\forall s.\varphi) & = & \neg\langle\!\langle s \rangle\!\rangle \neg Tr(\varphi) = [\![s]\!] Tr(\varphi) \quad)
\end{array}
$$

Examples

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$\forall y.Man(y) \rightarrow \exists x.(Woman(x) \land Loves(y,x)))$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x))) \;)$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))\ )$$
$$[\![y]\!](\ Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x))))$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y, x))))$$
$$[\![y]\!](Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y, x))))$$
$$[\![y]\!](Tr(Man(y)) \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y, x))))$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$
\begin{gathered}
Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))\ ) \\
[\![y]\!](\,Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))) \\
[\![y]\!](\,Tr(Man(y)) \rightarrow\ Tr(\exists x.(Woman(x) \wedge Loves(y,x)))) \\
[\![y]\!](y{:}Man \rightarrow\ Tr(\exists x.(Woman(x) \wedge Loves(y,x))))
\end{gathered}
$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$
\begin{gathered}
Tr(\forall y. Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y, x)))\ ) \\
[\![y]\!](Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y, x)))) \\
[\![y]\!](Tr(Man(y)) \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y, x)))) \\
[\![y]\!](y{:}Man \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y, x)))) \\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x \rangle\!\rangle(Tr((Woman(x) \wedge Loves(y, x)))))
\end{gathered}
$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$
\begin{gathered}
Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))\ ) \\
[\![y]\!](Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))) \\
[\![y]\!](Tr(Man(y)) \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y,x)))) \\
[\![y]\!](y{:}Man \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y,x)))) \\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(Tr((Woman(x) \wedge Loves(y,x))))) \\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(Tr(Woman(x)) \wedge Tr(Loves(y,x))))
\end{gathered}
$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$
\begin{gathered}
Tr(\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))\ )\\
[\![y]\!](Tr(Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x))))\\
[\![y]\!](Tr(Man(y)) \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y,x))))\\
[\![y]\!](y{:}Man \rightarrow Tr(\exists x.(Woman(x) \wedge Loves(y,x))))\\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(Tr((Woman(x) \wedge Loves(y,x)))))\\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(Tr(Woman(x)) \wedge Tr(Loves(y,x))))\\
[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(x{:}Woman \wedge y{:}\langle Loves\rangle x))
\end{gathered}
$$

Examples

- Let's write down a couple of formulas in First Order Logic and translate them to our language.

$$\forall y.Man(y) \rightarrow \exists x.(Woman(x) \wedge Loves(y,x)))$$
$$[\![y]\!](y{:}Man \rightarrow \langle\!\langle x\rangle\!\rangle(x{:}Woman \wedge y{:}\langle Loves\rangle x))$$

The Other Translation

The Other Translation

- Of course, we can do the translation in the other direction as well.

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$$\mathcal{M}, w \models \langle R \rangle \varphi \quad \text{iff} \quad \text{there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi$$

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$$\mathcal{M}, w \models \langle R \rangle \varphi \quad \text{iff} \quad \text{there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi$$

$$Tr(\langle R \rangle \varphi) \quad = \quad \exists w'.(R(w, w') \wedge Tr(\varphi))$$

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$$\mathcal{M}, w \models \langle R \rangle \varphi \quad \text{iff} \quad \text{there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi$$

$$Tr(\langle R \rangle \varphi) \quad = \quad \exists w'.(R(w, w') \wedge Tr(\varphi))$$

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$$\mathcal{M}, w \models \langle R \rangle \varphi \quad \text{iff} \quad \text{there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi$$

$$Tr_{w}(\langle R \rangle \varphi) \quad = \quad \exists w'.(R(w, w') \wedge Tr_{w'}(\varphi))$$

The Other Translation

- Of course, we can do the translation in the other direction as well.
- We only need to realize that the semantic definition of all the operators we introduced can be defined in first-order logic.

$$\mathcal{M}, w \models \langle R \rangle \varphi \quad \text{iff} \quad \text{there is } w' \text{ s.t. } wRw' \text{ and } \mathcal{M}, w' \models \varphi$$

$$Tr_w(\langle R \rangle \varphi) \;=\; \exists w'.(R(w, w') \wedge Tr_{w'}(\varphi))$$

- The w in Tr_w keeps track of where we are evaluating the formula in the model.

The Other Translation

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the $:$ and $\langle n \rangle$ operators.

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the : and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the $:$ and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$Tr_x(p) \quad = \quad P(x)$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the $:$ and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{aligned}
Tr_x(p) &= P(x) \\
Tr_x(n) &= (n = x)
\end{aligned}
$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the $:$ and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{aligned}
Tr_x(p) &= P(x) \\
Tr_x(n) &= (n = x) \\
Tr_x(\neg\varphi) &= \neg Tr_x(\varphi)
\end{aligned}
$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the : and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$\begin{array}{rcl} Tr_x(p) & = & P(x) \\ Tr_x(n) & = & (n = x) \\ Tr_x(\neg\varphi) & = & \neg Tr_x(\varphi) \\ Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \end{array}$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the : and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{array}{rcl}
Tr_x(p) & = & P(x) \\
Tr_x(n) & = & (n = x) \\
Tr_x(\neg\varphi) & = & \neg Tr_x(\varphi) \\
Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \\
Tr_x(\langle R \rangle \varphi) & = & \exists x.(R(x, y) \wedge Tr_y(\varphi)) \;\; \text{for } y \text{ a new variable}
\end{array}
$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the : and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{array}{rcl}
Tr_x(p) & = & P(x) \\
Tr_x(n) & = & (n = x) \\
Tr_x(\neg\varphi) & = & \neg Tr_x(\varphi) \\
Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \\
Tr_x(\langle R \rangle \varphi) & = & \exists x.(R(x, y) \wedge Tr_y(\varphi)) \text{ for } y \text{ a new variable} \\
Tr_x(n{:}\varphi) & = & Tr_n(\varphi)
\end{array}
$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R \rangle$ language extended with constants, and the : and $\langle n \rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{array}{rcl}
Tr_x(p) & = & P(x) \\
Tr_x(n) & = & (n = x) \\
Tr_x(\neg\varphi) & = & \neg Tr_x(\varphi) \\
Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \\
Tr_x(\langle R \rangle \varphi) & = & \exists x.(R(x, y) \wedge Tr_y(\varphi)) \text{ for } y \text{ a new variable} \\
Tr_x(n{:}\varphi) & = & Tr_n(\varphi) \\
Tr_x(\langle\!\langle n \rangle\!\rangle \varphi) & = & \exists n. Tr_x(\varphi)
\end{array}
$$

The Other Translation

- Let's see the details. Assume that we have a formula in the $\langle R\rangle$ language extended with constants, and the : and $\langle n\rangle$ operators.
  We will (recursively) define an equivalent first order formula:

$$
\begin{array}{rcl}
Tr_x(p) & = & P(x) \\
Tr_x(n) & = & (n = x) \\
Tr_x(\neg\varphi) & = & \neg Tr_x(\varphi) \\
Tr_x(\varphi \wedge \psi) & = & Tr_x(\varphi) \wedge Tr_x(\psi) \\
Tr_x(\langle R\rangle\varphi) & = & \exists x.(R(x,y) \wedge Tr_y(\varphi)) \text{ for } y \text{ a new variable} \\
Tr_x(n{:}\varphi) & = & Tr_n(\varphi) \\
Tr_x(\langle\!\langle n\rangle\!\rangle\varphi) & = & \exists n. Tr_x(\varphi)
\end{array}
$$

We can think of n as an FO variable now.

What we Covered Today

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.
- Students have told me

  I undestand PL, but I would never get how FOL works.

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.
- Students have told me

  I undestand PL, but I would never get how FOL works.

  NONSENSE!!

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.
- Students have told me

  I undestand PL, but I would never get how FOL works.

  NONSENSE!!
- As we saw today, they are not as different. You only need to look at them from the right perspective.

What we Covered Today

- In a way, the reason for today's talk was to show that there is nothing special about first order logic.
- It can be obtained in a natural way, following the ideas that we introduced in previous lectures.
- Students have told me

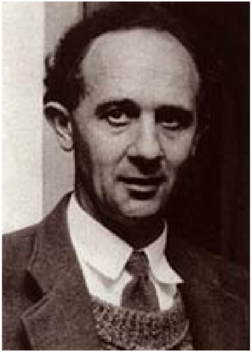
  I undestand PL, but I would never get how FOL works.

  NONSENSE!!
- As we saw today, they are not as different. You only need to look at them from the right perspective.
- If you really understand how one works, you already know how the other does.

Relevant Bibliography

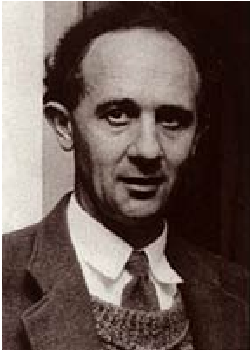
Relevant Bibliography

- As we mentioned in the first lecture one of the first polytheistic logicians was Arthur Prior.



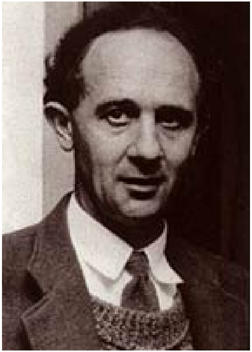
Relevant Bibliography

- As we mentioned in the first lecture one of the first polytheistic logicians was Arthur Prior.
- Prior is the father of Tense Logic, a logic that include the operators F and P to talk about the future and the past.



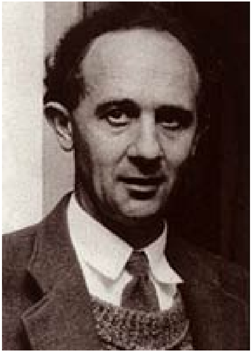
Relevant Bibliography

- As we mentioned in the first lecture one of the first polytheistic logicians was Arthur Prior.
- Prior is the father of Tense Logic, a logic that include the operators F and P to talk about the future and the past.
- He was a strong advocate of the bottom up way of viewing first-order logic that we presented today.



Relevant Bibliography

- As we mentioned in the first lecture one of the first polytheistic logicians was Arthur Prior.
- Prior is the father of Tense Logic, a logic that include the operators F and P to talk about the future and the past.
- He was a strong advocate of the bottom up way of viewing first-order logic that we presented today.



Prior, Arthur (1967). *Chapter V.6 of Past, Present and Future*. Clarendon Press, Oxford.

The Next Lecture

We Like it Complete and Compact

(and We have a Soft Spot for Löwenheim-Skolem)