A Tableau System for Natural Logic and Natural Reasoning

Reinhard Muskens

Tilburg Center for Logic and Philosophy of Science (TiLPS)

Workshop on Natural Logic, Proof Theory, and Computational Semantics, CSLI, April 9, 2011

Aim

- In the late 1960s Richard Montague wrote that there are no important theoretical differences between natural languages and logical languages.
- But if that is so it should be possible to find a calculus for the entailment relation in natural language that uses only linguistic forms.
- This calculus ideally should also use only rules that are linguistically relevant.
- In the best of all possible worlds we can also say more about natural reasoning with the help of such a logic.
- My aim is to bring such a logic closer to realization by providing a tableau calculus for linguistic representations.
- `tinyurl.com/tilnatlog`

Logical and Linguistic Representations

- At first blush a calculus that works directly with linguistic forms seems bound to run into problems: logic and linguistics seem to impose very different and conflicting requirements on the representations they work with.
- But in fact it is easy to see on the basis of Montague's work that a certain fragment of the simply typed lambda calculus is very close to the Logical Forms used in generative syntax.
- We will give a rough characterisation of this fragment and will call the representations in question Lambda Logical Forms.

Lambda Logical Forms (LLFs)

We consider terms of the simply typed lambda calculus in which no logical constants occur and in which abstraction is only over variables of basic type (or just over type e). Here are some examples.

(1) a. ((a woman)walk)

b. ((if((a woman)walk))((no man)talk))

c. (mary(think((if((a woman)walk))((no man)talk))))

d. ((a woman)(λx(mary(think((if(walk x))((no man)talk))))))

e. (few man)λx.(most woman)λy.like xy

f. (mary λx((try(run x)) x))

The constants here should be of the "right" type. Order is irrelevant, despite appearances.

Previous Work

- LLFs are not only related to LFs, but also to Cresswell's λ-categorical languages, to the proof terms of proofs in Lambek categorial grammar, and to the abstract terms of Abstract Categorial Grammars / Lambda Grammars (de Groote 2001, Muskens 2001).
- Given the connection with Categorial Grammar, the present ideas are related to previous work in natural logic by Van Benthem; Sánchez-Valencia; Bernardi; Zamansky, Francez and Winter; and others.
- But our strategy is not to annotate proofs in categorial grammar but to work directly with a tableau calculus for LLFs.
- We will also obviously build on work in Generalised Quantifier Theory.

Providing LLFs with an Underlying Logic

- LLFs are built up using non-logical constants, variables, application, and abstraction.
- In a sense they therefore have no logic at all: $\beta\eta$ conversion and that is it.
- We may assume that there is some homomorphic translation *sem* into a standard semantics, using rules like $sem(\texttt{some}) = \lambda P' \lambda P \exists x(P'x \wedge Px)$.
- $sem((\texttt{some woman})\, \lambda x.(\texttt{every man})(\texttt{loves}\ x)) =$ $\exists y(woman\, y \wedge \forall x(man\, x \rightarrow love\, xy))$
- That induces a logic on our terms. Let us call this the underlying interpretation.

Possible Incompleteness

- We will not directly use the underlying interpretation. We are interested in tableau rules that operate on LLFs.
- These rules should of course be sound with respect to the underlying interpretation.
- But, while it would be easy to find rules that are also complete with respect to this interpretation (provided we allow extra constants), there is no sense in working with such rules (in that case we could just as well use the underlying interpretation).
- LLFs are close to language and the hope is that this representation will help suggest ways of distinguishing the fast and automatic parts of reasoning from the the slow and concious ones.

Tableau Entries

- We will be working with a relational type logic.
- Read $\alpha_1 \to \ldots \to \alpha_n \to t$ or $\alpha_1 \ldots \alpha_n t$ (association in types is to the right) for $\langle \alpha_1 \ldots \alpha_n \rangle$ if you are unfamiliar with relational type logic. $\langle\rangle$ is just t. E.g. $\langle \langle es \rangle \langle es \rangle s \rangle$ is $(est)(est)st$.
- If A is an LLF of type $\langle \alpha_1 \ldots \alpha_n \rangle$ and $c_1, \ldots, c_n$ (abbreviated $\vec{c}$) are constants of types $\alpha_1, \ldots, \alpha_n$ respectively, then $T\vec{c} : A$ and $F\vec{c} : A$ are (tableau) entries.
- $T\vec{c} : A$ says that $A\vec{c}$ is true; $F\vec{c} : A$ says that $A\vec{c}$ is false.

The Boolean Core

- It is by now quite obvious that natural language has a Boolean core that is all-important.
- We basically have coordination in all categories. To some extent also negation.
- Entailment is being in the Boolean ordering relation, in all models (so that there is entailment in all linguistic categories).
- There are various properties of operators (like monotonicity) on a Boolean algebra that not only seem to be important for fast reasoning, but also correspond to certain distributional facts.

Reasoning with and, or, and not

- We will have constants and and or in all types $\langle\langle\vec{\alpha}\rangle\langle\vec{\alpha}\rangle\vec{\alpha}\rangle$ and write these between their arguments. Similarly, we will have constants not in all types $\langle\langle\vec{\alpha}\rangle\vec{\alpha}\rangle$. Adopt the following rules.
- $$\begin{array}{c} T\vec{c}: A \text{ and } B \\ | \\ T\vec{c}: A \\ T\vec{c}: B \end{array} \qquad \begin{array}{c} F\vec{c}: A \text{ and } B \\ \diagup\ \diagdown \\ F\vec{c}: A \quad F\vec{c}: B \end{array}$$
- $$\begin{array}{c} F\vec{c}: A \text{ or } B \\ | \\ F\vec{c}: A \\ F\vec{c}: B \end{array} \qquad \begin{array}{c} T\vec{c}: A \text{ or } B \\ \diagup\ \diagdown \\ T\vec{c}: A \quad T\vec{c}: B \end{array}$$
- $$\begin{array}{c} T\vec{c}: \texttt{not } A \\ | \\ F\vec{c}: A \end{array} \qquad \begin{array}{c} F\vec{c}: \texttt{not } A \\ | \\ T\vec{c}: A \end{array}$$

Monotonicity

- Upward Monotonicity: $X \subset Y \Longrightarrow F(X) \subset F(Y)$
  Alternatively: $F(X \cap Y) \subset F(X) \cap F(Y)$
  (I use $\subset$ interchangeably with $\subseteq$.)
- Downward Monotonicity: $X \subset Y \Longrightarrow F(Y) \subset F(X)$
  Alternatively: $F(X \cup Y) \subset F(X) \cap F(Y)$
- An axiom such as $\forall XY(X \subset Y \rightarrow GX \subset GY)$ can be replaced by a rule of the form

$$\begin{array}{c} T\vec{a} : GA \\ F\vec{a} : GB \\ \hline T\vec{c} : A \\ F\vec{c} : B \end{array}$$

  where the $\vec{c}$ are fresh.

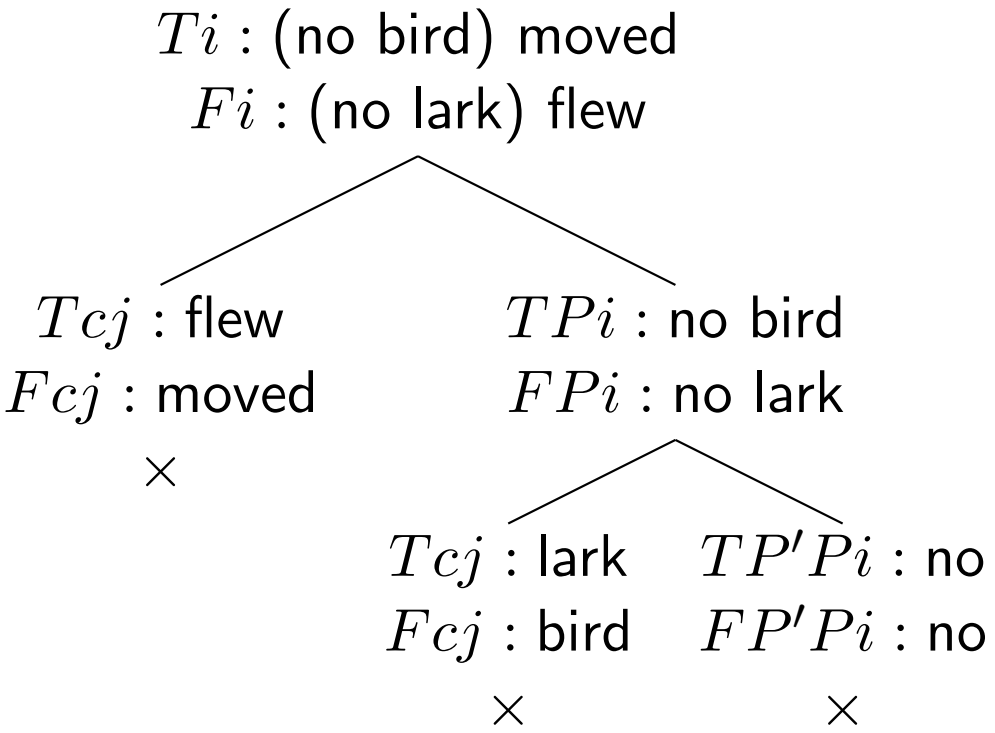
Rules for Monotone Operators

- $\dfrac{T\vec{a} : GA \qquad F\vec{a} : HB}{\begin{array}{ll} T\vec{c} : A & Tb\vec{a} : G \\ F\vec{c} : B & Fb\vec{a} : H \end{array}}$ where $\vec{c}$ and b are fresh, provided G or H is mon$\uparrow$
- Examples of mon$\uparrow$ functors: some, some N, every N, many N, most N.
- $\dfrac{T\vec{a} : GA \qquad F\vec{a} : HB}{\begin{array}{ll} T\vec{c} : B & Tb\vec{a} : G \\ F\vec{c} : A & Fb\vec{a} : H \end{array}}$ where $\vec{c}$ and b are fresh, provided G or H is mon$\downarrow$
- Examples of mon$\downarrow$ functors: no, no N, every, few, few N.

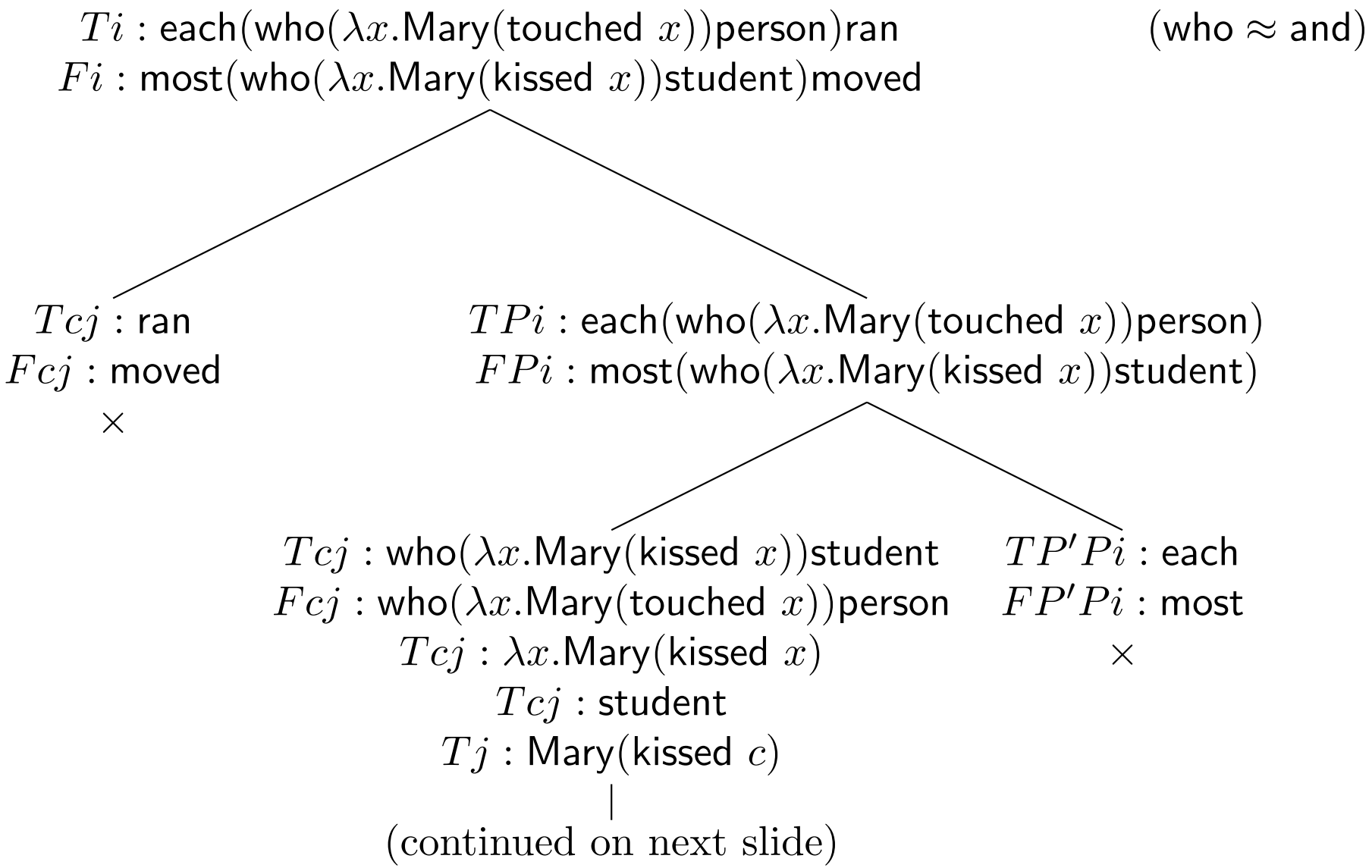
Rules Coming From the Format; Closure

- We will not distinguish between terms that are $\beta\eta$ equivalent.
- $$\begin{array}{l} X\vec{c}: Aa \\ \quad | \\ Xa\vec{c}: A \end{array}$$ where X is T or F.
- A branch is closed if it contains both $T\vec{c}: A$ and $F\vec{c}: A$.
- A branch is also closed if it contains $T\vec{c}: A$ and $F\vec{c}: B$, while $A \subset B$ is lexical knowledge.
- Closure of all branches means validity.

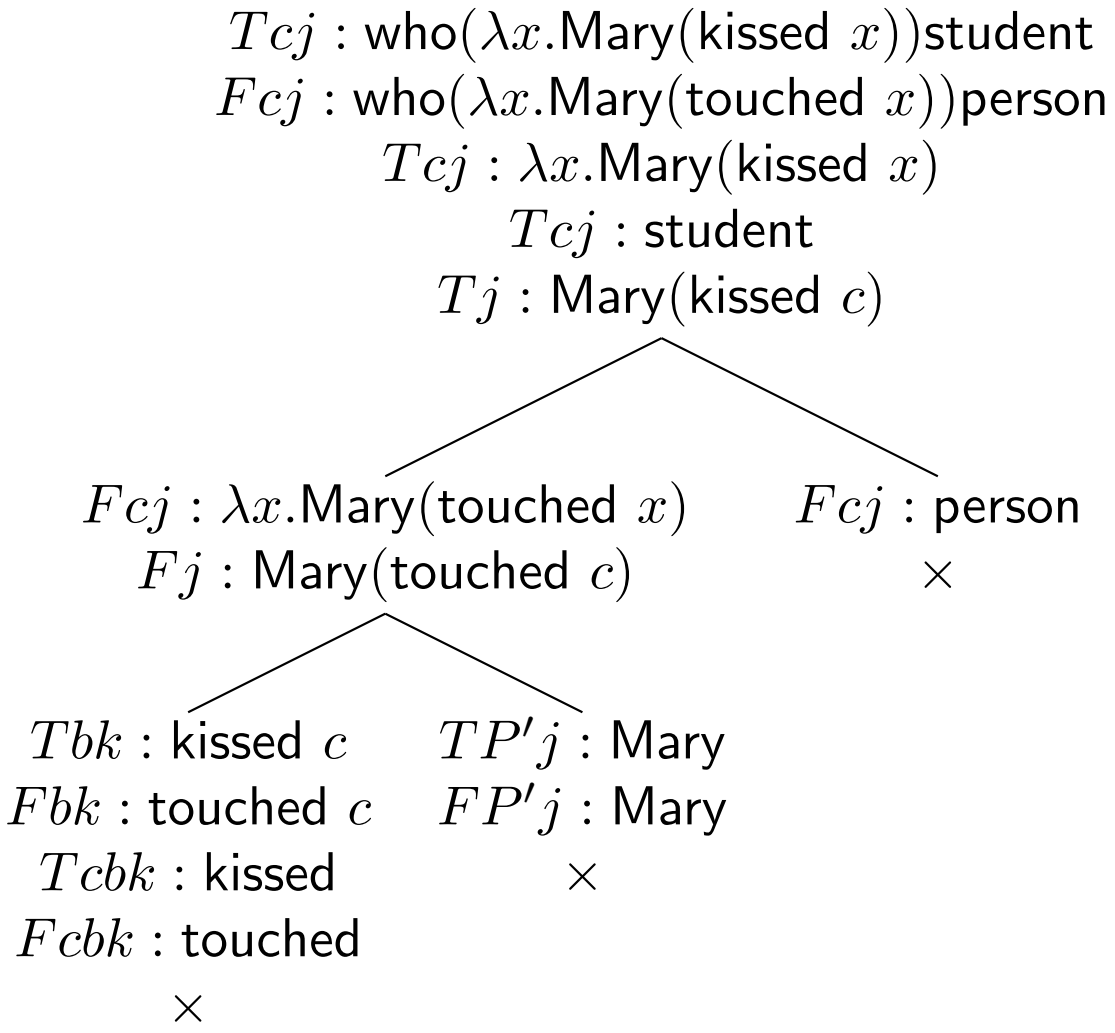
No Bird Moved. Therefore, No Lark Flew



Each Person Who Mary Touched Ran



Each Person Who Mary Touched Ran (continued)



Further Properties of Functions

- Anti-additivity: Downward Monotonicity +
  $F(X) \cap F(Y) \subset F(X \cup Y)$
- Antimorphicity: Anti-additivity +
  $F(X \cap Y) \subset F(X) \cup F(Y)$
  $F(X) \cup F(Y) \subset F(X \cap Y)$
- While e.g. few children is downward monotonic, no-one and without are also anti-additive, while not and n't are antimorphic.
- Zwarts (1981) shows that some negative polarity items in fact require more than just downward monotonicity, e.g. anti-additivity.

Rule for Anti-additive Operators

- $F\vec{c}: A(B \texttt{ or } C)$ if A is anti-additive.

  $F\vec{c}: AB$ $\quad$ $F\vec{c}: AC$

- Ti : (no bird) moved

  Ti : (no lark) sang

  Fi : (no lark) (flew or sang)

  Fi : (no lark) flew $\quad$ Fi : (no lark) sang

  $\vdots$ $\quad$ $\times$

  $\times$

  (as before)

Percolation of Boolean Operators

- Rules such as the one for anti-additivity sometimes allow us to get rid of Boolean operators that are embedded in the argument of a function.
- It is always possible to "percolate" these operators from a functor position and distribute them over lambdas. E.g.:
- $$\begin{array}{ccc} X\vec{c}: (A \texttt{ and } B)C & & X\vec{c}: (\lambda x.A \texttt{ and } B) \\ \Big| & & \Big| \\ X\vec{c}: AC \texttt{ and } BC & & X\vec{c}: (\lambda x.A) \texttt{ and } (\lambda x.B) \end{array}$$
- There are similar rules for `or` and `not`.

Derived Rules

- If G is mon$\uparrow$:

$$\begin{array}{c} T\vec{c} : G(A \texttt{ and } B) \\ | \\ T\vec{c} : GA \\ T\vec{c} : GB \end{array} \qquad \begin{array}{c} F\vec{c} : G(A \texttt{ or } B) \\ | \\ F\vec{c} : GA \\ F\vec{c} : GB \end{array}$$

- If G is mon$\downarrow$:

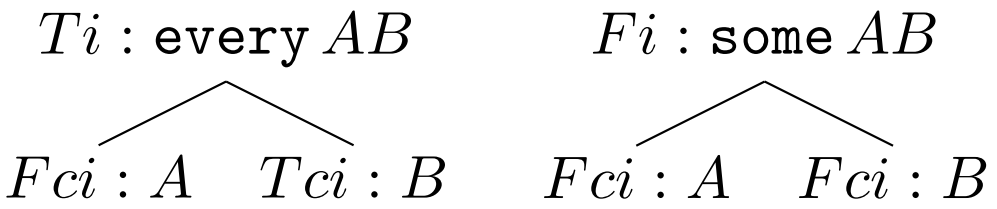
$$\begin{array}{c} T\vec{c} : G(A \texttt{ or } B) \\ | \\ T\vec{c} : GA \\ T\vec{c} : GB \end{array} \qquad \begin{array}{c} F\vec{c} : G(A \texttt{ and } B) \\ | \\ F\vec{c} : GA \\ F\vec{c} : GB \end{array}$$

- Allow us to get rid of `and` and `or` at least in some cases.
- There are more rules allowing such removals—depending on the properties of functors.

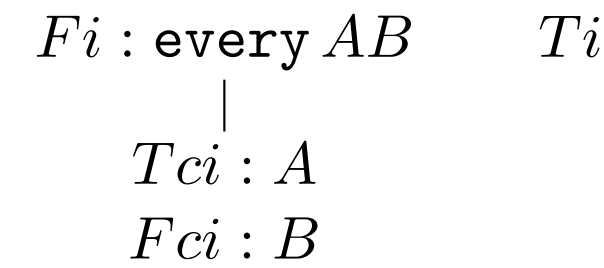
More Rules Removing Boolean Operators

- $$\frac{F\vec{c} : A(B \texttt{ and } C)}{F\vec{c} : AB \quad F\vec{c} : AC}$$ if A has meet.
- $$\frac{T\vec{c} : A(B \texttt{ or } C)}{T\vec{c} : AB \quad T\vec{c} : AC}$$ if A is splitting.
- The two properties above were taken from Van der Does (1992), in which a bunch of similar properties are discussed. no N and every N have meet, while some N is splitting.
- $$\frac{Xi : QA(A \texttt{ and } B)}{Xi : QAB}$$ if Q is conservative.
- $$\frac{Xi : \texttt{every}\, A(\texttt{not}\, B)}{-Xi : \texttt{some}\, AB}$$ (a duality rule)

Rules for every and some: Going All the Way

- 

c must be old.

- 

c must be fresh.

- The bleeding & feeding character of these rules will make the logic undecidable if we accept them in their full generality.

Taming these Rules

- One way to tame these rules (at the price of incompleteness of course) is to block the re-use of old constants if they were introduced as fresh witnesses earlier.
- Preference for certain derivable rules, such as the ones discussed, plus:

$$\begin{array}{l} Xi : \texttt{some}\, AB \\ \quad\quad | \\ Xi : \texttt{some}\, BA \end{array} \qquad \text{(symmetry)}$$

Tableaux Help Formalize Many Modes of Reasoning

- We have until now emphasized classical validity.
- But tableaux also formalize a simple form of model generation.
- They can formalize closed world reasoning: Generate a (minimal) model and then check whether certain things are true in that model. (Johnson-Laird)
- They can also be used to formalize abduction (Aliseda, Cialdea Mayer & Pirri).
- They might square well with probabilistic inference.
- As Van Lambalgen and Stenning have emphasized in a recent book, there are many modes of reasoning in natural inference. Tableaux seem a good general vehicle to formalize them.

Conclusion

- In this talk I have reported on ongoing work with the aim of giving an analytic tableau calculus that is based on representations very close to natural language.
- Unlike the usual tableau calculi, the calculus necessarily must have many rules and devising the calculus amounts to a form of linguistic description (proof-theoretic semantics).
- Rules of the calculus are intimately connected with the distributional properties of certain items.
- Calculi like these lend themselves well to implementation.
- Tableaux can formalize many modes of reasoning and that will be needed for modeling human inference.
- `tinyurl.com/tilnatlog`