# Formal Semantics in Modern Type Theories: Theory and Implementation

Stergios Chatzikyriakidis and Zhaohui Luo

(Lectures at ESSLLI 2014)

# Overview of the Course

- Introduce
  - Modern Type Theories (MTTs)
  - Formal semantics in MTTs
- Explicate
  - Why?
  - How?
  - Then what?
- Explain that MTTs provide
  - Full-scale powerful alternative to Montague semantics
  - Advantages, both theoretical and practical

# Historical remarks on NL semantics

- Montague Grammar (MG)
  - Richard Montague (1930 – 1971)
  - In early 1970s: Lewis, Cresswell, Parsons, ...
  - Later developments: Dowty, Partee, …
- Other formal semantics
  - "Dynamic semantics/logic" (cf, anaphora)
  - Discourse Representation Theory (Kemp 1981, Heim 1982)
  - Situation semantics (Barwise & Berry 1983)
- Formal semantics in modern type theories (MTTs)
  - Ranta 1994 and recent development (this talk), making it a full-scale alternative to MG, being better, more powerful & with applications to NL reasoning based on proof technology (Coq, …).

*RHUL project* [http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html](http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html)

# Why MTT-semantics?

- Powerful semantic tools
  - Much richer typing mechanisms for formal semantics
  - Powerful contextual mechanism to model situations
- Practical reasoning on computers
  - Existing proof technology: proof assistants (Coq, Agda, Lego, …)
  - Applications of to NL reasoning
- Leading to both
  - "Real-world" modelling as in model-theoretic semantics
  - Effective inference based on proof-theoretic semantics

*Remark: new perspective & new possibility not available before!*

# Material to distribute

- Lecture slides by SC and ZL
  - References on the last two slides of this lecture (and other lectures)
- Course proposal for this course (good summary)
- Some old notes on MTT-semantics in 2011.

(Available from course web site.)

# Course Plan: Lectures I-V

- I & II (ZL): MTTs and MTT-semantics: introduction
  - Basic concepts of MTTs
  - MTT-semantics of basic & more advanced language features
- III (SC): Coq proof-assistant and implementation of MTT-semantics
- IV (SC): NL inference in MTTs & Coq
- V (ZL): Advanced issues including
  - MTT-semantics as both model-theoretic and proof-theoretic
  - Proof technology for NL reasoning
  - Lexical semantics: mathematical modelling and combination with distributional semantics

# I. Modern Type Theories & MTT-semantics

- Introduction to
  - I.1. Montague semantics
  - I.2. MTTs
  - I.3. MTT-semantics

# I.1. Montague semantics

- Semantic language of Montague semantics
  - Church's simple type theory (1940)
  - IL – Montague's "Intensional Logic" (this aspect of intensionality is omitted for simplification here.)
- Syntactic categories of NLs
  - Sentences (S): "John walks."
  - Common Nouns (CN): bank, school, book, man
  - Intransitive Verbs (IV): run, walk, talk, work
  - Adjectives (Adj): pretty, tired, handsome
  - ...

# Semantic types in Montague semantics

| **Type** | **Informal explanation** |
|---|---|
| t | Type of truth values |
| e | Type of all entities |
| $e \rightarrow t$ | Type of subsets of entities |

# Montague's semantics of categories

| **Category** | **Semantic Type** |
|---|---|
| S | t |
| CN | $e \rightarrow t$ |
| IV | $e \rightarrow t$ |
| Adj (CN/CN) | $(e \rightarrow t) \rightarrow (e \rightarrow t)$ or $e \rightarrow t$ … |

Note: NL elements are then given set-theoretic interpretations within those of the semantic types.

# Montague semantics: examples

- Common nouns (as functional subsets of entities)
  - man – CN
  - [man] : $e \rightarrow t$
- Verbs (as predicates over entities)
  - walk – IV
  - [walk] : $e \rightarrow t$
  - [John walks] = [walk](j), if j = [John] : e.
- Adjectives (as functions from subsets to subsets)
  - handsome – CN/CN
  - [handsome] : $(e \rightarrow t) \rightarrow (e \rightarrow t)$
  - [handsome man] = [handsome]([man]) : $e \rightarrow t$

# Montague semantics: example problem

- New developments in lexical semantics
  - Generative lexicon (Pustegovsky 1995)
  - Copredication (Asher 2010)
- Limitation of the Montagovian setting
  - Formalisation of new lexical theories in Montagovian setting plus subtyping (Asher & Pustejovsky. 2005; Asher 2008/2010)
  - Difficulties of the above approach (Luo 2010)
  - Reason: incompatibility of the Montagovian setting with subtyping
- Problem and solution
  - Montagovian setting presents a problem …
  - Solution: types instead of functional subsets + coercive subtyping

# Prelude of MTT-semantics: types v.s. sets

- Types are "collections of objects"
  - May be thought of as "manageable sets"
  - Model-theoretic
- Modern type theories have meaning theories:
  - Proof-theoretic
  - Meanings given by means of inferential roles
- Some typical differences
  - Typing is decidable: "a:A" is decidable (in intensional TTs), while the set membership "$a \in S$" is not.
  - Type theories can have an embedded/consistent logic, by propositions-as-types, while set theory is only a theory in FOL.

*MTTs presents a new perspective and a new possibility!*

# I.2. Introduction to Modern Type Theories

- Introduction to Modern Type Theories by
  - Explaining basics of MTTs
  - Introducing propositions-as-types principle
- Introduction to MTTs by comparing
  - Types and sets
  - Simple TT (Church & Montague) and MTTs
- Introduction to subtyping in MTTs by
  - Showing inadequacy of traditional subsumptive subtyping
  - Introducing coercive subtyping and its crucial roles in MTT-semantics

*(This last aspect is to be introduced in next lecture.)*

# Typing – some general remarks

- The typing relation (or judgement)

  $a : A$

  Usually specified by means of a proof system.

  What can be "A" in "a : A"?

  - Types: eg,

    Nat, List(Nat), Table, Man, Man$\rightarrow$Prop, Phy$\times$Info, Phy$\bullet$Info

  - Propositions ("propositions-as-types"): eg,

    $\forall x{:}[man].\ [handsome](x) \rightarrow \neg[ugly](x)$

  - Advanced types: dependent types, type universes (see later)

- What typing is not:
  - "a : A" is not a logical formula.
    - 7 : Nat
    - Different from a logical formula is_nat(7)
  - "a : A" is different from the set-theoretic membership relation "a∈S" (the latter is a logical formula in FOL).
- What typing is related to:
  - Meaningfulness (ill-typed ➔ meaningless)
  - Semantic/category errors (eg, "A table talks.")
  - Type presuppositions (Asher 2011)

# Simple v.s. Modern Type Theories

- Church's simple type theory (Montague semantics)
  - Base types ("single-sorted"): e and t
  - Composite types: $e \to t$, $(e \to t) \to t$, …
  - Formulas in HOL (eg, membership of sets)
    - Eg, $s : e \to t$ is a set of entities ($a \in s$ iff $s(a)$)
- Modern type theories
  - Many types of entities – "many-sorted"
    - Table, Man, Human, Phy, … are all types (of certain entities).
  - Different MTTs have different embedded logics
    - Martin-Löf's type theory: first-order logic (but not the standard one)
    - Impredicative UTT: higher-order logic (standard one)

# MTTs (1) – Types

- Propositional types ("props-as-types")

| formula | type | example |
|---|---|---|
| $A \supset B$ | $A \rightarrow B$ | If …, then … |
| $\forall x{:}A.B(x)$ | $\Pi x{:}A.B(x)$ | Every man is handsome. |

- Inductive and dependent types
  - Nat, finite types (0, 1, 2,…), List(A), Vect(A,n), …
  - $\Sigma(A,B)$ (intuitively, { (a,b) | a : A & b : B(a) })
    - [handsome man] = $\Sigma$([man], [handsome])
  - $\Pi x{:}A.B(x)$ (intuitively, { f : $A \rightarrow \cup_{a \in A} B(a)$ | a : A & b : B(a) })
  - A+B, AxB, …
- Universes
  - A universe is a type of (some other) types.
  - Eg, CN – a universe of the types that interpret CNs
- Other types: Phy, Table, A•B, …

# Type of natural numbers

Formation $\dfrac{}{Nat : Type}$

Introduction $\dfrac{}{0 : Nat} \quad \dfrac{n : Nat}{n + 1 : Nat}$

Elimination

$$\frac{c : C(0) \qquad f(n) : C(n) \to C(n+1)\ [n : Nat]}{Rec(c, f)\ :\ \Pi x : Nat.\ C(x)}$$

Computation (behaviour of Rec, omitted)

# Elimination rule explained

Elimination

$$\frac{c : C(0) \qquad f(n) : C(n) \to C(n+1)\ [n : Nat]}{Rec(c, f)\ :\ \Pi x : Nat.\ C(x)}$$

Elimination and induction:

$$\frac{C(0) \qquad C(n) \supset C(n+1)\ [n : Nat]}{\forall x.\ C(x)}$$

"If C holds for all canonical nats, then C holds for every nat."

- General pattern (for all inductive types):

C holds for all *canonical* objects of ...

==========

C holds for *every* object of ...

# More inductive types: the Boolean type $\mathbb{2}$

Formation

$$\frac{}{2 : Type}$$

Introduction

$$\frac{}{true : 2} \qquad \frac{}{false : 2}$$

Elimination

$$\frac{c_1 : C(true) \qquad c_2 : C(false)}{\mathcal{E}_2(c_1, c_2) \;:\; \Pi x : 2.\, C(x)}$$

Computation (omitted)

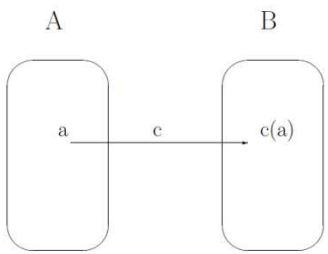
# MTTs (2): Coercive Subtyping

- Basic idea: subtyping as abbreviation
  - A≤B if there is a (unique) coercion c from A to B.

  Eg. Man ≤ Human; ∑(Man, handsome) ≤ Man; …
- Adequacy for MTTs (Luo, Soloviev & Xue 2012)
  - Coercive subtyping is adequate for MTTs
  - Note: traditional subsumptive subtyping is not.
- Subtyping essential for MTT-semantics
  - [walk] : Human→Prop, [Paul] = p : [handsome man]
  - [Paul walks] = [walk](p) : Prop

    because p : [handsome man] ≤ Man ≤ Human
- Very useful in modelling linguistic features

*(More in next lecture)*



# MTTs (3): examples

- Predicative type theories
  - Martin-Löf's type theory
  - Extensional and intensional equalities in TTs
- Impredicative type theories
  - Prop
    - Impredicative universe of logical propositions (cf, t in simple TT)
    - Internal totality (a type, and can hence form types, eg Table$\rightarrow$Prop, Man $\rightarrow$Prop, $\forall$X:Prop.X,
  - F/F$^{\omega}$ (Girard), CC (Coquand & Huet)
  - ECC/UTT (Luo, implemented in Lego/Plastic)
  - pCIC (implemented in Coq/Matita)

# MTTs (4): Technology and Applications

- Proof technology based on type theories
  - Proof assistants
    - MTT-based: ALF/Agda, Coq, Lego, NuPRL, Plastic, …
    - HOL-based: Isabelle, HOL, …
- Applications of proof assistants
  - Math: formalisation of mathematics (eg, 4-colour theorem in Coq; Kepler conjecture in Isabelle & HOL-Light in Flyspeck project)
  - CS: program verification and advanced programming
  - Computational Linguistics
    - E.g., MTT-sem based NL reasoning in Coq (Chatzikyriakidis & Luo 2014)

# I.3. MTT-semantics

- Formal semantics in modern TTs
  - Formal semantics in the Montagovian style
  - But, in modern type theories (not in simple TT)
- Key differences from the Montague semantics:
  - CNs interpreted as types (not predicates of type $e \rightarrow t$)
  - Rich type structure provides fruitful mechanisms for various linguistic features
    - CNs, Adj/Adv modifications, coordination, copredication, linguistic coercions, …
  - "Both" model-theoretic & proof-theoretic; hence
    - theoretically powerful & practically useful in computer-assisted reasoning, resp.
- Some work on MTT-semantics
  - Ranta (1994): basics of MTT-semantics
  - A lot of recent developments … …; for references, see for example: *http://www.cs.rhul.ac.uk/home/zhaohui/lexsem.html*

# MTT-semantics

| **Category** | **Semantic Type** |
|---|---|
| S | Prop |
| CNs (book, man, …) | types (each CN is interpreted as a type: [book]. [man], …) |
| IV | A→Prop (A is the "meaningful domain" of a verb) |
| Adj | A→Prop (A is the "meaningful domain" of an adjective) |

# MTT-semantics: examples

- Sentences as propositions: [A man walks] : Prop
- Common nouns as types: [man], [human], [table] : Type
- Verbs as predicates: [shout] : [human]$\rightarrow$Prop
  - [A man shouts] = $\exists$m:[man]. [shout](m) : Prop
  - Only well-typed because [man] $\leq$ [human] – subtyping is crucial.
- Adjectives as predicates: [handsome] : [man]$\rightarrow$Prop
  - Modified CNs as $\Sigma$-types: [handsome man] = $\Sigma$([man], [handsome])
  - Coercive subtyping is crucial: [handsome man] $\leq$ [man]
  - Other classes of adjectives (Chatzikyriakidis & Luo 2013)
- Adverbs as polymorphic functions:
  - [quickly] : $\prod$A:CN. (A$\rightarrow$Prop)$\rightarrow$(A$\rightarrow$Prop), where CN is universe of CNs
  - Cf, [Luo 2011, Chatzikyriakidis 2014]

# MTT-semantics (1): sentences & CNs

- Sentences (as propositions)
  - [John walks] : Prop
  - [A man walks] : Prop
- Common nouns are interpreted as types
  - [man], [book], [table] : Type (fine-grained)
  - Remark: not as sets of type $e \rightarrow t$ as in Montague semantics
- Other semantics types
  - Eg, Phy/Info – the type of physical/informational entities

# MTT-semantics (2): verbs

- Verbs are interpreted as predicates over "meaningful" domains
  - [shout] : [human]$\rightarrow$Prop
  - Note: "A table shouts" is meaningless (a "category error") in the sense that $\exists$t:[table]. [shout](t) is ill-typed (not "false", as in Montague's semantics).
- We need:
  - [John shouts] = [shout](j) : Prop, for j : [man]
  - [A man shouts] = $\exists$m:[man]. [shout](m) : Prop
  - But these are ill-typed! ([man] is not [human])
- Subtyping
  - [man] $\leq$ [human], the above become well-typed.
  - Subtyping is crucial for type-theoretical semantics! (Things only work in the presence of subtyping.)

# MTT-semantics (3): adjectives & modified CNs

- Adjectives, like verbs, are interpreted as predicates over "meaningful" domains
  - [handsome] : [man]→Prop
  - Note: "A table is handsome" is meaningless (a "category error") in the sense that ∃t:[table].[handsome](t) is ill-typed (not "false", as in Montague's semantics).
- Modified CNs
  - ∑-types for modified CNs
  - [handsome man] = ∑([man], [handsome])
  - Subtyping is needed as well (A handsome man is a man …)
  - More on subtyping & adjectives later

# MTT-sem (4): predicate-modifying adverbs

- Advanced features in MTTs are useful
  - Semantics to adverbs: example of using type universes & $\prod$
- Montague semantics:
  - [quickly] : $(e \to t) \to (e \to t)$
  - [John walked quickly] = [quickly]([walk], j) : t
- How in MTT?
  - Problem: We have many types that interpret CNs (Table, Man, Animated, …), not a single e.
  - Solution:
    - Introduce universe CN of types that interpret CNs
    - [quickly] : $\prod$A:CN. $(A \to Prop) \to (A \to Prop)$
    - [John walked quickly] = [quickly]([animated], [walk], j) : Prop
  - Remark: the above type of [quickly] is both polymorphic and dependent.

# MTT-sem (5): some advanced linguistic features

- Anaphora analysis and generalised quantifiers
  - MTTs provide alternative mechanisms for proper treatments via $\Sigma$-types [Sundholm 1989] (cf, DRTs, dynamic logic, …)
  - GQs [Sundholm 1989, Lungu & Luo 2014]
    - [every] : $\Pi$ A:CN. (A→Prop)→Prop
- Linguistic coercions
  - Coercive subtyping provides a promising mechanism (Asher & Luo 2012)
- Copredication
  - Cf, [Pustejovsky 1995, Asher 2011, Retoré et al 2010]
  - Dot-types [Luo 2009, Xue & Luo 2012]
- Other lexical sem: sense disambiguation etc. [Luo 2009, 2012]

# References


- N. Asher. *Lexical Meaning in Context: A Web of Words.* Cambridge University Press. 2011.
- N. Asher and Z. Luo. Formalisation of coercions in lexical semantics. Sinn und Bedeutung 17, Paris. 2012.
- S. Chatzikyriakidis. Adverbs in a Modern Type Theory. LACL 2014, LNCS 8535. 2014.
- S. Chatzikyriakidis and Z. Luo. Adjectives in a Modern Type-Theoretical Setting. The 18th Conf. on Formal Grammar, Dusseldorf. LNCS 8036. 2013.
- S. Chatzikyriakidis and Z. Luo. An Account of Natural Language Coordination in Type Theory with Coercive Subtyping. Constraint Solving and Language Processing 2012, LNCS 8114. 2013.
- S. Chatzikyriakidis and Z. Luo. Natural Language Reasoning Using Proof-assistant Technology: Rich Typing and Beyond. EACL Workshop on Type Theory and Natural Language Semantics (TTNLS), Goteborg, 2014.
- G. Lungu and Z. Luo. Monotonicity Reasoning in Formal Semantics Based on Modern Type Theories. LACL 2014, LNCS 8535. 2014.

# References


- Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science.* OUP, 1994.
- Z. Luo. Type-theoretical semantics with coercive subtyping. SALT20. 2010.
- Z. Luo. Contextual analysis of word meanings in type-theoretical semantics. Logical Aspects of Computational Linguistics (LACL'2011). LNAI 6736, 2011.
- Z. Luo. Common nouns as types. LACL'12, LNCS 7351. 2012.
- Z. Luo. Formal Semantics in Modern Type Theories with Coercive Subtyping. Linguistics and Philosophy, 35(6). 2012.
- Z. Luo. Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse. LNCS 8535. 2014.
- Z. Luo and P. Callaghan. Coercive subtyping and lexical semantics (extended abstract). Logical Aspects of Computational Linguistics (LACL'98). 1998.
- A. Ranta. *Type-Theoretical Grammar*. Oxford University Press. 1994.
- T. Xue and Z. Luo. Dot-types and their implementation. LACL'12, LNCS 7351. 2012.

# II. MTT-sem: Subtyping & Modelling Adjectives

- Subtyping (II.1 & II.2)
  - Needs for subtyping
  - Adequacy of coercive subtyping for MTTs
  - Uses of subtyping in MTT-semantics
- Adjectives in MTT-semantics (II.3)
  - Example case in linguistic modelling
  - Modelling intersective adjectival modifications
  - More advanced issues in modelling other adjectives

# II.1. Subtyping in MTT-semantics

- Need for subtyping
  - Problem in Montagovian setting (eg, copredication)
  - Also fundamentally needed for TT semantics
    - CNs as types – so subtyping needed:
      - A man shouts.
      - John : Man and [shout] : Human→Prop.
      - [shout](John) requires Man < Human.
- Coercive subtyping
  - Adequate (and powerful) framework for MTTs
    - Traditional "subsumptive subtyping" is inadequate for MTTs
  - Coercive subtyping are very useful in formal semantics (and, in particular, lexical semantics).

# Subtyping problem in the Montagovian setting

- Problematic example (in Montague semantics)
  - [heavy] : $(Phy \rightarrow t) \rightarrow (Phy \rightarrow t)$
  - [book] : $Phy \bullet Info \rightarrow t$
  - [heavy book] = [heavy]([book]) ?
  - In order for the above to be well-typed, we need

    $$Phy \bullet Info \rightarrow t \leq Phy \rightarrow t$$

    By contravariance, we need

    $$Phy \leq Phy \bullet Info$$

    But, this is <u>not</u> the case (the opposite is)!
- In MTT-semamtics, because CNs are interpreted as types, things work as intended.

# Subsumptive subtyping: traditional notion

- "Subsumptive subtyping":

$$\frac{a : A \qquad A \leq B}{a : B}$$

- Fundamental principle of subtyping

  *If $A \leq B$ and, wherever a term of type B is required, we can use a term of type A instead.*

  For example, the subsumption rule realises this.

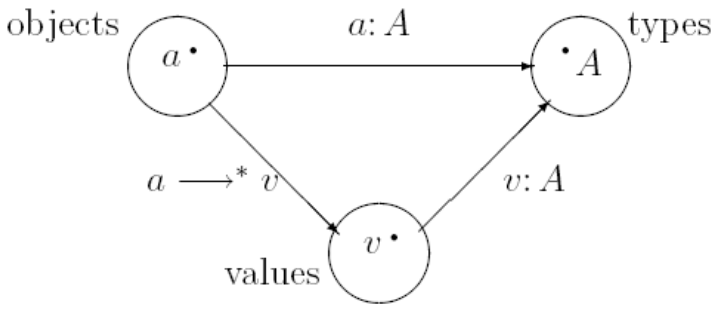
*Question:*

*Is subsumptive subtyping adequate for type theories with canonical objects?*

*Answer:*

*No (canonicity fails) and then what?*

# Canonicity




Examples:

- A = N, a = 3+4, v = 7.
- A = N×N, a = (λx:N.⟨x,x+1⟩)(2), v = ⟨2,3⟩.

- Definition

  *Any closed object of an inductive type is computationally equal to a canonical object of that type.*

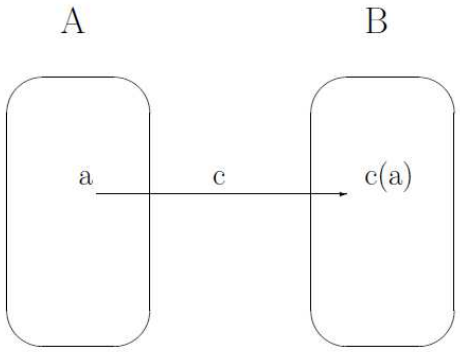
- This is a basis of TTs with canonical objects.
  - This is why the elimination rule is adequate.
  - Eg, Elimination rule for List(T):

    *"For any family C, if C is inhabited for all canonical T-lists nil(T) and cons(T,a,l), then so is C for all T-lists."*

- Canonicity is lost in subsumptive subtyping.
  - Eg, $$\frac{A \leq B}{List(A) \leq List(B)}$$
  - nil(A) : List(B), by subsumption;
  - But nil(A) $\neq$ any canonical B-list nil(B) or cons(B,b,l).
  - The elim rule for List(B) is inadequate: it does not cover nil(A) … …

# Coercive subtyping: basic idea

- A≤B if there is a coercion c from A to B:



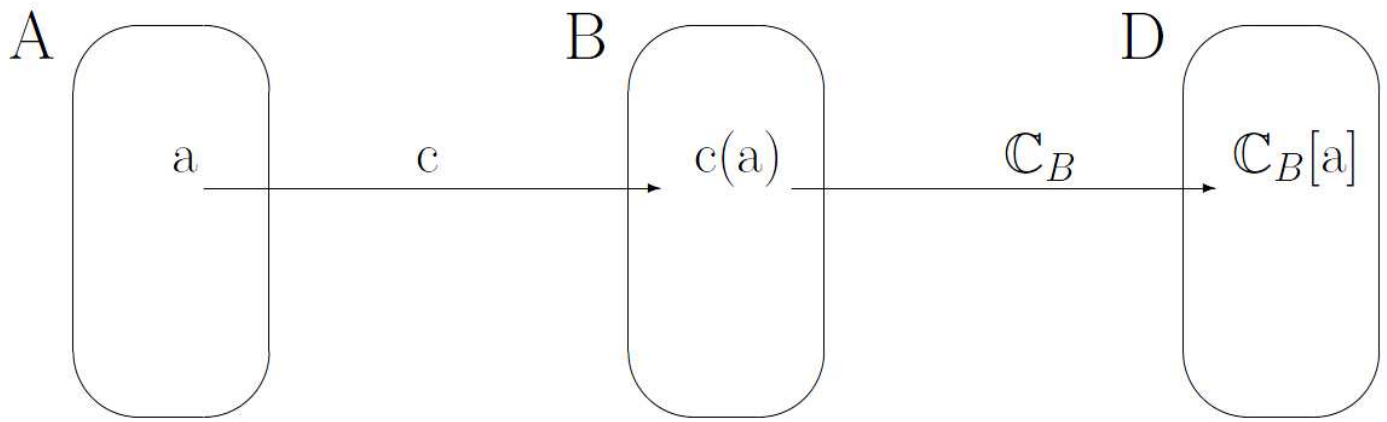

Eg. Even ≤ Nat; Man ≤ Human; ∑(Man, handsome) ≤ Man; …

- Subtyping as abbreviations:

  $a : A \leq_c B$

  ➔ "a" can be regarded as an object of type B

  ➔ $\mathbf{C}_B[a] = \mathbf{C}_B[c(a)]$, ie, "a" stands for "c(a)"


A
B
D
a
c
c(a)
$\mathbb{C}_B$
$\mathbb{C}_B$[a]

# Subtyping: basic need in MTT-semantics

- What about, eg,
  - "A man is a human."
  - "A handsome man is a man" ?
  - "Paul walks", with p=[Paul] : [handsome man]?
- Solution: coercive subtyping
  - [man] $\leq$ [human]
  - [handsome man] = $\Sigma$([man], [handsome]) $\leq_{\pi_1}$ [man]
  - [Paul walks] = [walk](p) : Prop
    because
    [walk] : [human]$\rightarrow$Prop and
    p : [handsome man] $\leq_{\pi_1}$ [man] $\leq$ [human]

# Coercive subtyping: summary

- Inadequacy of subsumptive subtyping
  - Canonical objects
  - Canonicity: key for TTs with canonical objects
  - Subsumptive subtyping violates canonicity.
- Adequacy of coercive subtyping
  - Coercive subtyping preserves canonicity & other properties.
  - Conservativity (Luo, Soloviev & Xue 2012)
- Historical development and applications in CS
  - Formal presentation (Luo 1996/1999, Luo, Soloviev & Xue 2012)
  - Implementations in proof assistants: Coq, Lego, Plastic, Matita

# II.2. Modelling Advanced Linguistic Features

- MTTs
  - Very useful in modelling advanced linguistic features
  - Partly because of
    - Rich/powerful typing (eg, dependent typing! See, eg, linguistic coercions in II.3)
    - Subtying
- Examples
  - As shown in a slide in last lecture, repeated here next.

*II.2 focusses on how coercive subtyping helps.*

# Recall the following slide on "linguistic features" (last lecture)

- Anaphora analysis and generalised quantifiers
  - MTTs provide alternative mechanisms for proper treatments via $\Sigma$-types [Sundholm 1989] (cf, DRTs, dynamic logic, …)
  - GQs [Sundholm 1989, Lungu & Luo 2014]
    - [every] : $\Pi$ A:CN. (A→Prop)→Prop
- Linguistic coercions
  - Coercive subtyping provides a promising mechanism (Asher & Luo 2012)
- Copredication
  - Cf, [Pustejovsky 1995, Asher 2011, Retoré et al 2010]
  - Dot-types [Luo 2009, Xue & Luo 2012]
- Other lexical sem: sense disambiguation etc. [Luo 2009, 2012]

# Remark on anaphora analysis

- Various treatments of "dynamics"
  - DRTs, dynamic logic, …
  - MTTs provide a suitable (alternative) mechanism.
- Donkey sentences
  - Eg, "Every farmer who owns a donkey beats it."
  - Montague semantics

    $\forall x.\ farmer(x) \ \&\ [\exists y.\ donkey(y) \ \&\ own(x,y)] \Rightarrow beat(x,?y)$
  - Modern TTs ($\Pi$ for $\forall$ and $\sum$ for $\exists$; Sundholme):

    $\Pi x{:}Farmer \Pi z{:}[\sum y{:}Donkey.\ own(x,y)]\ beat(x,\pi_1(z))$
- But, this is only an interesting point …

# Coercive subtyping in MTT-semantics

1. Needs for subtyping (earlier slides)
2. Sense enumeration/selection via. overloading
3. Coercion contexts and local coercions
4. Dot-types and copredication
5. Linguistic coercions

Notes:

- Focus on representation mechanisms, rather than NL semantic treatments.
- However, linguistic examples, rather than formal details.

# 2. Sense selection via overloading

- Sense enumeration (cf, Pustejovsky 1995 and others)
  - Homonymy
  - Automated selection
  - Existing treatments (eg, Asher et al via +-types)
- For example,

  1. John runs quickly.
  2. John runs a bank.

  with homonymous meanings

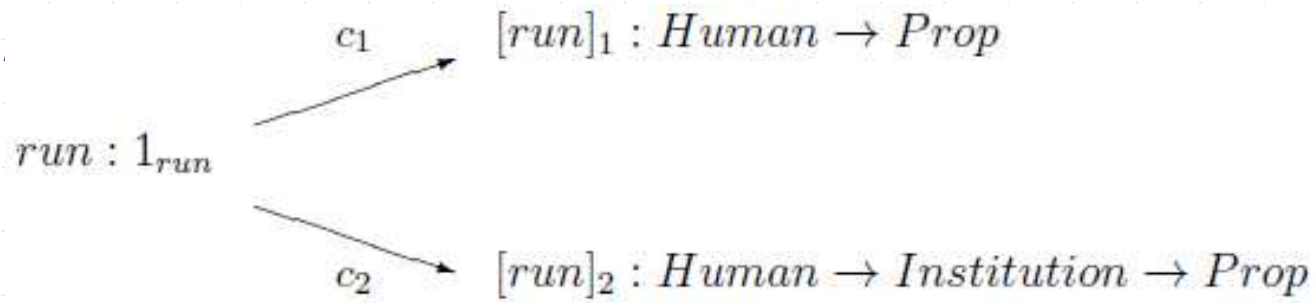
  1. $[\text{run}]_1$ : Human→Prop
  2. $[\text{run}]_2$ : Human→Institution→Prop

  "run" is overloaded – how to disambiguate?

# Overloading via coercive subtyping

- Overloading can be represented by coercions

  Eg




- Homonymous meanings can be represented.
- Automated selection according to typings

Question: What if typings cannot disambiguate (eg, bank)?

A solution: Local coercions

# 3. Coercion contexts and local coercions

- Coercion contexts

  $x{:}C, \ldots, A \leq_c B, \ldots \vdash \ldots \ldots$

- Useful in representing context-sensitivity
  - Eg, reference transfer

    *The ham sandwich shouts.*

    This can be interpreted in a context that contains, eg,

    [sandwich] < [human]

    that coerces sandwich into the person who has ordered a sandwich.

*Remark: "coherent contexts" needed, not just valid contexts.*

(Formal details omitted.)

- Local coercions (in terms/judgements)

  coercion $A \leq_c B$ in t
- Useful in disambiguation
  - Eg, "bank" has different meanings in

    (1) the bank of the river

    (2) the richest bank in the city
  - We might consider two coercions:

    $c_1 : 1_{bank} \rightarrow Type \quad c_1(bank) = [bank]_1$

    $c_2 : 1_{bank} \rightarrow Type \quad c_2(bank) = [bank]_2$

    But this is incoherent!

- Solution: local coercions
  - Rather than two coercions for "bank" in the same context, (which is incoherent), we can use

    coercion $1_{bank} \leq_{c1}$ Type in [(1)]

    coercion $1_{bank} \leq_{c2}$ Type in [(2)]

# 4. Dot-types and copredication

- Dot-types in Pustejovsky's GL theory
  - Example: PHY•INFO
  - PHY•INFO $\leq$ PHY and PHY•INFO $\leq$ INFO
- Copredication

  "John picked up and mastered the book."

  $[\text{pick up}]: \text{Human} \rightarrow \text{PHY} \rightarrow \text{Prop}$

  $\leq \text{Human} \rightarrow \text{PHY•INFO} \rightarrow \text{Prop}$

  $\leq \text{Human} \rightarrow [\text{book}] \rightarrow \text{Prop}$

  $[\text{master}]: \text{Human} \rightarrow \text{INFO} \rightarrow \text{Prop}$

  $\leq \text{Human} \rightarrow \text{PHY•INFO} \rightarrow \text{Prop}$

  $\leq \text{Human} \rightarrow [\text{book}] \rightarrow \text{Prop}$

*Remark: CNs as types in type-theoretical semantics – so things work.*

# Modelling dot-types in type theory

- What is A•B?
  - Inadequate accounts (cf, (Asher 08)):
    - Intersection type
    - Product type
- Proposal (SALT20, 2010)
  - A•B as type of pairs that do not share components
  - Both projections as coercions
- Implementation
  - Being implemented in proof assistant Plastic by Xue.

$$\frac{A : Type \quad B : Type \quad \mathscr{C}(A) \cap \mathscr{C}(B) = \emptyset}{A \bullet B : Type}$$

$$\frac{a : A \quad b : B}{\langle a, b \rangle : A \bullet B} \qquad \frac{c : A \bullet B}{p_1(c) : A} \qquad \frac{c : A \bullet B}{p_2(c) : B}$$

$$\frac{a : A \quad b : B}{p_1(\langle a, b \rangle) = a : A} \qquad \frac{a : A \quad b : B}{p_2(\langle a, b \rangle) = b : B}$$

$$\frac{A \bullet B : Type}{A \bullet B <_{p_1} A : Type} \qquad \frac{A \bullet B : Type}{A \bullet B <_{p_2} B : Type}$$

# Example

- "heavy book"
  - [heavy] : Phy$\rightarrow$Prop
    $\leq$ Phy$\bullet$Info$\rightarrow$Prop
    $\leq$ [book]$\rightarrow$Prop
  - So,
    [heavy book] = $\Sigma$([book], [heavy])
    is well-formed!

# 5. Linguistic Coercions

- Basic linguistic coercions
  - (*) Julie enjoyed a book.
  - (**) $\exists$x: Book. enjoy(j, x)
  - enjoy : Human → Event → Prop
  - Book $<_{reading}$ Event
  - (*) Julie enjoyed reading a book.
- Local coercions to disambiguate multiple coercions:
  - **coercion** Book $<_{reading}$ Event **in** (**)
  - **coercion** Book $<_{writing}$ Event **in** (**)

# Dependent typing

- What about

  *(#) Jill just started War and Peace, which Tolstoy finished after many years of hard work. But that won't last because she never gets through long novels.*

  - Overlapping scopes of "reading" and "writing".
- A solution with dependent typing
  - Evt : Human → Type (Evt(h) is the type of events conducted by h : Human.)
  - start, finish, last : $\prod$h: Human. (Evt(h)→Prop)
  - Read, write : $\prod$h: Human. (Book→Evt(h))
  - Book $<_{c(h)}$ Evt(h), where c(h,b)=writing if "h wrote b" & c(h,b)=reading if otherwise (parameterised coercion over h)

- Then, (#) is formalised as
  - start(j,wp)
    & finish(t,wp)
    & ¬last(j,wp)
    & ∀lb : LBook.finish(j, $\pi_1$(lb))

    which is (equal to)

    start(j,reading(j,wp))
    & finish(t,writing(t,wp))
    & ¬last(j,reading(j,wp))
    & ∀lb : LBook.finish(j, c(j,$\pi_1$(lb)))

    as intended.

# II.3. Modelling Adjectives in MTT-semantics

- Classification of adjectives
  - Intersective adjectives (eg, handsome)
    - Adj(N) ➔ N & Adj(N) ➔ Adj
  - Subsective, but non-intersective, adjectives (eg, large)
    - Adj(N) ➔ N (but not the 2nd above)
  - Privative adjectives (eg, fake)
    - Adj(N) ➔ ¬N
  - Non-committal adjectives (eg, alleged)
    - Adj(N) ➔ ?
  - Temporal adjectives (eg, former)
    - Adj(N) ➔ ¬N (not necessarily)
- Our proposals in MTT-semantics (Chatzikyriakidis and Luo 2013)

# Intersective adjectives

- In general, adjectives are of type A→Prop, ie, predicates over "meaningful" domains
  - [handsome] : [man]→Prop
- Modified CNs by intersective adjectives
  - $\Sigma$-types for modified CNs by intersective adjectives (Ranta 1994)
  - [handsome man] = $\Sigma$([man], [handsome])

# Subsective but non-intersective adjectives

- Nature of such adjectives
  - Their meanings are dependent on the nouns they modify
  - Eg, "a large mouse" is not a large animal
- This leads to our following proposal:
  - [large] : $\Pi$A:CN. (A→Prop)
    - CN – type universe of all (interpretations of) CNs
    - $\Pi$ is the type of dependent functions
      - [large]([mouse]) : [mouse] → Prop
      - [large mouse] = $\Sigma$([mouse], [large]([mouse]))
  - [skilful] : $\Pi$A:$CN_H$. (A→Prop)
    - $CN_H$ – sub-universe of CN of subtypes of Human
      - [skilful]([doctor]) : [doctor] → Prop
    - Excludes expressions like "skilful car".

# Privative adjectives

- Disjoint union types
  - A+B with two injections inl : A→A+B and inr : B→A+B
- "fake gun"
  - $G_R$ – type of real guns
  - $G_F$ – type of fake guns
  - $G = G_R + G_F$ – type of all guns
  - Declare inl and inr both as coercions: $G_R <_{inl} G$ and $G_F <_{inr} G$
- Now, eg,
  - Can define "real gun" or "fake gun" inductively as predicates of type G→Prop so that [real gun](g) iff ¬[fake gun](g).
  - We can interprete if f : $G_F$, "f is not a real gun" as ¬[real gun](f).
    - Note that, [real gun](f) is only well-typed because $G_F <_{inr} G$.

# Non-committal adjectives

- Modelled as beliefs
  - "John is an alleged criminal" is interpreted as "Somebody believes that John is a criminal".
  - Employ Ranta's "belief context" (Ranta 1994)
    - $\Gamma_p = x_1 : A_1, \ldots, x_n : A_n$
    - $B(p,A) = \Pi\, x_1{:}A_1 \ldots \Pi\, x_n : A_n.\ A$
  - [alleged criminal] = $\sum$p:Human. B(p, [criminal])

*Remark*

- Here, a belief context is finite – not general enough.
- See Lecture V or (Luo 2014) for infinite contexts/signatures.

# Temporal adjectives

- Temporal adjectives like "former" or "past"
  - As a privative adjective (Partee 2007), we can then use disjoint union types to interpret it.
  - Instead, we can employ type families (dependent types).
- Consider a simple model of times:
  - Time : Type,  now : Time,  < : Time→Time→Prop
  - Nouns parameterised by times:  [president] : Time→CN
- [former president] = $\sum$t:Time. (t<now) $\times$ [president](t)
  - If the intuition says that a "former president" is not a president, we can then add "$\neg$[president](now)".
  - [former] = $\lambda$p:Time→CN.  $\sum$t:Time. (t<now) $\times$ p(t)
    : (Time→CN)→CN

# References (for Lecture II)


- N. Asher. A type driven theory of predication with complex types. Fundamenta Informaticae 84(2). 2008.
- S. Chatzikyriakidis and Z. Luo. Adjectives in a Modern Type-Theoretical Setting. The 18th Conf. on Formal Grammar, Dusseldorf. LNCS 8036. 2013.
- Z. Luo. Coercive subtyping in type theory. CSL'96, LNCS 1258. 1996.
- Z. Luo. Coercive subtyping. J. of Logic and Computation, 9(1). 1999.
- Z. Luo. Type-theoretical semantics with coercive subtyping. SALT20. 2010.
- Z. Luo. Formal Semantics in Modern Type Theories with Coercive Subtyping. Linguistics and Philosophy, 35(6). 2012.
- Z. Luo. Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse. LNCS 8535. 2014.
- Z. Luo, S. Soloviev and T. Xue. Coercive subtyping: theory and implementation. Information and Computation 223. 2012.
- B. Partee. Compositionality and coercion in semantics: the semantics of adjective meaning. In *Cognitive Foundations of Interpretation*, Netherlands Academy of Arts and Sciences. 2007.
- J. Pustejovsky. *The Generative Lexicon*. MIT. 1995.
- A. Ranta. *Type-Theoretical Grammar*. Oxford University Press. 1994.

# Formal Semantics in Modern Type Theories: Theory and Implementation: Lecture 3 - Proof-assistant technology/Intro to Coq

Stergios Chatzikyriakidis and Zhaohui Luo

August 2014

# Interactive theorem provers

- Started in the early 60s
  - The need for formally verified proofs
  - The AUTOMATH project (De Bruijn 1983, 1967 onwards)
    - ★ Aim: a system for the mechanic verification of mathematics
    - ★ Several AUTOMATH systems have been implemented
    - ★ The first system to practically exploit the Curry-Howard isomorphism

# Interactive theorem provers

- Proof-assistant technology has gone a long way since then
  - Proliferation of proof-assistants implementing various logical frameworks
    - ⋆ Classical logics/set theory (Mizar, Isabelle)
    - ⋆ Constructive Type Theories (MTTs, Coq, Lego, Plastic, Agda among other things)
  - Important verified proofs
    - ⋆ Four Colour Theorem (Gonthier 2004, Coq)
    - ⋆ Jordan curve theorem (Kornilowicz 2005, Hales 2007, Mizar and HOL respectively)
    - ⋆ The prime number theorem (Avigad et al 2007, Isabelle)
    - ⋆ Feit-Thompson theorem (Gonthier et al. 2012, Coq (170.000 lines of code!))
  - Other uses: Software verification
    - ⋆ CompCert: an optimized, formally verified compiler for C (Leroy 2013, Coq)
    - ⋆ Coq in Coq (Barras 1997): Construct a model of Coq in Coq and show all tactics are sound w.r.t this model (verify the correctness of a system using the system itself)

# The Coq proof-assistant

- INRIA project
  - Started in 1984 as an implementation of Coquand's Calculus of Constructions (CoC)
  - Extension to the Calculus of Inductive Constructions (CiC) in 1991
  - Coq offers a program specification and mathematical higher-level language called *Gallina* based on CiC
  - CiC combines both expressive higher-order logic as well as a richly typed functional programming language
- Winner of the 2013 ACM software system award
- A collection of 100 mathematical theorems proven in Coq: http://perso.ens-lyon.fr/jeanmarie.madiot/coq100/

# The Coq proof-assistant

- An ideal tool for formal verification
  - Powerful and expressive logical language
  - Consistent embedded logic
  - Built-in proof tactics that help in the development of proofs
  - Equipped with libraries for efficient arithmetics in $N$, $Z$ and $Q$, libraries about lists, finite sets and finite maps, libraries on abstract sets, relations and classical analysis among others
  - Built-in automated tactics that can help in the automation of all or part of the proof process
  - Allows the definition of new proof-tactics by the user
    - ⋆ The user can develop automated tactics by using this feature

# Installing Coq

- Easy to install (http://coq.inria.fr/download)
- Use the installer
  - The code in this lecture is compatible with the earlier version of Coq 8.3 rather than 8.4
    - ★ 8.4 version has some minor improvements that lead to minor incompatibilities with the earlier version
    - ★ Download the earlier version if you want to directly use the code ( version 8.3) (http://coq.inria.fr/coq-8.3)
    - ★ If you feel adventurous, read the differences pertaining to the new version, and revise code accordingly

# Theorem proving in Coq

- Coq can be used in order to prove theorems
  - A theorem is declared with the command *Theorem* plus the name of the theorem we want to prove, plus the theorem itself
    - ★ *Theorem* $x : a \Rightarrow b$
    - ★ The goal is to reach a complete proof using the proof tactics provided by the assistant

# Basics of Coq

- Typing
  - All objects have a type in Coq
    - ★ All the pre-defined objects in Coq can be checked for type using the command *check*
    - ★ For example the type *nat* of natural numbers has type *Set* (*nat* : *Set*), while natural numbers like 1,2,3 and so on, type nat (1 : *nat*).

```
Coq < Check nat.
nat:Set
Coq <Check 1.
1:nat
```

- Function application
  - Applying a function to an argument
    - ★ The addition function is of type $nat \rightarrow nat \rightarrow nat$, takes two nat arguments and also returns a *nat* argument

```
Coq < Check plus.
plus:nat -> nat -> nat
Coq < Check plus 3 4.
3 + 4:nat
```

# Basics of Coq

- Declarations
  - Associating a name with a specification
  - Specifications classify the object declared
    - ★ Three kinds of specifications: *Prop*, *Set* and *Type*, logical propositions, mathematical collections of objects and abstract types
    - ★ We can declare new types either by *Parameter* or via *Variable*
    - ★ We can restrict the scope by using local contexts, using *section.*

```
Coq < Variable H:Set.
H is assumed
Warning: H is declared as a parameter because it is at
a global level
Coq < Parameter H:Set.
H is assumed
Coq < Section section.
Coq < Variable H1:Set.
H1 is assumed
```

# Basics of Coq

- Definitions
  - *Definitionident* : *term*1 := *term*2
  - It checks that the type of *term*2 is definitionally equal to *term*1, and registers *ident* as being of type *term*1, and bound to value *term*2.
  - We can define a constant three to be the successor of the successor of the successor of 0 (the successor is pre-defined).

    ```
    Definition three:nat:= S (S(S((0))).
    ```

  - Coq can infer the type in these cases, so it can be dropped:

    ```
    Definition three:= S (S(S((0))).
    ```

  - Defining functions
    - ★ Square number function
    - ★ Uses $\lambda$ abstraction. Takes a *nat* to return a *nat*

      ```
      Definition square:= fun n:nat=> n*n.
      ```

# Basics of Coq

- Inductive types
  - Inductive types without recursion
    - ★ The inductive type for booleans
    - ★ Pre-defined in Coq in the following manner:

```
Coq < Inductive bool : Set := true | false.
bool is defined
bool_rect is defined
bool_ind is defined
bool_rec is defined
```

    - ★ The above introduces a new *Set* type, *bool*. Then the constructors of this *Set true* and *false* are declared, and three elimination rules are provided, allowing to reason with this type of types
    - ★ The bool_ ind combinator for example allows us to prove that every $b : bool$ is either *true* or *false* (more on this later)

# Basics of Coq

- Inductive types
  - Inductive types with recursion: Natural numbers

```
Coq < Inductive nat : Set :=
| O : nat
| S : nat -> nat.
nat is defined
nat_rect is defined
nat_ind is defined
nat_rec is defined
```

  - Recursive types are closed types
    - ★ Their constructors define all the elements of that type
    - ★ Peano's induction axiom (*nat_ind*) as well as general recursion is defined (*nat_rec*)

# An example of a simple proof

- Transitivity of implication: $(P \to Q) \to (Q \to R) \to (P \to R)$
- What is needed before we get into proof-mode
  - Declaring $P, Q, R$ as propositional variables (only elements of type Prop can be the arguments of logical connectives)

    ```
    Variables P Q R:Prop.
    ```

  - With this declaration at hand, we can get into proof-mode:

    ```
    Theorem trans: (P->Q)->(Q->R)->(P->R)
    ```

  - *intro* tactic: introduction of $(P \to Q)$, $(Q \to R)$ and $P$ as assumptions

    ```
    1 subgoal

      H : P -> Q
      H0 : Q -> R
      H1 : P
      ============================
      R
    ```

# An example of a simple proof in Coq

- The *apply* tactic: It takes an argument which can be decomposed into a premise and a conclusion (e.g. $Q \rightarrow R$), with the conclusion matching the goal to be proven ($R$), and creates a new goal for the premise.

```
H : P -> Q
H0 : Q -> R
H1 : P
============================
 Q
```

- We now use *apply* for $H$

```
H : P -> Q
H0 : Q -> R
H1 : P
============================
 P
```

# An example of a simple proof in Coq

- The tactic *assumption*: matches a goal with an already existing hypothesis. Applying *assumption* completes the proof

```
1 subgoal

  H : P -> Q
  H0 : Q -> R
  H1 : P
  ============================
   P

trans < assumption.
Proof completed.
```

# An example of a more complicated proof in Coq

- Peirce's law: If the law of the excluded middle holds, then so is the following: $((A \to B) \to A) \to A$
  - We formulate in Coq notation:

```
Definition lem:= A \/ ~ A.
Definition Peirce:= ((A->B)->A)->A.
Theorem lemP: lem -> Peirce.
```

  - We first use unfold to unfold the definitions. So *lem* and *Peirce* will be substituted by their definition

```
 lemP < unfold lem.
1 subgoal
============================
 A \/ ~ A -> Peirce
lemP < unfold Peirce.
1 subgoal
============================
A \/ ~ A -> ((A -> B) -> A) -> A
```

# An example of a more complicated proof in Coq

- Applying intro twice (we can use intros to apply intro as many times possible)

```
lemP < intros.
1 subgoal
H : A \/ ~ A
H0 : (A -> B) -> A
============================
A
```

- We can now use the *elim* tactic on *H*, basically using the elimination rules for disjunction:

```
H : A \/ ~ A
H0 : (A -> B) -> A
============================
A -> A
subgoal 2 is: ~A -> A
```

# An example of a more complicated proof in Coq

- We use intro and assumption and the first subgoal is proven

```
lemP < intro. assumption.
H : A \/ ~ A
H0 : (A -> B) -> A
============================
~A -> A
```

- Intro and apply H0

```
lemP < intro. apply H0.
H : A \/ ~ A
H0 : (A -> B) -> A
H1 : ~ A
============================
A -> B
```

# An example of a more complicated proof in Coq

- *Intro* and *absurd A*:

```
  lemP < absurd A.
2 subgoals
H : A \/ ~ A
H0 : (A -> B) -> A
H1 : ~ A
H2 : A
============================
~ A
subgoal 2 is:
 A
```

# An example of a more complicated proof in Coq

- *Absurd A* proves the goal from *False* and generates to subgoals, *A* and *not A*
- Using assumption twice, the proof is completed

```
  lemP < assumption. assumption.
1 subgoal
H : A \/ ~ A
H0 : (A -> B) -> A
H1 : ~ A
H2 : A
============================
A
Proof completed.
```

# Other useful proof tactics

- We discuss some of the basic predefined Coq tactics
- Following Chipalla (2014) we categorize these according to the connective involved in each case
  - Conjunction
    - ★ *Elim*: Use of the elimination rule
    - ★ *Split*: Splits the conjunction into two subgoals
    - ★ Examples:

```
Theorem conj: A/\B->A.
Theorem conj: B/\(A/\C)->A/\B.
```

  - Disjunction
    - ★ *Elim:* Elimination rule
    - ★ *Left,Right:* Deals with one of the two disjuncts

```
Theorem disj: (B\/(B\/C))/\(A\/B)->A\/B.
```

  - *Implication* ($\Rightarrow$) and *Forall*
    - ★ *Intro(s)*
    - ★ *Apply*

# Other useful proof tactics

- We discuss some of the basic predefined Coq tactics
- Following Chipalla (2014) we categorize these according to the connective involved in each case
  - Existential
    - ★ *exists t*: Instantiates an existential variable
    - ★ *elim*: Elimination rule

      ```
      Theorem NAT: exists x: nat, le 0 x.
      ```

  - Equality (=)
    - ★ *reflexivity, symmetry, transitivity*: The usual properties of equality
    - ★ *congruence*: Used when a goal is solvable after a series of rewrites
    - ★ *rewrite, subst*:Rewrites an element of the equation with the other element of the equation. *Subst* is used when one of the terms is a variable

# Other useful proof tactics - Induction tactics

- *induction*: *induction x* decomposes the goal statement to a property applying to x and then applies *elim x*
- *elim*: Similar tactic, does not add hypotheses in the context
- An example using inductive types. We define the inductive type season, consisting of four members, corresponding to each season:

```
month1 < Inductive season:Set:= Winter|Spring|Summer|
Autumn.
season is defined
season_rect is defined
season_ind is defined
season_rec is defined
```

- Coq automatically adds several theorems that make reasoning about the type possible. In the case above these are season_ rect season_ ind and season_ rec

# Other useful proof tactics - Induction tactics

- season_ ind provides the induction principle associated with an inductive definition. In this case this amounts to:

```
month1 < Check season_ind.
season_ind
:forall P : season -> Prop,
P Winter -> P Spring -> P Summer ->
P Autumn -> forall s : season, P s
```

- Universal quantification on a property $P$ of seasons, followed by a succession of implications, each premise being $P$ applied to each of the seasons. The conclusion says that $P$ holds for all seasons

# Other useful proof tactics - Induction tactics

- Let us say we want to prove the following:

```
SEASONEQUAL < Theorem SEASONEQUAL: forall s: season,
s=Autumn\/s=Winter\/s=Spring\/s=Summer.
```

- We apply *intro* and call *elim*

```
s : season
============================
Autumn = Autumn \/ Autumn = Winter \/ Autumn = Spring \/
Autumn = Summer
Winter = Autumn \/ Winter = Winter \/ Winter = Spring
\/ Winter = Summer
Spring = Autumn \/ Spring = Winter \/ Spring = Spring
\/ Spring = Summer
Summer = Autumn \/ Summer = Winter \/ Summer = Spring
\/ Summer = Summer
```

- Can be easily proven using *left*, *right* and *reflexivity* or using *auto*.

# Automation tactics

- Tactics that are a combination of more simple tactics, in effect a macro of tactics
  - Used to automate parts or the whole proof
  - Examples of such tactics
    - ⋆ The *auto* tactic: Provides automation in case a proof can be found by using any of the tactics:*intros, apply, split, left, right and reflexivity*
    - ⋆ The *eauto* tactic: A variant of *auto*. Uses tactics that are variants of the tactics used in *auto*, the only difference being that they can deal with conclusions involving existentials (for example *eapply*, functions like *apply* but further introduces existential variables)

# Automation tactics

- An example exemplifying the difference between *auto* and *eauto*
  - We define a predicate nat_ predicate and then create a theorem:

```
Parameter nat_predicate: nat->Prop.
Theorem NATPR: nat_predicate(9) -> exists n: nat,
nat_predicate(n).
```

  - Due to the existential, *auto* cannot prove the above, while *eauto* can
- However, the following can be proven by *auto* as well:

```
Variable j:nat.
Let h:= j.
Theorem NATPR: nat_predicate(j) -> nat_predicate(h)\/
 exists n:nat, nat_predicate(n).
```

  - In effect, the existential does not have to be dealt with, only the left disjunct is used
    - ⋆ *Eauto* cannot however open up existentials or conjunctions from context. This is made possible with another tactic called *jauto* (see next lecture)

# Automation tactics

- The tactics *tauto*, *intuition*
  - The first is used for propositional intuitionistic tautologies
  - The latter for first-order intuitionistic logic tautologies

```
 Coq < Theorem TAUTO: A\/B->B\/A.
1 subgoal

  ============================
   A \/ B -> B \/ A

TAUTO < tauto.
Proof completed.
```

# Imported modules

- A number of other more advanced tactics can be used by importing different Coq packages
  - E.g. the *Classical* module can be imported, which includes classical tautologies rather than intuitionistic

    ```
    Theorem CLASSICAL: not (not A)-> A.
    ```
  - The *Omega* module can be used in order to deal with goals that need Presburger arithmetic in order to be solved

    ```
    Theorem neq_equiv : forall x:nat, forall y:nat, x <> y <->
    ```
  - *Libtactics* is a collection of advanced tactics, basically advanced variations of the standard tactics
    - ★ For example, the *destructs* tactic is the recursive application of the *destruct* tactic

      ```
      Theorem DESTRUCTS: (A/\B/\C/\D)->B.
      ```

# MTT semantics in Coq

- Encoding MTT semantics based on theoretical work using Type Theory with Coercive Subtyping in Coq
  - Coq is a natural toolkit to perform such a task
    - ★ The type theory implemented in Coq is quite close to Type Theory with Coercive Subtyping
    - ★ Thus, the TT does not need to be implemented!
    - ★ What we need, is a way to encode the various assumptions as regards linguistic semantics and then reason about them

# The CN universe

- Common nous in MTTs are seen as types rather than predicates
- Zhaohui Luo proposed the introduction of a universe of CN interpretations (Luo 2011, 2012 among others)
  - A collection of the names of types that interpret common nouns
  - Coq does not support universe construction
    - ★ Only the pre-defined universes can be used
    - ★ In this sense, we define *CN* to be Coq's pre-defined *Set* universe

```
Definition CN:= Set.
Parameters Man Human Animal Object:CN
```

# Subtyping relations

- In order for type many-sortedness to have any advantages over more coarse grained typing (like the *e* typing in MG), a subtyping mechanism is needed
  - We have already seen the use of coercive subtyping as an adequate subtyping mechanism
  - Coq uses a similar mechanism (albeit with minor formal differences)
  - Subtyping in Coq is also based on the notion of coercion.
    - ★ An example is shown below:

```
Axiom MH: Man->Human. Coercion MH: Man>->Human.
Axiom HA: Human->Animal. Coercion HA: Human>->Animal.
```

# Types for verbs

- The type of propositions is identified with *Prop*.
  - Verbs are function types returning a *Prop* type once one or more (depending on valency) arguments have been provided
    - ★ However, given type many sortedness the arguments needed for individual verbs will be dependent on the specific verb in each case
    - ★ Thus, Walk will be specified as $Animal \rightarrow Prop$ while fall as $Object \rightarrow Prop$

```
Parameter walk: Animal-> Prop.
Parameter fall: Object-> Prop.
```

# Quantifiers, adjectives, adverbs

- Following work by Luo (2011, 2012) and Chatzikyriakidis and Luo (2013a,b,2014), quantifiers are given an inductive type, taking an $A : CN$ argument and returns the type $(A \rightarrow Prop) \rightarrow Prop$
- Adjectives are defined as simple predicates.
- VP adverbs are defined as predicate modifiers extending over the universe $CN$, while sentence adverbs as functions from propositions to propositions

```
Parameter some: forall A:CN, (A->Prop)->Prop
Parameter handsome: Human -> Prop
Parameter slowly: forall A:CN, (A->Prop)->(A->Prop).
Parameter fortunately: Prop ->Prop.
```

# Quantifiers, adjectives, adverbs

- More must be said about the lexical semantics of all these categories.
- For example, in the case of *some* the following will be assumed
  - Same typing but has further information on the lexical semantics of some (i.e. existential quantification)

    ```
    Definition some:= fun A:CN, fun P:(A->Prop)=> exists x: A, P(x).
    ```

  - More will be said about the lexical semantics as we proceed

# Adjectival modification using dependent record types

- Intersective and subsective adjectival modification have been treated as involving $\Sigma$ types.
- This is the analysis we follow here
  - We however follow Luo (2012) and use dependent record types instead of $\Sigma$ types (which are equivalent)
    - ★ The first projection is declared as a coercion
    - ★ Thus, for *handsome man*, we get the inference *man*

```
Record handsomeman:CN:=mkhandsomeman{ c :>Man;_: handsome c }.
```

# Reasoning with NL

- As soon as NL categories are defined, Coq can be used to reason about them
  - In effect, we can view a valid NLI as a theorem
    - ⋆ Thus, we formulate NLIs as theorems
    - ⋆ The antecedent and consequent must be of type *Prop* in order to be used in proof mode
    - ⋆ Thus, the first can be formulated as a theorem, but not the second:

```
Theorem EX:(walk) John-> some Man (walk).
Theorem WA:walk -> drive.
```

# Reasoning with NL

- The same tactics that can be used in proving mathematical theorems are used for NL reasoning
  - The aim is to predict correct NLIs while avoiding unwanted inferences
    - ★ For example, given the semantics for quantifier *some*, one can formulate the following theorem and further try to prove it

```
Theorem EX:  (walk) John-> some Man (walk).
```

# An NLI example

- One of the inferences we should be able to get when a proper name acts as an argument of the verb is one where an element of the same type as the proper name acts as the argument of the same verb
  - Basically, from a sentence like *John walks*, we should infer that *a man walks*
  - We formulate the theorem

```
Theorem EX:  (walk) John-> some Man (walk).
```

  - We unfold the definition for *some* and use *intro*

```
EX < intro.
1 subgoal
H : walk John
============================
exists x : Man, walk x
```

  - We use the exists tactic to substitute $x$ for *John*. Using *assumption* the theorem is proven

# An NLI example

- To the contrary, we should not be able to prove the opposite

```
Theorem EX: some Man (walk) -> (walk) John.
```

- Indeed, no proof can be found in this case.
  - We unfold *some* and use *intro*

```
EX < intro.
1 subgoal
H : exists x : Man, walk x
============================
walk John
```

  - From this point on, we can use any of the *elim*, *induction*, *case* tactics but at the end we reach a dead end

```
EX < intro.
1 subgoal
H : exists x : Man, walk x
x : Man
H0 : walk x
============================
walk John
```

# Automation?

- From a theoretical point a view, having a system that can reason about NL semantics in such a straightforward way is already something
  - From the practical side however, in order to develop something like this into a more practical device, automation needs to be possible
    - ★ For the simple case we have been discussing, automation is possible once we unfold the definition for *some*
    - ★ The tactic *eauto* will solve the theorem in one step

```
EX< unfold some.
1 subgoal
============================
walk John -> exists x : Man, walk x
EX < eauto.
Proof completed.
```

    - ★ Still, this is not yet full automation. What can we do?

# The tactic language Ltac

- Besides the predefined tactics offered by Coq or these imported by various Coq packages, Coq offers a way for the user to define his own proof-tactics
  - This is achieved by Ltac
    - ⋆ A programming language inside Coq that can be used to build new user-defined tactics
    - ⋆ Using Ltac we can define the following tactic that will fully automate the example we are interested in

      ```
      Ltac EXTAC:= cbv; eauto.
      ```

    - ⋆ The *cbv* tactic performs all possible reductions using $\delta$, $\beta$, $\zeta$ and $\iota$
    - ⋆ In our case, $\delta$ reduction is applied first unfolding the definition and then $\beta$ reduction
    - ⋆ The tactic *compute* that embodies *cbv* can also be used

# The tactic language Ltac

- What we need is automated tactics that work for a range of examples and not tactics that work on a case by case basis
  - For example, the tactic *EXTAC*, though simple enough, has the power to automate quite a few inferences
    - ★ One can further prove:

```
all Man (walk)->walk John.
all Man (walk)->walk John->some Man (walk).
```

    - ★ Also cases where subtyping is involved, like the following:

```
all Animal (walk)->walk John.
```

# The tactic language Ltac

- However, the following cannot be proven with *EXTAC*:

```
all Man (walk)-> some Man (walk).
```

  - We get the following error:

```
Coq < Theorem EX2: all Man (walk) -> some Man (walk).
1 subgoal
============================
all Man (fun x : Man => walk x) -> some Man (fun x : Man =>
 walk x)
EX2< EXTAC.
No more subgoals but non-instantiated existential variables
Existential 1 = ?463 : [H : forall x : Man, walk x |- Man]
```

  - This means that *eauto* did not manage to instantiate an existential, which was then eliminated by a computation
  - The solution is to instantiate the value "manually"

# The tactic language Ltac

- For example, we can substitute $x$ for *John* using the *exists* tactic
  - We can define a similar tactic that instantiates the variable using *exists* and then calls *EXTAC*

    ```
    Ltac EXTAC1 x:= cbv; try exists x;EXTAC.
    ```
  - The command *try* + *tactic*, tries to perform the tactic, and if it fails, it moves on
  - This will suffice to prove automatically all the NL examples we have considered so far

# Recap

- The basics of working with Coq
  - Working with parameters, definitions
  - Coq's proof-mode and proof tactics
  - Inductive types
  - A suitable vehicle to represent MTT semantics
- Next lecture: Using Coq as a proper NL reasoner

# Formal Semantics in Modern Type Theories: Theory and Implementation: Lecture 4 - Natural Language Infererce using Coq

Stergios Chatzikyriakidis and Zhaohui Luo

August 2014

# NL Inference

- The task of determining whether an NL hypothesis H can be deduced from an NL premise P
- A central task in both theoretical and computational semantics
  - As Cooper et al. (1996) aptly put it: "inferential ability is not only a central manifestation of semantic competence but is in fact centrally constitutive of it"
    - ★ Inferential ability as the best way to test the semantic adequacy of NLP systems
    - ★ An adequate NLP system should be able to predict correct inferences like (1)-(3) without further generating unwanted inferences like (4) or (5)

(1) John walks and Mary talks $\Rightarrow$ Mary talks

(2) Some men run fast $\Rightarrow$ Some men run

(3) John walks $\Rightarrow$ Some one walks

(4) John walks and Mary talks $\not\Rightarrow$ If John walks, Mary talks

(5) No men run fast $\not\Rightarrow$ No men run

# NL Inference

- Three types of approaches to NL inference
  - Shallow approaches: no deep semantic analysis. Inferences are derived using a number of methods to establish some measure of syntactic or semantic similarity, which will then decide whether a given hypothesis follows (e.g. Glickman et al., 2005; Romero et al., 2006)
  - Deep approaches: Translation into a formal language which is then used to decide on the inferences (e.g. Blackburn & Bos, 2005; Bos & Market, 2005)
  - A combination of the two: approaches like MacCartney (2008)
  - The approach proposed here is a deep approach

# NL inference platforms: FraCas

- Platforms for NLI - The Fracas test suite
  - Came out of the FraCas consortium, a large collaboration in the 90's to create resources for computational semantics
  - Contains 349 NLIs, with one or more premises
    - ★ Categorized by semantic section: e.g. Quantifiers, adjectives temporal reference etc.
    - ★ A number of premises (usually single premised), followed by the hypothesis in the form of a question

# The FraCas test suite

- Typical examples

(6) No delegate finished the report.
Did any delegate finished the report on time? [No] (quantifier section)

(7) Either Smith, Jones or Anderson signed the contract. Did John sign the contract? [UNK] (plurals)

(8) Dumbo is a large animal. Is Dumbo a small animal? [NO] (adjectives)

(9) Smith believed that ITEL had won the contract in 1992. Did ITEL win the contract in 1992? [UNK] (Attitudes)

# The RTE challenges

- Recognizing textual entailment
  - Examples drawn from natural text.
    - ★ Organized every year from 2005 to 2013 with different datasets
    - ★ Premises are considerably longer than what is found in the FraCas test suite and can involve more than one sentence (25 word average for RTE1 and 39 in RTE4)
    - ★ Hypotheses are constructed, and are comprised of one sentence
    - ★ Binary classification in the first datasets (YES NO) tripartite in the next years (YES, NO and UNK)

# The RTE challenges

- Definition of entailment is not strict but as Maccartney (2008) says approximate
  - Whether a competent speaker with basic world knowledge would infer *h* from *p*

(10) As leaders gather in Argentina ahead of this weekends regional talks, Hugo Chavez, Venezuela's populist president is using an energy windfall to win friends and promote his vision of 21st-century socialism.

Hugo Chavez acts as Venezuela's president [YES]

(11) Democrat members of the Ways and Means Committee, where tax bills are written and advanced, do not have strong small business voting records.

Democrat members had strong small business voting records. [NO]

# Choice of platform

- We are going to deal with the FraCas test suite here
  - As already said, the FraCas test suite, even though it contains much simpler examples in terms of size and the amount of world knowledge needed for the entailement to succeed, it is however carefully designed to deal with a lot of semantic phenomena
    - ★ The well-known problem of deep approaches surfaces in more complex examples: Deep approaches are costly
    - ★ How much world knowledge can we encode?
    - ★ For this course, we are going to exemplify our approach by using a subset of the FraCas test suite

# Formulating the examples

- A note about parsing: In order to deal with the examples of the FraCas test suite, ideally we first need a parser for the sentences
  - Ranta's GF parser can be used to this end
    - ★ A GF extended parser for the FraCas already exists (Ljunglof et al, 2011)
    - ★ Ideally, then an automatic translation between well-formed parsed sentences and the syntax of Coq can be extablished
    - ★ At the moment we have not done this
    - ★ We will for the moment with the reasoner and do not worry about parsing
    - ★ Thus, the examples of the FraCas test suite will be translated to the language of the prover without worrying about parsing

# Formulating the examples

- As already said, the examples involve a number of premises, followed by a question ($h$).
  - We reformulate the examples as involving declarative forms in Coq (this is a usual approach, at least with deep approaches)
    - ★ In cases of *yes* in the FraCas test suite, we formulate a declarative hypothesis as following from the premise
    - ★ In cases of *no*, we formulate the negation of a declarative hypothesis as following from the premise
    - ★ In cases of *UNK*, for both the positive and the negated $h$, no proof should be found. If it is, we overgenerate inferences we do no want

# Formulating the examples

- A *YES* example

(12) A Swede won the Nobel Prize.
Every Swede is Scandinavian.
Did a Scandinavian win the Nobel prize? [Yes, FraCas ex. 3.49]

```
Theorem SWE:(a Swede)(Won(a Nobel_Prize))->(a
Scandinavian)(Won(a Nobel_Prize)).
```

# Formulating the examples

- A *NO* example

(13) A Swede did not win the Nobel Prize.
Every Swede is Scandinavian.
Did a Scandinavian win the Nobel prize? [No]

```
Theorem SWE:not((a Swede)(Won(a Nobel_Prize)))->not
(a Scandinavian)(Won(a Nobel_Prize)).
```

# Formulating the examples

- An *UNK* example

(14) A Scandinavian won the Nobel Prize.
Every Swede is Scandinavian.
Did a Swede win the Nobel prize? [UNK, 3.65]

```
Theorem SWE:(a Scandinavian)(Won(a
Nobel_Prize))->(a Scandinavian)(Won(a Nobel_Prize)).
```

```
Theorem SWE:(a Scandinavian)(Won(a Nobel_Prize))->not((a
Scandinavian)(Won(a Nobel_Prize))).
```

# Evaluating against the FraCas test suite - Quantifier monotonicity

- This section involves inferences due to quantifier monotonicity
  - Upwards monotonicity on the first argument

    (15) Some Irish delegates finished the survey on time
    Did any delegates finish the survey on time? [YES]

    - ★ Standard semantics for indefinites *some* and *any* (no presuppositions encoded)

      ```
      Definition some:= fun A:CN, fun P:A->Prop=> exists x:A, P(x).
      ```

# Modification

- The examples we are dealing involve instance of adjectival modification
  - *Irishdelegate* in this case says that something is a *delegate* and furthermore *Irish*
  - We follow the $\Sigma$ type treatment of adjectives. The first projection, $\pi 1$ is a coercion
  - We formulate it in Coq via means of dependent records

```
Record Irishdelegate:CN:=mkIrishdelegate{c:> Man;_:Irish c}
```

    - ⋆ With *Delegate* : *CN* and *Irish* : *Object* $\rightarrow$ *Prop*

# Modification

- With these assumptions, nothing more is needed
  - The inference can be proven given the coercion of $\pi 1$
  - We formulate the theorem:

```
Theorem IRISH: (some Irishdelegate(On_time(finish(the
survey)))->(some Delegate)(On_time (finish(the survey))).

compute.intro. elim H.intro.intro.exists x.auto.
```

  - Easy to prove. Subtyping does the work. Eliminating $H$ and using intro we get an $x : \mathit{Irishdelegate}$ that $\mathit{On\_time}(\mathit{finish}(\mathit{thesurvey}))(x))$ holds. Then, given subtyping, $\mathit{Irishdelegate} < \mathit{Delegate}$ via the first projection $\pi 1$, we also have that $\mathit{On\_time}(\mathit{finish}(\mathit{thesurvey}))(x))$ with $x : \mathit{Delegate}$

# Subtyping again

- Other similar examples involve more direct cases of subtyping

(16) A Swede won a Nobel prize
Every Swede is a Scandinavian
Did a Scandinavian win a Nobel Prize? [YES, 3.49]

- The above is multi-premised, i.e. more than one premise
  - We first define *Swede* and *Scandinavian* as being of type *CN*
    - ⋆ This is a case of nominalized adjectives. At least in this guise they function as *CNs*. One can give a Unit type capturing both guises (more on Unit types later)
  - Note that both arguments of the verb are quantifiers
  - In order to accommodate this, we have two options
    - ⋆ The first option is to define *won* as a regular transitive (leaving tense aside for the moment since it does not play a role in proving the inference). Then, in order to perform functional application, given the higher types for quantifiers, one must directly translate to something like the following: $\exists x : Man, \exists y : Object, win(x)(y)$ (scope issues are not going to be discussed here)

# Subtyping again

- Alternatively, one can follow the strategy employed by Montague and type shift the verb, thus lifting to type $((Object \rightarrow Prop) \rightarrow Prop) \rightarrow (Human \rightarrow Prop)$
  - We exemplify with both alternatives
  - The most important part in proving the inference is the declaration of a subtyping relation between *Swede* and *Scandinavian*, i.e. $Swede < Scandinavian$

```
Parameter Swede Scandinavian:CN
Won: Object->Human->Prop.
Won: ((Object->Prop)->Prop)->(Human->Prop)
Axiom ss: Swede->Scandinavian.
Coercion ss: Swede >-> Scandinavian.
Theorem SWEDE1: (a Swede)(won (a Nobel_Prize))->(a
Scandinavian (won(a Nobel_Prize)).
Theorem SWEDE2: exists x:Swede, exists y:Nobel_Prize, won(x)(y
```

# The Swede example

- Formulation with the verb type-lifted

```
SWEDE22<Theorem SWEDE22: (a Swede)(Won2(a Nobel_Prize))->
(a Scandinavian)(won(a Nobel_Prize)).
```

- We first use *cbv* to unfold the definitions. Then *intros*:

```
SWEDE22 < intros.
1 subgoal
H : exists x : Swede,
Won2 (fun P : Object -> Prop => exists x0: Nobel_Prize,
P x0) x
============================
exists x : Scandinavian,Won2 (fun P : Object -> Prop =>
exists x0 : Nobel_Prize, P x0) x
```

- Elimination (*elim*) can now be used followed by *eauto*. This suffices to prove the goal.

# The Swede example

- Formulation with the verb regularly typed

```
Theorem SWEDE2: exists x:Swede, exists y:Nobel_Prize,
Won(y)(x)->exists x:Scandinavian, exists y:Nobel_Prize,
won(y)(x).
```

- There are no definitions to unfold and *intro* cannot apply.
- The natural solution is to be use *eauto*. However, this will give us the following error:

```
SWED < eauto.
No more subgoals but non-instantiated existential
variables:
Existential 1 = ?535 : [ |- Swede]
Existential 2 = ?536 : [ |- Nobel_Prize]
```

- This basically says that non-instantiated variables generated by *eapply* have been lost prior to instantiation

# The Swede example

- The solution is to instantiate these variables
  - In this sense, we can introduce a number of variables (parameters) of type *Human* and a number of variables (parameters) of type *Object*
  - We can use one of these variables to instantiate the existentials
  - Starting with the proof, we basically instantiate both existentials
    - ★ We then apply *eauto*, and the proof is completed (with $d$ : *Swede* and $n$ : *Scandinavian*.

```
SWED < exists d.
1 subgoal
============================
exists y : Nobel_Prize,
Won1 y d -> exists x : Scandinavian, exists y0 : Nobel_Prize,
Won1 y0 x
SWED < exists n.
1 subgoal
============================
 Won1 n d -> exists x : Scandinavian, exists y : Nobel_Prize,
Won1 y x
SWED < eauto.
Proof completed.
```

# A NO example

- Monotonicity on the first argument

  (17) No delegate finished the report on time
  Did any Scandinavian delegate finish the report on time?
  [NO, FraCas 3.70]

- We try to prove the negation of the hypothesis

```
Theorem SCAN: (no delegate)(On_time Human(finish(the
report)))->not((some Scandinaviandelegate)(On_time Human
(finish(the report)))).
```

- We apply *cbv* to unfold the definitions followed by *intros*
- Then, the tactic jauto can be used to complete the proof
  - *jauto* is similar to *eauto* but can further open up conjunctions and existentials (what we need here)

# An UNK example

- Again from the monotonicity on the first argument part of the suite

  (18) Some delegates finished the survey on time
  Did any Irish delegates finish the survey on time? [UNK, FraCas 3.71]

- Indeed the above cannot be proven given that the subtyping relation is from *Irishdelegate* $<$ *delegate* and not the other way around
  - Basically, we end up with something like the following, and the proof cannot further continue

```
IRISH < AUTO.
H0 : exists x : delegate, On_time (finish (the survey)) x
x : delegate
H1 : On_time (finish (the survey)) x
============================
exists x0 : Irishdelegate,
On_time (finish (the survey)) (let (c0, _) := x0 in c0)
```

  - Trying to substitute $x$ for $x_0$ fails since the terms are of different types!

# Monotonicity on the second argument

- In this section we find examples like the following:

  (19) Some delegates finished the survey on time
  Did some delegates finish the survey? [UNK, FraCas 3.71]

- The inference in these cases comes from the veridicality of VP-adverbials like *ontime*
  - In order to capture this, we will have to see how VP veridical adverbials can be defined.
    - ★ In order to do this we first introduce the auxiliary object $ADV_{ver}$, for veridical VP-adverbials

      (20) $ADV_{ver} : \Pi A : \mathrm{CN}.\Pi v : A \to Prop.\ \Sigma p : A \to Prop.\forall x : A.p(x) \supset v(x)$

# Monotonicity on the second argument

- Continued

(21) $ADV_{ver} : \Pi A : \mathrm{CN}.\Pi v : A \to Prop.\ \Sigma p : A \to Prop.\forall x : A.p(x) \supset v(x)$

- Note that this is minimally different from $\forall A : CN, (A \to Prop) \to (A \to Prop)$, the only addition is the second part of the $\Sigma$ specifying that in case $p(x)$ holds (the clause with the adverbial), then $V(x)$ also holds (the same sentence without the adverbial)
- Now, we define on time to be the first projection of this auxiliary object

(22) $on\ time = \lambda A : \mathrm{CN}.\lambda v : A \to Prop.\ \pi_1(ADV(A, v))$

# Monotonicity on the second argument

- We formulate these assumptions in Coq

```
Parameter ADV: forall (A : CN) (v : A -> Prop),sigT
(fun p : A -> Prop => forall x : A, p x -> v x).
Definition on_time:= fun A:CN, fun v:A->Prop=> projT1
(ADV(v)).
```

- Let us see whether this definition suffices to prove the inference in (19).

```
IRISH2 < Theorem IRISH2: (some delegate)(on_time
(finish(the survey)))->(some delegate)((finish
(the survey))).
```

# Monotonicity on the second argument

- Continued
- We unfold the definitions and use destruct for *ADV* (basically it unfolds the definition for *ADV*)

```
IRISH2 < cbv. intro. destruct ADV in H.
1 subgoal
x : Human -> Prop
f0 : forall x0 : Human, x x0 -> finish (the survey) x0
H : exists x0 : delegate, x x0
============================
exists x0 : delegate, finish (the survey) x0
```

- We apply *induction* or *elim* to *H*
  - The difference between the two is that *induction* will add the inductive hypotheses into the context while *elim* will not
  - Applying *eauto* after this, will complete the proof

# A note on veridical adverbs/adverbials

- The way proposed to capture veridicality can be generalized to all VP adverbs/adverbials.
  - For example if one is interested in getting the veridicality inferences right, ignoring other issues pertaining to the lexical semantics of each adverbial, then the auxiliary object can be used in all these cases
  - Thus, adverbs like slowly, fast etc. can given a similar definition to *on_time*

$$(23) \quad adv_{ver} = \lambda A\colon \mathrm{CN}.\lambda v\colon A \to Prop.\ \pi_1(ADV_{ver}(A, v))$$

  - A similar strategy can be used for veridical sentence adverbs. We first define an auxiliary object:

$$(24) \quad ADV_{Sver}\colon \Pi v\colon Prop.\ \Sigma p\colon Prop.p \supset V$$

  - Then veridical sentence adverbs/adverbials like *fortunately, ironically* can be defined as:

$$(25) \quad adv_{Sver} = \lambda v\colon Prop.\ \pi_1(ADV_{Sver}(v))$$

# A note on veridical adverbs/adverbials

- We can check this in Coq

```
Coq < Theorem FORT: fortunately (walk John)-> walk John
1 subgoal
============================
fortunately (walk John) -> walk John
```

- We unfold the definitions and apply destruct to *ADVS*.

```
FORT < cbv. destruct ADVS.
x : Prop
w : x -> walk John
============================
x -> walk John
```

- Using *assumption* will complete the proof

# Cases with more premises

- Example cases involving more than one premise

  (26) Each European has the right to live in Europe
  Every European is a person
  Every person who has the right to live in Europe can travel freely within Europe
  Can each European travel freely within Europe? [Yes, FraCas 3.20]

- For reasons of brevity some elements will be treated non-compositionally
  - But: only those that do not play any role in inference
  - Thus, to leave in Europe will be assumed as a single lexical item, since its treatment does not play any role in the specific inference
    - ⋆ Interesting case: Does *each european hat the right to live in Europe* imply *that each European has the right to live*?

# Cases with more premises

- We assume *to_ live* to be a regular predicate. Then, we further assume that in_ Europe, freely and within_ Europe to be predicate modifiers
  - It is not difficult to give entries for prepositions *in* and *within* separately, but we will keep it simple in this case

```
Parameter in_Europe: forall A:CN, (A->Prop)->(A->Prop).
Parameter can: forall A:CN, (A->Prop)->(A->Prop).
Parameter travel: Object->Prop.
Parameter freely: forall A:CN, (A->Prop)->(A->Prop).
Parameter within_Europe: forall A:CN, (A->Prop)->(A->Prop).
```

# Cases with more premises

- Let us formulate the example:
  - The first premise is straightforward
  - The second premise is encoded as a coercion and thus does not have to be present in the proof explicitly
  - The third premise is an implication relation (if a person... then)
  - Careful with the parentheses: the above two premises must imply the conclusion

    ```
    Theorem EUROPEAN: ((each European)(have
    (the righttoliveinEurope))/\forall x:person, ((have
    (the righttoliveinEurope)x)->Can (within_Europe(freely
    (travel)))x))->(each European)(Can (within_Europe(freely
    (travel)))).
    ```
  - Once formulated correctly, it is to prove
  - Using *cbv* to unfold the definitions, we can use *intuition* and complete the proof

# One further example - at least two

- We define *at least two* as follows:

```
Definition at_least_two:= fun A:CN, fun P:A->Prop=>exists
exists y: A, P(x)/\(P(y))/\ not(x=y).
```

- With this one can deal with inferences like the following:

(27) At least two female commissioners spent time at home
At least two commissioners spent time at home [Yes, FraCas 3.63]

# Some preliminary results on the quantifier section

- Evaluation against 35 examples, 7 from each subsection
  - Usually the first 7 examples of each subsection
    - ★ Some examples are skipped to the next one, in case of consecutive similar examples (see Chatzikyriakidis and Luo 2014)
    - ★ 35/35 examples were correctly predicted! The plan is to evaluate against the whole section (in progress)

# Adjectives section

- The adjectival class is notoriously non-homogeneous and a rather problematic class
  - Behaviour in terms of inference depends on the specific adjective
    - ★ The FraCas test suite uses a somehow different terminology than that usually found in the literature
    - ★ Affirmative/non-affirmative distinction: This is basically the subsective, non-subsective distinction in mainstream terminology.

    a. Affirmative: Adj(N)(x) $\Rightarrow$ N(x)

    b. Non-affirmative: Adj(N)(x)$\Rightarrow \neg$ N(x) or undefined

# Affirmative adjectives

- The $\Sigma$ type account for adjectives suffices

  (28) John has a genuine diamond
  Does John have a diamond? [Yes, FraCas 3.197]

- Let us formulate the theorem

```
Theorem GENUINE: (a genuine_diamond)(has John)->
(a diamond)(has John).
```

- We unfold the definitions, use *intros*, *elim H* and *eauto*. The proof is completed

```
GENUINE < cbv. intros. elim H. eauto.
Proof completed.
```

- In a more economical way, *cbv* and *jauto* will also suffice to complete the proof

# Opposites

- In this category, we find opposite adjectives like *small/large* in the FraCas test suite
  - What we want to get are the following inferences

(29) $\text{Small(N)} \Rightarrow \neg\ \text{Large(N)}.$
$\text{Large(N)} \Rightarrow \neg\ \text{Small(N)}$
$\neg\ \text{Small(N)} \not\Rightarrow \text{Large(N)}.$
$\neg\ \text{Large(N)} \not\Rightarrow \text{Small(N)}$

  - These are a little bit tricky to get
    - ★ The problem is that there are other sizes than a binary opposition *small-large*, e.g. normalsized items
    - ★ We can use this intuition to find a way out of the problem
    - ★ First define the element that its negation is implied by the other, i.e. large in our case
    - ★ We just give a regular predicate type for large
    - ★ Now, small is going to be defined as not being large AND not being normalsized (in fact additional sizes can be introduced, depends on the sizing granularity one assumes)

# Opposites

- The definition for *small*

```
Definition small:= fun A:CN, fun a:A=> not (large (a)
/\ not (normalsized (a)).
```

- Checking against the examples

(30) Mickey is a small animal
Is Mickey a large animal? [No, FraCas 3.204)

- This is easily proven. We want to prove its negation

```
Theorem MICKEY: (Small Animal Mickey) ->not( Large Animal
Mickey).
```

- Unfolding the definitions, *intros*, *elim and eauto* or just *cbv* and *jauto* will complete the proof
  - Note how powerful *jauto* is. We are pretty much able to complete the proof in two steps (almost automation (we will exploit *jauto* when developing automated tactics))

# Opposites

- The next example from FraCas shows an inference that we should not get, i.e. $\neg$ large $\Rightarrow$ small

  (31) Fido is not a large animal
  Is Fido a small animal? [UNK, FraCas 3.207)

- We formulate the theorem

```
Theorem FIDO: not(Large Animal Fido) ->Small Animal Fido.
```

- We cannot complete the proof
- The same goes for the same theorem with the implicatum negated

# Comparison classes

- Adjectives that assume a comparison class like for example *small/big* (small for an *N*, big for an *N*) and adjectives that do not like *four-legged*
  - Let us see cases that do not assume a comparison class like *four-legged*
    - ★ We assume a simple predicate type $Animal \rightarrow Prop$
    - ★ Let us see a FraCas example

      (32) Dumbo is a four-legged animal
      Is Dumbo four-legged? [Yes, FraCas 3.203)

    - ★ We formulate the theorem (avoiding a discussion on how the copula should be treated here if at all)

      ```
      Theorem dfdss:exists x:Animal, four_legged(x)/\Dumbo=x->
      four_legged(Dumbo).
      ```

    - ★ We substitute *Dumbo* for *x* and use *jauto*. This suffices to complete the proof

# Comparison classes

- Adjectives like big/small assume a comparison class
  - The idea is that something like *big elephant*, means big for an elephant but not big in general
    - ⋆ This is basically the subsective class of adjectives where the adjective noun combination implies the noun only (e.g. skilful surgeon(x) ⇒ surgeon(x))
    - ⋆ Chatzikyriakidis and Luo (2013) deal with these types of adjectives by introducing a polymorphic type extending over the CNuniverse

      (33) $\Pi A : CN.A \rightarrow Prop$

    - ⋆ The idea is that typing is dependent on the choice of $A$. If $A$ is of type *Animal* then the type will be $Animal \rightarrow Prop$, if $A$ is of type *Human*, the typing would be $Human \rightarrow Prop$ and so on

# Comparison classes

- This polymorphic type along with the lexical semantics given for *small* will predict the correct inferences
  - Consider the following example

    (34) All mice are small animals
    Mickey is a large mouse
    Is Mickey a large animal? [No, FraCas 3.210)

  - We formulate the theorem

    ```
    Theorem MICKEY2: (all Mouse (Small Animal)/\ Large Mouse Mickey)->not( Large Animal Mickey)
    ```

  - We unfold the definitions and apply *intro*, followed by two applications of *induction* or *destruction* of *H*
  - In the second use, we have to introduce the value for *x* ourselves, *Mickey* in our case. Otherwise we can use *edestruct* or *einduction*

# Comparison classes

- Continued

```
H : forall x : Mouse,(Large Animal x -> False) /\
(Normalsized Animal x -> False)
H0 : Large Mouse Mickey
H1 : Large Animal Mickey -> False
H2 : Normalsized Animal Mickey -> False
============================
Large Animal Mickey -> False
```

- Applying *assumption* completes the proof
- The other examples in the section can be proven in a similar way

# Comparatives

- Two ways to dela with comparatives: One without measures, ione with measures
  - Both proposals were put forth in Chatzikyriakidis and Luo (2014)
  - The examples in the test suite do not need the explicit introduction of measures so we will concentrate on the approach without measures
    - ★ The same of idea of using an auxiliary object first is used. Thus, in the case of *SMALLER_THAN* one can define the following:

(35) $SHORTER\_THAN : \Sigma p : Human \rightarrow Human \rightarrow Prop.\ \forall h_1, h_2, h_3 : Human.\ p(h_1, h_2) \wedge p(h_2, h_3) \supset p(h_1, h_3) \wedge \forall h_1, h_2 : Human. p(h_1, h_2) \supset short(h_2) \supset short(h_1)$

(36) $shorter\ than = \pi_1(SHORTER\_THAN)$

    - ★ This basically captures the transitive properties of comparatives as well as the fact that an $x$ being $A\_er$ than something does not mean that this $x$ is also $A$ (being shorter than something does not guarantee shortness)
    - ★ It does however just in case the $y$ that $x$ is in a $A\_er$ relation with, is $A$

# Comparatives

- Let us see an example

(37) The PC-6082 is faster than the ITEL-XZ
The ITEL-xz is fast
Is the PC-6082 fast? [Yes, FraCas 3.220)

- We define *faster_than* in the sense described

```
Parameter FASTER_THAN : forall A : CN, {p : A -> A ->
Prop & forall h1 h2 h3 : A, (p h1 h2 /\ p h2 h3 ->
p h1 h3) /\ (forall h4 h5 : A, p h4 h5 -> Fast1 A h4 ->
Fast1 A h5)}.
Definition faster_than:= fun A:CN=>projT1 (FASTER_THAN A).
```

- With this, examples like (37) can be proven
- More on comparatives and inference in Chatzikyriakidis and Luo (2014)

# Epistemic, Intensional and Reportive Attitudes

- Section on the FraCas dealing with verbs that presuppose the truth of their propositional complement (e.g. *know*) and verbs that do not (e.g. *believe*)
  - For verbs like *believe* just a typing with no additional semantics will do

    (38) $\mathit{believe} : \mathit{Prop} \to \mathit{Human} \to \mathit{Prop}$

    - ⋆ For a treatment of belief intensionality in MTTs, see Ranta (1994), Chatzikyriakidis and Luo (2013), Chatzikyriakidis (2014)
  - For verbs that presuppose the truth of their complement, we can use a strategy similar to the one used for veridical adverbs
  - We define an auxiliary object first and then the lexical entry

    (39) $\mathit{KNOW} = \Sigma p : \mathit{Human} \to \mathit{Prop} \to \mathit{Prop}.\ \forall h : \mathit{Human} \forall P : \mathit{Prop}.\ p(h, P) \supset P$
    $\mathit{know} = \pi_1(\mathit{KNOW})$

# Epistemic, Intensional and Reportive Attitudes

- Examples like the following can be treated:

  (40) John knows that Itel won the contract
  Did Itel win the contract? [Yes, FraCas 3.334]

  (41) Smith believed that Itel had won the contract Did Itel win the contract? [UNK, FraCas 3.335]

```
Theorem KNOW:know John((Won1 (the Contract) ITEL))->
(Won1 (the Contract) ITEL) .
```

- We unfold the definitions, *destruct* the auxiliary object and then use *eapply*

# Plurals

- The section on plurals in the FraCas contains various subsections
- Conjoined plurals. Examples like the following

(42) Smith, Jones and Anderson signed the contract
Did Jones sign the contract? [Yes, FraCas 3.81]

- We can define conjunction using the same technique of using an auxiliary item
  - The following proposal was put forth in Chatzikyriakidis and Luo (2014) for the three place conjunction (see the paper on how to propose a generalized n-ary conjunction)

(43) $AND_3 : \Pi A : LType.\ \Pi x, y, z : A.\ \Sigma a : A.\ \forall p : A \rightarrow Prop.\ p(a) \supset p(x) \wedge p(y) \wedge p(z).$
$and_3 = \lambda A : LType.\lambda x, y, z : A.\ \pi_1(AND_3(A, x, y, z))$

# Plurals

- We formulate these assumptions in Coq
  - We use *Type* instead of *Ltype* given that universe construction is not an option in Coq
  - We these assumptions we can deal with examples like (42)
  - We formulate the theorem

```
Theorem CONJ:(Signed(the Contract)(and3 Smith Jones
Anderson)-> (Signed(the Contract)Smith)).
```

  - We unfold the definitions and destruct *AND*3

```
x : Man
a : forall p:Man->Prop,p x->p Smith/\ p Jones /\ p Anderson
============================
Signed (the Contract)x->Signed(the Contract) Smith
```

  - We use *apply a* and then *eauto* to complete the proof
  - Similar entries can be assumed for disjunction

# Plurals

- Dependent plurals

  (44) All APCOM managers have company cars
  John is an APCOM manager
  Does John have a company car? [Yes, FraCas 3.2.4)

- Again, we introduce some form non-compositionality for APCOM managers and company cars, since compositionaliity of these expressions does not play any role in the proof
  - The semantics given for *all* guarantee the completion of the proof

# Temporal reference

- We introduce a simple model of tense
  - We introduce first the parameter $Time : Type$
    - ⋆ We have a precedence relation $\leq$ and a specific object $now : Time$, standing for 'the current time' or the 'default time'
    - ⋆ We can define $Time$ as an inductive with one of its constructors being the following:

      (45) $DATE : date \rightarrow Time$

    - ⋆ Where date consists of the triples $(y, m, d)$ ranging over integers for years, months and days respectively

# Temporal reference

- Now, a present verb will say that the proposition expressed holds at the default time while a past tense verb at a time prior to the default time.
  - A number of inferences can be captured in this way. Let us see one:

    (46) ITEL has a factory in Birmingham
    Does ITEL currently have a factory in Birmingham? [Yes, FraCas 3.251]

  - We define currently to take an argument $P : \mathit{Time} \rightarrow \mathit{Prop}$ and specify that $P(\mathit{default}_t)$, $P$ holds in the default time
  - The present tense of the verb will also specify that $P$ holds at the default time.

```
Definition currently:=fun P : Time -> Prop=> P default_t
Definition Has:=fun (x : Object)(y : Human) (t : Time)=>
Have x y t /\ t = default_t.
```

# Temporal reference

- We formulate the theorem (we ignore the adverbial for the moment)

```
Theorem sCURRENTLY: (Has (a_factory))ITEL t-> currently ((
```

- We unfold the definitions and use *intros*, then we split the goal and destruct the hypothesis

```
H : Have a_factory ITEL t
H0 : t = DATE default_y default_m default_d
============================
Have a_factory ITEL (DATE default_y default_m default_d)
subgoal 2 is:
DATE default_y default_m default_d = DATE default_y
default_m default_d
```

- See Chatzikyriakidis and Luo (2014) for more examples
- We stop here as regards the phenomena to look at
  - See Chatzikyriakidis and Luo (2014) for more semantic phenomena e.g. bare plurals, elementary aspect and collective predication among others

# Automation

- We have seen that Coq is a powerful tool to reason about NL semantics
  - We have seen that using more composite tactics can shorten the proofs, e.g. using the *jauto* instead of the *eauto* tactic in cases of existentials.
    - ★ The question is whether we can fully automate our proofs
    - ★ It seems that we can, at least for the examples we are dealing
    - ★ We have seen that a number of examples can be proven using *jauto* or *intuition* after their definitions are unfolded. We have also seen in the end that *congruence* is also a very useful tactic to deal with equalities
    - ★ We can define a new composite tactic called *AUTO* that will basically formed out of the tactics just mentioned

      ```
      Ltac AUTO:= cbv delta;intuition;try repeat congruence;
      jauto;intuition.
      ```

# Automation

- Using the AUTO tactic
  - It turns out that *AUTO* is quite a powerful tactic
    - ★ It can actually automate many of the examples we were dealing with (and most importantly a lot more similar examples)

```
Theorem EX1:some Man (walk)->(some Human) walk
Theorem EX2: (walk) John-> some Man (walk).
Theorem IRISH: (some Irishdelegate)(On_time(finish(the
survey)))->(some delegate)(On_time (finish(the survey))).
Theorem SWEDE22: (a Swede) (Won2(a Nobel_Prize))->(a
Scandinavian)(Won2(a Nobel_Prize)).#
Theorem SCAN: (no delegate)(On_time Human(finish(the report))
->not((a Scandinaviandelegate)(On_time Human (finish(the
report)))).
Theorem EUROPE:((each European)(have(the righttoliveinEurope)
/\forall x:person, ((have(the righttoliveinEurope)x)->Can
(within_Europe(freely (travel)))x))->(each European)(Can
(within_Europe(freely(travel)))).
Theorem GENUINE: (a genuine_diamond)(has John)->(a diamond)
(has John).
Theorem MICKEY: (Small Animal Mickey) ->not( Large Animal
Mickey).
```

# Automation

- *AUTO* will fail in cases where destruct is needed, e.g. in the cases for factive complements, comparatives, conjunction etc.
  - We can remedy this by introducing a tactic which trie destruct before calling *AUTO* (the tactic is a little bit more complex but the details are not needed here)

    ```
    Ltac AUTOa  x i:= cbv;try destruct x;try intro;
    try ecase i;  AUTO;  try eapply i; try omega; AUTO;
    intuition; try repeat congruence; jauto;intuition.
    ```
  - Let us say we want to prove something which needs destruct

    ```
    Theorem KNOW:know John((Won1 (the Contract) ITEL))->(Won1 (
    ITEL) .
    ```
  - This can be automated with the new tactic now
  - Now, we can combine the two tactics into one generalized *GAUTO* tactic that tries to solve the goal via using one of the two automated tactics discussed

    ```
    Ltac GAUTO:= solve[AUTO|AUTOa].
    ```

# Automation

- *GAUTO* automates most of the proofs
  - There are some further cases like collective predication that need additional steps
    - ★ Extra AUTO tactics are defined in Chatzikyriakidis and Luo (2014) for these cases and are then added to *GAUTO*.
    - ★ All the examples discussed in the paper are given automated proofs
    - ★ How far can one go with automation?
    - ★ Is automation possible when NLIs are longer?

# Recap and issues for future research

- The use of Coq to deal with NLI semantics has provided us with fruitful results
  - Straightforward encoding of MTT semantics and reasoning about these
  - Valid NLIs as complete proofs
  - Using interaction to guide the assistant to the proof
  - Automation of the process, at least for the FraCas examples, seems feasible (at least to some good extent)
- Further issues
  - GF parser/translator translating between well-formed syntax structures and translating to the syntax of Coq
  - Work on phenomena that have not been attempted yet, i.e. aspect, modality
  - The issue of automation: What are its limits?
  - Entailment approximation

# V. MTT-semantics: More Advanced Issues

- MTT-semantics
  - Model-theoretic?
  - Proof-theoretic?
  - Both? If both, what are the implications?
    - Theoretical (& philosophical)
    - Practical
- Other advanced issues (not in this lecture)
  - Eg, Lexical semantics

*This lecture is mainly based on*

Z. Luo. Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014. LNCS 8535.

# Model-theoretic & Proof-theoretic Semamtics

- Model-theoretic (traditional):
  - Tarski, Montague, …
  - Logics/NL → set-theoretical models
    - Proof-theoretic reasoning is based on denotations/representations.
  - E.g., Montague: NL → simple type theory → set theory
- Proof-theoretic:
  - Gentzen, Prawitz, Dummett, …
  - Logics/NL → inferential roles
    - Meaning is use, in particular, uses in proof-theoretic reasoning.
  - E.g., logical operators given meaning via inference rules

- Example argument for traditional set-theoretic sem.
  - Or, an argument against non-set-theoretic semantics
- "Meanings are out in the world"
  - Portner's 2005 book on "What is Meaning" – typical view
  - Assumption that set theory represents (or even is) the world
  - Comments:
    - This is illusion! Set theory is just a theory in FOL, not "the world".
    - A good/reasonable formal system can be as good as set theory.

- Claim:

  *Formal semantics in Modern Type Theories*

  *is both model-theoretic and proof-theoretic.*

  - NL → MTT (representational, model-theoretic)
    - MTT as meaning-carrying language with its types representing collections (or "sets") and signatures representing situations
  - MTT → Meaning theory (inferential roles, proof-theoretic)
    - MTT-judgements, which are semantic representations, can be understood proof-theoretically by means of their inferential roles (c.f., Martin-Löf's meaning theory)

- Traditional model-theoretic semantics:

  Logics/NL → Set-theoretic representations
- Traditional proof-theoretic semantics of logics:

  Logics → Inferences
- Formal semantics in Modern Type Theories:

  NL → MTT-representations → Inferences

# This lecture

- Model-theoretic characteristics of MTT-sem
  - Signatures – extended notion of contexts to represent situations
- Proof-theoretic characteristics of MTT-sem
  - Meaning theory of MTTs – inferential role semantics of MTT-judgements

# V.1. MTT-sem: Model-theoretic Characteristics

- In MTT-semantics, MTT is a representational language.
- MTT-semantics is model-theoretic
  - Types represent collections (c.f., sets in set theory) – see earlier slides on using rich types in MTTs to give semantics.
  - Signatures represent situations (or incomplete possible worlds).

- Types and signatures/contexts are embodied in judgements:

$$\Gamma \vdash_{\Sigma} a : A$$

  where A is a type, $\Gamma$ is a context and $\Sigma$ is a signature.
- Contexts are of the form $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$
- Signatures, similar to contexts, are finite sequences of entries, but
  - their entries are introducing constants (not variables; i.e., cannot be abstracted – c.f, Edinburgh LF (Harper, Honsell & Plotkin 1993)), and
  - besides membership entries, allows more advanced ones such as manifest entries and subtyping entries (see later).

# Situations represented as signatures

- Beatles' rehearsal: simple example
  - Domain: $\Sigma_1 \equiv D : Type,$
    $John : D,\ Paul : D,\ George : D,\ Ringo : D,\ Brian : D,\ Bob : D$
  - Assignment: $\Sigma_2 \equiv B : D \to Prop,\ b_J : B(John),\ ...,\ b_B : \neg B(Brian),\ b'_B : \neg B(Bob),$
    $G : D \to Prop,\ g_J : G(John),\ ...,\ g_G : \neg G(Ringo),\ ...$
  - Signature representing the situation of Beatles' rehearsal:
    $\Sigma \equiv \Sigma_1,\ \Sigma_2,\ ...,\ \Sigma_n$
  - We have, for example,
    $\Gamma \vdash_\Sigma G(John)\ \ true \ \ and \ \ \Gamma \vdash_\Sigma \neg B(Bob)\ \ true.$

    "John played guitar" and "Bob was not a Beatle".

# Manifest entries

- More sophisticated situations
  - E.g., infinite domains
  - Traditional contexts with only membership entries are not enough
- In signatures, we can have a manifest entry:

$$x \sim a : A$$

  where $a : A$.
- Informally, it assumes x that behaves the same as a.

# Manifest entries: formal treatment

- Manifest entries are just abbreviations of special membership entries:
  - x ~ a : A abbreviates x : $1_A(a)$ where $1_A(a)$ is the unit type with only object $*_A(a)$.
  - with the following coercion:

$$\frac{\Gamma \vdash_\Sigma A : Type \quad \Gamma \vdash_\Sigma a : A}{\Gamma \vdash_\Sigma \mathbf{1}_A(a) \leq_{\xi_{A,a}} A : Type}$$

    where $\xi_{A,a}(z) = a$ for every $z : 1_A(a)$.
- So, in any hole that requires an object of type A, we can use x which, under the above coercion, will be coerced into a, as intended.

# Manifest entries: examples

$$\Sigma_1 \equiv D : Type,$$
$$John : D,\ Paul : D,\ George : D,\ Ringo : D,\ Brian : D,\ Bob : D$$
$$\Sigma_2 \equiv B : D \to Prop,\ b_J : B(John),\ ...,\ b_B : \neg B(Brian),\ b'_B : \neg B(Bob),$$
$$G : D \to Prop,\ g_J : G(John),\ ...,\ g_G : \neg G(Ringo),\ ...$$

➔➔➔

$$D \sim a_D : Type,\ B \sim a_B : D \to Prop,\ G \sim a_G : D \to Prop,$$

where

$$a_D = \{John,\ Paul,\ George,\ Ringo,\ Brian,\ Bob\}$$
$$a_B : D \to Prop, \text{ the predicate 'was a Beatle',}$$
$$a_G : D \to Prop, \text{ the predicate 'played guitar',}$$

with $a_D$ being a finite type and $a_B$ and $a_G$ inductively defined.
(Note: Formally, "Type" should be a type universe.)

- Infinity:
  - Infinite domain D represented by infinite type Inf

    $D \sim Inf : Type$
  - Infinite predicate with domain D:

    $f \sim f\text{-}defn : D \rightarrow Prop$

    with f-defn being inductively defined.
- "Animals in a snake exhibition":

  $Animal_1 \sim Snake : CN$

# Subtyping entries in signatures

- Subtyping entries in a signature:

  $c : A < B$

  where c is a functional operation from A to B.
- Eg, we may have

  $D \sim \{ \text{John}, \ldots \} : \text{Type}, c : D < \text{Human}$
- Note that, formally, for signatures,
  - we only need "coercion contexts" but do not need "local coercions" [Luo 2009, Luo & Part 2013];
  - this is meta-theoretically much simpler.

# Remarks

- Using contexts to represent situations: historical notes
  - Ranta 1994 (even earlier?)
  - Further references [Bodini 2000, Cooper 2009, Dapoigny/Barlatier 2010]
- We introduce "signatures" with new forms of entries: manifest/subtyping entries
  - Manifest/subtyping entries in signatures are simpler than manifest fields (Luo 2009) and local coercions (Luo & Part 2013).
- Preserving TT's meta-theoretic properties is important!
  - Ranta, Bodini, Dapoigny & Barlatier just use the traditional notion of contexts; so OK.
  - Our signatures with membership/manifest/subtyping entries are OK as well.
  - Other extensions/changes need be careful: e.g., one may ask: are we preserving logical consistency under propositions-as-types?

# V.2. MTT-sem: Proof-theoretic Characteristics

- Proof-theoretic semantics
  - Meaning is use (cf, Wittgenstein, Dummett, Brandom)
    - Conceptual role semantics; inferential semantics
    - Inference over reference/representation
  - Two aspects of use
    - Verification (how to assert a judgement correctly)
    - Consequential application (how to derive consequences from a correct judgement)

- Proof-theoretic semantics in logics
  - Two aspects of use via introduction/elimination rules, respectively.
  - Gentzen (1930s) and studied by Prawitz, Dummett, … (1970s)
  - Meaning theory for Martin-Löf's type theory (Martin-Löf 1984)
- Proof-theoretic semantics for NLs
  - Not much work so far
    - cf, Francez's work (eg, (Francez & Dyckhoff 2011))
  - Traditional divide of MTS & PTS might have a misleading effect.
  - MTT-semantics opens up new possibility – a meta/representational language (MTT) has a nice proof-theoretic semantics itself.

# Meaning Explanations in MTTs

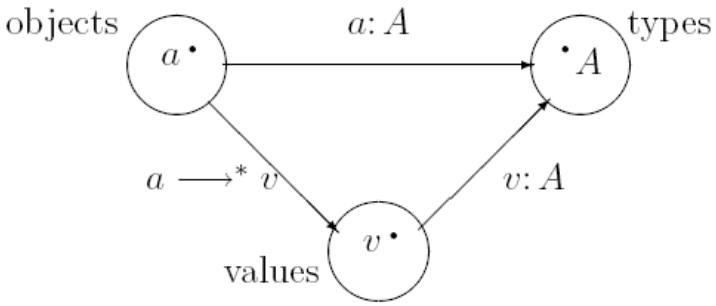
- Two aspects of use of judgements
  - How to prove a judgement?
  - What consequences can be proved from a judgement?
- Type constructors
  - They are specified by rules including, introduction rules & elimination rule.
  - Eg, for $\Sigma$-types

$$(\Sigma\text{-I}) \qquad \frac{\Gamma \vdash_\Sigma a : A \quad \Gamma \vdash_\Sigma b : B(a) \quad \ldots}{\Gamma \vdash_\Sigma p(a,b) : \Sigma(A,B)}$$

$$(\Sigma\text{-E}) \qquad \frac{\Gamma \vdash_\Sigma a : A \quad \Gamma \vdash_\Sigma b : B(a) \quad \Gamma \vdash_\Sigma C : (\Sigma(A,B))Type}{\Gamma \vdash_\Sigma \mathcal{E}_\Sigma(C,\ p(a,b)) : C(p(a,b))}$$

# Verificationist meaning theory

- Verification (introduction rule) as central
- In type theory, meaning explanation via canonicity (cf, Martin-Löf); recall the following picture:




cf, strong normalisation property.

# Pragmatist meaning theory

- Consequential application (elimination rule) as central
- This is possible for some logical systems
  - For example, operator &.
- For dependent types, impossible.
  - One can only formulate the elimination rules based on the introduction operators!

# Another view: both essential

- Both aspects (verification & consequential application) are essential to determine meanings.
  - Dummett
    - Harmony & stability (Dummett 1991), for simple systems.
  - For MTTs, discussions on this in (Luo 1994).
  - For a type constructor in MTTs, both introduction and elimination rules together determine its meaning.
- Argument for this view:
  - MTTs are much more complicated – a single aspect is insufficient.
  - Pragmatist view:
    - impossible for dependent types (see previous page)
  - Verificationist view:
    - Example of insufficiency – identity types

- Identity type $Id_A(a,b)$ (eg, in Martin-Löf's TT)
  - Its meaning cannot be completely determined by its introduction rule (Refl), for reflexivity, alone.
  - The derived elimination rule, so-called J-rule, is deficient in proving, eg, uniqueness of identity proofs, which can only be possible when we introduce the so-called K-rule [Streicher 1993].
  - So, the meaning of $Id_A$ is given by either one of the following:
    - (Refl) + (J)
    - (Refl) + (J) + (K)

    ie, elimination rule(s) as well as the introduction rule.

# Concluding Remarks

- Summary
  - NL ➔ MTT (model-theoretic)
  - MTT ➔ meaning theory (proof-theoretic)
- Future work
  - Proof-theoretic meaning theory
    - E.g. impredicativity (c.f., Dybjer's recent work in on "testing-based meaning theory")
    - Meaning explanations of hypothetical judgements
  - General model theory for MTTs? But ...
    - Generalised algebraic theories [Cartmell 1978, Belo 2007]
    - Logic-enriched Type Theories (LTTs; c.f., Aczel, Palmgren, …)

# Lexical Semantics – a further advanced issue

- Lexical semantics
  - Some features treated in formal semantics
    - Eg, copredication (work by Retore et al; our work on dot-types)
  - However, "real" lexical semantics is missing …
- Formal lexical semantics?
  - Formal and generative (cf, Pustejovsky (1995) nad other work on formal semantics)
  - Vector-space (statistical) (cf, recent work at Oxford etc.)
  - Mathematical texts (precise) (cf, existing work on math proofs)

(Realising the combined model based on the proof technology.)

# References (for Lecture V)

The cited references refer to either those in the following paper or those listed below.

Z. Luo. Formal Semantics in Modern Type Theories: Is It Model-theoretic, Proof-theoretic, or Both? Invited talk at Logical Aspects of Computational Linguistics 2014 (LACL 2014), Toulouse. LNCS 8535. 2014.


[Belo 2007] J. Belo. Dependently Sorted Logic. LNCS 4941.

[Bodini 2000] P. Bodini. Formalizing Contexts in Intuitionistic Type Theory. Fundamenta Informaticae 4(2).

[Cartmell 1978] Generalised algebraic theories and contextual categories, Ph.D. thesis, Oxford.

[Dapoigny/Barlatier 2010] Modelling Contexts with Dependent Types. Fundamenta Informaticae 104.

[Luo 2009] Type-theoretical semantics with coercive subtyping. SALT20.

[Portner 2005] What is Meaning? Blackwell.

[Pustejovsky 1995] The Generative Lexicon. MIT.

[Retoré et al 2010] Towards a Type-Theoretical Account of Lexical Semantics. JoLLI 19(2).

[Streicher 1993] T. Streicher. Investigations into Intensional Type Theory. Habilitation Thesis, 1993.

[Sundholm 1989] Constructive Generalized Quantifiers. Synthese 79(1).

[Xue & Luo 2012] Dot-types and their implementation. LACL 2012, LNCS 7351.