Constructive Hyperintensional Semantics

Jordan Needle, Carl Pollard, and Murat Yasavul

Department of Linguistics
Ohio State University
{needle.6,pollard.4,yasavul.1}@osu.edu

New Landscapes in Theoretical Computational Linguistics
Ohio State University
October 14, 2014

Introduction

This talk has several interrelated purposes:

Introduction

This talk has several interrelated purposes:

- to give an overview of the theory of **hyperintensional semantics** (HS), including some recent extensions,

Introduction

This talk has several interrelated purposes:

- to give an overview of the theory of **hyperintensional semantics** (HS), including some recent extensions,
- to point out some problems with HS that arise from limitations of the formalism in which it was written, namely **classical extensional HOL**,

Introduction

This talk has several interrelated purposes:

- to give an overview of the theory of **hyperintensional semantics** (HS), including some recent extensions,
- to point out some problems with HS that arise from limitations of the formalism in which it was written, namely **classical extensional HOL**,
- to introduce the **calculus of inductive constructions (CIC)**, a modern type theory closely related to HOL,

Introduction

This talk has several interrelated purposes:

- to give an overview of the theory of **hyperintensional semantics** (HS), including some recent extensions,
- to point out some problems with HS that arise from limitations of the formalism in which it was written, namely **classical extensional HOL**,
- to introduce the **calculus of inductive constructions (CIC)**, a modern type theory closely related to HOL,
- to explain how to fix the problems with the HOL-based implementation of HS (hereafter HHS) by shifting to a CIC-based implementation (hereafter CHS)

Outline of the talk

- Overview of static and dynamic HHS, noting some problematic aspects along the way

Outline of the talk

- Overview of static and dynamic HHS, noting some problematic aspects along the way
- Introduction to CIC as an elaboration of HOL

Outline of the talk

- Overview of static and dynamic HHS, noting some problematic aspects along the way
- Introduction to CIC as an elaboration of HOL
- Sketch of the transition from HHS to CHS

Static HHS

- HHS is a theory about **senses** (NL meanings), **worlds**, and **extensions** of senses at worlds

Static HHS

- HHS is a theory about **senses** (NL meanings), **worlds**, and **extensions** of senses at worlds
- HHS is intended as a synthesis/upgrade of Montague semantics (MS) and previous theories of dynamic semantics

Static HHS

- HHS is a theory about **senses** (NL meanings), **worlds**, and **extensions** of senses at worlds
- HHS is intended as a synthesis/upgrade of Montague semantics (MS) and previous theories of dynamic semantics
- HHS comes in both **static** (Plummer and Pollard, 2012; Pollard, 2008, 2015) and **dynamic** versions (Martin, 2013; Martin and Pollard, 2014; Martin, 2016; Yasavul, in progress)

Static HHS

- HHS is a theory about **senses** (NL meanings), **worlds**, and **extensions** of senses at worlds
- HHS is intended as a synthesis/upgrade of Montague semantics (MS) and previous theories of dynamic semantics
- HHS comes in both **static** (Plummer and Pollard, 2012; Pollard, 2008, 2015) and **dynamic** versions (Martin, 2013; Martin and Pollard, 2014; Martin, 2016; Yasavul, in progress)
- HHS is **compositional**: there is a straightforward interface to (linear-logical) categorial grammar (not discussed here)

Static HHS

- HHS is a theory about **senses** (NL meanings), **worlds**, and **extensions** of senses at worlds
- HHS is intended as a synthesis/upgrade of Montague semantics (MS) and previous theories of dynamic semantics
- HHS comes in both **static** (Plummer and Pollard, 2012; Pollard, 2008, 2015) and **dynamic** versions (Martin, 2013; Martin and Pollard, 2014; Martin, 2016; Yasavul, in progress)
- HHS is **compositional**: there is a straightforward interface to (linear-logical) categorial grammar (not discussed here)
- HHS is **formal**: it is expressed by **axioms**—aka **meaning postulates**—in **(classical extensional) HOL**

HOL: The Logic Underlying MS and HHS

- HOL is just Henkin's (1950) **simple theory of types**, a combination of typed lambda calculus and predicate logic with quantification over variables of all types

HOL: The Logic Underlying MS and HHS

- HOL is just Henkin's (1950) **simple theory of types**, a combination of typed lambda calculus and predicate logic with quantification over variables of all types
- HOL is familiar to linguistic semanticists under the name Ty2, which has one additional basic type w (worlds) in addition to the standard e (entities) and t (truth values)

HOL: The Logic Underlying MS and HHS

- HOL is just Henkin's (1950) **simple theory of types**, a combination of typed lambda calculus and predicate logic with quantification over variables of all types
- HOL is familiar to linguistic semanticists under the name Ty2, which has one additional basic type w (worlds) in addition to the standard e (entities) and t (truth values)
- Gallin (1975) showed how to translate Montague's Intensional Language (IL) into Ty2

HOL: The Logic Underlying MS and HHS

- HOL is just Henkin's (1950) **simple theory of types**, a combination of typed lambda calculus and predicate logic with quantification over variables of all types
- HOL is familiar to linguistic semanticists under the name Ty2, which has one additional basic type w (worlds) in addition to the standard e (entities) and t (truth values)
- Gallin (1975) showed how to translate Montague's Intensional Language (IL) into Ty2
- Points of notation: Montague's e and t correspond to Henkin's ι and o respectively; our w corresponds to Montague's and Gallin's s

HOL: The Logic Underlying MS and HHS

- HOL is just Henkin's (1950) **simple theory of types**, a combination of typed lambda calculus and predicate logic with quantification over variables of all types
- HOL is familiar to linguistic semanticists under the name Ty2, which has one additional basic type w (worlds) in addition to the standard e (entities) and t (truth values)
- Gallin (1975) showed how to translate Montague's Intensional Language (IL) into Ty2
- Points of notation: Montague's e and t correspond to Henkin's ι and o respectively; our w corresponds to Montague's and Gallin's s
- For HHS, we add another basic type p (propositions) for the senses of NL sentences (cf. Thomason 1980)

HHS Types

Pretty much as in MS/Ty2:

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**
   - e: the type of **entities**

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**
   - e: the type of **entities**
   - w: the type of **worlds**

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**
   - e: the type of **entities**
   - w: the type of **worlds**
   - p: the type of **propositions**

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**
   - e: the type of **entities**
   - w: the type of **worlds**
   - p: the type of **propositions**
2. If A and B are types, then $A \to B$ is a type

HHS Types

Pretty much as in MS/Ty2:

1. Basic types:
   - t: the type of **formulas/truth values**
   - e: the type of **entities**
   - w: the type of **worlds**
   - p: the type of **propositions**
2. If A and B are types, then $A \to B$ is a type
3. Nothing else is a type

Senses in HHS (1/2)

- As we saw, in HHS propositions get a type p of their own, instead of being defined as $w \to t$ as in MS/Ty2

Senses in HHS (1/2)

- As we saw, in HHS propositions get a type p of their own, instead of being defined as $\mathrm{w} \rightarrow \mathrm{t}$ as in MS/Ty2
- More generally, we'll see that in HHS, senses are **not** analyzed as intensions (functions from the set of worlds) the way they are in MS/Ty2

Senses in HHS (1/2)

- As we saw, in HHS propositions get a type p of their own, instead of being defined as $w \to t$ as in MS/Ty2
- More generally, we'll see that in HHS, senses are **not** analyzed as intensions (functions from the set of worlds) the way they are in MS/Ty2
- To put it another way: HHS makes **finer-grained** sense distinctions than MS/Ty2

Senses in HHS (1/2)

- As we saw, in HHS propositions get a type p of their own, instead of being defined as $w \rightarrow t$ as in MS/Ty2
- More generally, we'll see that in HHS, senses are **not** analyzed as intensions (functions from the set of worlds) the way they are in MS/Ty2
- To put it another way: HHS makes **finer-grained** sense distinctions than MS/Ty2
- This is what makes HHS **hyper-**intensional: two NL expressions can have the same intension but distinct senses

Senses in HHS (1/2)

- As we saw, in HHS propositions get a type p of their own, instead of being defined as $w \rightarrow t$ as in MS/Ty2
- More generally, we'll see that in HHS, senses are **not** analyzed as intensions (functions from the set of worlds) the way they are in MS/Ty2
- To put it another way: HHS makes **finer-grained** sense distinctions than MS/Ty2
- This is what makes HHS **hyper-**intensional: two NL expressions can have the same intension but distinct senses
- And that in turn is why HHS is immune from various foundational problems of MS (including but not limited to logical omniscience)

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types
  - If A and B are sense types, so is $A \to B$

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types
  - If A and B are sense types, so is $A \rightarrow B$
  - Nothing else is a sense type

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types
  - If A and B are sense types, so is $A \to B$
  - Nothing else is a sense type
- Notice that the family of sense types is defined in the metalanguage, not in the object language

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types
  - If A and B are sense types, so is $A \to B$
  - Nothing else is a sense type
- Notice that the family of sense types is defined in the metalanguage, not in the object language
- This is our first encounter with a problem that will surface over and over: HOL provides no way to recursively define, nor utilize any notion of, families of types internally

Senses in HHS (2/2)

- Not all types are types of senses of linguistic expressions
- Those that are are called **sense types**
- The set of sense types is recursively defined as follows:
  - e and p are sense types
  - If A and B are sense types, so is $A \rightarrow B$
  - Nothing else is a sense type
- Notice that the family of sense types is defined in the metalanguage, not in the object language
- This is our first encounter with a problem that will surface over and over: HOL provides no way to recursively define, nor utilize any notion of, families of types internally
- But CIC does

The Is-True-At Relation in HHS (1/2)

- We assume there is a relation of **being true at** that holds between propositions and worlds: $@ : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$

The Is-True-At Relation in HHS (1/2)

- We assume there is a relation of **being true at** that holds between propositions and worlds: $@ : \mathrm{p} \rightarrow \mathrm{w} \rightarrow \mathrm{t}$
- This is written infix, so $p@w$ can be read as either 'p is true at w' or 'the truth value of p at w' (remember that the interpretation of a formula is the same as its truth value)

The Is-True-At Relation in HHS (1/2)

- We assume there is a relation of **being true at** that holds between propositions and worlds: $@ : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$
- This is written infix, so $p@w$ can be read as either 'p is true at w' or 'the truth value of p at w' (remember that the interpretation of a formula is the same as its truth value)
- One possible extension of HHS is to identify p with $\mathrm{w} \to \mathrm{t}$ and identify @ with $\lambda p : \mathrm{w} \to \mathrm{t}.p$

The Is-True-At Relation in HHS (1/2)

- We assume there is a relation of **being true at** that holds between propositions and worlds: $@ : \mathrm{p} \rightarrow \mathrm{w} \rightarrow \mathrm{t}$
- This is written infix, so $p@w$ can be read as either 'p is true at w' or 'the truth value of p at w' (remember that the interpretation of a formula is the same as its truth value)
- One possible extension of HHS is to identify p with $\mathrm{w} \rightarrow \mathrm{t}$ and identify @ with $\lambda p : \mathrm{w} \rightarrow \mathrm{t}.p$
- The resulting theory is essentially the same as MS!

The Is-True-At Relation in HHS (1/2)

- We assume there is a relation of **being true at** that holds between propositions and worlds: $@ : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$
- This is written infix, so $p@w$ can be read as either 'p is true at w' or 'the truth value of p at w' (remember that the interpretation of a formula is the same as its truth value)
- One possible extension of HHS is to identify p with $\mathrm{w} \to \mathrm{t}$ and identify @ with $\lambda p : \mathrm{w} \to \mathrm{t}.p$
- The resulting theory is essentially the same as MS!
- In other words, HHS is a **weaker** theory than MS

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

  $\vdash \forall x : \mathrm{e}.\forall w : \mathrm{w}.x @_{\mathrm{e}} w = x$

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

  $\vdash \forall x : \mathrm{e}.\forall w : \mathrm{w}.x @_{\mathrm{e}} w = x$

  - This has as a consequence that the sense of a name is just its reference (similar to Kripke's view). Not everyone agrees this is right, but we'll use it for now.

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

  $\vdash \forall x : \mathrm{e}.\forall w : \mathrm{w}.x @_{\mathrm{e}} w = x$

  - This has as a consequence that the sense of a name is just its reference (similar to Kripke's view). Not everyone agrees this is right, but we'll use it for now.

  $\vdash \forall p : \mathrm{p}.\forall w : \mathrm{w}.p @_{\mathrm{p}} w = p @ w$

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

  $\vdash \forall x : \mathrm{e}.\forall w : \mathrm{w}.x@_{\mathrm{e}}w = x$

  - This has as a consequence that the sense of a name is just its reference (similar to Kripke's view). Not everyone agrees this is right, but we'll use it for now.

  $\vdash \forall p : \mathrm{p}.\forall w : \mathrm{w}.p@_{\mathrm{p}}w = p@w$

  $\vdash \forall f : A \to B.\forall w : \mathrm{w}.f@_{A \to B}w = \lambda x : A.(f\ x)@_B w$

The Is-True-At Relation in HHS (2/2)

- In general, we can define for each sense type A a term $@_A$ which maps terms of type A to their extensions:

  $\vdash \forall x\!:\!\mathrm{e}.\forall w\!:\!\mathrm{w}.x @_{\mathrm{e}} w = x$

  - This has as a consequence that the sense of a name is just its reference (similar to Kripke's view). Not everyone agrees this is right, but we'll use it for now.

  $\vdash \forall p\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.p @_{\mathrm{p}} w = p @ w$

  $\vdash \forall f\!:\!A \rightarrow B.\forall w\!:\!\mathrm{w}.f @_{A \rightarrow B} w = \lambda x\!:\!A.(f\ x) @_B w$

- This is yet another example of metalinguistic recursion. There is nothing in the formal theory that lets us think of these as the 'same' function

Dynamic HHS

- In dynamic HHS, contexts are modeled as functions from n-tuples of entities to a proposition

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{e}^n \rightarrow \mathrm{p}$$

Dynamic HHS

- In dynamic HHS, contexts are modeled as functions from n-tuples of entities to a proposition

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{e}^n \rightarrow \mathrm{p}$$

- The semantic objects in the n-tuple are indexed by what are commonly called **discourse referents (DRs)**

Dynamic HHS

- In dynamic HHS, contexts are modeled as functions from n-tuples of entities to a proposition

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{e}^n \rightarrow \mathrm{p}$$

- The semantic objects in the n-tuple are indexed by what are commonly called **discourse referents (DRs)**
- DRs are modeled as natural numbers

Dynamic HHS

- In dynamic HHS, contexts are modeled as functions from n-tuples of entities to a proposition

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{e}^n \rightarrow \mathrm{p}$$

- The semantic objects in the n-tuple are indexed by what are commonly called **discourse referents (DRs)**
- DRs are modeled as natural numbers
- The meanings of NL sentences are then taken to be functions which change the context

Dynamic HHS

- In dynamic HHS, contexts are modeled as functions from n-tuples of entities to a proposition

$$\mathrm{c}_n =_{\mathrm{def}} \mathrm{e}^n \rightarrow \mathrm{p}$$

- The semantic objects in the n-tuple are indexed by what are commonly called **discourse referents (DRs)**
- DRs are modeled as natural numbers
- The meanings of NL sentences are then taken to be functions which change the context

(1) $[\![$a donkey brayed$]\!] \approx$
$\lambda c : \mathrm{c}_n.\lambda \mathbf{x} : \mathrm{e}^{n+1}.(c\ x_1, \ldots, x_n)$ and (donkey x_0) and (bray x_0)

Problems with Dynamic HHS

- The n-tuples we use to hold our DRs correspond to the data structure known as **(homogeneous) vectors**

Problems with Dynamic HHS

- The n-tuples we use to hold our DRs correspond to the data structure known as **(homogeneous) vectors**
  - That is, a list/tuple whose components are all of the same type (e) of a certain length (n)

Problems with Dynamic HHS

- The n-tuples we use to hold our DRs correspond to the data structure known as **(homogeneous) vectors**
  - That is, a list/tuple whose components are all of the same type (e) of a certain length (n)
- Notice that the type of the input vector depends on a term of another type, namely natural numbers

Problems with Dynamic HHS

- The n-tuples we use to hold our DRs correspond to the data structure known as **(homogeneous) vectors**
  - That is, a list/tuple whose components are all of the same type (e) of a certain length (n)
- Notice that the type of the input vector depends on a term of another type, namely natural numbers
- The type system for HOL doesn't allow for types to be formed in this way

Problems with Dynamic HHS

- The n-tuples we use to hold our DRs correspond to the data structure known as **(homogeneous) vectors**
  - That is, a list/tuple whose components are all of the same type (e) of a certain length (n)
- Notice that the type of the input vector depends on a term of another type, namely natural numbers
- The type system for HOL doesn't allow for types to be formed in this way
- A metalinguistic set of "vector types" (on par with our set of sense types) won't suffice here, since we'll want many of our semantic expressions to be defined on all contexts

Non-entity anaphora

- We want the input vector to contain semantic objects that are not necessarily entities

Non-entity anaphora

- We want the input vector to contain semantic objects that are not necessarily entities

(2) A: My cousin thinks that Justin Bieber is Italian.
B: That's not true!

Non-entity anaphora

- We want the input vector to contain semantic objects that are not necessarily entities

(2) A: My cousin thinks that Justin Bieber is Italian.
B: That's not true!

- In general, we need to be able to have **heterogenous vectors** whose components are those semantic objects to which anaphoric reference is possible

Non-entity anaphora

- We want the input vector to contain semantic objects that are not necessarily entities

(2) A: My cousin thinks that Justin Bieber is Italian.
B: That's not true!

- In general, we need to be able to have **heterogenous vectors** whose components are those semantic objects to which anaphoric reference is possible
- By a **heterogeneous vector**, we mean a term h of a fixed length n (as before), and for each $0 \leq i < n$, the type of the ith element of h is also fixed (but not necessarily the same as the type of any jth element of h, $0 \leq j < n$, $i \neq j$)

Question-answer anaphora

- To handle question-answer anaphora, we extended the previous HHS context model such that a context is now a function from n-tuples of semantic objects to ordered pairs $\langle p, l\rangle$, where:

Question-answer anaphora

- To handle question-answer anaphora, we extended the previous HHS context model such that a context is now a function from n-tuples of semantic objects to ordered pairs $\langle p, l \rangle$, where:
  1. p is a (static) proposition, the CG, and

Question-answer anaphora

- To handle question-answer anaphora, we extended the previous HHS context model such that a context is now a function from n-tuples of semantic objects to ordered pairs $\langle p, l \rangle$, where:
  1. p is a (static) proposition, the CG, and
  2. l is a stack of (some of) the semantic objects in the n-tuple

Question-answer anaphora

- To handle question-answer anaphora, we extended the previous HHS context model such that a context is now a function from n-tuples of semantic objects to ordered pairs $\langle p, l\rangle$, where:
  1. p is a (static) proposition, the CG, and
  2. l is a stack of (some of) the semantic objects in the n-tuple
- We call the second component the **topics under discussion (TUD)** stack

Question-answer anaphora: the TUD-stack

- The TUD-stack is similar to the QUD-stack (Ginzburg, 1994; Roberts, 1996/2012) in the sense of keeping track of accepted questions in discourse

Question-answer anaphora: the TUD-stack

- The TUD-stack is similar to the QUD-stack (Ginzburg, 1994; Roberts, 1996/2012) in the sense of keeping track of accepted questions in discourse
- However, such questions are not stored as sets of propositions as in the QUD-stack but rather push onto the TUD-stack a DR for which further identification is sought

Question-answer anaphora: the TUD-stack

- The TUD-stack is similar to the QUD-stack (Ginzburg, 1994; Roberts, 1996/2012) in the sense of keeping track of accepted questions in discourse
- However, such questions are not stored as sets of propositions as in the QUD-stack but rather push onto the TUD-stack a DR for which further identification is sought
- Answers to questions are analyzed as anaphoric to this DR

Question-answer anaphora: the TUD-stack

- The TUD-stack is similar to the QUD-stack (Ginzburg, 1994; Roberts, 1996/2012) in the sense of keeping track of accepted questions in discourse
- However, such questions are not stored as sets of propositions as in the QUD-stack but rather push onto the TUD-stack a DR for which further identification is sought
- Answers to questions are analyzed as anaphoric to this DR
- The DRs on the TUD-stack form a **subvector** of the input vector

Interim summary

- HHS is a semantic theory written in HOL

Interim summary

- HHS is a semantic theory written in HOL
- Meanings of NL sentences are treated as a primitive type (p) rather than a set of worlds (w $\rightarrow$ t)

Interim summary

- HHS is a semantic theory written in HOL
- Meanings of NL sentences are treated as a primitive type (p) rather than a set of worlds (w $\to$ t)
- HHS works fine for static semantics, but proves problematic when we try to move to a dynamic system

Interim summary

- HHS is a semantic theory written in HOL
- Meanings of NL sentences are treated as a primitive type (p) rather than a set of worlds (w $\rightarrow$ t)
- HHS works fine for static semantics, but proves problematic when we try to move to a dynamic system
- In general, we would like to be able to define certain families of types in the object language, as well as functions which operate on/return terms whose types are members of these type families

From Curry to Howard

- Curry and Feys (1958) (paraphrased): if we think of the types of pure typed lambda calculus (with $\rightarrow$ as the only type constructor) as formulas of intuitionistic propositional logic (IPL), then the IPL theorems are the types which have an inhabitant (closed term of that type)

From Curry to Howard

- Curry and Feys (1958) (paraphrased): if we think of the types of pure typed lambda calculus (with $\rightarrow$ as the only type constructor) as formulas of intuitionistic propositional logic (IPL), then the IPL theorems are the types which have an inhabitant (closed term of that type)
- Howard 1969 sketched how to extend Curry and Feys's observation from IPL to intuitionistic first order logic (IFOL), more specifically Heyting arithmetic (HA)

From Curry to Howard

- Curry and Feys (1958) (paraphrased): if we think of the types of pure typed lambda calculus (with $\rightarrow$ as the only type constructor) as formulas of intuitionistic propositional logic (IPL), then the IPL theorems are the types which have an inhabitant (closed term of that type)
- Howard 1969 sketched how to extend Curry and Feys's observation from IPL to intuitionistic first order logic (IFOL), more specifically Heyting arithmetic (HA)
- In Howard's system, arithmetic formulas (equalities and formulas built up from them using implication and quantification over natural numbers) are themselves types

From Curry to Howard

- Curry and Feys (1958) (paraphrased): if we think of the types of pure typed lambda calculus (with $\rightarrow$ as the only type constructor) as formulas of intuitionistic propositional logic (IPL), then the IPL theorems are the types which have an inhabitant (closed term of that type)
- Howard 1969 sketched how to extend Curry and Feys's observation from IPL to intuitionistic first order logic (IFOL), more specifically Heyting arithmetic (HA)
- In Howard's system, arithmetic formulas (equalities and formulas built up from them using implication and quantification over natural numbers) are themselves types
- And the HA theorems are (again) the inhabited types

Key Ideas of Howard 1969

- It is possible to have a typed theory where some some basic types (e.g. nat) are types of the things the theory is about, and others are formulas about them (e.g. $1 + 1 = 1$)

Key Ideas of Howard 1969

- It is possible to have a typed theory where some some basic types (e.g. nat) are types of the things the theory is about, and others are formulas about them (e.g. $1 + 1 = 1$)
- For the latter kind of type, the type's inhabitants are thought of as its proofs

Key Ideas of Howard 1969

- It is possible to have a typed theory where some some basic types (e.g. nat) are types of the things the theory is about, and others are formulas about them (e.g. $1 + 1 = 1$)
- For the latter kind of type, the type's inhabitants are thought of as its proofs
- The type constructors can be be extended from just implication to include the other intuitionistic connectives and quantifiers

Key Ideas of Howard 1969

- It is possible to have a typed theory where some some basic types (e.g. nat) are types of the things the theory is about, and others are formulas about them (e.g. $1 + 1 = 1$)
- For the latter kind of type, the type's inhabitants are thought of as its proofs
- The type constructors can be be extended from just implication to include the other intuitionistic connectives and quantifiers
- Like terms, types can have terms as parts (including free variables)

From Howard 1969 to CIC

- CIC (Coquand and Huet, 1988) can be seen as a way of systematically extending the Curry-Howard correspondendence to (intuitionistic nonextensional) HOL, together with the following four new ingredients:

From Howard 1969 to CIC

- CIC (Coquand and Huet, 1988) can be seen as a way of systematically extending the Curry-Howard correspondendence to (intuitionistic nonextensional) HOL, together with the following four new ingredients:
- Whereas in HOL, only terms have a type, in CIC **all** expressions (i.e., both terms and types) have a type

From Howard 1969 to CIC

- CIC (Coquand and Huet, 1988) can be seen as a way of systematically extending the Curry-Howard correspondendence to (intuitionistic nonextensional) HOL, together with the following four new ingredients:
- Whereas in HOL, only terms have a type, in CIC **all** expressions (i.e., both terms and types) have a type
- There are variables of all types (including those whose inhabitants are types)

From Howard 1969 to CIC

- CIC (Coquand and Huet, 1988) can be seen as a way of systematically extending the Curry-Howard correspondence to (intuitionistic nonextensional) HOL, together with the following four new ingredients:
- Whereas in HOL, only terms have a type, in CIC **all** expressions (i.e., both terms and types) have a type
- There are variables of all types (including those whose inhabitants are types)
- Not only can types and terms contain subexpressions which are terms, they can also contain subexpressions which are types

From Howard 1969 to CIC

- CIC (Coquand and Huet, 1988) can be seen as a way of systematically extending the Curry-Howard correspondence to (intuitionistic nonextensional) HOL, together with the following four new ingredients:
- Whereas in HOL, only terms have a type, in CIC **all** expressions (i.e., both terms and types) have a type
- There are variables of all types (including those whose inhabitants are types)
- Not only can types and terms contain subexpressions which are terms, they can also contain subexpressions which are types
- Some types can be inductively defined in the object language

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, Type_i (for i a (meta-)natural number)

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, Type_i (for i a (meta-)natural number)
- We have the following axioms:

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, $\mathrm{Type_i}$ (for i a (meta-)natural number)
- We have the following axioms:
  - $\vdash \mathrm{Set} : \mathrm{Type}_0$

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, Type_i (for i a (meta-)natural number)
- We have the following axioms:
  - $\vdash \mathrm{Set} : \mathrm{Type}_0$
  - $\vdash \mathrm{Prop} : \mathrm{Type}_0$

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, $\mathrm{Type_i}$ (for i a (meta-)natural number)
- We have the following axioms:
  - $\vdash \mathrm{Set} : \mathrm{Type_0}$
  - $\vdash \mathrm{Prop} : \mathrm{Type_0}$
  - $\vdash \mathrm{Type_i} : \mathrm{Type_{i+1}}$

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, Type_i (for i a (meta-)natural number)
- We have the following axioms:
  - $\vdash \mathrm{Set} : \mathrm{Type}_0$
  - $\vdash \mathrm{Prop} : \mathrm{Type}_0$
  - $\vdash \mathrm{Type}_\mathrm{i} : \mathrm{Type}_{\mathrm{i}+1}$
- Furthermore, there is a sort hierarchy $\mathrm{Prop} \preceq \mathrm{Set} \preceq \mathrm{Type}_0 \preceq \mathrm{Type}_1 \preceq \ldots$, along with type **cumulativity**— For any sorts s, s':

CIC Sorts

- Types whose inhabitants are types are called **sorts** or **universes**; other types we call **mere** types
- The inventory of sorts is: Prop, Set, $\mathrm{Type_i}$ (for i a (meta-)natural number)
- We have the following axioms:
  - $\vdash \mathrm{Set} : \mathrm{Type_0}$
  - $\vdash \mathrm{Prop} : \mathrm{Type_0}$
  - $\vdash \mathrm{Type_i} : \mathrm{Type_{i+1}}$
- Furthermore, there is a sort hierarchy $\mathrm{Prop} \preceq \mathrm{Set} \preceq \mathrm{Type_0} \preceq \mathrm{Type_1} \preceq \ldots$, along with type **cumulativity**— For any sorts s, s':

$$\frac{\Gamma \vdash A : s \qquad s \preceq s'}{\Gamma \vdash A : s'}$$

$\forall$ Constructs Types of Dependent Functions

- Whereas the ‘workhorse’ of type formation in HOL is the function arrow $\rightarrow$, CIC’s primary tool is the universal quantifier $\forall$

$\forall$ Constructs Types of Dependent Functions

- Whereas the ‘workhorse’ of type formation in HOL is the function arrow $\rightarrow$, CIC’s primary tool is the universal quantifier $\forall$
- An expression $f : \forall x\!:\!A.B$ is a function that takes terms of type A

$\forall$ Constructs Types of Dependent Functions

- Whereas the 'workhorse' of type formation in HOL is the function arrow $\rightarrow$, CIC's primary tool is the universal quantifier $\forall$
- An expression $f : \forall x{:}A.B$ is a function that takes terms of type A
- However, the exact type f returns can depend on *which* term of type A f is applied to

$\forall$ Constructs Types of Dependent Functions

- Whereas the 'workhorse' of type formation in HOL is the function arrow $\rightarrow$, CIC's primary tool is the universal quantifier $\forall$
- An expression $f : \forall x \colon A.B$ is a function that takes terms of type A
- However, the exact type f returns can depend on *which* term of type A f is applied to
  - For example, if we have $f : \forall n \colon \mathrm{nat}.[n = (n + 0)]$, then $(f\ 0) : 0 = (0 + 0)$, $(f\ 5) : 5 = (5 + 0)$, etc

$\forall$ Constructs Types of Dependent Functions

- Whereas the 'workhorse' of type formation in HOL is the function arrow $\rightarrow$, CIC's primary tool is the universal quantifier $\forall$
- An expression $f : \forall x{:}A.B$ is a function that takes terms of type A
- However, the exact type f returns can depend on *which* term of type A f is applied to
  - For example, if we have $f : \forall n{:}\mathrm{nat}.[n = (n + 0)]$, then $(f\ 0) : 0 = (0 + 0)$, $(f\ 5) : 5 = (5 + 0)$, etc
- We can still use the HOL $A \rightarrow B$ notation for types $\forall x{:}A.B$ such that the variable x doesn't occur freely in B

Type Formation Rules using $\forall$

- Systems in which there are no restrictions on forming types via $\forall$ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent

Type Formation Rules using ∀

- Systems in which there are no restrictions on forming types via ∀ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent
- As a result, CIC has three type formation rules concerning ∀, one for each sort:

Type Formation Rules using $\forall$

- Systems in which there are no restrictions on forming types via $\forall$ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent
- As a result, CIC has three type formation rules concerning $\forall$, one for each sort:

For any sort s,

Type Formation Rules using ∀

- Systems in which there are no restrictions on forming types via ∀ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent
- As a result, CIC has three type formation rules concerning ∀, one for each sort:

$$\frac{\Gamma \vdash A : s \qquad \overset{\text{For any sort } s,}{\Gamma, x : A \vdash B : \text{Prop}}}{\Gamma \vdash (\forall x\!:\!A.B) : \text{Prop}}\ \text{Prod-Prop}$$

Type Formation Rules using $\forall$

- Systems in which there are no restrictions on forming types via $\forall$ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent
- As a result, CIC has three type formation rules concerning $\forall$, one for each sort:

For any sort s,

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : \mathrm{Prop}}{\Gamma \vdash (\forall x\!:\!A.B) : \mathrm{Prop}}\ \text{Prod-Prop}$$

$$\frac{\Gamma \vdash A : s \qquad s \preceq \mathrm{Set} \qquad \Gamma, x : A \vdash B : \mathrm{Set}}{\Gamma \vdash (\forall x\!:\!A.B) : \mathrm{Set}}\ \text{Prod-Set}$$

Type Formation Rules using $\forall$

- Systems in which there are no restrictions on forming types via $\forall$ suffer from **Girard's Paradox** (Girard, 1972), and are therefore inconsistent
- As a result, CIC has three type formation rules concerning $\forall$, one for each sort:

For any sort s,

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : \mathrm{Prop}}{\Gamma \vdash (\forall x{:}A.B) : \mathrm{Prop}}\ \text{Prod-Prop}$$

$$\frac{\Gamma \vdash A : s \qquad s \preceq \mathrm{Set} \qquad \Gamma, x : A \vdash B : \mathrm{Set}}{\Gamma \vdash (\forall x{:}A.B) : \mathrm{Set}}\ \text{Prod-Set}$$

$$\frac{\Gamma \vdash A : \mathrm{Type_i} \qquad \Gamma, x : A \vdash B : \mathrm{Type_i}}{\Gamma \vdash (\forall x{:}A.B) : \mathrm{Type_i}}\ \text{Prod-Type}$$

Translating a Theory from HOL to CIC

- HOL formulas, which are terms of type t, are replaced by (mere) types of sort Prop

Translating a Theory from HOL to CIC

- HOL formulas, which are terms of type t, are replaced by (mere) types of sort Prop
- So, a CIC formula ϕ is a (mere) type, and its inhabitants are thought of as its proofs (in HOL it had no inhabitants)

Translating a Theory from HOL to CIC

- HOL formulas, which are terms of type t, are replaced by (mere) types of sort Prop
- So, a CIC formula ϕ is a (mere) type, and its inhabitants are thought of as its proofs (in HOL it had no inhabitants)
- The other HOL basic types become types of sort Set

Translating a Theory from HOL to CIC

- HOL formulas, which are terms of type t, are replaced by (mere) types of sort Prop
- So, a CIC formula ϕ is a (mere) type, and its inhabitants are thought of as its proofs (in HOL it had no inhabitants)
- The other HOL basic types become types of sort Set
- Additionally, certain inductive types of sort Set are standard in CIC, including nat (natural numbers) and bool (booleans/truth values)

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.
- The word ‘proposition’ is now ambiguous between (interpretations of) terms of type p and (interpretations of) formulas

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.
- The word ‘proposition’ is now ambiguous between (interpretations of) terms of type p and (interpretations of) formulas
- To disambiguate, we call the latter ‘Props’

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.
- The word ‘proposition’ is now ambiguous between (interpretations of) terms of type p and (interpretations of) formulas
- To disambiguate, we call the latter ‘Props’
- The way to make a formula ϕ an axiom is to declare a constant (proof object, witness) of that type: $\vdash \mathsf{c} : \phi$

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.
- The word 'proposition' is now ambiguous between (interpretations of) terms of type p and (interpretations of) formulas
- To disambiguate, we call the latter 'Props'
- The way to make a formula ϕ an axiom is to declare a constant (proof object, witness) of that type: $\vdash \mathsf{c} : \phi$
- Terms of type bool are used, inter alia, for the truth values of propositions at worlds

Types for CHS

- $\vdash \mathrm{e}, \mathrm{w}, \mathrm{p} : \mathrm{Set}$
- The two roles of the HOL type t are now divided between the sort Prop and the Set bool.
- The word 'proposition' is now ambiguous between (interpretations of) terms of type p and (interpretations of) formulas
- To disambiguate, we call the latter 'Props'
- The way to make a formula ϕ an axiom is to declare a constant (proof object, witness) of that type: $\vdash \mathsf{c} : \phi$
- Terms of type bool are used, inter alia, for the truth values of propositions at worlds
- As in some other dynamic theories, nat is used for discourse referents (In CHS, these will be positions in argument vectors of contexts)

Going Sundholmian?

- In the context of CIC, ‘going Sundholmian’ means analyzing the senses of declarative sentences as Props

Going Sundholmian?

- In the context of CIC, 'going Sundholmian' means analyzing the senses of declarative sentences as Props
- This has been the norm in CIC-based semantic research

Going Sundholmian?

- In the context of CIC, 'going Sundholmian' means analyzing the senses of declarative sentences as Props
- This has been the norm in CIC-based semantic research
- CHS is the odd one out in this respect because it uses a distinct type p for that purpose

Going Sundholmian?

- In the context of CIC, ‘going Sundholmian’ means analyzing the senses of declarative sentences as Props
- This has been the norm in CIC-based semantic research
- CHS is the odd one out in this respect because it uses a distinct type p for that purpose
- For us, conflating Prop and p would be like conflating t and p in HHS, or conflating t and $w \rightarrow t$ in MS

Going Sundholmian?

- In the context of CIC, ‘going Sundholmian’ means analyzing the senses of declarative sentences as Props
- This has been the norm in CIC-based semantic research
- CHS is the odd one out in this respect because it uses a distinct type p for that purpose
- For us, conflating Prop and p would be like conflating t and p in HHS, or conflating t and $\mathrm{w} \rightarrow \mathrm{t}$ in MS
- Additionally, this would seem to predict that NL reasoning disallows the use of Excluded Middle, which strikes us as empirically incorrect

Basic Pieces of CHS (1/2)

- Constants for entities:

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$

  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \to \mathrm{p}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$

  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \to \mathrm{p}$

  etc.

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$
  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \rightarrow \mathrm{p}$
  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$
  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \to \mathrm{p}$
  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$
- The 'true-at' function: $\vdash @ : \mathrm{p} \to \mathrm{w} \to \mathrm{bool}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$
  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \rightarrow \mathrm{p}$
  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$
- The 'true-at' function: $\vdash @ : \mathrm{p} \rightarrow \mathrm{w} \rightarrow \mathrm{bool}$
- Basic propositional connectives:

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$
  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \rightarrow \mathrm{p}$
  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$
- The 'true-at' function: $\vdash @ : \mathrm{p} \rightarrow \mathrm{w} \rightarrow \mathrm{bool}$
- Basic propositional connectives:

  $\vdash \mathsf{truth}, \mathsf{falsity} : \mathrm{p}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$

  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \to \mathrm{p}$

  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$
- The 'true-at' function: $\vdash @ : \mathrm{p} \to \mathrm{w} \to \mathrm{bool}$
- Basic propositional connectives:

  $\vdash \mathsf{truth}, \mathsf{falsity} : \mathrm{p}$

  $\vdash \mathsf{not} : \mathrm{p} \to \mathrm{p}$

Basic Pieces of CHS (1/2)

- Constants for entities:

  $\vdash \mathsf{pedro}, \mathsf{john}, \mathsf{mary}, \mathsf{kim}, \mathsf{sandy}, \ldots : \mathrm{e}$

- Constants for n-ary propositions/predicates/relations:

  $\vdash \mathsf{rain} : \mathrm{p}$

  $\vdash \mathsf{run}, \mathsf{sleep}, \ldots : \mathrm{e} \to \mathrm{p}$

  etc.

- Constant for the 'actual' world: $\vdash w_0 : \mathrm{w}$
- The 'true-at' function: $\vdash @ : \mathrm{p} \to \mathrm{w} \to \mathrm{bool}$
- Basic propositional connectives:

  $\vdash \mathsf{truth}, \mathsf{falsity} : \mathrm{p}$

  $\vdash \mathsf{not} : \mathrm{p} \to \mathrm{p}$

  $\vdash \mathsf{and}, \mathsf{or}, \mathsf{implies} : \mathrm{p} \to \mathrm{p} \to \mathrm{p}$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg, \wedge, \vee, \Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg$,$\wedge$,$\vee$,$\Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$$\vdash \mathsf{truth\_ax} : \forall w \colon \mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg$,$\wedge$,$\vee$,$\Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$$\vdash \mathsf{truth\_ax} : \forall w : \mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$$

$$\vdash \mathsf{falsity\_ax} : \forall w : \mathrm{w}.[\mathsf{falsity}@w = \mathsf{false}]$$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg$,$\wedge$,$\vee$,$\Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$$\vdash \mathsf{truth\_ax} : \forall w{:}\mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$$

$$\vdash \mathsf{falsity\_ax} : \forall w{:}\mathrm{w}.[\mathsf{falsity}@w = \mathsf{false}]$$

$$\vdash \mathsf{not\_ax} : \forall p{:}\mathrm{p}.\forall w{:}\mathrm{w}.[(\mathsf{not}\ p)@w = \neg(p@w)]$$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg, \wedge, \vee, \Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$\vdash \mathsf{truth\_ax} : \forall w : \mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$

$\vdash \mathsf{falsity\_ax} : \forall w : \mathrm{w}.[\mathsf{falsity}@w = \mathsf{false}]$

$\vdash \mathsf{not\_ax} : \forall p : \mathrm{p}.\forall w : \mathrm{w}.[(\mathsf{not}\ p)@w = \neg(p@w)]$

$\vdash \mathsf{and\_ax} : \forall p, q : \mathrm{p}.\forall w : \mathrm{w}.[(p\ \mathsf{and}\ q)@w = (p@w \wedge q@w)]$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg, \wedge, \vee, \Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$$\vdash \mathsf{truth\_ax} : \forall w\!:\!\mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$$

$$\vdash \mathsf{falsity\_ax} : \forall w\!:\!\mathrm{w}.[\mathsf{falsity}@w = \mathsf{false}]$$

$$\vdash \mathsf{not\_ax} : \forall p\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(\mathsf{not}\ p)@w = \neg(p@w)]$$

$$\vdash \mathsf{and\_ax} : \forall p, q\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(p\ \mathsf{and}\ q)@w = (p@w \wedge q@w)]$$

$$\vdash \mathsf{or\_ax} : \forall p, q\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(p\ \mathsf{or}\ q)@w = (p@w \vee q@w)]$$

Basic Pieces of CHS (2/2)

Axioms for the connectives look similar to the ones in HHS (using $\neg,\wedge,\vee,\Rightarrow$ for boolean negation, conjunction, disjunction, and implication):

$$\vdash \mathsf{truth\_ax} : \forall w\!:\!\mathrm{w}.[\mathsf{truth}@w = \mathsf{true}]$$

$$\vdash \mathsf{falsity\_ax} : \forall w\!:\!\mathrm{w}.[\mathsf{falsity}@w = \mathsf{false}]$$

$$\vdash \mathsf{not\_ax} : \forall p\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(\mathsf{not}\ p)@w = \neg(p@w)]$$

$$\vdash \mathsf{and\_ax} : \forall p,q\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(p\ \mathsf{and}\ q)@w = (p@w \wedge q@w)]$$

$$\vdash \mathsf{or\_ax} : \forall p,q\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(p\ \mathsf{or}\ q)@w = (p@w \vee q@w)]$$

$$\vdash \mathsf{imp\_ax} : \forall p,q\!:\!\mathrm{p}.\forall w\!:\!\mathrm{w}.[(p\ \mathsf{implies}\ q)@w = (p@w \Rightarrow q@w)]$$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \text{stat} : \text{Set}$$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \rightarrow \mathrm{stat} \rightarrow \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

  $\vdash \text{stat} : \text{Set}$

  $\vdash \mathsf{ent} : \text{stat}$

  $\vdash \mathsf{prp} : \text{stat}$

  $\vdash \mathsf{func} : \text{stat} \rightarrow \text{stat} \rightarrow \text{stat}$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$
$(\mathrm{Sns}\ \mathsf{prp}) := \mathrm{p}$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \text{stat} : \text{Set}$$
$$\vdash \textsf{ent} : \text{stat}$$
$$\vdash \textsf{prp} : \text{stat}$$
$$\vdash \textsf{func} : \text{stat} \to \text{stat} \to \text{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\text{Sns } \textsf{ent}) := \text{e}$ $\quad$ $(\text{Ext } \textsf{ent}) := \text{e}$

$(\text{Sns } \textsf{prp}) := \text{p}$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$ $\quad$ $(\mathrm{Ext}\ \mathsf{ent}) := \mathrm{e}$
$(\mathrm{Sns}\ \mathsf{prp}) := \mathrm{p}$ $\quad$ $(\mathrm{Ext}\ \mathsf{prp}) := \mathrm{bool}$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{ent}) := \mathrm{e}$

$(\mathrm{Sns}\ \mathsf{prp}) := \mathrm{p}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{prp}) := \mathrm{bool}$

For any terms $a, b : \mathrm{stat}$,

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{ent}) := \mathrm{e}$

$(\mathrm{Sns}\ \mathsf{prp}) := \mathrm{p}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{prp}) := \mathrm{bool}$

For any terms $a, b : \mathrm{stat}$,

$(\mathrm{Sns}\ (\mathsf{func}\ a\ b)) := (\mathrm{Sns}\ a) \to (\mathrm{Sns}\ b)$

Defining the Family of Sense Types Internally (1/2)

- To allow us to refer to, and define functions over, the types of (static) linguistic senses, we first add an additional type, stat:

$$\vdash \mathrm{stat} : \mathrm{Set}$$
$$\vdash \mathsf{ent} : \mathrm{stat}$$
$$\vdash \mathsf{prp} : \mathrm{stat}$$
$$\vdash \mathsf{func} : \mathrm{stat} \to \mathrm{stat} \to \mathrm{stat}$$

- We then define two functions, Sns and Ext to map each term of type stat to its corresponding sense/extension type:

$(\mathrm{Sns}\ \mathsf{ent}) := \mathrm{e}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{ent}) := \mathrm{e}$

$(\mathrm{Sns}\ \mathsf{prp}) := \mathrm{p}$ $\qquad$ $(\mathrm{Ext}\ \mathsf{prp}) := \mathrm{bool}$

For any terms $a, b : \mathrm{stat}$,

$(\mathrm{Sns}\ (\mathsf{func}\ a\ b)) := (\mathrm{Sns}\ a) \to (\mathrm{Sns}\ b)$ $\qquad$ $(\mathrm{Ext}\ (\mathsf{func}\ a\ b)) := (\mathrm{Sns}\ a) \to (\mathrm{Ext}\ b)$

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

$$\mathrm{ext\_at} : \forall s : \mathrm{stat}.[(\mathrm{Sns}\ s) \to \mathrm{w} \to (\mathrm{Ext}\ s)]$$

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

$$\mathrm{ext\_at} : \forall s : \mathrm{stat}.[(\mathrm{Sns}\ s) \to \mathrm{w} \to (\mathrm{Ext}\ s)]$$

$$(\mathrm{ext\_at}\ \mathsf{ent}) := \lambda x : \mathrm{e}.\lambda w : \mathrm{w}.x$$

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

$$\mathrm{ext\_at} : \forall s : \mathrm{stat}.[(\mathrm{Sns}\ s) \to \mathrm{w} \to (\mathrm{Ext}\ s)]$$

$$(\mathrm{ext\_at}\ \mathsf{ent}) := \lambda x\!:\!\mathrm{e}.\lambda w\!:\!\mathrm{w}.x$$

$$(\mathrm{ext\_at}\ \mathsf{prp}) := \lambda p\!:\!\mathrm{p}.\lambda w\!:\!\mathrm{w}.p@w$$

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

$\text{ext\_at} : \forall s : \text{stat}.[(\text{Sns } s) \to \text{w} \to (\text{Ext } s)]$

$(\text{ext\_at } \mathsf{ent}) := \lambda x\!:\!\text{e}.\lambda w\!:\!\text{w}.x$

$(\text{ext\_at } \mathsf{prp}) := \lambda p\!:\!\text{p}.\lambda w\!:\!\text{w}.p@w$

For any terms $a, b : \text{stat}$,

Defining the Family of Sense Types Internally (2/2)

Using stat, Sns, and Ext, we can define HHS' $@_A$ as a single recursive function language-internally:

$$\text{ext\_at} : \forall s : \text{stat}.[(\text{Sns } s) \to \text{w} \to (\text{Ext } s)]$$

$$(\text{ext\_at } \mathsf{ent}) := \lambda x : \text{e}.\lambda w : \text{w}.x$$

$$(\text{ext\_at } \mathsf{prp}) := \lambda p : \text{p}.\lambda w : \text{w}.p@w$$

For any terms $a, b : \text{stat}$,

$$(\text{ext\_at } (\mathsf{func}\ a\ b)) :=$$
$$\lambda f : (\text{Sns } a) \to (\text{Sns } b).\lambda w : \text{w}.\lambda x : \text{Sns } a.\text{ext\_at } b\ (f\ x)\ w$$

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s\!:\!\mathrm{stat}.[((\mathrm{Sns}\ s) \to \mathrm{p}) \to \mathrm{p}]$$

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s{:}\mathrm{stat}.[((\mathrm{Sns}\ s) \rightarrow \mathrm{p}) \rightarrow \mathrm{p}]$$
$$\vdash \mathsf{equals} : \forall s{:}\mathrm{stat}.[(\mathrm{Sns}\ s) \rightarrow (\mathrm{Sns}\ s) \rightarrow \mathrm{p}]$$

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s\!:\!\mathrm{stat}.[((\mathrm{Sns}\ s) \to \mathrm{p}) \to \mathrm{p}]$$
$$\vdash \mathsf{equals} : \forall s\!:\!\mathrm{stat}.[(\mathrm{Sns}\ s) \to (\mathrm{Sns}\ s) \to \mathrm{p}]$$

We axiomatize these as follows (using $\sim$ for Prop negation):

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s\!:\!\mathrm{stat}.[((\mathrm{Sns}\ s) \to \mathrm{p}) \to \mathrm{p}]$$
$$\vdash \mathsf{equals} : \forall s\!:\!\mathrm{stat}.[(\mathrm{Sns}\ s) \to (\mathrm{Sns}\ s) \to \mathrm{p}]$$

We axiomatize these as follows (using $\sim$ for Prop negation):

$$\vdash \mathsf{fa\_ax} : \forall s\!:\!\mathrm{stat}.\forall R\!:\!(\mathrm{Sns}\ s) \to \mathrm{p}.\forall w\!:\!\mathrm{w}.$$
$$[((\mathsf{p\_forall}\ s\ R)@w = \mathsf{true}) \leftrightarrow \forall x : (\mathrm{Sns}\ s).[(R\ x)@w = \mathsf{true}]]$$

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s\!:\!\mathrm{stat}.[((\mathrm{Sns}\ s) \to \mathrm{p}) \to \mathrm{p}]$$
$$\vdash \mathsf{equals} : \forall s\!:\!\mathrm{stat}.[(\mathrm{Sns}\ s) \to (\mathrm{Sns}\ s) \to \mathrm{p}]$$

We axiomatize these as follows (using $\sim$ for Prop negation):

$$\vdash \mathsf{fa\_ax} : \forall s\!:\!\mathrm{stat}.\forall R\!:\!(\mathrm{Sns}\ s) \to \mathrm{p}.\forall w\!:\!\mathrm{w}.$$
$$[((\mathsf{p\_forall}\ s\ R)@w = \mathsf{true}) \leftrightarrow \forall x : (\mathrm{Sns}\ s).[(R\ x)@w = \mathsf{true}]]$$

$$\vdash \mathsf{ex\_ax} : \forall s\!:\!\mathrm{stat}.\forall R\!:\!(\mathrm{Sns}\ s) \to \mathrm{p}.\forall w\!:\!\mathrm{w}.$$
$$[((\mathsf{p\_exists}\ s\ R)@w = \mathsf{true}) \leftrightarrow \sim (\forall x : (\mathrm{Sns}\ s).[(R\ x)@w = \mathsf{false}])]$$

Propositional Quantifiers and Equality

With stat, we can also add single constants for propositional operators corresponding to the universal & existential quantifiers, as well as for (hyperintensional) equality:

$$\vdash \mathsf{p\_forall}, \mathsf{p\_exists} : \forall s{:}\mathrm{stat}.[((\mathrm{Sns}\ s) \to \mathrm{p}) \to \mathrm{p}]$$
$$\vdash \mathsf{equals} : \forall s{:}\mathrm{stat}.[(\mathrm{Sns}\ s) \to (\mathrm{Sns}\ s) \to \mathrm{p}]$$

We axiomatize these as follows (using $\sim$ for Prop negation):

$$\vdash \mathsf{fa\_ax} : \forall s{:}\mathrm{stat}.\forall R{:}(\mathrm{Sns}\ s) \to \mathrm{p}.\forall w{:}\mathrm{w}.$$
$$[((\mathsf{p\_forall}\ s\ R)@w = \mathsf{true}) \leftrightarrow \forall x : (\mathrm{Sns}\ s).[(R\ x)@w = \mathsf{true}]]$$

$$\vdash \mathsf{ex\_ax} : \forall s{:}\mathrm{stat}.\forall R{:}(\mathrm{Sns}\ s) \to \mathrm{p}.\forall w{:}\mathrm{w}.$$
$$[((\mathsf{p\_exists}\ s\ R)@w = \mathsf{true}) \leftrightarrow \sim (\forall x : (\mathrm{Sns}\ s).[(R\ x)@w = \mathsf{false}])]$$

$$\vdash \mathsf{eq\_ax} : \forall s{:}\mathrm{stat}.\forall x, y{:}(\mathrm{Sns}\ S).\forall w{:}\mathrm{w}.[((x\, \mathsf{equals}_s\, y)@w = \mathsf{true}) \leftrightarrow (x = y)]$$

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

$$\vdash \mathrm{Vect}_s : s \to \mathrm{nat} \to s$$

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

$\vdash \mathrm{Vect}_s : s \rightarrow \mathrm{nat} \rightarrow s$

$\vdash \mathsf{vnil}_s : \forall A\!:\!s.[\mathrm{Vect}_s\ A\ 0]$ (written [ ])

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

$$\vdash \mathrm{Vect}_s : s \to \mathrm{nat} \to s$$

$$\vdash \mathsf{vnil}_s : \forall A\!:\!s.[\mathrm{Vect}_s\ A\ 0] \qquad \text{(written [ ])}$$

$$\vdash \mathsf{vcons}_s : \forall A\!:\!s.\forall n\!:\!\mathrm{nat}.[A \to \mathrm{Vect}_s\ A\ n \to \mathrm{Vect}_s\ A\ (S\ n)] \qquad \text{(written infix as ::)}$$

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

  $\vdash \mathrm{Vect}_s : s \rightarrow \mathrm{nat} \rightarrow s$

  $\vdash \mathsf{vnil}_s : \forall A{:}s.[\mathrm{Vect}_s\ A\ 0]$ (written [ ])

  $\vdash \mathsf{vcons}_s : \forall A{:}s.\forall n{:}\mathrm{nat}.[A \rightarrow \mathrm{Vect}_s\ A\ n \rightarrow \mathrm{Vect}_s\ A\ (S\ n)]$ (written infix as ::)

- Of particular interest to us will be dependent pairs $\langle n, v\rangle$ where $n : \mathrm{nat}$ and $v : (\mathrm{Vect\ stat}\ n)$

(Homogeneous) Vectors

- The type for homogeneous vectors can be schematized over sorts s as follows:

$$\vdash \mathrm{Vect}_s : s \to \mathrm{nat} \to s$$
$$\vdash \mathsf{vnil}_s : \forall A\!:\!s.[\mathrm{Vect}_s\ A\ 0] \qquad \text{(written [ ])}$$
$$\vdash \mathsf{vcons}_s : \forall A\!:\!s.\forall n\!:\!\mathrm{nat}.[A \to \mathrm{Vect}_s\ A\ n \to \mathrm{Vect}_s\ A\ (S\ n)] \qquad \text{(written infix as ::)}$$

- Of particular interest to us will be dependent pairs $\langle n, v\rangle$ where $n : \mathrm{nat}$ and $v : (\mathrm{Vect\ stat}\ n)$
- Call the type of such pairs **Arity**

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types
- Therefore, we work with stat in the type of the vectors rather than the sense types themselves to ensure this property

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types
- Therefore, we work with stat in the type of the vectors rather than the sense types themselves to ensure this property
- Since all the sense types are in Set, our heterogeneous vectors need not be schematized over sorts:

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types
- Therefore, we work with stat in the type of the vectors rather than the sense types themselves to ensure this property
- Since all the sense types are in Set, our heterogeneous vectors need not be schematized over sorts:

$$\vdash \mathrm{HetVect} : \mathrm{Arity} \to \mathrm{Set}$$

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types
- Therefore, we work with stat in the type of the vectors rather than the sense types themselves to ensure this property
- Since all the sense types are in Set, our heterogeneous vectors need not be schematized over sorts:

$$\vdash \mathrm{HetVect} : \mathrm{Arity} \rightarrow \mathrm{Set}$$

$$\vdash \mathsf{hnil} : \mathrm{HetVect}\ \langle 0, [\ ]\rangle \qquad \text{(written } [\![\ ]\!]\text{)}$$

Heterogeneous Vectors (1/2)

- Note that the only kinds of heterogeneous vectors we're concerned with are those which only contain terms of our linguistic sense types
- Therefore, we work with stat in the type of the vectors rather than the sense types themselves to ensure this property
- Since all the sense types are in Set, our heterogeneous vectors need not be schematized over sorts:

$$\vdash \mathrm{HetVect} : \mathrm{Arity} \to \mathrm{Set}$$

$$\vdash \mathsf{hnil} : \mathrm{HetVect}\ \langle 0, [\,]\rangle \qquad \text{(written } [\![\,]\!]\text{)}$$

$$\vdash \mathsf{hcons} : \forall \langle n, v\rangle : \mathrm{Arity}.\forall s : \mathrm{stat}.$$
$$[\mathrm{Sns}\ s \to \mathrm{HetVect}\ \langle n, v\rangle \to \mathrm{HetVect}\ \langle S\ n, s :: v\rangle]$$
$$\text{(written infix as :::)}$$

Heterogeneous Vectors (2/2)

To ensure that our TUD stack is always a subvector of the DR vector, we define the subvector relation as follows:

Heterogeneous Vectors (2/2)

To ensure that our TUD stack is always a subvector of the DR vector, we define the subvector relation as follows:

$$\vdash (\sqsubseteq) : \forall \langle m, u\rangle, \langle n, v\rangle : \mathrm{Arity}.[\mathrm{HetVect}\ \langle m, u\rangle \to \mathrm{HetVect}\ \langle n, v\rangle \to \mathrm{Prop}]$$

Heterogeneous Vectors (2/2)

To ensure that our TUD stack is always a subvector of the DR vector, we define the subvector relation as follows:

$$\vdash (\sqsubseteq) : \forall \langle m, u \rangle, \langle n, v \rangle : \mathrm{Arity}.[\mathrm{HetVect}\ \langle m, u \rangle \rightarrow \mathrm{HetVect}\ \langle n, v \rangle \rightarrow \mathrm{Prop}]$$

$$\vdash \mathsf{hbot} : [\![\ ]\!] \sqsubseteq [\![\ ]\!]$$

Heterogeneous Vectors (2/2)

To ensure that our TUD stack is always a subvector of the DR vector, we define the subvector relation as follows:

$$\vdash (\sqsubseteq) : \forall \langle m, u\rangle, \langle n, v\rangle : \text{Arity}.[\text{HetVect } \langle m, u\rangle \rightarrow \text{HetVect } \langle n, v\rangle \rightarrow \text{Prop}]$$

$$\vdash \mathsf{hbot} : [\![\ ]\!] \sqsubseteq [\![\ ]\!]$$

$$\begin{aligned}\vdash \mathsf{hcon1} : \forall \langle m, u\rangle, \langle n, v\rangle : \text{Arity}.\forall h : \text{HetVect } \langle m, u\rangle. \\ \forall k : \text{HetVect } \langle n, v\rangle. \\ [h \sqsubseteq k \rightarrow \forall s : \text{stat}.\forall x : \text{Sns } s.[h \sqsubseteq (x ::: k)]]\end{aligned}$$

Heterogeneous Vectors (2/2)

To ensure that our TUD stack is always a subvector of the DR vector, we define the subvector relation as follows:

$$\vdash (\sqsubseteq) : \forall \langle m,u\rangle, \langle n,v\rangle : \text{Arity}.[\text{HetVect } \langle m,u\rangle \to \text{HetVect } \langle n,v\rangle \to \text{Prop}]$$

$$\vdash \mathsf{hbot} : [\![\ ]\!] \sqsubseteq [\![\ ]\!]$$

$$\begin{aligned}\vdash \mathsf{hcon1} : \forall \langle m,u\rangle, \langle n,v\rangle : \text{Arity}.\forall h : \text{HetVect } \langle m,u\rangle.\\ \forall k : \text{HetVect } \langle n,v\rangle.\\ [h \sqsubseteq k \to \forall s : \text{stat}.\forall x : \text{Sns } s.[h \sqsubseteq (x ::: k)]]\end{aligned}$$

$$\begin{aligned}\vdash \mathsf{hcon2} : \forall \langle m,u\rangle, \langle n,v\rangle : \text{Arity}.\forall h : \text{HetVect } \langle m,u\rangle.\\ \forall k : \text{HetVect } \langle n,v\rangle.\\ [h \sqsubseteq k \to \forall s : \text{stat}.\forall x : \text{Sns } s.[(x ::: h) \sqsubseteq (x ::: k)]]\end{aligned}$$

Coq

- Another advantage gained by switching over from HOL to CIC is the availability of Coq, a theorem prover written in (p)CIC

Coq

- Another advantage gained by switching over from HOL to CIC is the availability of Coq, a theorem prover written in (p)CIC
- Using Coq gives us the ability to write our theory out and ensure that it works

Coq

- Another advantage gained by switching over from HOL to CIC is the availability of Coq, a theorem prover written in (p)CIC
- Using Coq gives us the ability to write our theory out and ensure that it works
- The proof editing mode greatly simplifies the process of defining the functions of, and proving theorems about, CHS

Conclusion

- While HOL is the logical framework most familiar to many semanticists, it isn't powerful enough to be suited for the task of analyzing NL expressions

Conclusion

- While HOL is the logical framework most familiar to many semanticists, it isn't powerful enough to be suited for the task of analyzing NL expressions
- Moving to CIC allows us to work in a framework which is both similar enough to HOL that we can port what worked there to the new system, yet powerful enough to allow us to analyze a wider range of phenomena

References I

T. Coquand and G. Huet. The calculus of constructions. *Information and computation*, 76(2-3):95–120, 1988.

H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland, Amsterdam, 1958.

D. Gallin. *Intensional and Higher-order Modal Logic*. Amsterdam: North-Holland, 1975.

J. Ginzburg. An update semantics for dialogue. In H. Bunt, R. Muskens, and G. Rentier, editors, *The Tilburg International Workshop on Computational Semantics*. ITK, Tilburg, 1994.

References II

J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972.

L. Henkin. Completeness in the theory of types. *Symbolic Logics*, 15(2):81–91, 1950.

W. A. Howard. The formulae-as-types notion of construction. 1969.

S. Martin. *The dynamics of sense and implicature*. PhD dissertation, Ohio State University, 2013.

S. Martin. Supplemental update. *Semantics and pragmatics*, 9, 2016.

References III

S. Martin and C. Pollard. A dynamic categorial grammar. In G. Morrill, R. Muskens, R. Osswald, and F. Richter, editors, *Formal Grammar 2014, LNCS 8612*, pages 138–154. Springer-Verlag Berlin, 2014.

A. Plummer and C. Pollard. Agnostic possible world semantics. In *Logical Aspects of Computational Linguistics*, volume Lecture Notes in Computer Science 7351, pages 201–212. Springer, 2012.

C. Pollard. Hyperintensions. *Journal of Logic And Computation*, 18(2):257–282, 2008.

C. Pollard. Agnostic hyperintensional semantics. *Synthese*, 192: 535–562, 2015.

References IV

C. Roberts. Information structure: towards an integrated formal theory of pragmatics. In J. H. Yoon and A. Kathol, editors, *Ohio State University Working Papers in Linguistics (OSUWPL)*, volume 49, pages 91–136. Ohio State University, Department of Linguistics. Reprinted with a new Afterword in *Semantics and Pragmatics* volume 5, 2012, 1996/2012.

G. Sundholm. Proof theory and meaning. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 471-506. Dordrecht: Reidel, 1986.

R. H. Thomason. A model theory for propositional attitudes. *Linguistics and Philosophy*, 4:47–70, 1980.

M. Yasavul. *Questions and answers in K'iche'*. PhD thesis, Ohio State University, in progress.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.
- in terms of models: the interpretation of a formula is just its truth value.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.
- in terms of models: the interpretation of a formula is just its truth value.
- But in CIC, this is no longer the case: formulas are 'more intensional' than in classical extensional HOL.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.
- in terms of models: the interpretation of a formula is just its truth value.
- But in CIC, this is no longer the case: formulas are 'more intensional' than in classical extensional HOL.
- In fact, they are themselves **types**.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.
- in terms of models: the interpretation of a formula is just its truth value.
- But in CIC, this is no longer the case: formulas are 'more intensional' than in classical extensional HOL.
- In fact, they are themselves **types**.
- This fact gives rise to the option of using formulas not only to express the theory, but also as the translations of NL declarative sentences.

A Quandary about Formulas (1/2)

- As in MS/Ty2, HHS makes no distinction between the type for formulas and the type of truth values.
- in terms of models: the interpretation of a formula is just its truth value.
- But in CIC, this is no longer the case: formulas are 'more intensional' than in classical extensional HOL.
- In fact, they are themselves **types**.
- This fact gives rise to the option of using formulas not only to express the theory, but also as the translations of NL declarative sentences.
- And most researchers using CIC for NL semantics exercise that option.

A Quandary about Formulas (2/2)

- This approach is a version of Sundholm's (1986) idea of analyzing NL sentence meanings as types.

A Quandary about Formulas (2/2)

- This approach is a version of Sundholm's (1986) idea of analyzing NL sentence meanings as types.
- It can be seen as an extension to NL of Howard's (1980) treatment of formulas of Heyting arithmetic as types.

A Quandary about Formulas (2/2)

- This approach is a version of Sundholm's (1986) idea of analyzing NL sentence meanings as types.
- It can be seen as an extension to NL of Howard's (1980) treatment of formulas of Heyting arithmetic as types.
- And it is this idea, not an indifference to models, that really distinguishes proof-theoretic semantics from model-theoretic semantics.

A Quandary about Formulas (2/2)

- This approach is a version of Sundholm's (1986) idea of analyzing NL sentence meanings as types.
- It can be seen as an extension to NL of Howard's (1980) treatment of formulas of Heyting arithmetic as types.
- And it is this idea, not an indifference to models, that really distinguishes proof-theoretic semantics from model-theoretic semantics.
- Should CHS follow suit?

The Inductive Type nat (1/2)

- The Set nat of the natural numbers is defined in the usual way:

The Inductive Type nat (1/2)

- The Set nat of the natural numbers is defined in the usual way:

  $\vdash \mathrm{nat} : \mathrm{Set}$

The Inductive Type nat (1/2)

- The Set nat of the natural numbers is defined in the usual way:

  $\vdash \mathrm{nat} : \mathrm{Set}$

  $\vdash 0 : \mathrm{nat}$

The Inductive Type nat (1/2)

- The Set nat of the natural numbers is defined in the usual way:

$$\vdash \mathrm{nat} : \mathrm{Set}$$
$$\vdash 0 : \mathrm{nat}$$
$$\vdash S : \mathrm{nat} \rightarrow \mathrm{nat}$$

The Inductive Type nat (1/2)

- The Set nat of the natural numbers is defined in the usual way:

  $\vdash \mathrm{nat} : \mathrm{Set}$
  $\vdash 0 : \mathrm{nat}$
  $\vdash S : \mathrm{nat} \to \mathrm{nat}$

- That is, a fully normalized term of type nat will be either the nullary constant 0 or the unary constructor S applied to another term $i : \mathrm{nat}$

The Inductive Type nat (2/2)

- nat also comes with three elimination schemes—$\mathsf{nat\_rect_i}$, $\mathsf{nat\_rec}$, and $\mathsf{nat\_ind}$ for
  eliminating into the sorts $\mathrm{Type_i}$, Set, and Prop, respectively:
  $\mathsf{nat\_ind} : \forall P : \mathrm{nat} \to \mathrm{Prop}.[(P\ 0) \to \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)] \to \forall n : \mathrm{nat}.[P\ n]]$

The Inductive Type nat (2/2)

- nat also comes with three elimination schemes—$\mathsf{nat\_rect_i}$, $\mathsf{nat\_rec}$, and $\mathsf{nat\_ind}$ for
  eliminating into the sorts $\mathrm{Type_i}$, Set, and Prop, respectively:
  $\mathsf{nat\_ind} : \forall P : \mathrm{nat} \to \mathrm{Prop}.[(P\ 0) \to \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)] \to \forall n : \mathrm{nat}.[P\ n]]$
- These destructors reduce eliminate in the expected way—
  ie for any $P : \mathrm{nat} \to \mathrm{Prop}$, $b : (P\ 0)$,
  $f : \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)]$, $n : \mathrm{nat}$:

The Inductive Type nat (2/2)

- nat also comes with three elimination schemes—$\mathsf{nat\_rect_i}$, $\mathsf{nat\_rec}$, and $\mathsf{nat\_ind}$ for
  eliminating into the sorts $\mathrm{Type_i}$, Set, and Prop, respectively:
  $\mathsf{nat\_ind} : \forall P : \mathrm{nat} \to \mathrm{Prop}.[(P\ 0) \to \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)] \to \forall n : \mathrm{nat}.[P\ n]]$
- These destructors reduce eliminate in the expected way—
  ie for any $P : \mathrm{nat} \to \mathrm{Prop}$, $b : (P\ 0)$,
  $f : \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)]$, $n : \mathrm{nat}$:
  - $(\mathsf{nat\_ind}\ P\ b\ f\ 0) \Rightarrow 0$

The Inductive Type nat (2/2)

- nat also comes with three elimination schemes—$\mathsf{nat\_rect_i}$, $\mathsf{nat\_rec}$, and $\mathsf{nat\_ind}$ for
  eliminating into the sorts $\mathrm{Type_i}$, Set, and Prop, respectively:
  $\mathsf{nat\_ind} : \forall P : \mathrm{nat} \to \mathrm{Prop}.[(P\ 0) \to \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)] \to \forall n : \mathrm{nat}.[P\ n]]$
- These destructors reduce eliminate in the expected way—
  ie for any $P : \mathrm{nat} \to \mathrm{Prop}$, $b : (P\ 0)$,
  $f : \forall i : \mathrm{nat}.[P\ i\ \to\ P\ (S\ i)]$, $n : \mathrm{nat}$:
  - $(\mathsf{nat\_ind}\ P\ b\ f\ 0) \Rightarrow 0$
  - $(\mathsf{nat\_ind}\ P\ b\ f\ (S\ n)) \Rightarrow (f\ n\ (\mathsf{nat\_ind}\ P\ b\ f\ n))$