Coq for natural language semantics day 3: FraCoq and some other goodies[1]

Stergios Chatzikyriakidis

November 24, 2017

[1]Parts of this talk are based on Jean-Philippe Bernardy's tutorial at the GF summer school, Riga 2017

Brief summary of the talk

- Present a type-theoretical framework for formal semantics leveraging two well-studied tools
  - Grammatical Framework (GF, [Ranta(2011)])
  - Coq
- Providing a compositional resource semantics for GF
  - A tutorial on FraCoq
- Evaluation on the FraCaS test suite
- State-of-the-art results
- Other goodies
  - Some Type Theory with Records (TTR)
  - Some Montagovian Generative Lexicon
  - Some neo-Davidsonian semantics
  - Co-predication and individuation

CLASP centre for linguistic theory and studies in probability

Structure the talk

- Brief intro to the systems used, GF and Coq
- Presenting the FraCoq system
  - We concentrate on the most linguistically relevant features and also the features relevant for the FraCaS
- Evaluation against the FraCaS test suite
  - Some brief remarks about the FraCaS and NLI platforms
    - Results and comparison with previous approaches
    - The issue of automation
- Conclusions and Future work

CLASP centre for linguistic theory and studies in probability

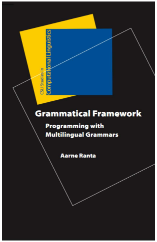
Background: Grammatical Framework plus a proof-assistant

- Grammatical Framework (GF, [Ranta(2011)])

CLASP centre for linguistic theory and studies in probability

Background: Grammatical Framework plus a proof-assistant

- Grammatical Framework (GF, [Ranta(2011)])
  - Chalmers based
  - Programming language for multilingual applications



CLASP centre for linguistic theory and studies in probability

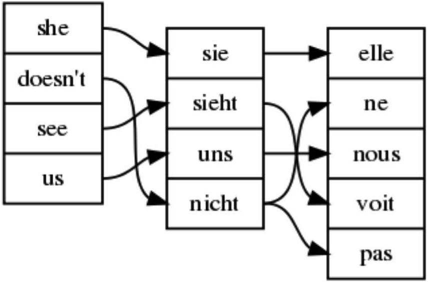
Background: Grammatical Framework (GF)

- Involves an abstract syntax, comprised of:
  - A number of syntactic categories
  - A number of syntactic construction functions, which provide the means to compose basic syntactic categories into more complex ones
    - ★ $AdjCN{:}AP \rightarrow CN \rightarrow CN$ (appending an adjectival phrase to a common noun and obtaining a new common noun)
- GF comes with a library of mappings from abstract syntax to concrete
  - These mappings can be inverted by GF, thus offering parsers from natural text into abstract syntax
  - We use the parse trees constructed by [Ljungloöf and Siverbo(2011)] thereby avoiding any syntactic ambiguity (GF FraCaS treebank).

CLASP centre for linguistic theory and studies in probability

Background: Grammatical Framework

- One abstract syntax (set of rules), lots of concrete ones (linearizations of the rules)



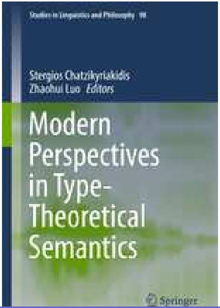
Background: Grammatical Framework

- Precise symbolic parser
  - Mildly context-sensitive expressive power (equivalent to a parallel multiple CFG)
    - At least a version of GF (there is also unrestricted GF)
    - Have a look at Ljunglöf's thesis [Ljunglöf(2004)] if you want to know more

CLASP centre for linguistic theory and studies in probability

Background: Type-Theoretical Semantics (you've heard all this before!)

- We use the type of logics that have been traditionally dubbed as constructive
  - Initiated by the work of Martin-Löf [Martin-Löf(1975), **?**]
  - In linguistics this types of logics go back to Ranta's seminal work [Ranta(1994)] or even earlier to [Sundholm(1986)]
    - ★ More recent approaches can be found as well. Please see [Chatzikyriakidis and Luo(2017)] for a collection of papers on constructive type theories for NL semantics



CLASP centre for linguistic theory and studies in probability

Background: Type Theoretical Semantics

- Main features of MTTs
  - Type many-sortedness.
  - Dependent sum and product types
    - Σ-types, often written $\sum_{x:A} B[x]$ and which have product types $A \times B$ as a special case when B does not depend on x.
    - Dependent product, Π-types, often written $(\prod_{x:A} B[x])$, and which have arrow-types $A \to B$ as a special case
    - They generalize universal quantification and function types and they offer type polymorphism
  - Proof-theoretical specification and support for effective reasoning.
    - Most powerful proof assistants implement MTTs (e.g. Coq, Agda)

Background: Coq (some more repetition!)

- Proof assistant based on the calculus of inductive constructions (extension of CoC, see [Paulin-Mohring(2015)])
  - Arguably one of the leading proof assistants
    - a proof of the four-color theorem [Gonthier(2008)]
    - a proof of the odd order theorem [Gonthier et al.(2013)Gonthier, Asperti, Avigad, Bertot, Cohen, Garillot, L
    - developing CompCert, a formally verified compiler for C [Leroy(2013)]
    - One of the assistants used in the Univalent Foundations project (Homotopy Type Theory, [Program(2013)])

Background: Coq

- Important features used
  - Π types
    - in Coq: $\prod_{x:A} B[x]$ is written `forall (x:A), B` or (simply `A` $\to$ `B` when `B` does not depend on `x`)
  - Record types
    - Generalization of Σ-types and are encoded as (trivial) inductive types with a single constructor.
    - $\Sigma x{:}A.B(x)$ can be expressed as a dependent record type in Coq:

```
Record AB:Type:=mkAB{x:> A;P:B x}
```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- We use Ljünglof's FraCaS treebank and take these trees to their semantic counterparts
- The structure of the semantic representation
  1. Every GF syntactic category C is mapped to a Coq *Set*, noted $[\![C]\!]$.
  2. GF Functional types are mapped compositionally : $[\![A \to B]\!] = [\![A]\!] \to [\![B]\!]$
  3. Every GF syntactic construction function $f{:}X$ is mapped to a function $[\![f]\!]$ such that $[\![f]\!] : [\![X]\!]$.
  4. GF function applications are mapped compositionally: $[\![t(u)]\!] = [\![t]\!]([\![u]\!])$.

CLASP centre for linguistic theory and studies in probability

From GF to Coq in practice

- Note the following toy GF grammar (taken from Bernardy's tutorial on the GF summer school, August 2017):

```
abstract Grammar = {
flags startcat = S ;
cat
S ; Cl ; NP ; VP ; AP ; CN ; PN ; Det ; N ; A ; V ; V2 ;
AdA ;  Pol ; Conj ;
data
UseCl   : Pol -> Cl -> S ; PredVP  : NP -> VP -> Cl ;
ComplV2 : V2 -> NP -> VP ; DetCN   : Det -> CN -> NP ;
ModCN   : AP -> CN -> CN ; CompAP  : AP -> VP ;
AdAP    : AdA -> AP -> AP ; ConjS   : Conj -> S  -> S  ->
ConjNP  : Conj -> NP -> NP -> NP ; UseV     : V -> VP ;
UsePN   : PN -> NP ; UseN     : N -> CN ;
UseA    : A -> AP ; some_Det, every_Det : Det ;
i_NP, you_NP : NP ; very_AdA : AdA ;
Pos, Neg : Pol ; and_Conj, or_Conj : Conj ;
```

CLASP centre for linguistic theory and studies in probability

From GF to Coq in practice

```
}
abstract Test = Grammar ** {

fun
man_N, woman_N, house_N, tree_N : N ;
big_A, small_A, green_A : A ;
walk_V, arrive_V : V ;
love_V2, please_V2 : V2 ;
john_PN, mary_PN : PN;

} ;
```

CLASP centre for linguistic theory and studies in probability

From GF to Coq in practice

- A trivial way of doing this

```
Parameter S: Type. Parameter Cl: Type. Parameter VP: Type.
Parameter PN: Type. Parameter NP: Type. Parameter AP: Type.
Parameter A: Type. Parameter CN: Type. Parameter Det: Type.
Parameter N: Type. Parameter V: Type. Parameter V2: Type.
Parameter AdA: Type. Parameter Pol: Type. Parameter Conj: Type
...
Parameter man_N:  N. Parameter woman_N: N .
Parameter house_N:  N. Parameter tree_N: N .
...
Parameter john_PN: PN . Parameter mary_PN: PN.
```

CLASP centre for linguistic theory and studies in probability

From GF to Coq in practice

- This is of course quite trivial but, now, for every abstract syntax expression in GF, there is a well-typed expression in Coq
  - You can even do some reasoning :)

```
Theorem thm0 : UseCl Pos (PredVP (UsePN john_PN) walk_V) ->
UseCl Pos (PredVP (UsePN john_PN) walk_V).
intro H. exact H. Qed.
```

CLASP centre for linguistic theory and studies in probability

From GF to Coq in practice

- Getting more proper semantics step by step

```
Definition S     : Type := Prop .
Definition Cl    : Type := Prop .
Definition Pol   : Type := Prop -> Prop .
Definition Pos : Pol := fun p => p.
Definition Neg : Pol := fun p => not p.
Definition UseCl : Pol -> Cl -> S :=
fun pol c => pol c.Definition S     : Type := Prop .
```

- We can get clausal negation yay!

From GF to Coq in practice

- Quantifiers, PN and VP

```
Parameter object : Type.
Definition VP    : Type :== object -> Prop.
Definition V     : Type := object -> Prop.
Definition UseV  : V -> VP := fun v => v.
Definition PredVP  : NP -> VP -> Cl := fun np vp => vp np.
Definition NP: Type := VP -> Prop .
Definition UsePN: PN -> NP := fun pn vp => vp pn.
Definition PredVP: NP -> VP -> Cl := fun np vp => np vp.
Definition everyoneNP : NP := fun vp => forall x, vp x.
```

- Getting better, we can do some elementary quantification now!

From GF to Coq in practice

- Introducing quantifiers like *all* and *some* needs a definition of quantifier
  - This we do with *Det*, a function from *CNs* to *NPs*.
  - We define *all* and *some*

```
Definition CN    : Type := PN -> Prop .
Definition N    : Type := CN .
Definition Det    : Type := CN -> NP .
Definition DetCN : Det -> CN -> NP := fun det cn => det cn.
Definition every_Det : Det := fun cn vp =>
forall x, cn x -> vp x.
Definition some_Det : Det := fun cn vp =>
exists x, cn x /\ vp x.
```

  - You get the picture!
  - Let us now get fine-grained and see FraCoq!

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Sentences
  - As we said, we interpret sentences as propositions: $[\![S]\!] = Prop$.
  - To verify that P entails H, we prove the proposition $[\![P]\!] \rightarrow [\![H]\!]$.

  ```
  Definition S := Prop.
  ```

- Common Nouns
  - Predicates over an abstract object type

  ```
  Parameter object : Set.
  Definition CN := object->Prop.
  ```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Verb phrases
  - Parameterize over the *noun* of the subject (using Π types)

```
Definition VP := forall (subjectClass : CN)
object -> Prop.
```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Adjectives
  - Functions from cn to cn (predicates to predicates)

    ```
    Definition A := CN -> CN.
    ```

  - Different classes of adjectives are captured using coercions (subtyping). All special classes of adjectives are subtypes of A.

    ```
    Definition IntersectiveA := object -> Prop.
    Definition wkIntersectiveA : IntersectiveA -> A
    := fun a cn (x:object) => a x /\ cn x.
    Coercion wkIntersectiveA : IntersectiveA >-> A.
    ```

  - Provision is made for intersective, subsective, privative and non-committal adjectives
- intersective adjectives are then declared as follows:

```
Parameter green_A : IntersectiveA.
```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Another example: privative adjectives

```
Inductive PrivativeA : Type :=
mkPrivativeA : ((object -> Prop) -> (object -> Prop)) ->
PrivativeA.
Definition wkPrivativeA : PrivativeA -> A
:= fun aa cn (x:object) => let (a) :=
aa in a cn x /\ not (cn x).
Coercion wkPrivativeA : PrivativeA >-> A.
Definition NonCommitalA := A.
```

- Again adjectives are declared as: $fake_N$:$Privative_A$
- For non commitals, it suffices to declare them as A.

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Adverbs
  - Similar method to adjectives but instead the modification is on verbal predicates
  - The adverb cases in the FraCaS are all veridical and covariant.
  - We define such a subclass *VeridicalAdv* and declare it as a coercion *Adv*
    - ★ Adverbs of type *VeridicalAdv* are also of type *Adv*

```
Definition VeridicalAdv :=
{ adv : (object -> Prop) -> (object -> Prop)
& prod (forall (x : object) (v : object -> Prop), (adv v) x
 -> v x) (forall (v w : object -> Prop),
(forall x, v x -> w x) -> forall (x : object),
adv v x -> adv w x)
}.
Definition WkVeridical : VeridicalAdv -> Adv
:= fun adv => projT1 adv.
Coercion WkVeridical : VeridicalAdv >-> Adv.
Parameter on_time_Adv : VeridicalAdv .
```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Noun Phrases and Predeterminers
  - A clean definition of NPs as functions from predicates to truth values will not work
    - ★ Problem with GF's abstract syntax: existence of pre-determiners which include cases like *most*, *all* among others and are defined as functions from NPs to NPs
    - ★ In general, the category includes elements that are naturally interpreted as GQs
    - ★ Solution: Remember the components of NPs (number, quantifier and common noun)
    - ★ Pre-determiners then are able to substistute the quantifier part with the appropriate quantifier
    - ★ This has to be done, otherwise pre-determiners introduce a dummy indefinite in these cases

The FraCoq system

- Predeterminers

```
Definition all_Predet : Predet
:= fun np => let (num,qIGNORED,cn) := np
in mkNP num all_Quant cn.

Definition at_least_Predet : Predet
:= fun np => let (num,qIGNORED,cn) := np
in mkNP num (fun num cn vp => interpAtLeast num (CARD
(fun x => cn x /\ vp cn x))) cn.
```

- For *all_Predet* the number is ignored and the quantifier part is substituted with a universal

The FraCoq system

- The function *interpAtLeast* is an interpretation function of number for *at least*

```
Fixpoint interpAtLeast (num:Num) (x:nat) :=
match num with
| singular => x >= 1
| plural   => x >= SOME
| unknownNum => True
| moreThan n => interpAtLeast n x
| cardinal n => x >= n
end.
```

- Thus, in the case of *at least 3* what we would get is a situation where the cardinality of x is equal or more than 3.
- CARD a context-dependent abstract function which turns a predicate into a natural number is used to get the correct semantics
- Other predeterminers are determined accordingly

centre for linguistic theory and studies in probability

The FraCoq system

- Generalized Quantifiers
  - They turn a number and a common noun into a noun-phrase (which we call *NP*0).

    ```
    Definition Quant := Num -> CN -> NP0.
    ```

  - Some quantifiers ignore the number and are given usual definitions (e.g. *some* or *all*), whereas others make essential use of number (e.g. *at most*)
    - ★ In the latter case, the function `CARD` is used

The FraCoq system

- Quantifiers *some*

```
Definition someSg_Det : Det:= (singular, fun num P Q=>
exists x,  P x /\ Q P x ).
```

- Number ignored here

CLASP centre for linguistic theory and studies in probability

The FraCoq system: Quantifiers

- Quantifier *at most*

  ```
  Definition atMost_quant : Quant
  := fun num cn vp => interpAtMost num
  (CARD (fun x => cn x /\ vp cn x))
  ```

- Essential use of number
- *interpAtMost* checks that the given number is less than the given cardinality

The FraCoq system: Cardinalities

- CARD is a context-dependent abstract function which turns a predicate into a natural number.
  - Common-sense axioms of set cardinality, such as monotonicity are provided

    ```
    Variable CARD_monotonous : forall a b:CN,
    (forall x, a x -> b x) -> CARD a <= CARD b.
    ```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- The definite article
  - Checks for plural noun phrases
    - ★ If found, then universal quantification
    - ★ If not, it looks up the object of discourse in an abstract *environment*, which is a function which turns a common noun into an object

```
Definition DefArt:Quant:= fun (num : Num) (P:CN)=> fun Q:VP
=> match num with plural => (forall x, P x -> Q P x)
/\ Q P (environment P) /\ P (environment P) |
_ => Q P (environment P) /\ P (environment P) end.
```

CLASP centre for linguistic theory and studies in probability

The FraCoq system

- Prepositions
  - Takes a NP and returns a higher order predicate (1)
  - Covariant and veridical (2 and 3)

```
Definition NP1 := (object -> Prop) ->Prop.
Inductive Prep : Type :=
mkPrep : forall
(prep : NP1 -> (object -> Prop) -> (object -> Prop)), (*1*)
(forall (prepArg : NP1) (v : object -> Prop)
(subject : object), prep prepArg v subject -> v subject) ->
(forall (prepArg : NP1) (v w : object -> Prop), (*2*)
(forall x, v x -> w x) -> forall x, prep prepArg v x ->
prep prepArg w x)  -> Prep. (*3*)
```

CLASP centre for linguistic theory and studies in probability

Evaluation: The FraCaS test suite

- A test suite for NLI
  [Cooper et al.(1996)Cooper, Crouch, van Eijck, Fox, van Genabith, Jasp
  - 346 NLI examples in the form of one or more premises followed by a question along with an answer to that question
  - Three potential answers
    - YES: The declarative sentence formed out of the question follows from the premises
    - NO: The declarative sentence does not follow from the premises
    - UNK: The declarative sentence neither follows nor does not follow fro the premises

CLASP centre for linguistic theory and studies in probability

Evaluation: The FraCaS test suite

(1) A Swede won the Nobel Prize.
Every Swede is Scandinavian.
Did a Scandinavian win the Nobel prize? [Yes, FraCas 049]

(2) No delegate finished the report on time..
Did any Scandinavian delegate finish the report on time? [No, FraCaS 070]

(3) A Scandinavian won the Nobel Prize.
Every Swede is Scandinavian.
Did a Swede win the Nobel prize? [UNK, FraCaS 065]

Evaluation: The FraCaS test suite

- The FraCaS has considerable weaknesses
  - Small size
  - Artificial nature of the examples
- However, it covers a lot of phenomena associated with NLI
- It is still a very good suite to test logical approaches
  - And it is actually the one (or one of the suites) used in these approaches!

Evaluation

- Evaluation against 5 sections of the FraCaS
  - Total of 174 examples
  - Excluded sections where a lot of context-dependency has to be taken into consideration (e.g. the section on ellipsis)
    - Note that no one has ever made a full run of the suite
- YES: a proof can be constructed from the premises to the hypothesis
- NO: a proof of the negated hypothesis can be constructed
- UNK: otherwise

Evaluation

- The following table presents the results (Ours) as well as a comparison with the approach in Mineshima et al. [Mineshima et al.(2015)Mineshima, Martınez-Gómez, Miyao, and Bekki] Bos [Bos(2008)] and Abzianidze [Abzianidze()]

	Section	# examples	Ours	MINE	Nut	Langpro
1	Quantifiers	75	.96	.77	.53	.93 (44)
2	Plurals	33	.76	.67	.52	.73 (24)
3	Adjectives	22	.95	.68	.32	.73 (12)
4	Comparatives	31	.56	.48	.45	-
5	Attitudes	13	.85	.77	.46	.92 (9)
6	Total	174 (181)	0.83	0.69	0.50	0.85

- Our approach outerperforms Mineshema et al. by 13 percentage points.
- The approach by Abzianidze has an accuracy of 0.85 without involving the comparative section. If this section is taken out, our system's accuracy rises to 0.88

centre for linguistic theory and studies in probability

Error Analysis

- Improvement over earlier approaches. Still, there were a couple of difficulties
  - Comparatives: cases that needed one to provide adequate semantics for *more* but also to take care of ellipsis

(4) ITEL won more orders than APCOM.
ITEL won some orders.
Did ITEL win some orders? [Yes, FraCaS 233]

  - Definite Plurals: Universal reading was captured. Cases of existential readings were not

(5) The inhabitants of Cambridge voted for a Labour MP.
very inhabitant of Cambridge voted for a Labour MP.
Did every inhabitant of Cambridge vote for a Labour MP?
[UNK, FraCaS 094]

CLASP centre for linguistic theory and studies in probability

Automation

- So far, our proofs are not automated
  - A couple of steps (usually very few) to reach a proof
  - Earlier approaches using Coq (e.g. Chatzikyriakidis 2014 and Mineshima et al. 2015) use Coq's tactical language LTac to define automated macros of actions
    - ★ This is not difficult to do in our case as well
    - ★ Just go through all the proof tactics or observe the tactics that are used in the proofs to create a macro that will automate the proofs
    - ★ The question remains: can that macro of tactics generalize outside the suite?
    - ★ Answer: only to a limited extent, i.e. when exactly the same set of tactics yields a proof
    - ★ For this reason, we have not automated proof search to obtain the results presented in this paper, even though this can be done easily

CLASP centre for linguistic theory and studies in probability

Automation

- Automating would also make an unprincipled use of higher-order logic (HOL)
  - No algorithm which can decide if a proposition has a proof or not
    - We must use heuristics both to search for proofs and to decide when to give up searching
- Most problems have either obvious proofs or obviously lack a proof (fortunately)
  - Due to its heuristic nature the proof search necessarily contains a human component
    - Problematic to make a statement about the suitability of FraCoq outside FraCas
    - Small dataset and lack of separation between a development and a test set does not help the situation either
    - Related shortcoming: specialized semantics for specific lexical entries

CLASP centre for linguistic theory and studies in probability

Future Work

- Address the issue of automation
  - Define a decidable fragment of the logic and only work within such fragment
    - Possible to concisely characterize how the approach generalises
  - Train a neural network on a body of freely available proofs on the net and see whether it can generalize to automatically provide the proof tactics for the cases interested
- Improvement at the GF level: make the abstract syntax more compatible with compositional semantics
  - For example, do something about problematic syntactic categories like pre-determiners or cases where the syntax makes it impossible to recover elliptical fragments
- Extend into the whole suite (first attempt to do anaphora using monads on the way!)

CLASP centre for linguistic theory and studies in probability

Conclusions

- We have connected two well-defined systems based on type-theory
  - GF and Coq
  - Providing resource semantics for GF
- The issue of generalization remains a shortcoming
  - It is possible to achieve very precise semantics for specific domains
    - Our system outerperforms previous logical systems w.r.t accuracy
- Useful in performing inference tasks on controlled natural language domains
- Hybrid NLI systems

CLASP centre for linguistic theory and studies in probability

Conclusions

- The system can be found here: https://github.com/GU-CLASP/FraCoq

CLASP centre for linguistic theory and studies in probability

Lasha Abzianidze.
A tableau prover for natural logic and language.

Johan Bos.
Wide-coverage semantic analysis with boxer.
In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 277–286. Association for Computational Linguistics, 2008.

Stergios Chatzikyriakidis and Zhaohui Luo.
Modern Perspectives in Type-Theoretical Semantics.
Springer Publishing Company, Incorporated, 1st edition, 2017.
ISBN 3319504207, 9783319504209.

R. Cooper, D. Crouch, J. van Eijck, C. Fox, J. van Genabith, J. Jaspars, H. Kamp, D. Milward, M. Pinkal, M. Poesio, and S. Pulman.
Using the framework.
Technical Report LRE 62-051r, 1996.
http://www.cogsci.ed.ac.uk/ fracas/.

CLASP centre for linguistic theory and studies in probability

Georges Gonthier.
Formal proof–the four-color theorem.
Notices of the AMS, 55(11):1382–1393, 2008.

Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell OConnor, Sidi Ould Biha, et al.
A machine-checked proof of the odd order theorem.
In *Interactive Theorem Proving*, pages 163–179. Springer, 2013.

X. Leroy.
The compcert c verified compiler: Documentation and users manual.
http://compcert.inria.fr/man/manual.pdf, 2013.

Peter Ljunglöf.
Expressivity and complexity of the grammatical framework.
2004.

P. Ljungloöf and M. Siverbo.
A bilingual treebank for the FraCas test suite.
Clt project report, University of Gothenburg, 2011.

P. Martin-Löf.
An intuitionistic theory of types: predicative part.
In H.Rose and J.C.Shepherdson, editors, *Logic Colloquium'73*, 1975.

Koji Mineshima, Pascual Martınez-Gómez, Yusuke Miyao, and Daisuke Bekki.
Higher-order logical inference with compositional semantics.
In *Proceedings of EMNLP 2015*, 2015.

Christine Paulin-Mohring.
Introduction to the calculus of inductive constructions, 2015.

The Univalent Foundations Program.
Homotopy type theory: Univalent foundations of mathematics.
Technical report, Institute for Advanced Study, 2013.

A. Ranta.
Type-Theoretical Grammar.
Oxford University Press, 1994.

A. Ranta.

CLASP centre for linguistic theory and studies in probability

Grammatical Framework: Programming with Multilingual Grammar.
CSLI Publications, 2011.

G. Sundholm.
Proof theory and meaning.
In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic III: Alternatives to Classical Logic*, pages 471–506. Reidel, 1986.

CLASP centre for linguistic theory and studies in probability