Some Recent Logical Frameworks

Randy Pollack

LFCS, University of Edinburgh

Version of September 13, 2010

Outline

Outline

Reminder by Example: Syntax of FOL

```
i    : type.                  % Individuals
zero : i.
suc  : i -> i.
plus : i -> i -> i.

o      : type.                % Propositions
imp    : o -> o -> o.
and    : o -> o -> o.
or     : o -> o -> o.
forall : (i -> o) -> o.
eq     : i -> i -> o.
```

Framework handles binding

Reminder by Example: Some Natural Deduction Rules

$$\frac{\begin{array}{c}[A]\\ B\end{array}}{A \Rightarrow B} \qquad \frac{A \Rightarrow B \quad A}{B} \qquad \frac{A}{A \vee B} \qquad \frac{A \vee B \quad \begin{array}{c}[A]\\ C\end{array} \quad \begin{array}{c}[B]\\ C\end{array}}{C}$$

```
prf  : o -> type.

impi : {A, B : o} (prf A -> prf B) -> prf (A imp B).
impe : prf (A imp B) -> prf A -> prf B.
ori1 : prf A -> prf (A or B).
ore  : prf (A or B) ->
       (prf A -> prf C) -> (prf B -> prf C) ->
       prf C.
```

Framework handles hypothetical and schematic judgements.

Reminder by Example: Some more rules

$$\frac{A(x)}{\forall x.A}\,(x \text{ fresh}) \qquad \frac{\forall x.A}{A(t)} \qquad \frac{A \Rightarrow B \qquad \begin{bmatrix} [A] \\ B \end{bmatrix} \\ C}{C}$$

```
foralli : ({x:i} prf (A x)) -> prf (forall A).
foralle : prf (forall A) -> {T:i} prf (A T).
-- A Schroeder-Heister version of imp elim
SH_impe : prf (A imp B) ->
          ((prf A -> prf B) -> prf C) ->
          ---------------------------
          prf C
```

Framework handles freshness and substitution.

The SH rule is clear in the framework (it is third order).

Remark on Canonical Forms

- We want to have a compositional isomorphism between object language expressions and *canonical* forms of the LF.
- In order for the framework to handle substitution, we identify meta-terms up to β.
- But β is not enough, we need η-equivalence in the notion of *canonical*.
- Example: the two well-typed LF terms

  ```
  suc                [x:i](suc x)
  ```

  represent the same object expression (a function symbol of the language of FOL).

Some Problems with ELF: η equivalence

- Problem η reduction of raw ELF terms is not Church–Rosser: for $x \notin B$

$$[x : A]C \quad \overset{\beta}{\longleftarrow} \quad [x : A](([x : B]C)x) \quad \overset{\eta}{\longrightarrow} \quad [x : B]C$$

  where A and B may be anything.
- Well typed terms are $\beta\eta$-Church–Rosser.
- But equality becomes mutually recursive with the typing relation.
- Thus, the meta theory of $\beta\eta$-equivalence for the ELF language (with type tags) is complicated.
  - See [Geu93], [Sal], [HP05], [Gog05].
- Further, many extensions (singleton types, dependent pairs, linear types, ...) interact badly with η.

We want equality of raw terms to be decidable.

Some Problems with ELF: Too many terms are typed

- The correct ELF terms

  ```
  plus zero        eq zero        imp A
  ```

  represent no object language term or proposition at all.

Some Problems with ELF: Too many terms are typed

- The correct ELF terms

  `plus zero` `eq zero` `imp A`

  represent no object language term or proposition at all.

We consider some LFs that address these problems.

Two related ideas:

- Canonical Syntax: only canonical terms are syntactically well-formed.
- Hereditary Substitution: substitution propogates
  - since redexes cannot be formed for syntactic reasons.

Outline

Plotkin's DMBEL: Syntax

identifiers	S (type constants), F (term constants), x (term variables), ϕ (abstraction variables)
types	$\sigma, \tau \ ::= S(\overrightarrow{A})$
abstraction types	$\alpha, \beta \ ::= (\Gamma)\sigma$
terms	$t, u \ ::= x \mid F(\overrightarrow{A}) \mid \phi(\overrightarrow{t})$
abstraction terms	$A, B \ ::= (\Gamma)t$
contexts	$\Gamma, \Delta \ ::= \bullet \mid \Gamma, x{:}\sigma$
abstraction contexts	$\Phi, \Psi \ ::= \bullet \mid \Phi, \phi{:}\alpha$
signatures	$\Sigma \ ::= \bullet \mid \Sigma, S(\Phi) \mid \Sigma, F(\Phi){:}\sigma$

- Typechecking: abstraction and application arities match.
- No β or η redexes.
  - Only constants and variables are applied ...
  - ... and they are fully applied.
- Equality is identity of raw expressions.

Example in Plotkin's DMBEL: First order logic

Concrete syntax: type constants start with `*`
term constants start with `@`
abstraction variables start with a capital, `X`
term variables start with lower case, `x`

```
*i,                              -- type of individuals
@zero     : *i,          -- order 0
@suc(*i)  : *i,          -- order 1
*o,                              -- type of propositions
@imp(*o, *o)  : *o,      -- order 1
@all((*i)*o)  : *o,      -- order 2
@eq(*i, *i)   : *o       -- order 1
```

- Partially applied terms, `@eq(@zero)`, are not well-typed

Example in Plotkin's DMBEL: First order rules

```
*prf(*o),
@impI(X Y:*o, (*prf(X))*prf(Y)) : *prf(@imp(X, Y)),
@impE(X Y:*o, *prf(@imp(X, Y)), *prf(X)) : *prf(Y),
@allI(P:(*i)*o, (x:*i)*prf(P(x))) :
                          *prf(@all((x:*i)P(x))),
               -- note eta expansion ^^^^^^^^^^^
@allE(P:(*i)*o, X:*i, *prf(@all((x:*i)P(x))) :
                          *prf(P(X))
```

- The Schroeder–Heister rule is not representable in this system
  - requires third order types.
- This system is not easily represented in itself or any other current LF
  - because of the abstraction/application vectors.

Plotkin's DMBEL: Judgement Forms

Validity of Signatures, Abstraction Contexts and Contexts:

$$\vdash \Sigma \qquad \vdash_\Sigma \Phi \qquad \Phi \vdash_\Sigma \Gamma$$

Checking correctness of Types and Abstraction Types:

$$\Phi\ ;\ \Gamma \vdash_\Sigma \sigma \qquad \Phi\ ;\ \Gamma \vdash_\Sigma \alpha$$

Inferring (Abstraction) Types of (Abstraction) Terms:

$$\Phi\ ;\ \Gamma \vdash_\Sigma t \Rightarrow \sigma \qquad \Phi\ ;\ \Gamma \vdash_\Sigma A \Rightarrow \alpha$$

Checking the types of (Abstraction) Records:

$$\Phi\ ;\ \Gamma \vdash_\Sigma \overrightarrow{t} \Leftarrow \overrightarrow{\Delta} \qquad \Phi\ ;\ \Gamma \vdash_\Sigma \overrightarrow{A} \Leftarrow \overrightarrow{\Psi}$$

- *Bi-directional* typechecking, completely syntax directed.
- Cannot infer dependent types of records, as they are not unique.
  - As usual for dependent tuples, e.g. in FOL have
    $(\mathrm{True}, \mathrm{id}) : (\mathrm{x{:}o},\ \mathrm{y{:}prf(x)})$ and $(\mathrm{True}, \mathrm{id}) : (\mathrm{x{:}o},\ \mathrm{y{:}prf(True)})$.

Plotkin's DMBEL: Typing Rules

Types and Abstraction Types

$$\frac{\Phi\ ;\ \Gamma \vdash_\Sigma \overrightarrow{A} \Leftarrow \Psi}{\Phi\ ;\ \Gamma \vdash_\Sigma S(\overrightarrow{A})}\ S(\Psi) \in \Sigma \qquad \frac{\Phi\ ;\ \Gamma, \Delta \vdash_\Sigma \sigma}{\Phi\ ;\ \Gamma \vdash_\Sigma (\Delta)\sigma}$$

Terms

$$\frac{\Phi \vdash_\Sigma \Gamma}{\Phi\ ;\ \Gamma \vdash_\Sigma x \Rightarrow \sigma}\ x{:}\sigma \in \Gamma \qquad \frac{\Phi\ ;\ \Gamma \vdash_\Sigma \overrightarrow{A} \Leftarrow \Psi}{\Phi\ ;\ \Gamma \vdash_\Sigma F(\overrightarrow{A}) \Rightarrow \sigma[\overrightarrow{A}/\overrightarrow{\phi}]}\ \begin{array}{l} F(\Psi){:}\sigma \in \Sigma \\ \Psi = \overrightarrow{\phi}{:}\overrightarrow{\alpha} \end{array}$$

$$\frac{\Phi\ ;\ \Gamma \vdash_\Sigma \overrightarrow{t} \Leftarrow \Delta}{\Phi\ ;\ \Gamma \vdash_\Sigma \phi(\overrightarrow{t}) \Rightarrow \sigma[\overrightarrow{t}/\overrightarrow{x}]}\ \begin{array}{l} \phi{:}(\Delta)\sigma \in \Phi \\ \Delta = \overrightarrow{x}{:}\overrightarrow{\sigma} \end{array}$$

Abstraction Terms

$$\frac{\Phi\ ;\ \Gamma, \Delta \vdash_\Sigma t \Rightarrow \sigma}{\Phi\ ;\ \Gamma \vdash_\Sigma (\Delta)t \Rightarrow (\Delta)\sigma}$$

Plotkin's DMBEL: Typing Rules

Records

$$\frac{\Phi \vdash_\Sigma \Gamma}{\Phi \,;\, \Gamma \vdash_\Sigma \Diamond \Leftarrow \bullet} \qquad \frac{\Phi \,;\, \Gamma \vdash_\Sigma t \Rightarrow \tau \qquad \Phi \,;\, \Gamma \vdash_\Sigma \vec{u} \Leftarrow \Delta[t/x]}{\Phi \,;\, \Gamma \vdash_\Sigma t, \vec{u} \Leftarrow x{:}\sigma, \Delta}\ \tau = \sigma$$

Abstraction Records

$$\frac{\Phi \vdash_\Sigma \Gamma}{\Phi \,;\, \Gamma \vdash_\Sigma \Diamond \Leftarrow \bullet} \qquad \frac{\Phi \,;\, \Gamma \vdash_\Sigma A \Rightarrow \beta \qquad \Phi \,;\, \Gamma \vdash_\Sigma \vec{B} \Leftarrow \Psi[A/\phi]}{\Phi \,;\, \Gamma \vdash_\Sigma A, \vec{B} \Leftarrow \phi{:}\alpha, \Psi}\ \beta = \alpha$$

Context and Abstraction Context Validity

$$\frac{\vdash_\Sigma \Phi}{\Phi \vdash_\Sigma \bullet} \qquad \frac{\Phi \,;\, \Gamma \vdash_\Sigma \sigma}{\Phi \vdash_\Sigma \Gamma, x{:}\sigma} \qquad \frac{\vdash \Sigma}{\vdash_\Sigma \bullet} \qquad \frac{\Phi \,;\, \bullet \vdash_\Sigma \alpha}{\vdash_\Sigma \Phi, \phi{:}\alpha}$$

Signature Validity

$$\frac{}{\vdash \bullet} \qquad \frac{\vdash_\Sigma \Phi}{\vdash \Sigma, S(\Phi)} \qquad \frac{\Phi \,;\, \bullet \vdash_\Sigma \sigma}{\vdash \Sigma, F(\Phi){:}\sigma}$$

Plotkin's DMBEL: Hereditary Substitution ...

What is the notion of substitution that appears in these rules?

- First-order substitution as usual:
  - $u[\overrightarrow{t}/\overrightarrow{x}]$, $A[\overrightarrow{t}/\overrightarrow{x}]$, $\sigma[\overrightarrow{t}/\overrightarrow{x}]$, $\alpha[\overrightarrow{t}/\overrightarrow{x}]$, $\Gamma[\overrightarrow{t}/\overrightarrow{x}]$,
  - congruence, generated by the occurrences of x in terms,
  - termination as usual: structurally decreasing.

Plotkin's DMBEL: Hereditary Substitution (cont'd)

- Second-order substitution, $\_[A/\phi]$.
  - congruence, generated by the occurrences of ϕ in terms:

$$\begin{aligned}
x[A/\phi] &= x \\
F(\overrightarrow{B})[A/\phi] &= F(\overrightarrow{B}[A/\phi]) \\
\phi(\overrightarrow{t})[A/\phi] &= \phi(t_1,\ldots,t_n)[(x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n)u/\phi] \\
&= u[t_1[A/\phi],\ldots,t_n[A/\phi]/x_1,\ldots,x_n] \\
\phi(\overrightarrow{t})[A/\phi'] &= \phi(\overrightarrow{t}[A/\phi']) \qquad (\phi \neq \phi')
\end{aligned}$$

- Termination: second order substitutions are structurally decreasing.
  - May give rise to (terminating) first order substutution.
  - Really need a syntactic notion of *arity* to be precise.

Outline

Luo's PAL$^+$

- Developed in late 1990's; journal version [Luo03].
- Named after Automath language PAL
- A *lambda-free* version of Luo's LF [Luo94].
  - Basic principles: parameterisation (with instantiation) and definition.
  - Also allows inductive definition.
- Parameterisation and instantiation have arities; arities must match.
  - Allowing precise specification of object language, as in DMBEL.
- Arbitrary orders of types, but no nested parameterisation.
- Parametric definitions, both local and global.
  - Thus abstraction can be expressed.
  - Has real computation: eta and delta rules.
- Meta-theory by *Typed Operational Semantics*: complex.
- Some work towards support of *coercive subtyping*.

Robin Adams' Hierarchy of Frameworks, TF

- Robin Adams' thesis [Ada05] precedes Plotkins DMBEL.
- In his thesis, Adams gives a hierarchy of systems that extend DMBEL to orders 3, 4, ...
- Adams' thesis contains many interesting and explanatory examples.

Outline

Canonical LF

- Accepts exactly the canonical judgements accepted by ELF.
- Developed by Watkins, Pfenning, and co-workers [WCPW02, HL06].
- This work precedes both Adams' TF and Plotkin's DMBEL.
- Only β-normal terms are syntactically formed.
- Only canonical terms (η-expanded, β-normal) are well-typed.
  - Equality is syntactic identity of expressions.
  - A notion of hereditary substitution is needed.
- Bi-directional typechecking: syntax directed and decidable.

Canonical LF: Syntax

identifiers	a (type constants), c (term constants), x (term variables)
kinds	$K ::= \text{type} \mid \Pi x{:}A.K$
types	$A, B ::= P \mid \Pi x{:}A.B$
atomic types	$P ::= a \mid P\ M$
terms	$M ::= R \mid \lambda x.M$
atomic terms	$R ::= c \mid x \mid R\ M$
contexts	$\Gamma, \Delta ::= \bullet \mid \Gamma, x{:}A$
signatures	$\Sigma ::= \bullet \mid \Sigma, a{:}K \mid \Sigma, c{:}A$

Canonical LF: Judgement Forms

Validity of signatures and contexts:

$$\vdash \Sigma \text{ sig} \qquad\qquad \vdash_\Sigma \Gamma \text{ cxt}$$

Checking correctness of kinds and types:

$$\Gamma \vdash_\Sigma K \text{ kind} \qquad\qquad \Gamma \vdash_\Sigma A \text{ type}$$

Inferring kinds of atomic types, and the types of atomic terms:

$$\Gamma \vdash_\Sigma P \Rightarrow K \qquad\qquad \Gamma \vdash_\Sigma R \Rightarrow A$$

Checking the types of terms:

$$\Gamma \vdash_\Sigma M \Leftarrow A$$

- *Bi-directional* typechecking, completely syntax directed.
- Cannot infer the type of $\lambda x.M$: no type annotation on x.

Canonical LF: Rules

Signatures

$$\frac{}{\vdash \bullet \text{ sig}} \qquad \frac{\vdash \Sigma \text{ sig} \qquad \vdash_\Sigma K \text{ kind}}{\vdash \Sigma, a{:}K \text{ sig}} \qquad \frac{\vdash \Sigma \text{ sig} \qquad \vdash_\Sigma A \text{ type}}{\vdash \Sigma, c{:}A \text{ sig}}$$

Contexts

$$\frac{\vdash \Sigma \text{ sig}}{\vdash_\Sigma \bullet \text{ cxt}} \qquad \frac{\vdash_\Sigma \Gamma \text{ cxt} \qquad \Gamma \vdash_\Sigma A \text{ type}}{\vdash_\Sigma \Gamma, x{:}A \text{ cxt}}$$

Canonical LF: Rules

Kinds

$$\frac{\vdash_\Sigma \Gamma \text{ cxt}}{\Gamma \vdash_\Sigma \text{type} \;\; \text{kind}} \qquad \frac{\Gamma, x{:}A \vdash_\Sigma K \;\; \text{kind}}{\Gamma \vdash_\Sigma \Pi x{:}A.K \;\; \text{kind}}$$

Types

$$\frac{\Gamma, x{:}A \vdash_\Sigma B \;\; \text{type}}{\Gamma \vdash_\Sigma \Pi x{:}A.B \;\; \text{type}} \qquad \frac{\Gamma \vdash_\Sigma P \Rightarrow \text{type}}{\Gamma \vdash_\Sigma P \;\; \text{type}}$$

Atomic Types

$$\frac{\vdash_\Sigma \Gamma \text{ cxt}}{\Gamma \vdash_\Sigma a \Rightarrow K}\ (a{:}K \in \Sigma) \qquad \frac{\Gamma \vdash_\Sigma P \Rightarrow \Pi x{:}A.K \qquad \Gamma \vdash_\Sigma M \Leftarrow A}{\Gamma \vdash_\Sigma P\, M \Rightarrow [M^{(A)^-}/x]K}$$

Canonical LF: Rules

Terms

$$\frac{\Gamma, x{:}A \vdash_\Sigma M \Leftarrow B}{\Gamma \vdash_\Sigma \lambda x.M \Leftarrow \Pi x{:}A.B} \qquad \frac{\Gamma \vdash_\Sigma R \Rightarrow P}{\Gamma \vdash_\Sigma R \Leftarrow P}\ (*)$$

Atomic Terms

$$\frac{\vdash_\Sigma \Gamma\ \mathrm{cxt}}{\Gamma \vdash_\Sigma x \Rightarrow A}\ (x{:}A \in \Gamma) \qquad \frac{\vdash_\Sigma \Gamma\ \mathrm{cxt}}{\Gamma \vdash_\Sigma c \Rightarrow A}\ (c{:}A \in \Sigma)$$

$$\frac{\Gamma \vdash_\Sigma R \Rightarrow \Pi x{:}A.B \qquad \Gamma \vdash_\Sigma M \Leftarrow A}{\Gamma \vdash_\Sigma R\, M \Rightarrow [M^{(A)^-}/x]B}$$

$(*)$ This restriction to atomic types, P, makes judgements canonical.

Hereditary Substitution: Pure λ-terms

Let L, M, N be lambda terms.

$$
\begin{array}{lcll}
[M\ /x]x & = & M & \\
[M\ /x]y & = & y & \text{if } x \neq y \\
[M\ /x](\lambda y.N) & = & \lambda y.[M\ /x]N & y \text{ fresh} \\
[M\ /x](L\ N) & = & ([\hat{N}\ /y]M') & \text{if } \hat{L} = (\lambda y.M') \\
 & = & \hat{L}\ \hat{N} & \text{otherwise}
\end{array}
$$

where $\hat{L} = [M\ /x]L$ and $\hat{N} = [M\ /x]N$

- If M and N have no β-redexes, then $[M/x]N$
  - is β-equal to the usual substitution $M[x/N]$
  - contains no β-redexes
- However $[M/x]N$ may not terminate.
- We use types to control how deep the substitution goes.

Hereditary Substitution: Simply typed terms [Abe06]

Let L, M, N be lambda terms.
Let A, B, C be simple types

$$
\begin{array}{lcll}
[M^A/x]x & = & M^A & \\
[M^A/x]y & = & y & \text{if } x \neq y \\
[M^A/x](\lambda y.N) & = & \lambda y.[M^A/x]N & y \text{ fresh} \\
[M^A/x](L\,N) & = & ([\hat{N}^B/y]M')^C & \text{if } \hat{L} = (\lambda y.M')^{B\to C} \\
 & = & \hat{L}\,\hat{N} & \text{otherwise}
\end{array}
$$

$$\text{where } \hat{L} = [M^A/x]L \text{ and } \hat{N} = [M^A/x]N$$

- If M and N have no β-redexes, then $[M/x]N$
  - is β-equal to the usual substitution $M[x/N]$
  - contains no β-redexes if the type annotation is correct
- Always terminates: newly created substitutions operate on subterms, or have syntactically smaller types.

Hereditary Substitution: Canonical LF

- LF types exactly the same lambda-terms as simple types . . .
- . . . just throw away the dependency.

$$\begin{array}{lcl} (a)^- & = & a \\ (P\ M)^- & = & (P)^- \\ (\Pi x{:}A.B)^- & = & (A)^- \to (B)^- \end{array}$$

  Only the shape of the simple type matters: can take $(a)^- = \bullet$.
- One more problem: canonical terms cannot contain redexes.
  - Instead of returning a term with a redex (when the given type is incorrect), the canonical hereditary substitution algorithm must return failure.
- Finally, $[M^{(A)^-}/x]\_$ is a congruence, generated by the replacement of term variables in terms, as explained above.

Judgements are Canonical by Stratified Syntax

We want to know that no two derivable judgements differ only by $\beta\eta$.

- Let $\Sigma = a$:type.
  - $(a \to a)$ is a non-atomic type, syntactic class A, not class P.

Direction $\Rightarrow$:

- f:$(a \to a) \vdash_\Sigma f \Rightarrow (a \to a)$ is derivable,
- f:$(a \to a) \vdash_\Sigma \lambda x.f\ x \Rightarrow (a \to a)$ is not derivable:
  - $\lambda x.f\ x$ is of class M, not of class R,
  - there are no rules of shape $\Gamma \vdash_\Sigma M \Rightarrow A$.

Judgements are Canonical by Stratified Syntax

Direction $\Leftarrow$:

- $f{:}(a \to a) \vdash_\Sigma f \Leftarrow (a \to a)$ is not derivable.
  - f is not of shape $\lambda x.M$ and $a \to a$ is not of shape P, so no rule applies.
- $f{:}(a \to a) \vdash_\Sigma \lambda x.f\ x \Leftarrow (a \to a)$ is derivable:
  Let $\Gamma = f{:}(a \to a), x{:}a$.

$$\dfrac{\dfrac{f{:}(a \to a) \vdash_\Sigma a \Rightarrow \text{type}}{f{:}(a \to a) \vdash_\Sigma a \Leftarrow \text{type}} \qquad \dfrac{\dfrac{\Gamma \vdash_\Sigma f \Rightarrow (a \to a) \quad \dfrac{\Gamma \vdash_\Sigma x \Rightarrow a}{\Gamma \vdash_\Sigma x \Leftarrow a}\ (*) \qquad [x/\_]a = a}{\Gamma \vdash_\Sigma f\ x \Rightarrow a}}{\Gamma \vdash_\Sigma f\ x \Leftarrow a}\ (*)}{f{:}(a \to a) \vdash_\Sigma \lambda x.f\ x \Leftarrow (a \to a)}$$

$(*)$ At atomic type.

Outline

References

[Abe06] Andreas Abel. Implementing a normalizer using sized heterogeneous types.
In *Workshop on Mathematically Structured Functional Programming, MSFP 2006*.

[Ada05] Robin Adams. *A Modular Hierarchy of Logical Frameworks*).
PhD thesis, University of Manchester, 2005.

[Geu93] Herman Geuvers. *Logics and Type Systems*.
PhD thesis, Univ. of Nijmegen, 1993.

[Gog05] Healfdene Goguen. A syntactic approach to eta equality in type theory.
In *POPL 2005*.

[HL06] Robert Harper and Daniel R. Licata. Mechanizing Metatheory in a Logical Framework.
Submitted for publication. Available from `http://www.cs.cmu.edu/~drl/`, Oct. 2006.

[HP05] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory.
ACM Trans. on Computational Logic, 6(1), 2005.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*.
International Series of Monographs on Computer Science. Oxford Univ. Press, 1994.

[Luo03] Zhaohui Luo. PAL $^+$: a lambda-free logical framework.
Journal of Functional Programming, 13(2):317–338, 2003.

[Sal] Anne Salvesen. The Church-Rosser theorem for LF with beta/eta reduction.
Talk given at First Workshop on Logical Frameworks, Antibes, 1990.

[WCPW02] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker.
A concurrent logical framework I: Judgments and properties.
Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.