Optimizing Higher-Order Pattern Unification

Brigitte Pientka and Frank Pfenning

Carnegie Mellon University

CADE-19, Miami Beach, Florida, August 2003

Logical Frameworks

- *Logical frameworks*: meta-languages for deductive systems
  - High-level specifications (e.g. logics, type systems)
  - Direct implementations (e.g. proof search, type checking)
  - Feasible meta-reasoning (e.g. cut elimination, type preservation)
- Examples: λProlog, Isabelle, Twelf
- Results may apply to other higher-order systems (e.g. Coq, HOL, NuPrl, PVS)

Higher-Order Unification

- *Higher-order unification*: solving equations in the presence of λ-abstraction
- Central in higher-order logic and type theories
  - General proof search
  - Logic program execution
  - Type and term reconstruction
  - Definitional reflection and case analysis
- Undecidable for second order [Huet'73] [Goldfarb'81]

Tractable Cases

- Pre-unifi cation often practical [Huet'75]
  - Some solvable equations are postponed
  - Non-determinism major drawback in practice
- Pattern unifi cation decidable [Miller'91]
  - Restricting β-reduction to β_0: $(\lambda x.M)\, y \longrightarrow [y/x]M$
  - Most general unifiers exist!
  - Extends to complex type theories [Pf'91]
  - Higher-order patterns as a calculus of variable binding, variable occurrences, and renaming

Implementation

- In practice, β_0 often too weak [Michaylov & Pf'92]
- Implementation based on constraints
  - Solve all pattern equations
  - Postpone all other equations
- Many equations ($\approx 95\%$) essentially fi rst-order
- First-order equations with fi rst-order effi ciency?

Outline

- A logical foundation for meta-variables
- Techniques for effi cient implementation
- Experimental results
- Conclusions and future work

Example: Quantifier Manipulation

- Object logic is simple first-order logic

  Formulas $A \ ::= \ P \mid A \supset A \mid \forall x.A \mid \exists x.A \mid \ldots$

- Specifying manipulation of quantifier scope
- Sample rule

$$(\forall x.A(x) \supset B) \Leftrightarrow ((\exists x.A(x)) \supset B)$$

- Proviso: x may not be free in B

Specification in LF

- Language using higher-order abstract syntax

$$
\begin{array}{lcl}
\mathsf{i} & : & \mathsf{type}. \\
\mathsf{o} & : & \mathsf{type}. \\
\mathsf{imp} & : & \mathsf{o} \longrightarrow \mathsf{o} \longrightarrow \mathsf{o}. \\
\mathsf{forall} & : & (\mathsf{i} \longrightarrow \mathsf{o}) \longrightarrow \mathsf{o}. \\
\mathsf{exists} & : & (\mathsf{i} \longrightarrow \mathsf{o}) \longrightarrow \mathsf{o}.
\end{array}
$$

- Scope relation $(\forall x.A(x) \supset B) \Leftrightarrow ((\exists x.A(x)) \supset B)$

$$
\begin{array}{lcl}
\mathsf{eq} & : & \mathsf{o} \longrightarrow \mathsf{o} \longrightarrow \mathsf{type}. \\
\mathsf{eq}_1 & : & \Pi \mathrm{A} : \mathsf{i} \longrightarrow \mathsf{o}.\, \Pi \mathrm{B} : \mathsf{o}. \\
 & & \quad \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\, \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\, \mathrm{A}\ \mathsf{x})\ \mathrm{B}).
\end{array}
$$

Meta-Variables

- Sample query

$$?- \quad \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\, \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C}.$$

- Copy clause and unify

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\, \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\, \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\, \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C} \end{array}$$

- Here $\mathrm{A} : \mathsf{i} \to \mathsf{o}$, $\mathrm{B} : \mathsf{o}$, $\mathrm{C} : \mathsf{o}$ are *meta-variables* (also called *unification variables* or *existential variables* or *logic variables*)

Closed Meta-Variables

- Unifi cation problem

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\ \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C} \end{array}$$

- Solution

$$\begin{array}{rcl} \mathrm{A} & = & \lambda \mathsf{z}.\ \mathsf{imp}\ (\mathsf{p}\ \mathsf{z})\ (\mathsf{p}\ \mathsf{z}) \\ \mathrm{B} & = & \mathsf{q} \\ \mathrm{C} & = & \mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x})\ \mathrm{B}) \\ & = & \mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathsf{p}\ \mathsf{x})\ (\mathsf{p}\ \mathsf{x}))\ \mathsf{q}) \end{array}$$

- Meta-variables contain no free ordinary variables!

  [Huet'75]

Closed Meta-Variables

- A failing query

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\ \mathsf{imp}\ \mathsf{q}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))))\ \mathrm{C} \end{array}$$

- No solution

$$\begin{array}{lcll} \mathrm{A} & = & \lambda \mathsf{z}.\ \mathsf{q} & \\ \mathrm{B} & = & (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y})) & \text{fails} \\ \mathrm{C} & = & \ldots & \end{array}$$

- In fact:

$$(\forall x.q \supset (p(x) \supset p(x))) \not\Leftrightarrow ((\exists x.q) \supset (p(x) \supset p(x)))$$

Modal Logic of Necessity

- Modal logic of necessity captures closedness
- In Hilbert Style

$$\frac{\vdash A}{\vdash \Box A}$$

- With proof terms in natural deduction

$$\frac{\bullet \vdash M : A}{\Gamma \vdash \mathsf{box}\ M : \Box A}$$

- $\Gamma ::= \bullet \mid \Gamma, x{:}A$
- Rule enforces that M closed

Necessity via Judgments

- Separate judgments from propositions [Martin-Löf'83]
- Extended to modal logic [Pf & Davies'01]
- Two basic judgments
  - $A\ true$ (proposition A is *true*)
  - $A\ valid$ (proposition A is *valid*)
- Form hypothetical judgment

$$\underbrace{A_1\ valid, \ldots A_n\ valid}_{\Delta}; \underbrace{B_1\ true, \ldots, B_m\ true}_{\Gamma} \vdash C\ true$$

Necessity via Judgments

- Defi nitional rule for validity

$$\frac{\Delta; \bullet \vdash C\ true}{\Delta \vdash C\ valid}$$

- Hypothesis rule

$$\frac{}{\Delta, A\ valid, \Delta'; \Gamma \vdash A\ true}$$

- Substitution principle

  If $\Delta; \bullet \vdash A\ true$ (means: $\Delta \vdash A\ valid$)
  and $\Delta, A\ valid; \Gamma \vdash C\ true$
  then $\Delta; \Gamma \vdash C\ true$

Adding Proof Terms

- Two kinds of variables for two kinds of hyps
- Modal variables
  - $\Delta ::= \bullet \mid \Delta, u{::}A$
  - $u{::}A$ expresses $A\ valid$ (stands for closed terms)
  - Represent (closed) meta-variables
- Ordinary variables
  - $\Gamma ::= \bullet \mid \Gamma, x{:}A$
  - $x{:}A$ expresses $A\ true$ (stands for any term)
  - Ordinary variables can be bound by lambda

Adding Proof Terms

- Judgment $\Delta; \Gamma \vdash M : A$
- Hypothesis rule

$$\frac{}{(\Delta, u{::}A, \Delta); \Gamma \vdash u : A}$$

- Substitution principle (without dependencies)

  If $\Delta; \bullet \vdash M : A$
  and $\Delta, u{::}A; \Gamma \vdash N : C$
  then $\Delta; \Gamma \vdash [\![M/u]\!]N : C$

- Need substitution operation $[\![M/u]\!]N$

Substitution for Meta-Variables

- Terms $M ::= c \mid x \mid u \mid (\lambda x.M) \mid M_1\, M_2$

$$
\begin{aligned}
[\![M/u]\!]c &= c \\
[\![M/u]\!]x &= x \\
[\![M/u]\!]u &= M \\
[\![M/u]\!]w &= w \quad \text{for } u \neq w \\
[\![M/u]\!](N_1\, N_2) &= ([\![M/u]\!]N_1)\,([\![M/u]\!]N_2) \\
[\![M/u]\!](\lambda x.N) &= \lambda x.[\![M/u]\!]N
\end{aligned}
$$

- No side condition on last rule (M closed)!
- *Can implement with destructive update!*

Example Revisited

- Recall unifi cation problem with meta-variables

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\ \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C} \end{array}$$

- First meta-variable / term subproblem:

$$\mathsf{z} : \mathsf{i} \vdash \mathrm{A}\ \mathsf{z} \doteq \mathsf{imp}\ (\mathsf{p}\ \mathsf{z})\ (\mathsf{p}\ \mathsf{z})$$

- To solve, traverse $\mathsf{imp}\ (\mathsf{p}\ \mathsf{z})\ (\mathsf{p}\ \mathsf{z})$
  - Check for occurrences of A (occurs-check)
  - Ensure A is closed (pruning)
  - Build $\lambda \mathsf{z}.\ \mathsf{imp}\ (\mathsf{p}\ \mathsf{z})\ (\mathsf{p}\ \mathsf{z})$

Avoiding Term Traversal

- To solve $\Delta; \Gamma \vdash u\, x_1 \ldots x_n \doteq M$ we traverse M
  - Check there is no occurrence of u (occurs-check)
  - Ensure $\{x_1, \ldots, x_n\} \subseteq FV(M)$ (pruning to M')
  - Build $u = \lambda x_1 \ldots . \lambda x_n . M'$
- Goal: avoid traversal of right-hand side M
- Condition: remain sound (unlike Prolog)
- Highly signifi cant optimization for logic programming and theorem proving

Eliminating Occurs-Check

- We obtain left-hand side of

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\ \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C} \end{array}$$

  by copying

$$\begin{array}{lll} \mathsf{eq}_1 & : & \Pi \mathrm{A} : \mathsf{i} \to \mathsf{o}.\ \Pi \mathrm{B} : \mathsf{o}. \\ & & \quad \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x})\ \mathrm{B}). \end{array}$$

- The meta-variables A and B are *fresh*
  - A and B cannot occur on right-hand side
  - Eliminate occurs-check for first variable occurrences

Eliminating Occurs-Check

- Rename duplicate occurrences
- Construct residual equations (variable defns)
- Example: compile

  $\mathsf{s} \;:\; \Pi \mathrm{A} : \mathsf{o}.\, \mathsf{same}\ \mathrm{A}\ \mathrm{A}.$

  to

  $\mathsf{s}' \;:\; \Pi \mathrm{A} : \mathsf{o}.\, \Pi \mathrm{A}' : \mathsf{o}.\, \mathsf{same}\ \mathrm{A}\ \mathrm{A}' \leftarrow \mathrm{A} = \mathrm{A}'.$

- Technical complication due to dependent types
- See paper in proceedings for solution

Eliminating Pruning

- Recall

$$\begin{array}{rl} & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\ \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\ \mathrm{A}\ \mathsf{x}))\ \mathrm{B}) \\ \doteq & \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{y}.\ \mathsf{imp}\ (\mathsf{imp}\ (\mathsf{p}\ \mathsf{y})\ (\mathsf{p}\ \mathsf{y}))\ \mathsf{q}))\ \mathrm{C} \end{array}$$

- Would like to replace $(\mathrm{A}\ \mathsf{x})$ by A' such that

$$\mathsf{x} : \mathsf{i} \ \vdash\ \mathrm{A}' : \mathsf{o}$$

- Yields

$$\mathsf{x} : \mathsf{i} \ \vdash\ \mathrm{A}' \doteq \mathsf{imp}\ (\mathsf{p}\ \mathsf{x})\ (\mathsf{p}\ \mathsf{x})$$

- Then substitute (with apparent capture)

$$[\![\mathsf{imp}\ (\mathsf{p}\ \mathsf{x})\ (\mathsf{p}\ \mathsf{x})/\mathrm{A}']\!]$$

Eliminating Pruning

- Note that meta-variable A' with $\mathsf{x} : \mathsf{i} \vdash \mathrm{A}' : \mathsf{o}$ is no longer closed.
- General form: $\Psi_u \vdash u : A_u$ for context Ψ_u of ordinary variables [Dowek, Hardin, C.Kirchner'95]
- Does it have a logical foundation?
- Generalize validity to *relative validity*

Relative Validity

- Defi nitional rule for validity

$$\frac{\Delta; \Psi \vdash C \ true}{\Delta \vdash C \ valid \ \Psi}$$

- Hypothesis rule

$$\frac{\Delta, A \ valid \ \Psi, \Delta'; \Gamma \vdash B_i \ true \quad \text{for all } B_i \ true \text{ in } \Psi}{\Delta, A \ valid \ \Psi, \Delta'; \Gamma \vdash A \ true}$$

- Substitution principle (w/o dependent types)

  If $\Delta; \Psi \vdash A \ true$
  and $\Delta, A \ valid \ \Psi; \Gamma \vdash C \ true$
  then $\Delta; \Gamma \vdash C \ true$

Adding Proof Terms

- $u{::}(\Psi \vdash A)$ expresses $A\ valid\ \Psi$
- Need simultaneous substitutions σ
- Generalized hypothesis rule

$$\frac{(\Delta, u{::}(\Psi \vdash A), \Delta); \Gamma \vdash \sigma : \Psi}{(\Delta, u{::}(\Psi \vdash A), \Delta); \Gamma \vdash u[\sigma] : A}$$

- Substitution principle (w/o dependent types)

  If $\Delta; \Psi \vdash M : A$
  and $\Delta, u{::}(\Psi \vdash A); \Gamma \vdash N : C$
  then $\Delta; \Gamma \vdash [\![M/u]\!]N : C$

Simultaneous Substitution

- Substitutions $\sigma ::= \bullet \mid \sigma, M/x$
- Show the fully dependent case

$$\frac{}{\Delta; \Gamma \vdash (\bullet) : (\bullet)} \qquad \frac{\Delta; \Gamma \vdash \sigma : \Psi \quad \Delta; \Gamma \vdash M : [\sigma]A}{\Delta; \Gamma \vdash (\sigma, M/x) : (\Psi, x{:}A)}$$

- Applying a substitution is straightforward
- Substitution $[\sigma]M$ and $[\sigma]\tau$ is capture avoiding

Meta-Variable Substitution

- Terms $M ::= c \mid x \mid u[\sigma] \mid (\lambda x.M) \mid M_1\, M_2$
- Note closures only for meta-variables
- Changed cases in meta-variable substitution

$$\begin{aligned} [\![M/u]\!](u[\sigma]) &= [[\![M/u]\!]\sigma]M \\ [\![M/u]\!](w[\tau]) &= w[[\![M/u]\!]\tau] \quad \text{for } u \neq w \end{aligned}$$

- Can still substitute in place (grafting) if $[[\![M/u]\!]\sigma]M$ is postponed
- Implemented via a calculus of explicit substitutions

Abstraction Over Meta-Variables

- We can directly abstract over meta-variables
- Modal logic and judgments dictate the rules
- New type $\Pi^{\Box}\mathsf{u}::(\Psi \vdash \mathrm{A}).\,\mathrm{B}$
- New application and abstraction

$$M ::= \ldots \mid \lambda^{\Box} u.M \mid M_1 \,\Box\, M_2$$

- Theory not yet fully investigated
- Currently only used in prefi x form

Example Revisited

- Recall

$$\mathsf{eq}_1 \;:\; \Pi \mathrm{A} : \mathsf{i} \to \mathsf{o}.\, \Pi \mathrm{B} : \mathsf{o}.$$
$$\mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\, \mathsf{imp}\ (\mathrm{A}\ \mathsf{x})\ \mathrm{B}))\ (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\, \mathrm{A}\ \mathsf{x})\ \mathrm{B}).$$

- Compile as

$$\begin{aligned}
\mathsf{eq}_1 \;:\; & \Pi^{\square} \mathrm{A}' :: (\mathsf{x} : \mathsf{i} \vdash \mathsf{o}).\, \Pi^{\square} \mathrm{B}' :: (\mathsf{x} : \mathsf{i} \vdash \mathsf{o}). \\
& \Pi^{\square} \mathrm{A}'' :: (\mathsf{x} : \mathsf{i} \vdash \mathsf{o}).\, \Pi^{\square} \mathrm{B}'' :: (\mathsf{x} : \mathsf{i} \vdash \mathsf{o}). \\
& \Pi^{\square} \mathrm{B} :: (\bullet \vdash \mathsf{o}). \\
& \quad \mathsf{eq}\ (\mathsf{forall}\ (\lambda \mathsf{x}.\, \mathsf{imp}\ \mathrm{A}'[\mathsf{id}]\ \mathrm{B}'[\mathsf{id}])) \\
& \qquad (\mathsf{imp}\ (\mathsf{exists}\ (\lambda \mathsf{x}.\, \mathrm{A}''[\mathsf{id}])\ \mathrm{B}''[\mathsf{id}]) \\
& \leftarrow (\Pi \mathsf{x} : \mathsf{i}.\mathrm{A}'[\mathsf{id}] = \mathrm{A}''[\mathsf{id}]) \\
& \leftarrow (\Pi \mathsf{x} : \mathsf{i}.\mathrm{B}[\bullet] = \mathrm{B}'[\mathsf{id}] = \mathrm{B}''[\mathsf{id}]).
\end{aligned}$$

Operational Semantics

- Unifi cation in clause head always by *assignment*
  - No occurs-check
  - No pruning
  - No copying
- Assignment as in-place update (as in Prolog), but safe
- Residual equations solved by ordinary pattern unifi cation (almost)

Experimental Results

- Meta-interpreter for ordered linear logic

example	opt	stand	speed-up	residual eqns	
				none	fail
sqnt (bf)	0.84	2.09	149%	44%	46%
sqnt (dfs)	0.93	2.35	152%	44%	47%
sqnt (perm)	4.44	7.11	60%	44%	52%
sqnt (rev)	1.21	1.70	40%	45%	48%
sqnt (mergesort)	2.26	3.39	50%	46%	53%

Experimental Results

- Foundational proof-carrying code

example	opt	stand	speed-up	residual eqns	
				none	fail
inc	5.8	9.19	58%	64%	46%
switch	36.00	49.69	38%	64%	48%
mul2	5.51	9.520	72%	64%	46%
div2	121.96	153.610	26%	63%	48%
divx	333.69	1133.150	239%	63%	50%
listsum	1073.33	∞	∞	65%	45%
polyc	2417.85	∞	∞	65%	41%
pack	197.07	1075.610	445%	66%	45%

Experimental Results

- Intuitionistic sequent calculus

example	opt	stand	speed-up	residual eqns	
				none	fail
dist-1	53.00	57.11	8%	100%	0%
distImp	0.40	0.44	10%	100%	0%
pierce	1520.77	1563.35	3%	100%	0%
trans	0.13	0.13	0%	100%	0%

Experimental Results

- Proof search in classical natural deduction

example	opt	stand	speed-up	residual eqns	
				none	fail
andEff1-nk	7.67	13.14	71%	100%	0%
andEff2-nk	3.86	6.58	70%	100%	0%
assocAnd-nk	2.24	3.74	67%	100%	0%
combS-nk	3.85	6.64	72%	100%	0%

Summary

- A foundation for meta-variables in modal logic
- Resulting techniques for effi cient implementation
- Clearly signifi cant in practice
  - Logic programming
  - Generic theorem proving
- Also insights into some internal invariants (“just because”)

Some Related Work

- Higher-order unifi cation via explicit substitutions
  [Dowek, Hardin, C.Kirchner'95] [Dowek, Hardin, C.Kirchner, Pf'96]
  - Meta-variables $\Gamma_X \vdash X : A_X$
  - Substitution via grafting, missing other optimizations
  - No dependent types
  - No logical explanation
- Teyjus compiler for λProlog [Nadathur et al.]
  - WAM-style optimization for occurs-check
  - Uses Huet's algorithm, not constraint-based patterns
  - No dependent types

Future Work

- Higher-order term indexing [Pientka'03]
- Internal redundancy elimination
  [Michaylov & Pf'93] [Necula & Lee'98]
  - Specifi c to dependent types
  - Avoid some unifi cations altogether
  - Interacts with linearization
- Accounting for mode information
- More on modal type theory
  [Nanevski, Pientka, Pf'03]