Contextual modal type theory: a foundation for meta-variables

Brigitte Pientka

School of Computer Science

McGill University

Montreal, Canada

Joint work with A. Nanevski and F. Pfenning

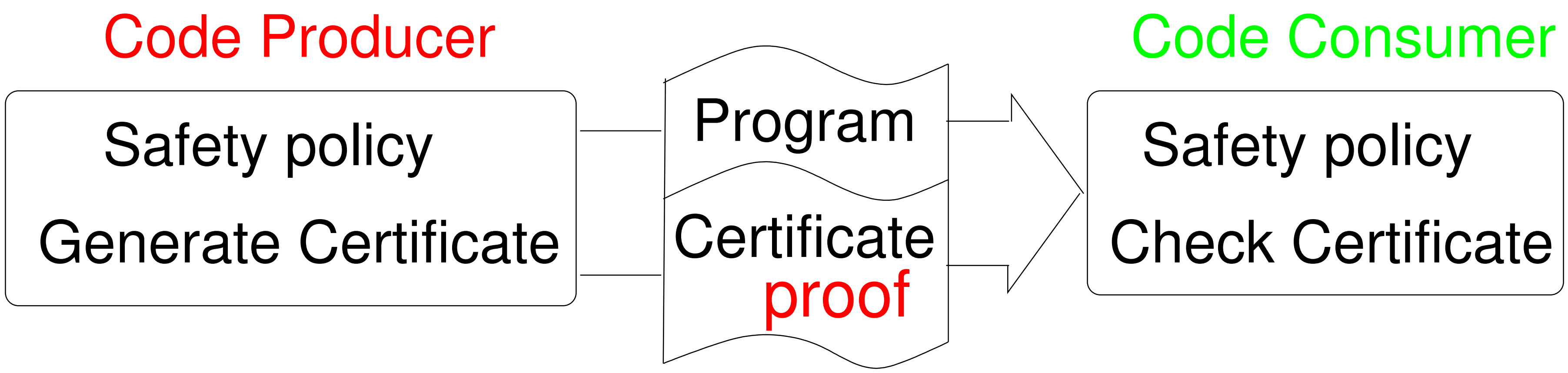
Outline

- Logical frameworks and certified code
- Contextual modal type theory
- Applications: Higher-order unification
- Conclusion and future work

Logical frameworks

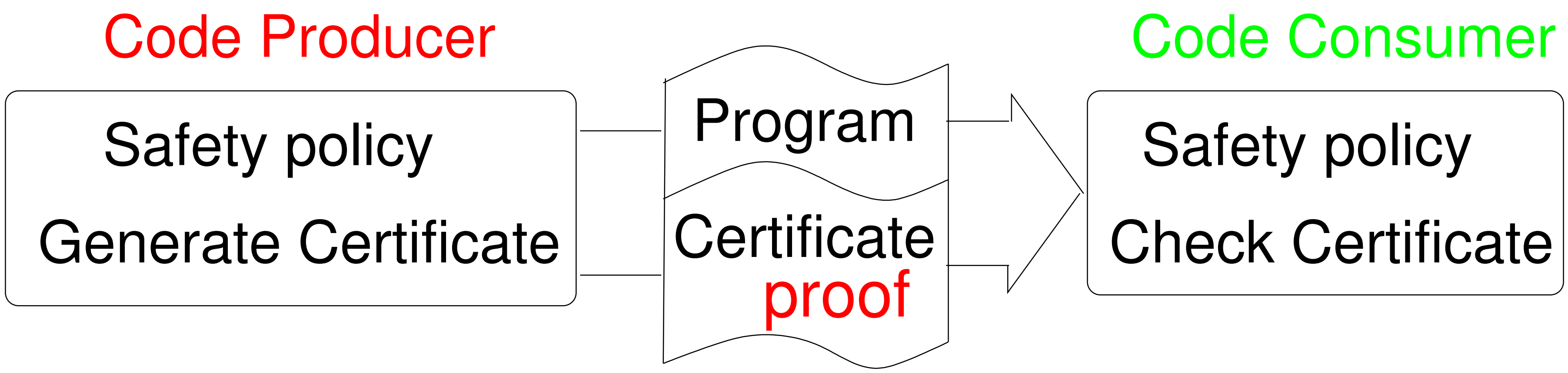
- Meta-languages for deductive systems
  - High-level specification (e.g. logics, type systems)
  - Direct implementations (e.g. proof search, type checking)
  - Meta-reasoning (e.g. cut elim., type preservation)
- Examples:
  λProlog, Twelf, Isabelle
- Other higher-order systems:
  Coq, PVS, NuPRL, HOL, ...

Application: certified code



- Foundational proof-carrying code : [Appel, Felty 00]
- Temporal-logic proof carrying code [Bernard,Lee02]
- Foundational typed assembly language : [Crary 03]
- Proof-carrying authentication: [Felten, Appel 99]

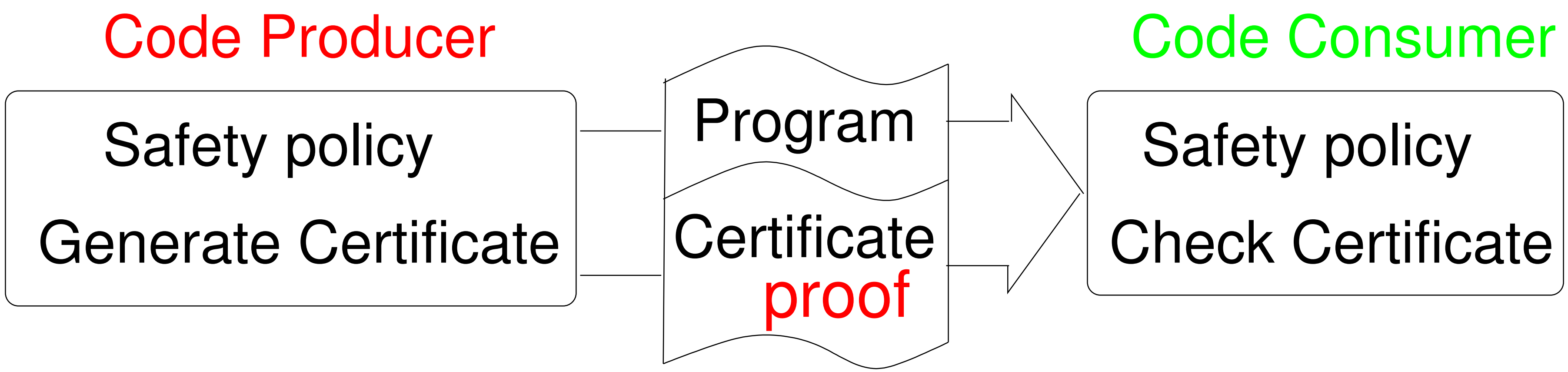
Application: certified code



Large-scale applications

- Typical code size: 70,000 – 100,000 lines
  includes data-type definitions and proofs
- Higher-order logic program: 5,000 lines
- Over 600 – 700 clauses

Application: certified code



Special purpose logical frameworks:

- Efficient representation of proofs

  LF$_i$ [Necula,Lee'98] (2-level, restricting dependent types)

- Proof checking via
  "higher-order" logic programming

  Oracle-based checking[Necula'01], Fleat[Wu'03]

  No higher-order terms

Application: POPLmark Challenge

- Challenge: How can we
  - encode elegantly prog. languages?
  - experiment easily with proposed systems?
  - facilitate interactive proof developments?
  - prove or check meta-properties?
- Goal: Verify mechanically every POPL paper by 2010.

State of the art

- Logical frameworks are widely used.
- Many challenges remain:
  - Higher-order systems are not efficient enough in practice.
  - Complexity of higher-order issues poorly understood.
  - Higher-order systems lack automatic support.
  - ...

State of the art

- Logical frameworks are widely used.
- Many challenges remain:
  - Higher-order systems are not efficient enough in practice.
  - Complexity of higher-order issues poorly understood.
  - Higher-order systems lack automatic support.
  - ...
- This talk: Contextual modal logic and type theory
  - Foundation for meta-variables and explicit substitutions
  - Relativized truth and validity

Outline

- Logical frameworks and certified code
- Contextual modal type theory
- Applications: Higher-order unification
- Conclusion and future work

Example: Quantifier Manipulation

- Object logic: First-order logic

$$\text{Formula} \;\; ::= \;\; P \mid A \supset A \mid \forall x.A \mid \exists x.A \mid \ldots$$

- Specifying manipulation of quantifier scope
- Sample rule:

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

if x is not free in B

Specification in LF

- Based on higher order abstract syntax:

i	:	type.
o	:	type.
imp	:	o $\longrightarrow$ o $\longrightarrow$ o.
all	:	(i $\longrightarrow$ o) $\longrightarrow$ o.
exists	:	(i $\longrightarrow$ o) $\longrightarrow$ o.

- Quantifier manipulation:

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

eq	: o $\longrightarrow$ o $\longrightarrow$ type.
eq_all	: eq (all (λ x. imp (A x) B)) (imp (exists (λx. A x)) B).

Meta-variables

- Clause :

  eq_all : eq (all (λ x. imp (A x) B)) (imp (exists (λx. A x)) B).

- A: i $\longrightarrow$ o and B: o are meta-variables
  also sometimes called *existential variables* or *logic variables*

- Unification problem:

$$\text{eq (all } (\lambda \text{ y. imp (imp (p y) (p y))) q) C}$$
$$\doteq$$
$$\text{eq (all } (\lambda \text{ x. imp (A x) B)) (imp (exists } (\lambda\text{x. A x)) B)}$$

Closed instantiation for meta-variables

- Unification problem:

$$\text{eq (all } (\lambda\, \text{y. imp (imp (p y) (p y))) q) C}$$
$$\doteq$$
$$\text{eq (all } (\lambda\, \text{x. imp (A x) B)) (imp (exists } (\lambda \text{x. A x)) B)}$$

- Solution:

$$\begin{array}{lll} \text{A} & = & \lambda\, \text{z. imp (p z) (p z)} \\ \text{B} & = & \text{q} \\ \text{C} & = & \text{(imp (exists } (\lambda \text{x. A x)) B)} \\ & = & \text{(imp (exists } (\lambda \text{x. imp (p x) (p x))) q)} \end{array}$$

- Instantiations for meta-variables contain no free ordinary variables![Huet75]

No closed instantiation of meta-variables

- Unification problem

$$\text{eq (all } (\lambda \text{ y. imp q (imp (p y) (p y)))) C}$$
$$\doteq$$
$$\text{eq (all } (\lambda \text{ x. imp (A x) B)) (imp (exists } (\lambda \text{x. A x)) B)}$$

- No solution:

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

FAILURE

Questions concerning meta-variables

1. Which parameters are allowed to occur in a term that instantiates a meta-variable.

Questions concerning meta-variables

1. Which parameters are allowed to occur in a term that instantiates a meta-variable.
2. What constraints does the foundation impose on
   - occurrences of meta-variables in the contexts
   - types of other meta-variables.

Questions concerning meta-variables

1. Which parameters are allowed to occur in a term that instantiates a meta-variable.
2. What constraints does the foundation impose on
   - occurrences of meta-variables in the contexts
   - types of other meta-variables.
3. How do we implement meta-variables and the instantiation operation.

Questions concerning meta-variables

1. Which parameters are allowed to occur in a term that instantiates a meta-variable.
2. What constraints does the foundation impose on
   - occurrences of meta-variables in the contexts
   - types of other meta-variables.
3. How do we implement meta-variables and the instantiation operation.
4. Which algorithm do we use for unification or constraint simplification.

Modal logic and type theory

- Distinguish between truth and validity[Pf,Davies'01]
- Two basic judgments

  A true : proposition A is true

  A valid : proposition A is valid

  A is true in any world

- Two contexts – two kinds of variables:

  Γ : $x_1{:}A_1\mathsf{true}, \ldots x_n{:}A_n\mathsf{true}$

  Δ : $u_1{::}B_1\mathsf{valid}, \ldots u_k{::}B_k\mathsf{valid}$

Validity and truth

- Hypothetical judgment: $\Delta; \Gamma \vdash C$ true

  $\Delta; \Gamma \vdash C$ valid

- Definition of validity: $$\frac{\Delta; \cdot \vdash A \text{ true}}{\Delta; \Gamma \vdash A \text{ valid}}$$

- Hypothesis rule: $$\frac{}{(\Delta, u{::}A \,\text{valid}, \Delta'); \Gamma \vdash A \text{ true}}$$

- Substitution principle for validity:

  If $\Delta; \cdot \vdash A$ true and $(\Delta, u{::}A \,\text{valid}, \Delta'); \Gamma \vdash C$ true
  then $(\Delta, \Delta'); \Gamma \vdash C$ true.

Contextual validity

- Validity: A valid $\Longleftrightarrow$ A is true in any world
- Contextual validity: A *is valid relative* Ψ
  if A is true in every world in which Ψ is true

$$A \text{ valid}[\underbrace{y_1{:}B_1 \text{true}, \ldots, y_n{:}B_m \text{ true}}_{\Psi}]$$

- Terminology: Ψ : context
  $A[\Psi]$: contextual validity
- Generalization of validity

Definition of contextual validity

- Definition of contextual validity

$$\frac{\Delta; \Psi \vdash A \;\mathsf{true}}{\Delta; \Gamma \vdash A \;\mathsf{valid}[\Psi]}$$

- Contextual Hypothesis Rule

$$\frac{(\Delta, u{::}A\,\mathsf{valid}[\Psi], \Delta'); \Gamma \vdash \Psi}{(\Delta, u{::}A\,\mathsf{valid}[\Psi], \Delta'); \Gamma \vdash A \;\mathsf{true}}\;\mathsf{ctxhyp}_u$$

- Contextual Entailment.

$$\frac{\Delta; \Gamma \vdash B_1 \;\mathsf{true} \quad \ldots \quad \Delta; \Gamma \vdash B_m \;\mathsf{true}}{\Delta; \Gamma \vdash y_1{:}B_1 \;\mathsf{true}, \ldots, y_m{:}B_m \;\mathsf{true}}\;\mathsf{ctx}$$

Meta-theoretical Properties

- Contextual substitution principle:

  If $\Delta; \Psi \vdash A$ true and $(\Delta, u{::}A \text{ valid}[\Psi], \Delta'); \Gamma \vdash C$ true then $(\Delta, \Delta'); \Gamma \vdash C$ true.

- Internalize modality via introduction and elimination rules
- Meta-theoretic properties:
  - Locally sound and complete
  - Cut-elimination for ordinary and contextual cut

Towards a contextual modal type-theory

- Terms $M ::= x \mid u[\sigma] \mid \lambda x.M \mid M_1\ M_2$
- Ordinary hypothesis rule:

$$\frac{}{\Delta; (\Gamma, x{:}A, \Gamma') \vdash x : A}\ \mathsf{hyp}$$

- Contextual hypothesis rule:

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

Contextual modal type theory

- Modal variables u defined in a modal context Δ
  - $u::A[\Psi]$: term which may refer to ordinary variables in Ψ
  - modal variables = meta-variable
- Ordinary variables x defined in a context Γ
  - $x{:}A$ stands for any term
  - ordinary variables can be bound by lambda

Modal substitutions

$$
\begin{array}{lcl}
[\![M/u]\!](x) & = & x \\
[\![M/u]\!](\lambda y{:}B.\ N) & = & \lambda y{:}B.\ [\![M/u]\!]N \\
[\![M/u]\!](N_1\ N_2) & = & ([\![M/u]\!]N_1)\ ([\![M/u]\!]N_2) \\
[\![M/u]\!](u[\tau]) & = & [[\![M/u]\!]\tau]M \\
[\![M/u]\!](v[\tau]) & = & v[[\![M/u]\!]\tau] \\
 & & \text{provided } v \neq u
\end{array}
$$

- No side condition on the λ-rule!
- Modal substitution allows in place update!

Parameter occurences

$$\frac{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash \sigma : \Psi}{\Delta, u{::}A[\Psi], \Delta'; \Gamma \vdash u[\sigma] : [\sigma]A} \ \mathsf{mvar}$$

A meta-variable u can depend exactly on the ordinary variables in Ψ.

Meta-variable occurrences

$$\frac{}{\vdash \cdot \ \mathsf{mctx}}$$

$$\frac{\vdash \Delta \ \mathsf{mctx} \quad \Delta \vdash \Psi \ \mathsf{ctx} \quad \Delta; \Psi \vdash A : \mathsf{type}}{\vdash (\Delta, u{::}A[\Psi]) \ \mathsf{mctx}}$$

- There must be a linear order for meta-variables.
- Clarifies dependencies among meta-variables in the dependently typed case.

Meta-theoretic properties

- Subject reduction and expansion.
- Strong normalization.
- Lowering and raising.
- Bi-directional type-checking decidable.

Related Work: meta-variables

- Explicit substitution calculi
  - Simple types[Dowek'95, Dowek'96], Dependent types[Munoz'01, 00]
  - Assosciate explicit substitutions with any term
  - Pre-cooking and grafting
- Calculus with meta-variables [Strecker'99]
  - Meta-variables are associated with substitutions
  - No context for meta-variables
  - Decidability of type-checking not obvious.

Related Work

- Incomplete proofs (or Proofs with holes) [Magnusson' 95,Geuvers'02,Jojgov'02,Bognar'01]
  - Holes in proofs = meta-variables
  - Build on explicit substitution calculi
  - Reduction and instantiation of meta-variables do not commute
- Calculus of meta-variables [Sato'03]
  - Meta-variables are associated with levels
  - Textual substitution with capture
  - Loss of confluence and decidability of type-checking.

Contribution

- Logical foundation for calculi with meta-variables
  ICML (Nanevski, Pfenning, Pientka'05)
  (earlier version LFM'03)
- Applications:
  - Meta-variables (Theorem proving)
  - Staged computation (Functional programming)
  - Reasoning about different view-points/contexts (AI)

Outline

- Logical frameworks and certified code
- Contextual modal type theory
- Applications: Higher-order unification
- Conclusion and future work

Higher-order unification

- Solving equations in the presence of λ-abstraction
- Undecidable for second order [Huet'73] [Goldfarb'81]
- Central in higher-order logic and type theory
  - General proof search
  - Logic program execution
  - Type and term reconstruction
  - Partial proofs

Tractable cases

- Pre-unification often practical [Huet'75]
  - Some solvable equations are postponed
  - Non-determinsm major drawback in practice
- Pattern unification decidable [Miller'91]
  - Restricting β-reduction to $\beta_0 : (\lambda x.M)y \longrightarrow [y/x]M$
  - Most general unifiers exist.
  - Extends to complex theories [Pf'91]
  - Higher-order patterns as a calculus of variable binding, variable occurrence and renaming

Example revisited

- Unification problem:

$$\text{eq (all } (\lambda \text{ y. imp (imp (p y) (p y))) q) } w[\cdot]$$
$$\doteq$$
$$\text{eq (all } (\lambda \text{ y. imp } u[y/x]\ v[\cdot]))\quad \text{(imp (exists } (\lambda \text{ y. } u[y/x]))\ v[\cdot])$$

- Meta-variables: Δ = u::o[x:i], v::o[·], w::o[·]
- Solve sub-problems:

$$\Delta \text{ ; y:o} \quad \vdash \quad u[y/x] \doteq \text{(imp (p y) (p y))}$$
$$\Delta \text{ ; y:o} \quad \vdash \quad v[\cdot] \doteq q$$
$$\Delta \text{ ; y:o} \quad \vdash \quad \text{(imp (exists } (\lambda \text{ y. } u[y/x]))\ v[\cdot]) \doteq w[\cdot]$$

Solving higher-order patterns

- Higher-order patterns:

  Terms $M \quad ::= \quad x \mid u[\sigma] \mid \lambda x{:}A.M \mid M_1\ M_2$

  Subst. $\sigma \quad ::= \quad \cdot \mid \sigma, y/x$

- Judgment: $\Delta; \Gamma \vdash M \doteq N/(\theta, \Delta')$

- To solve : Δ ; y:o $\vdash$ u[y/x] $\doteq$ (imp (p y) (p y))

  where Δ = u::o[x:i], v::o[$\cdot$], w::[$\cdot$]

  - Check for occurrences of u (occurs-check)
  - Check that $[y/x]^{-1}$ (imp (p y) (p y)) exists (pruning)
  - Substitute (with apparent capture): $[\![$(imp (p x) (p x)) / u$]\!]$

Occcurs check

Case: $(\Delta_1,\ u{::}Q[\Psi],\ \Delta_2); \Gamma \vdash u[\sigma] \doteq M / \ldots$

- $\Delta; \Gamma \vdash M : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : Q'$
- Can u occur in Q'?

Occcurs check

Case: $(\Delta_1,\ u{::}Q[\Psi],\ \Delta_2); \Gamma \vdash u[\sigma] \doteq M / \ldots$

- $\Delta; \Gamma \vdash M : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : [\sigma]Q$
- $\Delta; \Gamma \vdash \sigma : \Psi$. by rule

Occcurs check

Case: $(\Delta_1,\ u{::}Q[\Psi],\ \Delta_2); \Gamma \vdash u[\sigma] \doteq M / \ldots$

- $\Delta; \Gamma \vdash M : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : [\sigma]Q$
- $\Delta; \Gamma \vdash \sigma : \Psi$. — by rule
- $Q' = [\sigma]Q$ — by previous lines

Occcurs check

Case: $(\Delta_1,\ u{::}Q[\Psi],\ \Delta_2); \Gamma \vdash u[\sigma] \doteq M / \ldots$

- $\Delta; \Gamma \vdash M : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : Q'$
- $\Delta; \Gamma \vdash u[\sigma] : [\sigma]Q$
- $\Delta; \Gamma \vdash \sigma : \Psi$. by rule
- $Q' = [\sigma]Q$ by previous lines
- $\Delta_1; \Psi \vdash Q$: type well-typed
- $\Delta_1; \Gamma \vdash [\sigma]Q$: type since σ is a pattern substitution

Occurs check

Case: $(\Delta_1,\ u{::}Q[\Psi],\ \Delta_2);\Gamma \vdash u[\sigma] \doteq M/\ \ldots$

- $\Delta;\Gamma \vdash M : Q'$
- $\Delta;\Gamma \vdash u[\sigma] : Q'$
- $\Delta;\Gamma \vdash u[\sigma] : [\sigma]Q$
- $\Delta;\Gamma \vdash \sigma : \Psi$. by rule
- $Q' = [\sigma]Q$ by previous lines
- $\Delta_1;\Psi \vdash Q$: type well-typed
- $\Delta_1;\Gamma \vdash [\sigma]Q$: type since σ is a pattern substitution

No occurs check on Q' necessary!

Unification with a meta-variable

Case: $\Delta; \Gamma \vdash u[\sigma] \doteq M / \ldots$
where $\Delta = \Delta_1,\ u{::}Q[\Psi],\ \Delta_2$

Prune M with respect to σ s.t. (u does not occur in M)

- ρ is a (modal) pruning substitution s.t. $[\sigma]^{-1}\left([\![\rho]\!]M\right)$ exists
- $\Delta' \vdash \rho : \Delta$

Unification with a meta-variable

Case: $\Delta; \Gamma \vdash u[\sigma] \doteq M / \ldots$
where $\Delta = \Delta_1,\ u{::}Q[\Psi],\ \Delta_2$

Prune M with respect to σ s.t. (u does not occur in M)

- ρ is a (modal) pruning substitution s.t.$[\sigma]^{-1}([\![\rho]\!]M)$ exists
- $\Delta' \vdash \rho : \Delta$ and $\rho = (\rho_1, u/u, \rho_2)$
- $\Delta' = (\Delta_1', u{::}[\![\rho]\!]Q[[\![\rho]\!]\Psi], \Delta_2')$

Unification with a meta-variable

Case: $\Delta; \Gamma \vdash u[\sigma] \doteq M / (\theta, \Delta^*)$
where Δ = $\Delta_1,\ u{::}Q[\Psi],\ \Delta_2$

Prune M with respect to σ s.t. (u does not occur in M)

- ρ is a (modal) pruning substitution s.t. $[\sigma]^{-1}\left([\![\rho]\!]M\right)$ exists
- $\Delta' \vdash \rho : \Delta$ and $\rho = (\rho_1, u/u, \rho_2)$
- $\Delta' = (\Delta_1', u{::}[\![\rho]\!]Q[[\![\rho]\!]\Psi], \Delta_2')$
- $$\begin{array}{rcl} \theta & = & (\rho_1, [\sigma]^{-1}\left([\![\rho]\!]M\right)/u, \rho_2) \\ & = & (\mathsf{id}_{\Delta_1'}, [\sigma]^{-1}\left([\![\rho]\!]M\right)/u, \mathsf{id}_{\Delta_2'})\rho \end{array}$$
- $\Delta^* = (\Delta_1', [\![\mathsf{id}_{\Delta_1'}, [\sigma]^{-1}\left([\![\rho]\!]M\right)/u]\!]\Delta_2')$

Meta-theoretic properties [Pi'03]

- Modal substitutions and pattern substitutions commute.
- Correctness of pruning for dependent types.
  - Pruning of types or context not necessary.
  - Occurs check on types not necessary.
- Correctness of higher-order pattern unification for dependent types.
  - Well-typed simultanous substitutions for meta-variables

Contributions

- High-level description of higher-order pattern unification for dependent types
  no de Bruijn indices
- Logical foundation based on modal type-theory:
  - Meta-variables = modal variables
  - Strong invariants about modal and ordinary variables.
- Post-hoc justification of implementation in Twelf
- Insights into optimizations:
  - Linearization [Pi'03]
  - Omitting redundant dependent types (current work)

Outline

- Logical frameworks and certified code
- Contextual modal type theory
- Applications: Higher-order unification
- Conclusion and future work

Summary and future work

- Contextual modal type theory
  Foundation for meta-variables and explicit substitutions
- High-level explanation (no de Bruijn indices!)
- Basis for other algorithms:
  - Higher-order term indexing [Pi'03]
  - Proof search [Pie'02]
  - Redundant type elimination (current work)

Future work

- Further development of contextual modal logic
  - Contextual possibility
  - Dependent necessity
  - Internalizing explicit substitutions
- Applications to logical frameworks:
  - Omitting redundant dependent types
  - General higher-order unification
  - Variable definitions