In Memoriam

I’d like to dedicate my talk to the memory of Gilles Kahn whose life and work are an inspiration to me and to the many people whose lives he touched.
Acknowledgements

Thanks to my collaborators:

- Michael Ashley-Rollman, Daniel Lee, Susmit Sarkar
- Karl Crary, Frank Pfenning

Thanks to UTexas for the invitation!
Language Definitions

What does it mean for a programming language to exist?

The “standard” answer is exemplified by C.

- Informal description (a la K&R, say).
- A “reference” implementation (gcc, say).
- Social processes such as standardization committees.
Language Definitions

- The PL research community has developed better definitional methods.
  - Classically, various grammatical formalisms, denotational and axiomatic semantics.
  - Most successfully, type systems and operational semantics.
- Nearly all theoretical studies use these methods! (e.g., every other POPL paper)
Language Definitions

What good is a language definition?

- Precise specification for programmers.
- Ensures compatibility among compilers.
- Admits rigorous analysis of properties.

The Definition of Standard ML has proved successful in these respects!
Language Definitions

But a language definition is also a burden!

- Someone has to maintain it.
- Not easy to make changes.
- Definitions can be mistaken too!
- Internally incoherent.
- Difficult or impossible to implement.
Language Definitions

- A definition alone is not enough! Must maintain a body of meta-theory as well.

- Type safety: coherence of static and dynamic semantics.

- Decidability of type checking, determinacy of execution, ....

- Developing and maintaining the meta-theory is onerous.
Mechanized Definitions

- Can we alleviate some of the burden through mechanization?
- Formalize the definition in a logical framework.
- Automatically or semi-automatically verify key meta-theoretic properties.
- Can we do this at scale?
Formalizing Languages

This talk is about using Twelf to

- Formalize language definitions.
- Reason about their meta-theory.

Several groups are using Coq and Isabelle for similar purposes.

It’ll be interesting to compare results.
What I’ve Learned

- Twelf is a very convenient and effective tool for mechanized meta-theory.
  - Natural, pattern-matching style of presentation.
  - Easy to state and verify simple, but informative, invariants.
  - A “type system” for language definitions: simple sanity checks are powerful!
What I’ve Learned

- One cannot (and should not) expect a “waterfall” process.
- Definitional technique is influenced by the demands of mechanization.
- Mechanization process uncovers mistakes, ambiguities, infelicities in the language.
- LF tends to enforce good hygiene (rather than require contortions).
What I’ve Learned

- LF and Twelf are not the last word!
- The methodology is robust and likely to remain useful.
- There is a clear path to improvement (e.g., linearity, structural congruences).
- But it will never be everything to everyone.
LF Methodology

The LF language is a dependently typed λ-calculus in the AUTOMATH family.

The type \( \{x:A\}B(x) \) consists of functions \([x:A]M(x)\) such that \(x:A \vdash M(x) : B(x)\).

Parameterized by constants that generate the types and terms.

Paradoxically, the weakness of LF is the source of its strength!
LF Methodology

- Encode syntactic categories as types.
- abstract syntax, including binding and scoping conventions
- typing derivations
- evaluation derivations
- Populate types by terms corresponding to these entities.
Encoding $\lambda$-Calculus

- Syntactic categories for typed $\lambda$-calculus:
  - $\tau$ type, $\tau$ is a type
  - $e$ term, $e$ is a term
  - $e : \tau$, $e$ has type $\tau$
  - $e \mapsto e'$, $e$ steps to $e'$
Encoding \( \lambda \)-Calculus

- Judgements are represented as types.
  
  \[ \text{tp} : \text{type}. \]
  \[ \text{tm} : \text{type}. \]
  \[ \text{of} : \text{tp} \to \text{tm} \to \text{type}. \]
  \[ \text{step} : \text{tm} \to \text{tm} \to \text{type}. \]

- The elements are **derivations** of judgements.
Abstract Syntax

nat : tp.

arrow : tp -> tp -> tp.

\[
\begin{array}{c}
\text{nat type} \\
\hline \\
\text{\(\tau_1\) type \quad \(\tau_2\) type} \\
\hline \\
\text{\(\tau_1 \rightarrow \tau_2\) type}
\end{array}
\]
Abstract Syntax

\[ \text{lam} : \tau \rightarrow (\text{tm} \rightarrow \text{tm}) \rightarrow \text{tm}. \]

\[ \text{app} : \text{tm} \rightarrow \text{tm} \rightarrow \text{tm}. \]
Abstract Syntax

\[ \text{lam} : \text{tp} \rightarrow (\text{tm} \rightarrow \text{tm}) \rightarrow \text{tm}. \]
\[ \text{app} : \text{tm} \rightarrow \text{tm} \rightarrow \text{tm}. \]
Abstract Syntax

\[ \text{lam} : \tau \text{p} \to (\text{tm} \to \text{tm}) \to \text{tm.} \]

\[ \text{app} : \text{tm} \to \text{tm} \to \text{tm.} \]

"hoas"
Typing Rules

\[
\frac{e_1 : \tau \rightarrow \sigma \quad e_2 : \tau}{\text{ap}(e_1, e_2) : \sigma}
\]

\text{of}_{\text{app}} : \\
\text{of E1 (arr T U)} \rightarrow \text{of E2 T} \rightarrow \text{of (app E1 E2) U.}
Typing Rules

\[ \frac{x: \sigma \vdash e : \tau}{\lambda(x: \sigma.e) : \sigma \to \tau} \]

of_lam :

\[
\{x : \text{tm}\}{dx : \text{of } x \ T} \ \text{of} \ (E \ x) \ U) \rightarrow
\text{of} \ (\text{lam} \ T \ E) \ (\text{arr} \ T \ U).
\]
Evaluation

\[
\text{ap}(\lambda(x: \tau. e_2), e_1) \mapsto [e_1/x] e_2
\]

\text{beta}:
step (app (lam T F) E) (F E).
Evaluation

\[
\text{ap}[\lambda(x: \tau. e_2), e_1] \mapsto [e_1/x]e_2
\]

\text{substitution}

\text{beta : step (app (lam T F) E) (F E).}
Evaluation

\[
\text{ap}[\lambda(x:\tau.e_2), e_1] \mapsto [e_1/x]e_2
\]

\[
\text{beta: step} \ (\text{app} \ (\text{lam} \ T \ F) \ E) \ (F \ E).
\]
Evaluation

\[
\text{ap}(\lambda(x:\tau.e_2), e_1) \mapsto [e_1/x]e_2
\]

\text{beta :}
\text{step (app (lam T F) E) (F E).}
LF Methodology: Adequacy

- Establish a compositional bijection between objects of each syntactic category of object language
- Canonical forms of associated types of the LF $\lambda$-calculus
- "Compositional" means "commutes with substitution" (semantics of variables).
## Adequacy Theorem

<table>
<thead>
<tr>
<th>Cat’y</th>
<th>Rep’n</th>
<th>Contexts/World</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$ type</td>
<td>$T : \text{tp}$</td>
<td></td>
</tr>
<tr>
<td>$e$ term</td>
<td>$E : \text{tm}$</td>
<td>$x : \text{tm}$</td>
</tr>
<tr>
<td>$e : \tau$</td>
<td>$D : \text{of } E \text{ T}$</td>
<td>$x : \text{tm}$, $dx : \text{of } x \text{ U}$</td>
</tr>
</tbody>
</table>
Meta-Reasoning

- Adequacy ensures that the object language “exists” as canonical forms of LF type.
- Canonical forms are long $\beta\eta$ normal forms.
- Prove theorems about canonical forms.
- Simultaneous and iterated structural induction over canonical forms.
- Applies to informal and formal reasoning!
Meta-Reasoning in Twelf

Twelf can find proofs of $\forall \exists$ prop's over canonical forms in specified contexts (world).

   Enough for preservation, progress, ... 

These are totality assertions for a relation between inputs ($\forall$) and outputs ($\exists$).

   Thus totality checking is central!
Relational Meta-Theory
Relational Meta-Theory

Preservation Theorem as a relation:
\[ \text{pres} : \text{of} \ E \ T \rightarrow \text{step} \ E \ E' \rightarrow \text{of} \ E' \ T \rightarrow \text{type}. \]

Axiomatize this relation:
\[ \text{pres}_\beta : \]
\[ \text{pres} \ (\text{of}_\text{app} \ (\text{of}_\text{lam} \ D) \ D') \]
\[ \text{beta} \]
\[ (D \_ \ D'). \]
Relational Meta-Theory

Preservation Theorem as a relation:
\[ \text{pres} : \text{of } E \ T \rightarrow \text{step } E \ E' \rightarrow \text{of } E' \ T \rightarrow \text{type.} \]

Axiomatize this relation:
\[ \text{pres}_\beta : \]
\[ \text{pres} (\text{of}_\text{app} (\text{of}_\text{lam} D) D') \]
\[ \beta \]
\[ (D \_ D'). \]
Relational Meta-Theory

- Each axiom corresponds to one case of the inductive proof of preservation.
- Induction is on derivation of transition.
- One case for each transition rule.
- Natural pattern-matching style.
- The collection of these axioms constitute the inductive proof!
Why do these axioms constitute a proof?

Type checking ensures that the assumptions entail the conclusion.

Coverage and totality checking ensures that all cases are covered.

A (constructive) proof of preservation defines a function from derivations to derivations.
Relational Meta-Theory

Twelf can verify the totality of the relation corresponding to the meta-theorem.

- Specify the contexts to consider.
- Specify input/output mode of the relation.
- Specify induction principle to use.

Checks that all cases are covered, and induction is used appropriately.
Relational Meta-Theory
Relational Meta-Theory

For preservation this consists of declaring

```
%mode pres +D1 +D2 -D3.
%worlds () (pres _ _ _).
%total D (pres _ D _).
```

Twelf performs a mode check, world check, and a coverage and termination check.

Very similar to ML exhaustiveness check.
Relational Meta-Theory

For preservation this consists of declaring

%mode pres +D1 +D2 -D3.
%worlds () (pres _ _ _).
%total D (pres _ D _).

Twelf performs a mode check, world check, and a coverage and termination check.

Very similar to ML exhaustiveness check.
Relational Meta-Theory

For preservation this consists of declaring

%mode pres +D1 +D2 -D3.
%worlds () (pres _ _ _).
%total D (pres _ D _).

Twelf performs a mode check, world check, and a coverage and termination check.

Very similar to ML exhaustiveness check.
Relational Meta-Theory

For preservation this consists of declaring

%mode pres +D1 +D2 -D3.
%worlds () (pres _ _ _).
%total D (pres _ D _).

Twelf performs a mode check, world check, and a coverage and termination check.

Very similar to ML exhaustiveness check.
Relational Meta-Theory

For preservation this consists of declaring

\%mode pres +D1 +D2 -D3.
\%worlds () (pres _ _ _).
\%total D (pres _ D _).

Twelf performs a mode check, world check, and a coverage and termination check.

Very similar to ML exhaustiveness check.
For simple examples like this we can easily prove progress and preservation.

Formulate the inductive steps using pattern matching.

Check coverage and totality.

Substitution, weakening, contraction are all provided “free” by the framework.
Life’s Not Always Easy

- It’s not always so straightforward!
- Some features present challenges.
- Often the challenges uncover implicit structure or expose design problems.
- Sometimes you just suffer.
- But you can get surprisingly far with a bit of experience and ingenuity.
POPLmark Challenge

Pierce, et al. at Penn and Cambridge posed a challenge problem for the field.

see TPHOL's 2005 paper

Mechanically verify the meta-theory of languages typically found in a POPL paper.

eg, meta-theory of F<: (polymorphism with subtyping).
Reasoning About Variables

Quite often one wishes to prove a meta-theorem about the behavior of variables.

- eg, substitution preserves typing
- eg, narrowing a variable to a subtype

Since the context is represented only implicitly in LF, these can be a bit tricky.

- eg, POPLmark challenge for F<:
For example, why does this type ...

\[ \{ \{ x : \text{tm} \} \{ dx : \text{assm x T} \} \text{ of } (F \, x) \, U \} \rightarrow \]
\[ \text{of } E \, T \rightarrow \text{of } (F \, E) \, U \rightarrow \text{type.} \]

... codify this substitution principle?

if \( G, x : T, G' \models F : U \) and \( G \models E : T \),

then \( G, G' \models [E/x]F : U \)
Reasoning About Variables

- The key is permutation, which permits us to regard $G, x : T, G'$ as $G, G', x : T$ in STLC.
- If permutation is available, it is easy to prove properties of variables.
- Any given variable may be thought of as occurring “last”.
- But what if we don’t have permutation?
Reasoning About Variables

From the POPLmark challenge:

Stated relationally,
narrow :

$(\{X : \text{tp}\} \{dX : \text{assm} X Q\} \text{sub} A B) \rightarrow$
sub $P Q \rightarrow$
$(\{X : \text{tp}\} \{dX : \text{assm} X P\} \text{sub} A B) \rightarrow$
type.
Reasoning About Variables

But this statement cannot be proved!

Descending into a binder introduces an additional assumption, say $Y <: X$.

Cannot permute $Y <: X$ before $X <: Q$!

So we must consider a general $G'$, which cannot be done uniformly in LF.

The context $G'$ is not a “single thing”.
Reasoning About Variables

Adequacy for $F_\eta$ is for worlds of the form $X : \text{tp}, \; dX : \text{assm } X \; T$

For example,

$\text{tlam\_of} :$

$\lambda ( \{ X : \text{tp} \} \{ dX : \text{assm } X \; T \} ) \rightarrow$

$\lambda ( \text{of} \; (F \; X) \; (U \; X) ) \rightarrow$

$\lambda ( \text{of} \; (\text{tlam } T \; F) \; (\text{all } T \; U))$. 
Reasoning About Variables

We cannot, in general, permute such pairs past one another due to dependencies.

But, a limited form of permutation is OK:

\[
\{ X : \text{tp} \} \{ Y : \text{tp} \} \\
\{ dY : \text{assm} Y X \} \{ dX : \text{assm} X P \}
\]

The strategy is to permit “mixed” permutations so that an \text{assm} can be last!
Reasoning About Variables

Revised relational statement of narrowing permits $X$ to be separated from $dX$:

$$\{X:tm\} (\{dX : \text{assm} X Q\} \text{ sub } A B) \rightarrow \text{sub } P Q \rightarrow (\{dX : \text{assm} X P\} \text{ sub } A B) \rightarrow \text{type.}$$

But now $\text{assm} X Q$ no longer ensures that $X$ is a variable!
Reasoning About Variables

We “tag” each variable and “link” it to an assumption:

\[
\begin{align*}
\text{var} : \text{tm} \rightarrow \text{type}. \\
\text{assm}_\text{var} : \text{assm} \ X \ T \rightarrow \text{var} \ X \rightarrow \text{type}. \\
\text{%mode} \ \text{assm}_\text{var} +\text{D1} -\text{D2}.
\end{align*}
\]

Consider context blocks of these forms:

\[
\begin{align*}
\text{X} : \text{tp}, \ \text{vX} : \text{var} \ X \\
\text{dX} : \text{assm} \ X \ T, \ \text{dvX} : \text{assm}_\text{var} \ dX \ vX
\end{align*}
\]
Solving POPLmark

This was the hardest problem in the POPLmark challenge!

The rest was handled easily using standard methods with no serious complications.

This solution is a simplification of another that was much harder.

We solved the challenge in about a week!
Scaling Up

- We use Twelf daily to verify metatheory.
- Type safety for experimental languages.
- Compiler transformation verification.
- Full-scale certification infrastructure for ConCert trust-free grid.
- But does it scale to “real” languages?
Case Study: SML

- We are currently using these techniques to verify the type safety of Standard ML.
- Full scale higher-order language.
- Rich modularity mechanisms.
- Work is in progress as we speak!
Scaling Up To SML

A full-scale language introduces many complications, such as:

- Scope resolution for variables.
- Type inference and reconstruction.
- Pattern compilation.
- Equality compilation.

Can we scale up our methods to SML?
Scaling Up To SML

These complexities have significantly impeded mechanization.

To my knowledge, there is no complete proof of soundness of Standard ML!

The Definition of Standard ML is not as amenable to proof as one might expect.

e.g., van Inwegen's effort with HOL.
A Type-Theoretic Definition of SML

- Formalize the elaboration of SML into a well-behaved type theory.
  - Type inference, equality compilation, pattern compilation, signature matching.
  - Takes care of the “conveniences” of ML.
- Use Twelf to prove safety of the target.
- Using methods sketched above.
A Type-Theoretic Definition of SML

- Elaboration judgements (schematic):
  \[ \mathcal{E} \vdash \text{exp} \Rightarrow e : \tau \]

- Elaboration context includes typing context, plus a lot of bookkeeping data.

- Reconstructs types.

- Resolves identifiers, etc.
A Type-Theoretic Definition of SML

- Internal language based on CMU PhD’s by Lillibridge, Stone, Dreyer.

- Type theory of modularity that accounts for crucial concept of type sharing.

- Dynamic semantics is defined by SOS rules on the internal language.

- Much as above, but with stores, exceptions, modules.
Mechanizing the Meta-Theory of SML

The meta-theory decomposes into:

- Elaborated programs are well-typed.
- Well-typed programs are safe.

The hypothesis is that this approach will ease mechanization.

Experience so far bears this out.
Scaling Up To SML

Current state of development:

- Progress for the IL largely finished.
- Preservation for the IL in progress.
- Elaboration remains “to do”.

- Making steady progress week-by-week.
- But it’s not all smooth sailing either!
Scaling Up To SML

- We’ve overcome a few obstacles.
- explicit store and label management
- had to re-formulate dynamic semantics
- And uncovered a few bugs in type system.
- Missing rules, missing premises.
- An unsound typing rule!
Scaling Up To SML

One serious sticking point is type equality!

- Defined type constructors.
- Type sharing specifications.
- Progress theorem requires injectivity.

eg, if $A \rightarrow B = A' \rightarrow B'$, then $A = A'$ and $B = B'$.

Non-trivial for a “declarative” formulation.
Scaling Up To SML

- Our solution is to use an “algorithmic” formulation of equality.
- What an implementation would do.
- Injectivity principles are immediate.
- But we leave open whether the algorithmic formulation is equivalent to the declarative.
- May (or may not) be beyond Twelf’s
Conclusions

Lots of meta-theory for language definitions can be readily mechanized today.

We do this routinely for small-scale languages and logics.

It's not yet clear whether we can scale up to languages such as SML.

No "show stoppers" so far, but we've had to make some compromises.
Conclusions

- A language definition must be formulated with the demands of mechanization in mind.
- Often good hygiene anyway.
- Culmination of years of research.
- What we need now are more experiments!
- Different languages.
- Different frameworks and tools.
Questions?

Twelf info:  
www.cs.cmu.edu/~twelf

POPLMark Challenge:  
www.cis.upenn.edu/proj/plclub/mmm