Real World Binding Structures

Susmit Sarkar

with Peter Sewell and Francesco Zappa Nardelli

Paradigm Binding

- Single binders

$$
\begin{array}{lcll}
exp & ::= & X & \\
 & \mid & \lambda\, X\ .\ exp & \text{bind } X \text{ in } exp \\
 & \mid & exp\ exp' &
\end{array}
$$

Paradigm Binding

- Single binders

$$
\begin{array}{lcll}
exp & ::= & X & \\
 & | & \lambda\, X \;.\; exp & \text{bind } X \text{ in } exp \\
 & | & exp\;\; exp' &
\end{array}
$$

- Lots of work on representations
  - deBruijn
  - HOAS
  - Locally nameless
  - Nominal
  - ...

How about: Patterns?

- Many binders

$$\textbf{let}\ (\ x\ ,\ \ y\ )\ =\ z\ \textbf{in}\ x\ y$$

How about: Patterns?

- Many binders

$$\mathbf{let}\ (\ x\ ,\ \ y\ )\ =\ z\ \mathbf{in}\ x\ y$$

$$
\begin{array}{lcll}
exp & ::= & X & \\
 & \mid & (\ exp\ ,\ \ exp'\ ) & \\
 & \mid & \mathbf{let}\ pat\ =\ exp\ \mathbf{in}\ exp' & \quad \text{bind } b(pat) \text{ in } exp' \\
pat & ::= & X & \\
 & \mid & \_ & \\
 & \mid & (\ pat\ ,\ \ pat'\ ) &
\end{array}
$$

How about: Patterns?

- Many binders

$$\textbf{let}\ (\ \color{red}{x}\ ,\ \ \color{blue}{y}\ )\ =\ z\ \textbf{in}\ \color{red}{x}\ \color{blue}{y}$$

$$\begin{array}{lcll}
exp & ::= & X & \\
 & \mid & (\ exp\ ,\ \ exp'\ ) & \\
 & \mid & \textbf{let}\ pat\ =\ exp\ \textbf{in}\ exp' & \text{bind } b(pat) \text{ in } exp' \\
pat & ::= & X & b = X \\
 & \mid & _ & b = \{\} \\
 & \mid & (\ pat\ ,\ \ pat'\ ) & b = b(pat) \cup b(pat')
\end{array}$$

How about: Let rec?

- Binding one variable in multiple scopes

$$\textbf{letrec}\ x\ =\ (\ x\ ,\ \ y\ )\ \textbf{in}\ (\ x\ ,\ \ y\ )$$

How about: Let rec?

- Binding one variable in multiple scopes

$$\textbf{letrec}\ x\ =\ (\ x\ ,\ \ y\ )\ \textbf{in}\ (\ x\ ,\ \ y\ )$$

$$
\begin{array}{lcll}
exp & ::= & x & \\
 & | & () & \\
 & | & (\ exp\ ,\ \ exp'\ ) & \\
 & | & \textbf{let rec}\ x\ =\ exp\ \textbf{in}\ exp' & \text{bind } x \text{ in } exp \\
 & & & \text{bind } x \text{ in } exp'
\end{array}
$$

How about: Or-patterns?

- A variable does not have a binding occurrence

$$
\begin{array}{l}
\textbf{let} \\
\qquad \left(\quad \begin{array}{ll} & (\textbf{None} \,,\, \textbf{Some}\; x\,) \\ \| & (\,\textbf{Some}\; x \,,\, \textbf{None}\,) \end{array} \quad \right) \;=\; w \\
\textbf{in} \\
\qquad\qquad (\, x \,,\, x \,)
\end{array}
$$

How about: Dependent Patterns?

- Binding within binders

$$
\begin{array}{l}
\textbf{let} \\
\quad \textbf{val}\,[X\ <:\ \textbf{top}, x\ :\ X\,]\ =\ w \\
\textbf{in} \\
\quad [\ X\ ,\ (\ x\ ,\ y\ )\ ]
\end{array}
$$

This work

- A language for binding structures
- What does it mean, mathematically?
- *What does it really mean, mechanically?*

Bindspec language annotations

$$
\begin{array}{l}
\mathit{element},\ e ::= \\
\quad |\ \ \mathit{terminal} \\
\quad |\ \ \mathit{metavar} \\
\quad |\ \ \mathit{nonterm} \\[1em]
\mathit{prod},\ p ::= \\
\quad |\ \ \texttt{|}\ \mathit{element}_1 \mathinner{..} \mathit{element}_m\ \texttt{::}\ \texttt{::}\ \mathit{prodname}\ \texttt{(+}\ \mathit{bs}_1 \mathinner{..} \mathit{bs}_n\ \texttt{+)} \\
\mathit{bindspec},\ \mathit{bs} ::= \\
\quad |\ \ \texttt{bind}\ \mathit{mse}\ \texttt{in}\ \mathit{nonterm} \\
\quad |\ \ \ldots
\end{array}
$$

Metavariable set expressions

- Bind arbitrary sets of metavariables in declared nonterminals

$$
\begin{array}{lll}
\multicolumn{3}{l}{metavar\_set\_expression,\ mse ::=} \\
 & | \ \texttt{\{\}} & \text{Empty} \\
 & | \ metavar & \text{Singleton} \\
 & | \ mse\ \texttt{union}\ mse' & \text{Union} \\
 & | \ auxfn\,\texttt{(}\,nonterm\,\texttt{)} & \text{Auxiliary function}
\end{array}
$$

Auxiliary Functions

- Collect some particular set of metavariables
- User-defined, primitive recursive functions
- Annotation of bindspec language

$$
\begin{aligned}
&bindspec,\ bs ::= \\
&\quad |\ \ldots \\
&\quad |\ auxfn\ =\ mse
\end{aligned}
$$

Example: Multiple Letrec

$$
\begin{array}{lcll}
exp & ::= & X & \\
 & | & \textbf{let rec } lrbs \textbf{ in } exp & (\!+\ \text{bind } b(lrbs) \text{ in } lrbs\ +\!) \\
 & & & (\!+\ \text{bind } b(lrbs) \text{ in } exp\ +\!) \\
lrb & ::= & X\ pat\ =\ exp & (\!+\ b = X\ +\!) \\
 & & & (\!+\ \text{bind } bpat(pat) \text{ in } exp\ +\!) \\
lrbs & ::= & lrb & (\!+\ b = b(lrb)\ +\!) \\
 & | & lrb \textbf{ and } lrbs & (\!+\ b = b(lrb) \cup b(lrbs)\ +\!) \\
pat & ::= & X & (\!+\ bpat = X\ +\!) \\
 & | & (\ pat\ ,\ pat'\ ) & (\!+\ bpat = bpat(pat) \cup bpat(pat')\ +\!)
\end{array}
$$

What does it mean?

- There is no notion of binding occurrence
  - Recall: binders collected by user-defined auxfns

What does it mean?

- There is no notion of binding occurrence
  - Recall: binders collected by user-defined auxfns
- Let us think about alpha-equivalence classes

Alpha-equivalence classes

- Concrete variables that must all vary together
- Relate by partial equivalence relations of occurrence of variables

Alpha-equivalence classes

- Concrete variables that must all vary together
- Relate by partial equivalence relations of occurrence of variables

$$
\begin{aligned}
&\textbf{let rec } f\ x \ = \ g\ (\ x - 1\ ) \\
&\quad \textbf{and } g\ x \ = \ f\ x + h\ x \\
&\quad \textbf{and } h\ x \ = \ 0 \\
&\textbf{in } (\ g\ 5\ )
\end{aligned}
$$

Alpha-equivalence classes

- Concrete variables that must all vary together
- Relate by partial equivalence relations of occurrence of variables

$$
\begin{aligned}
&\textbf{let rec } f\ x = g\ (\ x - 1\ ) \\
&\quad \textbf{and } g\ x = f\ x + h\ x \\
&\quad \textbf{and } h\ x = 0 \\
&\textbf{in } (\ g\ 5\ )
\end{aligned}
$$

- Alpha-equivalence is equivalence upto identity of these concrete variables

Calculating closed PER

- Calculated by induction on term structure

Calculating closed PER

- Calculated by induction on term structure
- Case: bind mse in nt annotation

$$exp \quad ::= \quad \textbf{let rec}\ lrbs\ \textbf{in}\ exp \qquad \text{(+ bind } b(lrbs) \text{ in } lrbs \text{ +)}$$

$$\text{(+ bind } b(lrbs) \text{ in } exp \text{ +)}$$

Calculating closed PER

- Calculated by induction on term structure
- Case: bind mse in nt annotation

$$
\begin{aligned}
exp \quad ::= \quad & \textbf{let rec}\ lrbs\ \textbf{in}\ exp \qquad & \textsf{(+ bind}\ b(lrbs)\ \textsf{in}\ lrbs\ \textsf{+)} \\
& & \textsf{(+ bind}\ b(lrbs)\ \textsf{in}\ exp\ \textsf{+)}
\end{aligned}
$$

- Collect relevant occurences of variables and relate them

$$
\begin{aligned}
&\textbf{let rec}\ f\ x\ =\ f\ (\ x\ -\ 1\ ) \\
&\textbf{in}\ f\ 4
\end{aligned}
$$

Calculating closed PER

- Calculated by induction on term structure
- Case: bind mse in nt annotation

$$exp \quad ::= \quad \textbf{let rec}\ lrbs\ \textbf{in}\ exp \qquad (+\ \mathsf{bind}\ b(lrbs)\ \mathsf{in}\ lrbs\ +)$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad (+\ \mathsf{bind}\ b(lrbs)\ \mathsf{in}\ exp\ +)$$

- Collect relevant occurences of variables and relate them

$$\textbf{let rec}\ f\ x\ =\ f\ (\ x\ -\ 1\ )$$
$$\textbf{in}\ f\ 4$$

- Seal the equivalence relation of all such variables (forget its identity)...

Open PER

- …but not always!
- Consider when there is binding within binding

$$\begin{array}{l}\textbf{let} \\ \quad \textbf{val}\,[\,X \;<:\; \textbf{top}, x \;:\; X\,] \;=\; w \\ \textbf{in}\,[\,X,\;\ldots]\end{array}$$

Open PER

- ...but not always!
- Consider when there is binding within binding

$$[X <: \mathbf{top}, x : X]$$

- Cannot forget the concrete variable (more binding possible)
- Syntactically analyze when safe to seal

Well-formed Substitution

- Defined over our alpha-equivalence classes
- Must avoid capture (PER's undisturbed)
- When substituting closed terms, cheap solution possible
  - Check for equality when descending binders
  - Clearly not what you want to use in general

What does it *Really* Mean?

- Proof assistant representations
- Translations to a proper alpha-equivalent representation: deBruijn, HOAS, locally nameless, nominal…
- Not clear how to translate the entire language

The way forward?

- Simple cases are easy
  - Single binders in one or more terms

The way forward?

- Simple cases are easy
  - Single binders in one or more terms
- Translate (almost) everything to single binders?
  - Possibly, cases without nested binding

The way forward?

- Simple cases are easy
  - Single binders in one or more terms
- Translate (almost) everything to single binders?
  - Possibly, cases without nested binding
- …without loss of expressiveness?

The way forward?

- Simple cases are easy
  - Single binders in one or more terms
- Translate (almost) everything to single binders?
  - Possibly, cases without nested binding
- …without loss of expressiveness?
- …making idiomatic proofs possible?

Related work

- Much work on single binders
- Rich binding specifications: FreshML, Cαml
  - Cαml: similar goals, but different expressivities
  - Alpha-equivalence classes coincides on large subset
  - Multiple auxiliary functions, or multiple binding occurences, in Cαml?
  - Bind only in some subterms in Ott bindspec?

Current and future work

- Mechanized rich theory of binding (mini-Ott in Ott)
- Showed correspondence with usual notions in simple cases
- Define a notion of correctness (aka adequacy)
- Want: a translation to a practical representation

Thank you!

`http://www.cl.cam.ac.uk/~pes20/ott`

Inexpressible binding

- Binding non-terminals in non-terminals

$$\begin{array}{l} \mathbf{let}\ x : \mathbf{bool}\ =\ e \\ \mathbf{in}\ (\ x : \mathbf{bool}, x : \mathbf{int}\ ) \end{array}$$

  - Note: It is handled in the implementation with concrete atoms
- First match patterns
  - First occurrence of variable in pattern is binding, others bound