Real World Binding Structures

Susmit Sarkar

with Peter Sewell and Francesco Zappa Nardelli

Paradigm Binding

- Single binders

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

Paradigm Binding

- Single binders

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

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

How about: Patterns?

- Many binders

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

How about: Patterns?

- Many binders

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

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

How about: Patterns?

- Many binders

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

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

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

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

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

bind x in exp

bind x in exp'

How about: Or-patterns?

- A variable does not have a binding occurrence

$$
\begin{array}{l}
\textbf{let} \\
\qquad \begin{array}{rl}
(& (\textbf{None}\ ,\ \textbf{Some}\ x\ ) \\
\| & (\ \textbf{Some}\ x\ ,\ \textbf{None}\ )\ \ )\ =\ w
\end{array} \\
\textbf{in} \\
\qquad (\ x\ ,\ x\ )
\end{array}
$$

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

$element,\ e ::=$

| $terminal$

| $metavar$

| $nonterm$

$prod,\ p ::=$

| | $element_1$ .. $element_m$ `:::` $prodname$ `(+` bs_1 .. bs_n `+)`

$bindspec,\ bs ::=$

| `bind` mse `in` $nonterm$

| ...

Metavariable set expressions

- Bind arbitrary sets of metavariables in declared nonterminals

$metavar\_set\_expression,\ mse ::=$

$\mid$ `{}`	Empty
$\mid$ $metavar$	Singleton
$\mid$ mse `union` mse'	Union
$\mid$ $auxfn$ `(` $nonterm$ `)`	Auxiliary function

Auxiliary Functions

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

$$
\begin{array}{l}
bindspec,\ bs ::= \\
\quad \mid \ \ldots \\
\quad \mid \ auxfn\ =\ mse
\end{array}
$$

Example: Multiple Letrec

exp	$::=$	X	
	$\mid$	**let rec** $lrbs$ **in** exp	(+ bind $b(lrbs)$ in $lrbs$ +)
			(+ bind $b(lrbs)$ in exp +)
lrb	$::=$	$X\ pat\ =\ exp$	(+ $b = X$ +)
			(+ bind $bpat(pat)$ in exp +)
$lrbs$	$::=$	lrb	(+ $b = b(lrb)$ +)
	$\mid$	lrb **and** $lrbs$	(+ $b = b(lrb) \cup b(lrbs)$ +)
pat	$::=$	X	(+ $bpat = X$ +)
	$\mid$	$(\ pat\ ,\ pat'\ )$	(+ $bpat = bpat(pat) \cup bpat(pat')$ +)

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

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

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

Calculating closed PER

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

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

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

- Collect relevant occurences of variables and relate them

$$\mathbf{let\ rec}\ f\ x\ =\ f\ (\ x\ -\ 1\ )$$

$$\mathbf{in}\ f\ 4$$

Calculating closed PER

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

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

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

- 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?

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

Inexpressible binding

- Binding non-terminals in non-terminals

$$\begin{aligned}&\textbf{let } x : \textbf{bool} \;=\; e\\ &\textbf{in } (\; x : \textbf{bool}, x : \textbf{int} \;)\end{aligned}$$

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