CS 524
Lecture 16
J. Meseguer

λ-CINNI : A User Friendly and Machine Friendly Lambda Calculus

The de Bruijn notation achieves three great advantages:

1. machine friendliness
2. canonical representation of α-equivalent terms
3. avoidance of "variable capture" in substitutions without need for creating new names.

However, as a notation, it is not as user-friendly as the standard λ-calculus. This raises the question:

> Can we have it both ways? That is,
> Can we have a Lambda Calculus notation that is user friendly and keeps the de Bruijn advantages (1)-(3)?

The answer is Yes! we can! This is what Mark-Oliver Stehr's CINNI Lambda notation provides.

We can motivate the idea of CINNI in two simple steps:

Step 1 Consider reading a DB λ-term like:

$$S = \lambda.\,\lambda.\,\lambda.\,(2\ 0)\ (1\ 0)$$

as:

$$S = \lambda x.\,\lambda x.\,\lambda x.\,(x_{\{2\}}\ x_{\{0\}})\ (x_{\{1\}}\ x_{\{0\}})$$

This is a fantastic idea! In fact, an idea impossible!

in standard λ-calculus, since

$$\lambda x.\ \lambda x.\ \lambda x.\ (x\ x)\ (x\ x)$$

is the DB λ-term: $\lambda.\ \lambda.\ \lambda.\ (0\ 0)\ (0\ 0)$

we can use a single name at any λ-depth, but can unambiguously identify at which depth each variable is bound by the index & in $x_{\{k\}}$.

Step 1 But why restricting to a single name? why not using any names we want as in the standard λ-calculus?

Q: Yes, but how about the indices $x_{\{n\}}$, $y_{\{m\}}$, $z_{\{k\}}$

A: $x_{\{n\}}$ now means:
we need to cross n λ-binders $\lambda x.-$ to get to the name x binding $x_{\{n\}}$

In particular, we can achieve the notation closest to the standard λ-calculus by never using the same name X in a λ-binder $\lambda X.-$

For example, we can write the standard

$$S = \lambda x.\ \lambda y.\ \lambda z.\ (x\ z)\ (y\ z)$$

as:

$$S = \lambda x.\ \lambda y.\ \lambda z.\ (x_{\{0\}}\ z_{\{0\}})\ (y_{\{0\}}\ z_{\{0\}})$$

But if we insist on the opposite idea of using a single name we of course get the equivalent expressions à la de Bruijn,

$$S_{\text{"DB"}} = \lambda x.\ \lambda x.\ \lambda x.\ (x_{\{2\}}\ x_{\{0\}})\ (x_{\{1\}}\ x_{\{0\}})$$

Q: In which sense "equivalent"?

A: In the sense of α-conversion, as this makes sense for CINNI.

Q: Yes, but we can of course have infinitely many α-equivalent terms. How can we obtain a unique representative among all of them, and therefore be able to compare results?

A: very simple: choose a fixed name, e.g. 'x' and get the unique repentative as the $S_{\text{"DB"}}$ term above in the case of S.
That is, unique representative of α-equivalance classes are now their de Bruijn-like representations with a fixed chosen name.

Of course, many intermediate possibilities, where some names are reused, and other names are used only once as possible. For example, the following are also λ-equivalent representations of S in λ-CINNI:

$$S = \lambda x.\ \lambda x.\ \lambda z.\ (x_{\{1\}}\ z_{\{0\}})\ (x_{\{0\}}\ z_{\{0\}})$$

$$S = \lambda x.\ \lambda y.\ \lambda y.\ (x_{\{0\}}\ y_{\{0\}})\ (y_{\{1\}}\ y_{\{0\}})$$

Q: How about substitution?

A: It is a straightforward generalization of DB substitution, namely, that the operators have to be generalized to become name aware. Therefore, if, say, the name y is 'y' we get operators:

DB	CINNI	
$[U]$	$[y := U]$	[substitution]
$[\uparrow]$	$[\uparrow y]$	[shift]
$[\uparrow \sigma]$	$[\uparrow_y \sigma]$	[lift]

where σ denotes a substitution.

The substitution equations are just the "name aware" ~~version~~ generalization of the DB substitution equations:

with X, Y ranging over Names, U V over λ-expressions and SU ranges over Substitutions, and where if(B, U, V) is an if-then-else operator we have:

~~eq[0]: [X := U] if(eq(X,Y), U,~~

eq[1]: $[X := U]\ Y_{\{0\}} = \text{if}(\text{eq}(X,Y), U, Y_{\{0\}})$.

eq[2]: $[X := U]\ Y_{\{s(m)\}} = \text{if}(\text{eq}(X,Y), Y_{\{m\}}, Y_{\{s(m)\}})$.

eq[3]: $[\lambda X]\ Y_{\{m\}} = \text{if}(\text{eq}(X,Y), Y_{\{s(m)\}}, Y_{\{m\}})$.

eq[4]: $[\lambda X\ SU]\ Y_{\{0\}} = \text{if}(\text{eq}(X,Y), Y_{\{0\}}, [\lambda X](SU\ Y_{\{0\}}))$.

eq[5]: $[\lambda X\ SU]\ Y_{\{s(m)\}} = \text{if}(\text{eq}(X,Y), [\lambda Y](SU\ Y_{\{m\}}), [\lambda X](SU\ Y_{\{s(m)\}}))$.

eq[6]: $SU\ (U\ V) = (SU\ U)\ (SU\ V)$.

eq[7]: $SU\ (\lambda X.\ U) = \lambda X.([\lambda X\ SU]\ U)$.

And β-reduction is the rule:

rl [beta]: $(\lambda X.\ U)\ V \rightarrow [X := V]\ U$.

We can illustrate how the β rule and the substitution equations work and how in CINNI the need for creating fresh names to avoid variable capture evaporates, by revisiting the following "variable capture trouble" example in Lecture 14.6:

$(\lambda x.\, \lambda y.\, (y\ x))\ (\lambda x.\, (x\ y)) \rightarrow_\beta$

$\lambda z.\, (z\ \ (\lambda x.\, (x\ y))$ with fresh variable z,

which in CINNI becomes:

$(\lambda x.\, \lambda y.\, (y_{\{0\}}\ x_{\{0\}}))\ (\lambda x.\, (x_{\{0\}}\ y_{\{0\}})) \rightarrow_\beta$

$[x := \lambda x.\, (x_{\{0\}}\ y_{\{0\}})]\ (\lambda y.\, (y_{\{0\}}\ x_{\{0\}})) =_{[7]\&[6]}$

$\lambda y.\, ([\lambda y\ [x := \lambda x.\, (x_{\{0\}}\ y_{\{0\}})]]\ y_{\{0\}}$

$\quad [\lambda y\ [x := \lambda x.\, (x_{\{0\}}\ y_{\{0\}})]]\ x_{\{0\}}) =_{[4]}$

$\lambda y.\, (y_{\{0\}}\ ([\lambda y]\ [x := \lambda x.\, (x_{\{0\}}\ y_{\{0\}})]\ x_{\{0\}})) =_{[1]}$

$\lambda y.\, (y_{\{0\}}\ ([\lambda y]\ (\lambda x.\, (x_{\{0\}}\ y_{\{0\}}))) =_{[7]\&[6]}$

$\lambda y.\, (y_{\{0\}}\ \lambda x.\, ([\lambda x\ [\lambda y]]\ x_0\ [\lambda x\ [\lambda y]]\ y_{\{0\}})) =_{[4]}$

$\lambda y.\, (y_{\{0\}}\ \lambda x.\, (x_{\{0\}}\ [\lambda x]\ [\lambda y]\ y_{\{0\}})) =_{[3]}$

$\lambda y.\, (y_{\{0\}}\ \lambda x.\, (x_{\{0\}}\ [\lambda x]\ y_{\{1\}})) =_{\beta}$

$\lambda y.\, (y_{\{0\}}\ \lambda x.\, (x_{\{0\}}\ y_{\{1\}}))$

where the need to create a fresh name has evaporated.

Achieving Unique representation of λ-terms up to α-equivalence for closed terms [without variables]

CINM is parametric on the chosen set of names. It can therefore be defined as a parameterized module in Maude, parameterized on the theory EQ* that assumes the existance of an equality predicate $eq(X, Y)$ between any two names, and of a constant * to choose a fixed name. Alpha conversion is then achived by the conditional equation:

$$ceq\ [alpha] : \lambda X.U = \lambda * [X := *[0]][\wedge *]U \bullet \quad \text{if } eq(X, *) = false \bullet$$

For example, if we choose quoted identifiers as names and interpret * as the name 'x, then the unique representation of

$$S = \lambda\,'x \,.\, \lambda\,'y \,.\, \lambda\,'z \,.\, ('x_{[0]}\ 'z_{[0]})('y_{[0]}\ 'z_0)$$

becomes

$$S_{\downarrow alpha} = \lambda\,'x \,.\, \lambda\,'x \,.\, \lambda\,'x \,.\, (x_{[2]}\ 'x_{[0]})(x_{[1]}\ x_{[0]})$$

Appendix lecture16-A1.pdf contains the parameterized specification of λ-CINNI, its instantiation to the case where names are quoted identifiers, which is quite user-friendly, and examples showing how [alpha] works. Note that [alpha] leaves free variables alone. So, uniqness of representation for a λ-term is only achieved if the terms is closed [has no free variables]. CINNI has the following advantages:

1. machine friendly: no compilation to another formalism is needed
2. user friendly
3. canonical representation of closed λ-terms, which becomes unique and the same for α-equivalent terms
4. avoidance of the variable capture problem
5. Applicable not just to the λ-calculus, but to any binding operators like:

$\forall x.\varphi \quad \exists x.\varphi \quad \nu x.U$ [in the π-calculus]

and so on: any such binders, not just λ can be handled.