CS5860

Thur. Sept 29, 2011 Computational First-Order Logic continued

PLAN

1. Review of concepts that seemed unclear in discussions
   functions, meaning of $P(x)$ in evidence semantics,
   "dependent typing" vs ML typing

2. Semantics of functions and logical operators

   $\lambda(x.b(x))$, $decide(d; p.left(p); r.right(r))$
   (if $isl(d)$ then $left(outl(d))$ else $right(outr(d))$)

   $spread(p; x,y.b(x,y))$ ($(x,y) = p$ in $b(x,y)$)

3. Why First-Order Logic (FOL) is a programming language,
   Call it a First-Order Programming Language.

4. Another "programming pattern"

   $$\underbrace{\exists x.P(x)}_{\text{data}} \Rightarrow \underbrace{\forall x.(P(x) \Rightarrow \exists y Q(y))}_{\text{function}} \Rightarrow \underbrace{\exists y.Q(y)}_{\text{data}}$$

5. Proof rules continued

   Stress the apseq rule, the most subtle.

   $\bar{H}_1, f: A \Rightarrow B, \bar{H}_2 \vdash G$ by $apseq(f; \_\_; v.\_\_)$
   " " " " $\vdash A$ by a
   (input)
   " " " "
   " " " " $v: B \vdash G$ by $g(v)$
   (output)

   Note $v = ap(f; a)$ but a might not be available until after the input subproof is complete.

CS 5860

Thur Sept. 29, 2011

Review Some of you have had little experience with functions in programming. They are central to ML, Lisp, Scheme and other functional programming languages. Haskell is one of the "pure" modern functional languages. C programmers know about functions as well, but in Java they are called methods and their application is implicit, e.g.

```
class C { double x, y:
            f1() { return exp}
            f2() { return x*y} }
```

We might then see $a := c.f_1$ which "calls f_1" on x, y.

In Lisp/Scheme you can see (lambda (a) (lambda(x) (eq? x a))).

In ML this is ~~fun~~ \a.\x. if (x=a then true else false).

You should also know about functions from calculus and discrete mathematics. For example $\int_a^b f(x)dx$ has type

$\int : x{:}\mathbb{R} \to y{:}\mathbb{R} \to f{:}\,[x,y] \to \mathbb{R} \to \mathbb{R}$. In discrete mathematics $\sum_{i=n}^{m} f(i)$ has inputs $n, m : \mathbb{N}$, $f : \mathbb{N} \to \mathbb{N}$ and the result is type $\mathbb{N}$.

CS 5860

Thur. Sept. 29, 2011

Also see Sept 15 lecture : Semantics and Proof Rules

The computational semantics of functions and logical operators.

To apply a function f to an argument (input) a is to evaluate the expression $ap(f;a)$ also written $f(a)$, fa

to evaluate $ap(f;a)$, first evaluate the principal argument f, the result must be $\lambda(x.b(x))$ for some expression $b(x)$ with free variable x; then substitute a for x in $b(x)$, to get $b(a)$; then evaluate $b(a)$ to a value v; this v is the value of $ap(f;a)$.

The above method is lazy evaluation or call by name. In this case, if the expression a has any free variables, say it is $(3+y)$, then we need to make sure that "a is free for x in $b(x)$", that is, $b(x)$ can't be an expression such as $\lambda(y.x)$ because then after substitution we get $\lambda(y.3+y)$ which changed the meaning of $(3+y)$ by "capturing y."

Another method of evaluation is call by value.

to evaluate $ap(f;a)$, first evaluate f to some $\lambda(x.b(x))$, then evaluate a to a value, a'; then substitute a' for x in $b(x)$ to get $b(a')$; then evaluate $b(a')$ to a value v.

CS 5860

Thur Sept 29, 2011

Another programming paradigm

$$(\exists x. P(x) \ \& \ \forall x.(P(x) \Rightarrow \exists y.Q(y))) \Rightarrow \exists y.Q(y)$$

Data & Property (an assertion) — Program (with assertions) — Output with Assertion

Here is an evidence term, also called a realizer.

$$\lambda(h.\ spread(h;\ data, f.\ spread(data;\ x, p.\ ap(ap(f;x);p))))$$

We could also write $ap(ap(f;x);p)$ as $(f(x))(p)$.

Note f is the function. We could call it a program too, but we want to use p of the evidence for $P(x)$.

Write the realizer as an ML program. Notice, ML does not have the dependent types such as $x{:}D \times P(x)$, the dependent product, nor $x{:}D \to (P(x) \to y{:}D \times Q(y))$, the dependent function space, needed to state the task.