Computing in the λ-Calculus
Programming Languages (CSCI 3300), Fall 2014

Prof. Harley Eades (heades@gru.edu).

1 Inference Systems

A lot of the design of programming languages in general involve the careful definition of certain algorithms that manipulate programs. This is similar to the idea of defining functions on programs like the depth function we saw in class. A large number of algorithms can be described using inference systems.

An inference system consists of one or more inference rules. An inference rule has the following shape:

$$\frac{P_1 \qquad \cdots \qquad P_n}{C}\ \textsc{Name}$$

We call $P_1, \cdots, P_n$ the premises of the rule, and C the conclusion. Note that it is completely satisfactory to have an inference rule with no premises at all. In fact, an inference rules with no premises are called axioms. Inference rules are read from the top down as an if-statement. So the above rule should be read "If P_1, P_2, ..., and P_n all hold, then C holds." If an inference rule has no premises at all, then its conclusion always holds, hence, the term axiom.

An example may help. The following set of inference rules define what it means to be a binary tree:

$$\frac{n \in \mathbb{N}}{(\mathsf{leaf}\, n)\, Tree}\ \textsc{leaf} \qquad \frac{T_1\, Tree \qquad T_2\, Tree \qquad n \in \mathbb{N}}{(\mathsf{node}\, n\, T_1\, T_2)\, Tree}\ \textsc{node}$$

The first rule states, "for all $n \in \mathbb{N}$, $\mathsf{leaf}\, n$ is a tree." Now the second states that "for any T_1, T_2, and $n \in \mathbb{N}$, if T_1 and T_2 are trees, then $\mathsf{node}\, n\, T_1\, T_2$ is a tree."

So what are these rules good for? Inference system should be thought of as an algorithm where the conclusion, C, is an input, and the output essentially amounts to yes or no. The former indicating that the input adheres to the rules, and no otherwise. So for example, given the rules above we can ask is $\mathsf{node}\, 1\, (\mathsf{node}\, 2\, (\mathsf{leaf}\, 4)\, (\mathsf{leaf}\, 5))\, (\mathsf{node}\, 3\, (\mathsf{leaf}\, 6)\, (\mathsf{leaf}\, 7))$ a tree, but how can we actually answer this question? We answer this question by building derivations of the rules. This is all very similar to how we used grammars.

A derivation is the application of one or more rules. For example, one of the simplest derivations using the rules for trees is the following:

$$\frac{1 \in \mathbb{N}}{(\mathsf{leaf}\, 1)\, Tree}\ \textsc{leaf}$$

In the above $\mathsf{leaf}\, 1$ is the input, and the derivation consists of only the rule `leaf`, and this successfully shows that $\mathsf{leaf}\, 1$ is indeed a tree. A more complete derivation is the follows:

$$\dfrac{\dfrac{\dfrac{4 \in \mathbb{N}}{(\mathsf{leaf}\, 4)\, Tree}\ \textsc{leaf} \qquad \dfrac{5 \in \mathbb{N}}{(\mathsf{leaf}\, 5)\, Tree}\ \textsc{leaf} \qquad 2 \in \mathbb{N}}{(\mathsf{node}\, 2\, (\mathsf{leaf}\, 4)\, (\mathsf{leaf}\, 5))\, Tree}\ \textsc{node} \qquad \dfrac{\dfrac{6 \in \mathbb{N}}{(\mathsf{leaf}\, 6)\, Tree}\ \textsc{leaf} \qquad \dfrac{7 \in \mathbb{N}}{(\mathsf{leaf}\, 7)\, Tree}\ \textsc{leaf} \qquad 3 \in \mathbb{N}}{(\mathsf{node}\, 3\, (\mathsf{leaf}\, 6)\, (\mathsf{leaf}\, 7))\, Tree}\ \textsc{node} \qquad 1 \in \mathbb{N}}{(\mathsf{node}\, 1\, (\mathsf{node}\, 2\, (\mathsf{leaf}\, 4)\, (\mathsf{leaf}\, 5))\, (\mathsf{node}\, 3\, (\mathsf{leaf}\, 6)\, (\mathsf{leaf}\, 7)))\, Tree}\ \textsc{node}$$

Derivations are constructed from the bottom up. In the above we begin by asking the question, "does $(\mathsf{node}\, 1\, (\mathsf{node}\, 2\, (\mathsf{leaf}\, 4)\, (\mathsf{leaf}\, 5))\, (\mathsf{node}\, 3\, (\mathsf{leaf}\, 6)\, (\mathsf{leaf}\, 7)))\, Tree$ hold?" To prove that it holds we try to apply the rules of the tree inference system one at a time, from the bottom up. So after asking the question we

try and pattern match on the conclusion of each rule. If what we are asking matches the conclusion of the rule, then we may then move to the premises of that rule. For example, we construct the above derivation by starting like this, we first asking the question:

$$(\mathsf{node}\,1\,(\mathsf{node}\,2\,(\mathsf{leaf}\,4)\,(\mathsf{leaf}\,5))\,(\mathsf{node}\,3\,(\mathsf{leaf}\,6)\,(\mathsf{leaf}\,7)))\,Tree$$

Then we try to pattern match on the previous statement against the conclusion of each rule, and we can see that it matches the conclusion of the `node` rule, where $n = 1$, $T_1 = (\mathsf{node}\,2\,(\mathsf{leaf}\,4)\,(\mathsf{leaf}\,5))$, and $T_2 = (\mathsf{node}\,3\,(\mathsf{leaf}\,6)\,(\mathsf{leaf}\,7))$. So we are allowed to apply that rule:

$$\frac{(\mathsf{node}\,2\,(\mathsf{leaf}\,4)\,(\mathsf{leaf}\,5))\,Tree \qquad (\mathsf{node}\,3\,(\mathsf{leaf}\,6)\,(\mathsf{leaf}\,7))\,Tree \qquad 1 \in \mathbb{N}}{(\mathsf{node}\,1\,(\mathsf{node}\,2\,(\mathsf{leaf}\,4)\,(\mathsf{leaf}\,5))\,(\mathsf{node}\,3\,(\mathsf{leaf}\,6)\,(\mathsf{leaf}\,7)))\,Tree}\ \textsc{node}$$

Then we ask, which rule can be applied to each premise? Then we proceed until no further rules can be applied. Eventually we end up with the derivation from above.

Now how do we know when a derivation is successful? First, notice that the derivations are themselves tree structures. The axioms are the leaves of the tree, and the premises start branches of the tree. For example, in the above example of the complete derivation using the tree inference system we can see that the `node` rule forms two branches, one for each subtree. Now a derivation is **successful** if and only if every branch of the derivation ends in an axiom. That is, the very last rules on the top of the derivation must be all axioms. If not, then the derivation is a failure, and the conclusion is said not to hold. The following is an example of a failure:

$$\frac{\dfrac{2 \in \mathbb{N}}{(\mathsf{leaf}\,2)\,Tree}\ \textsc{leaf} \qquad 42 \qquad 1 \in \mathbb{N}}{(\mathsf{node}\,1\,(\mathsf{leaf}\,2)\,42)\,Tree}\ \textsc{node}$$

The previous derivation fails, because there is no rule to apply 42 to and thus the derivation does not end in all axioms – that is the `leaf` rule. Therefore, we must conclude that $\mathsf{node}\,1\,(\mathsf{leaf}\,2)\,42$ is not a valid tree.

Inference systems usually have a particular goal called a judgment. The inference system is said to judge whether not something is provable or not. For example, the tree inference system judges whether or not some mathematical object is a tree, and we can read the judgement $t\,Tree$ as "t is a tree." We then say that a judgment holds or is provable or is derivable if and only if we can construct a successful derivation of it.

Inference systems are everywhere in programming language design. They allow for a very nice, compact, and easy definitions of algorithms on the objects of a programming language. Now we can use inference systems to define how we can compute in the λ-calculus.

2 Computing in the λ-Calculus

The locus of computation in the λ-calculus is function application, $t_1\,t_2$, but even more specifically it is the application of a λ-abstraction to an argument, that is, $(\lambda x.t)\,t'$. It turns out that computation in the λ-calculus is really just a matter of replacing every occurrence of x in t with t', but how do we do this formally? We use an inference system:

$$\frac{}{(\lambda x.t)\,t' \rightsquigarrow [t'/x]t}\ \textsc{Beta} \qquad \frac{t \rightsquigarrow t'}{\lambda x.t \rightsquigarrow \lambda x.t'}\ \textsc{Lam}$$

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\,t_2 \rightsquigarrow t_1'\,t_2}\ \textsc{App1} \qquad \frac{t_2 \rightsquigarrow t_2'}{t_1\,t_2 \rightsquigarrow t_1\,t_2'}\ \textsc{App2}$$

The judgment here is $t_1 \rightsquigarrow t_2$ which can be read as t_1 reduces to t_2. We call this set of inference rules the reduction rules for the λ-calculus. Note that one should think of the terms in these rules as schemas of terms

that we match pattern against. So some care has to be taken when checking to see if a rule applies. The reduction rules should be used just as we used the tree inference system above. That is, we begin with a judgment that looks like $t_1 \rightsquigarrow t_2$, and then try to prove that t_1 reduces to t_2 if we can construct a successful derivation of $t_1 \rightsquigarrow t_2$ using the reduction rules. We can think of this style as starting with some solution to a problem, and then using the rules to check to see if the solution is correct.

The most important rule is the β-rule:

$$\frac{}{(\lambda x.t)\, t' \rightsquigarrow [t'/x]t} \quad \textsc{Beta}$$

This rule says that any term matching the pattern $(\lambda x.t)\, t'$ can be replaced by $[t'/x]t$ which replaces every occurrence of x with t' in t.

Definition 1 (Redex)**.** *Any term matching* $(\lambda x.t)\, t'$ *is called a* ***redex****.*

So we can see that the term in the lefthand side of the arrow in the β-rule is the redex, and we call the righthand side the **contractum**. Notice that variable substitution is an important part of the β-rule, and now we can see that computing really amounts to using the β-rule which amounts to just using the variable substitution function to simplify function applications.

Now lets consider an example using the β-rules. The following shows that $(\lambda x.x)\, y$ reduces to y.

$$\frac{}{(\lambda x.x)\, y \rightsquigarrow y}\ \textsc{Beta}$$

This holds because the term $(\lambda x.x)\, y$ matches the pattern $(\lambda x.t)\, t'$ where the binder x is x, t is x, and t' is t, and $[y/x]x = y$. So sometimes this rule is hard to apply because we have to reverse the substitution function, that is, we had to realize the y is the same as $[y/x]x$ when just given y.

The reduction rules are literal. One can only apply the rule if the terms given match the term patterns in the conclusion. For example, can we apply the β-rule to conclude $x\,((\lambda y.y\, z)\, r) \rightsquigarrow x\,(r\, z)$? No, because $x\,((\lambda y.y\, z)\, r)$ does not match the pattern $(\lambda x.t)\, t'$, and $x\,(r\, z)$ does not match the pattern $[t'/x]t$ so that rule does not apply, but we should be allowed to reduce that redex, right? This is the job for the other reduction rules.

There are three other reduction rules:

$$\frac{t \rightsquigarrow t'}{\lambda x.t \rightsquigarrow \lambda x.t'} \quad \textsc{Lam} \qquad \frac{t_1 \rightsquigarrow t_1'}{t_1\ t_2 \rightsquigarrow t_1'\ t_2} \quad \textsc{App1}$$

$$\frac{t_2 \rightsquigarrow t_2'}{t_1\ t_2 \rightsquigarrow t_1\ t_2'} \quad \textsc{App2}$$

These rules allow us to move inside of terms. For example, we can now show $x\,((\lambda y.y\, z)\, r) \rightsquigarrow x\,(r\, z)$:

$$\frac{\dfrac{}{(\lambda y.y\, z)\, r \rightsquigarrow (r\, z)}\ \textsc{Beta}}{x\,((\lambda y.y\, z)\, r) \rightsquigarrow x\,(r\, z)}\ \textsc{App2}$$

Consider this term for example, $(\lambda x.x\, x)\,((\lambda y.y)\, r)$, how many redexes are there in this term? There are two, the entier term, and the subterm $(\lambda y.y)\, r$. So how do we reduce them both? In fact, we cannot do this, because these rules only support reducing exactly one redex. In fact, the reduction rules we have considered so far are called the **single-step full β-reduction.** The "full" part has to do with the ability to contract a redex anywhere within a term.

To allow for the reduction of multiple redexes we define a new inference system called **multi-step full β-reduction.** The inference system is as follows:

$$\frac{}{t \rightsquigarrow^* t}\ \textsc{Refl} \qquad \frac{t \rightsquigarrow t'}{t \rightsquigarrow^* t'}\ \textsc{SStep} \qquad \frac{t_1 \rightsquigarrow^* t_2 \quad t_2 \rightsquigarrow^* t_3}{t_1 \rightsquigarrow^* t_3}\ \textsc{MStep}$$

Now we can use this inference system to reduce more than one redex at a time. For example we can prove the following reductions (full derivations in class):

$$x\,((\lambda y.y)\,((\lambda z.z\ z)\ r)) \rightsquigarrow^* x\,(r\ r)$$
$$\lambda y.(\lambda x.\lambda y.y)\,((\lambda x.x\ x)\,(\lambda x.x\ x))\ z \rightsquigarrow^* \lambda y.z$$

There is one downfall of applying the reduction rules the way that we have. The redex is often the program the programmer has written, and the contractum is either the solution, or an intermediate point to the solution. So from here on out we – unless directed differently like in homeworks or exams – we are going to adopted a different way of applying these rules. We will use these rules a guiding principle for how to transform a term into another by contracting its redexes. So instead of writing down long and big derivations we will simply write down a chain of transformations. These transformations a very similar to the derivations we did on grammars. Lets consider an example.

$$\begin{aligned}
&\underline{(\lambda n.\lambda m.(\lambda s.\lambda z.(n\ s)\,(m\ s\ z)))\,(\lambda s.\lambda z.s\ z)\,(\lambda s.\lambda z.s\,(s\ z))}\\
&\quad\rightsquigarrow \underline{(\lambda m.(\lambda s.\lambda z.((\lambda s.\lambda z.s\ z\ s)\,(m\ s\ z))))\,(\lambda s.\lambda z.s\,(s\ z))}\\
&\quad\rightsquigarrow \lambda s.\lambda z.(((\lambda s.\lambda z.s\ z)\ s)\,\underline{((\lambda s.\lambda z.s\,(s\ z))\ s\ z)})\\
&\quad\rightsquigarrow \lambda s.\lambda z.(\underline{((\lambda s.\lambda z.s\ z)\ s)}\,(s\,(s\ z)))\\
&\quad\rightsquigarrow \lambda s.\lambda z.\underline{(\lambda z.s\ z)\,(s\,(s\ z))}\\
&\quad\rightsquigarrow \lambda s.\lambda z.s\,(s\,(s\ z))
\end{aligned}$$

Notice how in the above I underlined the redex I was contracting in each step. This is required when writing these down in your homework and exams, because it makes it apparent which redex we are contracting at each step. Also, notice that where we can put the underline is determined by the rule of the inference system. Lastly, each transformation is step in the single-step full β-reduction inference system. However, if one can give a chain of one or more transformations from t_1 to t_2, then we can derive $t_1 \rightsquigarrow^* t_2$ in the multi-step full β-reduction inference system.