Constructive Logic (15-317), Fall 2016
Recitation 8: Quantifiers and Logic programming

Evan Cavallo (ecavallo@cs.cmu.edu), Oliver Daids (ojd@andrew), Giselle Reis (greis@andrew)

1 Quantifiers

$$\frac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B} \supset R \qquad \frac{\Gamma, A \supset B \longrightarrow A \quad \Gamma, B \longrightarrow C}{\Gamma, A \supset B \longrightarrow C} \supset L$$

$$\frac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B}{\Gamma \longrightarrow A \wedge B} \wedge R \qquad \frac{\Gamma, A, B \longrightarrow C}{\Gamma, A \wedge B \longrightarrow C} \wedge L$$

$$\frac{\Gamma \longrightarrow A}{\Gamma \longrightarrow A \vee B} \vee R_1 \qquad \frac{\Gamma \longrightarrow B}{\Gamma \longrightarrow A \vee B} \vee R_2 \qquad \frac{\Gamma, A \longrightarrow C \quad \Gamma, B \longrightarrow C}{\Gamma, A \vee B \longrightarrow C} \vee L$$

$$\frac{}{\Gamma, P \longrightarrow P} \, init \qquad \frac{}{\Gamma \longrightarrow \top} \top R \qquad \frac{}{\Gamma, \bot \longrightarrow C} \bot L$$

$$\frac{\Gamma \longrightarrow A(c)}{\Gamma \longrightarrow \forall x.A(x)} \forall R \qquad \frac{\Gamma, \forall x.A(x), A(t) \longrightarrow C}{\Gamma, \forall x.A(x) \longrightarrow C} \forall L \qquad \frac{\Gamma \longrightarrow A(t)}{\Gamma \longrightarrow \exists x.A(x)} \exists R \qquad \frac{\Gamma, A(c) \longrightarrow C}{\Gamma, \exists x.A(x) \longrightarrow C} \exists L$$

Task 1. Prove the following formula in KeYmaera I (try to prove it by hand first to see the pattern).

$$p(z) \wedge (\forall x.p(x) \supset p(s(x))) \supset p(s^9(z))$$

Solution 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p(z), \forall x.p(x) \supset p(s(x)), p(z) \supset p(s(z)) \longrightarrow p(z)} \, init \quad \dfrac{\varphi}{p(z), \forall x.p(x) \supset p(s(x)), p(s(z)) \longrightarrow p(s^9(z))} \forall L}{p(z), \forall x.p(x) \supset p(s(x)), p(z) \supset p(s(z)) \longrightarrow p(s^9(z))} \supset L}{p(z), \forall x.p(x) \supset p(s(x)) \longrightarrow p(s^9(z))} \forall L}{p(z) \wedge (\forall x.p(x) \supset p(s(x))) \longrightarrow p(s^9(z))} \wedge R}{\longrightarrow p(z) \wedge (\forall x.p(x) \supset p(s(x))) \supset p(s^9(z))} \supset R$$

Where φ is:

$$\dfrac{\dfrac{\dfrac{}{p(z), \forall x.p(x) \supset p(s(x)), p(s(z)), p(s(z)) \supset p(s^2(z)) \longrightarrow p(s(z))} \, init \quad \dfrac{\text{and so on and so forth...}}{p(z), \forall x.p(x) \supset p(s(x)), p(s(z)), p(s^2(z)) \longrightarrow p(s^9(z))} \forall L}{p(z), \forall x.p(x) \supset p(s(x)), p(s(z)), p(s(z)) \supset p(s^2(z)) \longrightarrow p(s^9(z))} \supset L}{p(z), \forall x.p(x) \supset p(s(x)), p(s(z)) \longrightarrow p(s^9(z))} \forall L$$

Observe how the sequent would not be provable if we erase the $\forall$ formula on the left after instantiating it the first time.

Task 2. Can you think of a cut-formula that can make this proof shorter?

Solution 2: Cut-formula: $\forall x.p(x) \supset p(s^3(x))$.

$$\dfrac{\dfrac{\dfrac{\overset{\varphi_l}{p(z), \forall x.p(x) \supset p(s(x)) \longrightarrow \forall x.p(x) \supset p(s^3(x))} \quad \overset{\varphi_r}{p(z), \forall x.p(x) \supset p(s(x)), \forall x.p(x) \supset p(s^3(z)) \longrightarrow p(s^9(z))}}{p(z), \forall x.p(x) \supset p(s(x)) \longrightarrow p(s^9(z))}\ \mathit{cut}}{p(z) \wedge (\forall x.p(x) \supset p(s(x))) \longrightarrow p(s^9(z))}\ \wedge R}{\longrightarrow p(z) \wedge (\forall x.p(x) \supset p(s(x))) \supset p(s^9(z))}\ \supset R$$

Where φ_l is ($\Gamma_l = \{p(z), \forall x.p(x) \supset p(s(x))\}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_l, p(\alpha), \ldots \longrightarrow p(\alpha)}\ \mathit{init} \quad \dfrac{\dfrac{\dfrac{}{\Gamma_l, p(s(\alpha)), \ldots \longrightarrow p(s(\alpha))}\ \mathit{init} \quad \dfrac{\dfrac{\dfrac{}{\Gamma_l, p(s^2(\alpha)), \ldots \longrightarrow p(s^2(\alpha))}\ \mathit{init} \quad \dfrac{}{\Gamma_l, p(s^3(\alpha)), \ldots \longrightarrow p(s^3(\alpha))}\ \mathit{init}}{\Gamma_l, p(\alpha), p(s(\alpha)), p(s^2(\alpha)), p(s^2(\alpha)) \supset p(s^3(\alpha)) \longrightarrow p(s^3(\alpha))}\ \supset L}{\Gamma_l, p(\alpha), p(s(\alpha)), p(s^2(\alpha)) \longrightarrow p(s^3(\alpha))}\ \forall L}{\Gamma_l, p(\alpha), p(s(\alpha)), p(s(\alpha)) \supset p(s^2(\alpha)) \longrightarrow p(s^3(\alpha))}\ \supset L}{\Gamma_l, p(s(\alpha)) \longrightarrow p(s^3(\alpha))}\ \forall L}{\Gamma_l, p(\alpha), p(\alpha) \supset p(s(\alpha)) \longrightarrow p(s^3(\alpha))}\ \supset L}{\Gamma_l, p(\alpha) \longrightarrow p(s^3(\alpha))}\ \forall L}{\Gamma_l \longrightarrow p(\alpha) \supset p(s^3(\alpha))}\ \supset R}{\Gamma_l \longrightarrow \forall x.p(x) \supset p(s^3(x))}\ \forall R$$

And φ_r is ($\Gamma_r = \{p(z), \forall x.p(x) \supset p(s(x)), \forall x.p(x) \supset p(s^3(z))\}$):

$$\dfrac{\dfrac{\dfrac{}{\Gamma_r, p(z) \supset p(s^3(z)) \longrightarrow p(z)}\ \mathit{init} \quad \dfrac{\dfrac{\dfrac{}{\Gamma_r, p(s^3(z)), p(s^3(z)) \supset p(s^9(z)) \longrightarrow p(s^3(z))}\ \mathit{init} \quad \dfrac{}{\Gamma_r, p(s^3(z)), p(s^9(z)) \longrightarrow p(s^9(z))}\ \mathit{init}}{\Gamma_r, p(s^3(z)), p(s^3(z)) \supset p(s^9(z)) \longrightarrow p(s^9(z))}\ \supset L}{\Gamma_r p(s^3(z)) \longrightarrow p(s^9(z))}\ \forall L}{\Gamma_r, p(z) \supset p(s^3(z)) \longrightarrow p(s^9(z))}\ \supset L}{\Gamma_r \longrightarrow p(s^9(z))}\ \forall L$$

Note that, using cuts, we only need to instantiate quantifiers 6 times, as opposed to 9 times in a cut-free proof.

2 Logic programming

You might be familiar with functional and imperative programming. Today we will see yet another programming paradigm: logic programming. Logic programming can be seen as a fragment of intuitionistic logic[1] called *Horn clauses* (remember last homework?). A Horn clause is either an atom or a formula of the shape $A_1 \wedge ... \wedge A_n \supset H$, where H is called the *head* and $A_1 \wedge ... \wedge A_n$ is the *body*. In prolog syntax, this is written as:

```
h :- a1, a2, ..., an.
```

Let's step through a simple prolog program to understand how computation (or proof search) works. Consider the following simple code:

```
ocean_level(rising).
temperature(extreme).
global_warming(conspiracy) :- ocean_level(stable), temperature(normal).
global_warming(real) :- ocean_level(rising), temperature(extreme).
```

If we query prolog for `global_warming(X)`, it will look at the head of all (four) clauses trying to find one that "matches" (*unifies*) with the goal. In this case, it finds the clauses in lines 3 and 4. Prolog will process the options in order, so it will first go to clause in line 3 and unify `X` with `conspiracy`, generating the new goals `ocean_level(stable)` and `temperature(normal)`. A proof-theoretic interpretation of this step is the following (predicate names are abbreviated for the sake of space):

$$\dfrac{\dfrac{\mathrm{ol(ris), temp(xtr), ...} \longrightarrow \mathrm{ol(sta)} \quad \mathrm{ol(ris), temp(xtr), ...} \longrightarrow \mathrm{temp(nml)}}{\mathrm{ol(ris), temp(xtr), ...} \longrightarrow \mathrm{ol(sta)} \wedge \mathrm{temp(nml)}} \wedge R \quad \dfrac{\texttt{X is csp}}{\mathrm{ol(ris), temp(xtr), gw(csp), ...} \longrightarrow \mathrm{gw(X)}} \mathit{init}}{\mathrm{ol(ris), temp(xtr), ol(sta)} \wedge \mathrm{temp(nml)} \supset \mathrm{gw(csp), ol(ris)} \wedge \mathrm{temp(xtr)} \supset \mathrm{gw(real)} \longrightarrow \mathrm{gw(X)}} \supset L$$

In this derivation, X is a special variable that is unified on initial rules, and this unification propagates to the next branch if there were occurrences of X there as well. When trying to prove the two open sequents, or the new goals, prolog will realize that `ocean_level(stable)` or `temperature(normal)` are not true... oops, are not in the context nor they are unifiable with any clause head. Time to backtrack. We know that $\wedge R$ is an invertible rule, so no use in backtracking there. We go back to the choice of clauses (i.e., $\supset L$) and try to use the one on line 4. This time the unification will be `X is real` and the new goals `ocean_level(rising)` and `temperature(extreme)`, which can be proved.

As a final note, logic programs hold some resemblance to functional programs in the way programs are written. You will find that sometimes the clauses used look a lot like the cases you would need in, say, SML. This kind of programming style is referred to as *declarative* programming (you write *what* your program does as opposed to *how* it does it).

Task 3. Implement a prolog program that computes the truncated subtraction between natural number along the same lines as the `plus` and `times` implementations given in the lecture notes.

Solution 3: pred(z,z).
pred(s(M), M).
minus(N, z, N).
minus(N, s(M), Q) :- minus(N, M, P), pred(P, Q).

[1]It is also a fragment of classical logic. Since it is such a simple fragment, intuitionistic and classical logic coincide.