Topic: Refinement Logic, single-conclusioned calculi Smullyan p.

Assignments: RL Proofs for Smullyan p 24. 6-8

Two remarks on handouts

1. Confusion about brackets in matrix methods. I clarified that and improved the handout
2. On the other handout there was a typo – I corrected that on the web and I will come to that when I resume the relation between Gentzen Systems and Blocked Tableaux

7 Refinement Logic

1. Why are the two calculi identical

Compare on one slide note the slightly different notation for logical connectives. The right hand side is what we use in NuPRL
So every sequent proof can be directly simulated by a blocked tableau and vice versa. So the duality is actually an *isomorphism*. And as a result we get

The multi-succedent sequent calculus is consistent and complete

2. Differences

Let me conclude by a few remarks about the semantical differences between tableaux and Gentzen systems. Although thses two methods are formally almost identical, they have different intentions.

	Tableau	Sequent
Goal	$\mathsf{F}(X)$: search for counterexample	$\vdash X$: try to prove X
α-step	counterexample must involve both α_1 and α_2 (AND branch)	proof must show either α_1 or α_2 (OR-branch)
β-step	counterexample uses either β_1 or β_2 (OR-branch)	proof must show both β_1 and β_2 (AND branch)
closed branch	counterexample impossible	partial proof sucessful
open leaf	consistent valuation possible (counterexample found)	proof fails at this leaf
	implicit proof of validity – no counterexample	explicit justification of validity
	negative / indirect approach	positive / constructive proof

But the technical similarity shows us that the indirect proof can be converted into a direct one. Actually, there are even methods which start way in the very compact matrix proofs and end up in the refinement logic that we will discuss next today

7.1 Refinement Logic

Today we want to go one step further and talk about a Gentzen system that doesn't use multiple succedents. This *refinement logic* has many properties that are interesting for a computer scientists, because it catches the notion of a construction in a very nice and natural way. It is also the logic that we use in our NuPRL system and I will show you how we can use this system to support and automate the development of formal proofs.

3. Motivation

So let us first consider the question why one would need yet another calculus when trying to prove a formula? Isn't the multi-conclusioned calculus good enough. Well theretically it is, but it is somewhat weird when you look at it from the perspective of a computer scientist
Pierce's law $((p \supset q) \supset p) \supset p$ f

- $\vdash ((p \supset q) \supset p) \supset p$ by `impR`
- $(p \supset q) \supset p \vdash p$ by `impL`
- 1. $p \vdash p,\ p \supset q$ by `axiom`
  2. $p \vdash p$ by `axiom`

Hey, that is cheating – by applying `impL` we created the proof goal $p \supset q$ but then we changed our mind and decided to prove the other goal that we already had. Well, that is perfectly legal – we proved the rule to be consistent – but it still looks as if something is wrong about this proof. Constable used to draw an analogy to politics – the old "bait and switch" strategy that promises you something nice but before you look, you have also accepted something that you did not want. If you accept truth tables, you must also accept this proof.
Anyway, the main reason why this seems so difficult to accept is that we don't see the construction of the proof argument anymore.
So the multiple conclusions on the right of the turnstyle do not really lead to intuitive proofs. So what happens if we simply drop them.

4. Single-Conclusioned Sequent Systems

Here are the rules for Single-Conclusioned Sequent Systems that come out of this. There are two minor changes. We need to have two rules for disjunctions on the right, which express that now we have to *decide* which goal we want to consider further. For the right negation rule, dropping Γ leads to no conclusion at all – no goal formula can be true – which is the same as false.

5. Try it out

Now let's try it out – did we miss anything important by removing the multiple conclusions? What happens to our proof of Pierce's law?
It doesn't work anymore!
But we are convinced that this is true. We have a tableau proof, a multi-conclusioned sequent proof, even a truth table proof would work.
So what do we do?
We must add something to the rule system again that apparently got lost.
Well the point in truth tables is, that we always assume that a proposition can only be true or false. Or, in terms of a logical formula: it is true or its negation is true.

6. Add magic and cut

So let us add this assumption to the proof. We assert it first and get two subgoals. In the first we have to prove our asserted formula.
Let's go back to the rules ... ... actually, we can't prove this with our rules, because we have to decide which of the two shall be true and we can't just say either one must be the case. So ... ... go back to our proof and use a new rule .. that's the one that must have been missing.
And now the rest of the proof goes through fine.
O.k. – we need these two rules `cut` and `magic` to complete the proof.

7. Facts

And we will show on Thursday that this is enough to make the single-conclusioned calculus complete. Actually, the cut rule is not necessary. This is another famous theorem, which is quite tricky to prove.
The magic rule is under dispute because it still violates the idea of a construction. So mathematicians make a distinction between what they call classical logic and a constructive logic, that is more appealing for a computer scientist because it does not allow arguments like – "well – I don't know which, one of the two MUST be true". You will find out that Pierce's law holds only in classical logic but is not constructive at all.
Finally, we make one more minor change to the logic and create a version that is more suitable for machine support. We modify replace the *set notation* for assumptions by a *list* of hypotheses.

8. Basic rules

Instead of sets of formulae (Δ) we consider lists, and indicate that a formula might be somewhere in that list by putting two lists H, H' around it. Because of the list representation left rules must provide the index of the formula to which the rule shall be applied.
One more change: in the `impL` and `notL` rule we keep the formula that is being analyzed. We might need it again. In the set notation we would not have to write it again because it could be as well in the remaining set. But in lists we must make this explicit.

	left	right	
`andL`	$H,\ A \wedge B,\ H' \vdash G$ $H,\ A,\ B,\ H' \vdash G$	$H \vdash A \wedge B$ $H \vdash A$ $H \vdash B$	`andR`

`orL`	$H,\ A \vee B,\ H' \vdash G$
	$H,\ A,\ H' \vdash G$
	$H,\ B,\ H' \vdash G$
		$H \vdash A \vee B$
		$H \vdash A$
		
		$H \vdash A \vee B$
		$H \vdash B$
			`orR1`
			
			`orR2`

`impL`	$H,\ A \supset B,\ H' \vdash G$ $H,\ A \supset B,\ H' \vdash A$ $H,\ H',\ B \vdash G$	$H \vdash A \supset B$ $H,\ A \vdash B$	`impR`
`notL`	$H,\ \sim A,\ H' \vdash G$ $H,\ \sim A,\ H' \vdash A$	$H \vdash \sim A$ $H,\ A \vdash \text{false}$	`notR`
`falseL`	$H,\ \text{false},\ H' \vdash G$		
`hypothesis`	$H,\ A,\ H' \vdash A$		

9. Special Rules

Special Rules	
	$H \vdash G$ `cut` A $H \vdash A$ $H, A \vdash G$
`magic` $H \vdash A \vee \sim A$	$H \vdash G$ `ClassCases` A $H, A \vdash G$ $H, \sim A \vdash G$
`lemma` *name* $H \vdash L$	$H \vdash G$ `cut_lemma` *name* $H, L \vdash G$

In the lemma-rules L is the conclusion of a previously proven lemma with the given *name*. In `cut` and `ClassCases` A is a formula that has to be provided. The rules `ClassCases` and `cut_lemma` are composed out of simpler rules.

Exercises

Well, this leads me to a few exercises that I would like to assign before we go into the demo.

7.2 A demo of NuPRL

1. **Set up a few theorems, already opened (Smullyan 24)**
2. **Explain the system,**
   a) structure – library, command window, proof editor

   b) how to read a theorem window, how to proceed when proving a goal.
3. **play with a few proofs, interactively – let them guess rules**
   – be quick with Smullyan 6,7,8 if at all –
4. **talk about tactics: `THEN`, `ORELSE`, `Repeat` – use them in the proofs**
5. **talk about proof methodology**
   Show Prover_simple.ml
6. **Run tactic prover**
7. **More about the system:**
   how to enter theorems etc.