Applied Logic	Lecture 12: Properties of Refinement Logic
CS 4860 Spring 2009	Thursday, February 26, 2009

On Tuesday I introduced Refinement Logic as single-conclusioned Gentzen System with two extra rules, `magic` and `cut`. I gave a few examples for using the `magic` rule but it turned out that this rule isn't easy to handle and the resulting proofs are occasionally a bit complicated.

Today we want to investigate the properties of Refinement Logic. There are five questions that we need to look at.

1. *Is it consistent?*
2. *Is it complete?*
3. *Can a proof search get wedged?*
4. *What kind of logic do we get if drop the* `magic` *rule?*
5. *Is it decidable?*

To answer the first two questions we will compare Refinement Logic (with `magic`) to the multi-conclusioned sequent Gentzen system, which we already know to be consistent and complete - because it is isomorphic to block tableaux and that block tableau proofs can be converted into ordinary tableau proofs and vice versa. We will proceed in three steps.[1]

1. Refinement Logic with `cut` can be mapped onto a Refinement Logic without `cut`. This *cut elimination theorem* makes it much easier to prove consistency.
2. Refinement Logic with `magic` and `cut` can be mapped onto the multi-conclusioned sequent calculus. That gives us *correctness* and *consistency*.
3. The multi-conclusioned sequent calculus can be mapped onto Refinement Logic with `magic` and `cut`. That gives us *completeness*.

12.1 The cut elimination theorem

We start with the cut elimination theorem, which will help proving consistency. The cut rule

$$\begin{array}{ll} H \vdash G & \texttt{cut}\ A \\ H \vdash A & \\ H, A \vdash G & \end{array}$$

is very useful for structuring a proof. One introduces a fact A, or a lemma. This fact will be proven separately and the be used as assumption in the rest of the proof. This technique is particularly useful if an uses the lemma several times in a proof. But it also helps keep the proof comprehensible by cutting it into smaller pieces.

It is easy to explain why this rule must be correct but its disadvantage is that it cannot be used in automated proofs. All other rules (except for `magic` of course) depend on the outer structure of a formula on the left or right of the turnstyle. The cut rule, however, can be applied *anytime* and with *any* formula. Nobody has found a way of automating this properly – finding lemmata automatically is still a very difficult task.

[1] Because of the amount of detail I will use slides for this in class.

So the question was – do we actually *need* the cut rule? Do we lose anything if we drop this rule? Can we prove more formulas if we use cuts than if we don't? No, we cannot because the cut rule is not really necessary – one can prove the same theorems without it.

Theorem 12.1 (Cut elimination) *Any formula provable in the single-conclusioned sequent calculus with cut has a single-conclusioned sequent proof that does not use the cut rule.*

While this theorem is plausible, a precise proof of this theorem is quite tricky. If you're interested you can look up Smullyan's proof on pages 110–115, but I can't really recommend this because it is *very* dense.

Q: *Why should one believe that we can do without the cut rule?*

Essentially, it is because *we have to prove the formula that we use*, so we can *use the full proof of A whenever A is used.* This is the central idea of the proof of the cut elimination theorem. But a precise proof that uses this idea is difficult to formulate and one has to look at many details. Let me illustrate why.

Assume we have used the cut rule to prove a goal G and let pf be the sequent proof for the fact A. Now let us look at the proof for $H, A \vdash G$. If A were to remain untouched during that proof and we would use it just at the end in a `axiom` rule $H, A, H' \vdash A$ then we could simply replace this rule application by the proof pf.

But imagine if A has the form $P \wedge Q$ and A is decomposed somewhere in the proof *before* being used. Then we would have to deal with goals of the form $H, P, H' \vdash P$. Now we have to analyze the proof pf for A. If it used the rule `andR` to decompose A into P and Q, we could simply take the proof up to that point, drop the `andR` rule and the proof part for Q and insert the rest as proof for P instead of applying the `axiom` rule. But if `andR` was not used to prove A, then H must have contained a superformula of A, which at some point was decomposed into an occurrence of A on the left hand side. In this case we could take this proof fragment, decompose A further into P and Q and insert this whole proof before applying the `axiom` rule.

Now one has to take into account all the other connectives: disjunction, implication, negation and do the same. And to be precise, you has to provide an inductive proof because A may be a nested formula that is decomposed into very small pieces. All this tells us how to *eliminate a single application of the cut rule* from a proof. Now one proceeds inductively and eliminates all the cuts from bottom to top – assuming that after elimination the remaining proof branch is cut-free.

Of course, after this complex proof of the cut elimination theorem is complete, one tries to make it more elegant and the result of these considerations is what you see in Smullyan's book. It is not an easy proof but the fundamental ideas are not so complicated.

Actually, the argument becomes much easier if one takes evidence construction into account.

$$\begin{array}{lll} H \vdash G & \texttt{ev} = (\texttt{fun}\ a \to b[a])(pf) & \texttt{cut}\ A \\ \quad H \vdash A & \quad \texttt{ev} = pf & \\ \quad H, a{:}A \vdash G & \quad \texttt{ev} = g[a] & \end{array}$$

If a is a label for A and $g[a]$ the evidence for G in the second subgoal and pf the evidence for A in the first then constructing the evidence for G in the main sequent simply means applying the function that constructs $g[a]$ from a to the actual evidence pf for A. Eliminating the cut rule corresponds to simply evaluating that function, that is calculating the evidence that results from

replacing every occurrence of a in $g[a]$ by pf and continuing the computation until the evidence term cannot be reduced further. Converting the final evidence back into a proof gives us the result of cut elimination.[2]

There is also a cost to eliminating cuts. Our proofs will grow. In fact they can grow more than superexponentially – in some cases the size reduction achieved by introducing cuts is non-elementary. This of course holds only for pathological cases, but one should be aware of this nevertheless.

12.2 Consistency

To prove this consistency, we have to show how to convert a proof in refinement logic into a proof in a Gentzen system. In fact, we will show that every Refinement Logic rule can be expressed by a proof fragment in Gentzen's calculus.

If you start with a single conclusion, then all the right rules except for `orR1` and `orR2` are the same in both calculi. The `andL`, `orL`, and `axiom` rule are identical as well.

The refinement logic rules `orR1`, `orR2`, `impL`, and `notL` are subsumed by the corresponding Gentzen rules, which do exactly the same except for the fact that they keep more than one conclusion. So any RL proof that uses these rules without can certainly be simulated by a proof that uses the corresponding rules and keeps the extra conclusions around without ever using them. The `cut` rule can be eliminated and `magic` rule can be simulated as follows:

```
      H ⊢ G                 by cut A ∨ ∼A
[1]     H ⊢ A ∨ ∼A          by orR
        H ⊢ A, ∼A           by notR
        H, A ⊢ A            by axiom
[2]     H, A ∨ ∼ A  ⊢ G
```

What about `falseL`? This rule is actually *derivable* if one has the cut rule, since `f` is the same as a contradiction like `A ∧ ∼A`. The following proof fragment shows how to express the `falseL` rule in refinement logic with `magic` and `cut`.

```
     f ⊢ G                    by magic G
     f, G∨∼G ⊢ G              by orL 2
[1]    f, G ⊢ G               by axiom
[2]    f, ∼G ⊢ G              by cut ∼∼G
[2.1]    f, ∼G ⊢ ∼∼G          by notR
         f, ∼G, ∼G ⊢ f        by axiom
[2.2]    f, ∼G, ∼∼G ⊢ G       by notL 3
         f, ∼G ⊢ ∼G           by axiom
```

Thus we know how to simulate all the rules of refinement logic in the multi-conclusioned calculus and get the following theorem.

Theorem 12.2 *Refinement Logic is consistent*

[2]Obviously the details are equally complex because we need to define evaluation of evidence terms. But for this we can rely on the operational semantics of functional programming languages like ML. Note, however, that the resulting term can grow superexponentially depending on how often the lemmas are actually used.

12.3 Completeness

Finally, let us show completeness. Can we really do everything in refinement logic that we could do in multi-conclusioned sequent systems?

Theorem 12.3 *Any formula provable by Gentzen systems can be proved in the single-conclusioned sequent calculus with magic and cut.*

This seems plausible, but how do we prove it? Assuming that we start with a single conclusion – *can we simulate the multi-conclusioned rules by single-conclusioned ones?*

Look at the rules and compare them: $\wedge$L, $\wedge$R, $\vee$L, $\supset$R, $\sim$R, and `axiom` are really the same. But the other rules, $\vee$R, $\supset$L, and $\sim$L, *create* an additional conclusion, so *we can't simulate them directly.*

So we have to do something different. If we cannot express the rules directly, we have to *mimic a multi-conclusioned proof* with just one conclusion. Look at the meaning of multiple conclusions. We have said that $H \vdash G$ should be understood as: "under the assumption that all the formulas in H are true we can show that one of the formulas in G is true". So if G is $G_1, ..., G_n$ then

proving $H \vdash G$ *is the same as proving* $H \vdash G_1 \vee G_2 ... \vee G_n$

We will use this idea to simulate multi-conclusioned proofs in refinement logic. That is, we will express every multi-succedent rule that operates on $H \vdash G_1, G_2 ..., G_n$ by a series of refinement logic rules that operate on $H \vdash G_1 \vee G_2 ... \vee G_n$. Let me introduce an abbreviation:

For a list of formulas $G = G_1, G_2 ..., G_n$ *we define* G^* *to be the formula* $G_1 \vee (G_2 \vee (... \vee G_n))$.

I put the parentheses around the disjunctions to avoid ambiguities. If I omit parentheses, I mean exactly this way of parenthesizing. With this abbreviation we can state our simulation theorem.

Theorem 12.4 *Let* $H = H_1, ..., H_m$ *and* $G = G_1, ..., G_n$ *be lists of formulas. If* $H \vdash G$ *is provable in the multi-conclusioned sequent calculus, then* $H \vdash G^*$ *is provable in refinement logic.*

Q: *Why is this enough?*

An initial sequent has exactly one conclusion, and the theorem tells us that if $H \vdash G_1$ *is provable in the multi-conclusioned sequent calculus then it is also provable in refinement logic.*

Note that I said "lists of formulas" and not "sets". Although multi-succedent rules operate on sets, we can always assume that we have a standard way of listing the elements of these sets so that we know how G^* is related to $G = G_1, G_2 ..., G_n$. I need this only for proof purposes – I don't change the calculus.

Proof: We will proceed by induction over the depth of the multi-conclusioned sequent proof and show how to simulate each proof step in refinement logic.

I will sketch each case and leave the details as an exercise to the reader.

Base Case: If the depth of the proof for $H \vdash G$ is 1 then the proof must be the application of the `axiom` rule to some G_i, which also occurs in H as H_j.

$$H_1, ...G_i = H_j..., H_m \vdash G_1, G_2, .., G_i, ., G_n$$

We can essentialy do the same in refinement logic by using the rule `axiom` j. But we must keep in mind that the multi-conclusioned sequent rules can operate on *any* element of G, while we need to prove

$$H_1, ...G_i = H_j..., H_m \vdash G_1 \vee (G_2 \vee (..(G_i \vee .. \vee G_n)))$$

that is we have a goal formula with a fixed structure. So, to isolate G_i, we must move it to the left of the disjunction. We need four derived proof rules for this purpose.

$H \vdash$ `A∨(B∨C)` `by orAssocL` $H \vdash$ `(A∨B)∨C`

$H \vdash$ `(A∨B)∨C` `by orAssocR` $H \vdash$ `A∨(B∨C)`

$H \vdash$ `A∨B` `by orCommut` $H \vdash$ `B∨A`

$H \vdash$ `(A∨B)∨C` `by orCommutL` $H \vdash$ `(B∨A)∨C`

These rules can be derived in Refinement Logic. For instance, we can represent the rule `orCommut` by the following proof fragment, which tells us how to write a program (*tactic*) that applies proof rules in a way that we get the same effect as applying `orCommut`.

```
     H ⊢ A∨B                  by cut B∨A
1.     H ⊢ B∨A
2.     H, B∨A ⊢ A∨B           by orL m+1
2.1.   H, A ⊢ A∨B             by orR1
2.1.1.   H, A ⊢ A             by axiom m+1
2.2.   H, B ⊢ A∨B             by orR2
2.2.1.   H, B ⊢ B             by axiom m+1
```

So we can write a tactic that performs all these steps and we can consider `orCommut` as a derived inference rule.

By applying the four rules we can transform

$$H_1, ...G_i = H_j..., H_m \vdash G_1, G_2, .., G_i, ., G_n$$

into $H_1, ...G_i = H_j.., H_m \vdash G_i \vee (G_1 \vee (G_2 \vee (..(G_{i-1} \vee (Gi{+}1 \vee .. \vee G_n)))))$

and then proceed by applying `orR1` to get

$$H_1, ...G_i = H_j..., H_m \vdash G_i$$

and now we can apply `axiom` j to complete the proof.

Let me show you how this works when you want to isolate G_2 out of $G_1 \vee (G_2 \vee G_3)$.

```
     H ⊢ G1∨(G2∨G3)          by orAssocL
1.     H ⊢ (G1∨G2)∨G3        by orCommutL
1.1.   H ⊢ (G2∨G1)∨G3        by orAssocR
1.1.1.   H ⊢ G2∨(G1∨G3)      by orAssocR
```

In fact, it is possible to write tactics that implement the following two derived rules

$H \vdash G^*$ `by SwapOut` i $H \vdash G_i \vee G_i^*$

$H \vdash G_i \vee G_i^*$ `by SwapIn` i $H \vdash G^*$

Induction case: Assume that for all H,G, and all multi-conclusioned sequent proofs for $H \vdash G$ of depth n we find a refinement logic proof for $H \vdash G^*$. We show how to convert a multi-conclusioned sequent proof of depth $n{+}1$ into a refinement logic proof by looking at the *first* proof rule that was applied.

$\wedge$L**:** If the first step in the deduction was

$$\begin{array}{l} H_1, .H_j = A \wedge B .., H_m \vdash G \quad \texttt{by} \ \wedge\texttt{L} \\ H_1, .H_j = A, B .., H_m \vdash G \end{array}$$

then we can simulate this step by

$$\begin{array}{l} H_1, .H_j = A \wedge B .., H_m \vdash G^* \quad \texttt{by andL } j \\ H_1, .H_j = A, B .., H_m \vdash G^* \end{array}$$

By the induction assumption we can find a refinement proof for this subgoal because the multi-conclusioned sequent proof for $H_1, .H_j = A, B .., H_m \vdash G$ has depth n.

$\wedge$R**:** If the first step in the deduction was

$$\begin{array}{l} H \vdash G_1, .. G_{i-1}, \ A \wedge B, \ Gi{+}1, .. G_n \quad \texttt{by} \ \wedge\texttt{R} \\ \quad H \vdash G_1, .. G_{i-1}, \ A, \ Gi{+}1, .. G_n \\ \quad H \vdash G_1, .. G_{i-1}, \ B, \ Gi{+}1, .. G_n \end{array}$$

We need to show that we can express the following rule in refinement logic:

$$\begin{array}{l} H \vdash G_1 \vee .. G_{i-1} \vee \ A \wedge B \ \vee \ G_{i+1} \vee \ .. \vee G_n \\ H \vdash G_1 \vee .. G_{i-1} \vee \ A \ \vee \ G_{i+1} \vee \ .. \vee G_n \\ H \vdash G_1 \vee .. G_{i-1} \vee \ B \ \vee \ G_{i+1} \vee \ .. \vee G_n \end{array}$$

By the induction assumption we can find a refinement proof for the "subgoals" because the corresponding multi-succedent proofs have depth n.

It is sufficient to describe the following derived rule:

$$\begin{array}{ll} H \vdash A \wedge B \ \vee \ G_i^* & \texttt{andR}^* \\ \quad H \vdash A \ \vee \ G_i^* & \\ \quad H \vdash B \ \vee \ G_i^* & \end{array}$$

because if we apply `SwapOut` i before this rule and `SwapIn` i afterwards, we have exactly the rule we want to have.

In the rest of this proof we won't bother with this distinction anymore but always assume that we operate on the first formula of G^*.

So how do we implement `andR`*? We first cut in the desired subgoals $A \ \vee \ G_i^*$ and $B \ \vee \ G_i^*$ and then show that they are sufficient to prove $A \wedge B \ \vee \ G_i^*$:

$$\begin{array}{ll} H \vdash A \wedge B \ \vee \ G_i^* & \texttt{by cut } (A \ \vee \ G_i^*) \ \wedge \ (B \ \vee \ G_i^*) \\ \texttt{1.} \ \ H \vdash (A \ \vee \ G_i^*) \ \wedge \ (B \ \vee \ G_i^*) & \texttt{by andR} \\ \texttt{1.1.} \ \ H \vdash A \ \vee \ G_i^* & \\ \texttt{1.2.} \ \ H \vdash B \ \vee \ G_i^* & \\ \texttt{2.} \ \ H, \ (A \ \vee \ G_i^*) \ \wedge \ (B \ \vee \ G_i^*) \vdash A \wedge B \ \vee \ G_i^* & \texttt{by andL} \\ \texttt{2.1.} \ \ H, \ A \ \vee \ G_i^*, \ B \ \vee \ G_i^* \vdash A \wedge B \ \vee \ G_i^* & \texttt{by orL 1} \\ \texttt{2.1.1.} \ \ H, \ A, \ B \ \vee \ G_i^* \vdash A \wedge B \ \vee \ G_i^* & \texttt{by orL 2} \\ \texttt{2.1.1.1.} \ \ H, \ A, \ B \vdash A \wedge B \ \vee \ G_i^* & \texttt{by orR1} \\ \texttt{2.1.1.1.1.} \ \ H, \ A, \ B \vdash A \wedge B & \texttt{by andR} \\ \texttt{2.1.1.1.1.1.} \ \ H, \ A, \ B \vdash A & \texttt{by axiom } m{+}1 \\ \texttt{2.1.1.1.1.2.} \ \ H, \ A, \ B \vdash B & \texttt{by axiom } m{+}2 \end{array}$$

$2.1.1.2.\quad H,\ A,\ G_i^* \vdash A \wedge B \ \vee\ G_i^*$ by orR2
$2.1.1.2.1.\quad H,\ A,\ G_i^* \vdash G_i^*$ by axiom m+2

$2.1.2.\quad H,\ G_i^*,\ B\ \vee\ G_i^* \vdash A \wedge B \ \vee\ G_i^*$ by orR2
$2.1.2.1\ H,\ G_i^*,\ B\ \vee\ G_i^* \vdash G_i^*$ by axiom m+1

So we have found an "implementation" of andR*

$\vee$L: can be simulated directly as in the case of andL

$\vee$R: Assuming that we operate on the first formula of the set the first step was

$H \vdash A \vee B,\ G$ by $\vee$R
$H \vdash A,\ B,\ G$

We need to show that we can express the following rule in refinement logic:

$H \vdash A \vee B \ \vee\ G^*$ by orR*
$H \vdash A \vee (B \ \vee\ G^*)$

which is exactly the same as orAssocR

$\supset$L: If the first step in the deduction has been

$H_1, .H_j = A \supset B .., H_m \vdash G$ by $\supset$L
$H_1, .H_j = A \supset B .., H_m \vdash A, G$
$H_1, .H_j = B .., H_m \vdash G$

Then we can simulate this first step by

$H_1, .H_j = A \supset B .., H_m \vdash G$ by impL*
$H_1, .H_j = A \supset B .., H_m \vdash A \vee G$
$H_1, .H_j = B .., H_m \vdash G$

To implement impL* we proceed as before

$H_1, .H_j = A \supset B .., H_m \vdash G$ by cut $(A \vee G) \ \wedge\ (B \supset G)$
$1.\quad H_1, .H_j = A \supset B .., H_m \vdash (A \vee G) \ \wedge\ (B \supset G)$ by andR
$1.1.\quad H_1, .H_j = A \supset B .., H_m \vdash A \vee G$
$1.2.\quad H_1, .H_j = A \supset B .., H_m \vdash B \supset G$ by impR
$1.2.1\ H_1, .H_j = A \supset B .., H_m, B \vdash G$

$2.\quad H_1, .H_j = A \supset B .., H_m,\ (A \vee G) \ \wedge\ (B \supset G) \vdash G$ by andL m+1
$2.1.\quad H_1, .H_j = A \supset B .., H_m,\ A \vee G,\ \ B \supset G\ \vdash G$ by orL m+1
$2.1.1.\quad H_1, .H_j = A \supset B .., H_m,\ A,\ \ B \supset G\ \vdash G$ by impL j
$2.1.1.1.\quad H_1, .H_j = A \supset B .., H_m,\ A,\ \ B \supset G\ \vdash A$ by axiom m+1
$2.1.1.2.\quad H_1, .H_j = B .., H_m,\ A,\ \ B \supset G\ \vdash G$ by impL m+2
$2.1.1.2.1.\quad H_1, .H_j = B .., H_m,\ A,\ \ B \supset G\ \vdash B$ by axiom j
$2.1.1.2.2.\quad H_1, .H_j = B .., H_m,\ A,\ \ G\ \vdash G$ by axiom j

$2.1.2.\quad H_1, .H_j = A \supset B .., H_m,\ G,\ \ B \supset G\ \vdash G$ by axiom m+1

$\supset$R: If the first step in the deduction has been

$H \vdash A \supset B,\ G$ by $\supset$R
$H,\ A \vdash B,\ G$

then we need to show that we can express the following rule in refinement logic:

$H \vdash A \supset B \ \vee\ G^*$ impR*
$H,\ A\ \vdash B \ \vee\ G^*$

I leave this as exercise.

∼L: If the first step in the deduction has been

$$\begin{array}{l} H_1, .H_j = {\sim}\mathtt{A}.., H_m \vdash G \quad \mathtt{by}\ {\sim}\mathtt{L} \\ H_1, .H_j = {\sim}\mathtt{A}.., H_m \vdash A, G \end{array}$$

then we can simulate this first step by

$$\begin{array}{l} H_1, .H_j = {\sim}\mathtt{A}.., H_m \vdash G \quad \mathtt{by}\ \mathtt{notL}^* \\ H_1, .H_j = {\sim}\mathtt{A}.., H_m \vdash A \vee G \end{array}$$

Since $\sim A$ can be viewed as abbreviation for $A \supset \mathtt{f}$ this works precisely as $\mathtt{impL}^*$

∼R: If the first step in the deduction has been

$$\begin{array}{l} H \vdash {\sim}A,\ G \quad \mathtt{by}\ {\sim}\mathtt{R} \\ H,\ A \vdash G \end{array}$$

then we need to show that we can express the following rule in refinement logic:

$$\begin{array}{l} H \vdash {\sim}A \vee G^* \quad \mathtt{notR}^* \\ H,\ A \vdash G^* \end{array}$$

Since $\sim A$ can be viewed as abbreviation for $A \supset \mathtt{f}$ this works precisely as $\mathtt{impR}^*$

So in all cases we have shown how to simulate multi-succedent proofs using refinement logic. We need the `magic` rule only for implication and negation on the right, so we see where it is actually needed to achieve completeness. So as a result we get:

Theorem 12.5 *The Refinement Logic calculus is complete*

I have presented the inductive proof in a rule-fashion because this enables us to write an algorithm that converts multi-succedent proofs into refinement logic proofs. Because of the rules `SwapOut` and `SwapIn` this proof will in the end be much bigger than the original multi-succedent proof. It gets even worse if we try to eliminate the cuts as well.

But there is no way out. Although refinement logic proofs are sufficient to prove that a formula is a tautology, they cannot be as short as multi-succedent proofs. In the worst case, they must be exponentially longer. For those of you who are interested, here is an example:

For $n \geq 0$ consider the propositional formula class F_n

$$F_n \quad \equiv \quad A_n \wedge \bigwedge_{i=1}^{n-1} \underbrace{(B_{i-1} \supset (B_i \vee A_i) \vee B_i)}_{O_i} \wedge \underbrace{((B_0 \vee A_0) \vee B_0)}_{O_0} \quad \supset \quad A_0 \vee \bigvee_{i=0}^{n-1} \underbrace{B_i \wedge A_{i+1}}_{N_i}$$

Now if you look at the properties of these formulas you find out

- F_0 needs one leaf in Gentzen Systems and Refinement Logic
- there exist a Gentzen proof of F_n with $6n - 2$ leaves (branches)
- F_n implies a reduction ordering $O_j \longmapsto O_{j+1} \longmapsto N_j$ on Refinement Logic proofs
- each (cut free) Refinement Logic proof of F_n requires at least $5(2^n - 1)$ leaves

This result, however, describes a pathological case. In the average the translation gets much better results.

12.4 Can a proof search get wedged?

The example proof of Pierce's law that we discussed on Tuesday seems to indicate that. Just imagine we had used a strategy that first tries to decompose the available formulas before attempting `magic`. This was a reasonable strategy for the tableau method and for Gentzen Systems. We could prove it to terminate. But when we apply it to Pierce's law, our proof gets stuck at the same goal where the calculus without `magic` got stuck.

```
    ⊢ ((P⊃Q)⊃Q)⊃P       by impR
    (P⊃Q)⊃P ⊢ P          by impL
[1]    (P⊃Q)⊃P ⊢ P⊃Q    by impR
       (P⊃Q)⊃P, P ⊢ Q    by ???
[2]     P ⊢ P              by axiom
```

No application of the magic rule will help us anymore. Once we have applied `impL` we are required to prove `Q` and a case analysis will not help us to do that.

We get a similar problem with the following two proof attempts for the law of excluded middle.

```
⊢ P∨∼P      by ∨R1          ⊢ P∨∼P      by ∨R2
⊢ P         ??              ⊢ ∼P        ??
```

Q: *What is the reason? Why do proofs in refinement logic get stuck?*

The reason is that as soon as we apply the `impL` or the `orR`i rules, we have lost information that we had before. Applying `impL` drops the conclusion that we had before and we are left with a completely different conclusion in the first subgoal. Formerly, we preserved the original goal as another alternative and we could decide later which one we wanted to prove. Now we make the choice as soon as we apply `impL`. Similarly, the `orR`i rules force us to make choices early in the proof and to drop the other alternative.

Both kinds of rules are *irreversible*, that is the subgoals are not equivalent to the main goal anymore but stronger. Applying the rule forces us to prove something stronger than the original goal and that may get us stuck. At that point we need to backtrack to the application of `impL` or the `orR`i and proceed in a different fashion. This makes searching for a proof much more difficult.[3]

12.4.1 What kind of logic do we get if drop the `magic` rule?

The original idea of the sequent calculus is to show that we can construct a proof for a goal G from a given set of hypotheses H. If G is a single conclusion, then the whole construction has to focus on that goal and the proof rules tell us exactly how to construct *evidence* for the truth of G.

The `andR` rule, for instance tells us: "to prove $A \wedge B$ you have to prove both A and B" – evidence for A and evidence for B together is sufficient evidence for $A \wedge B$. The `orR`i rules for $A \vee B$ require us to make a choice between A and B. If we find evidence for either of them, we have sufficient evidence for $A \vee B$. `impR` is similar: if we show how to build evidence for B from evidence for A then we certainly have enough evidence for the implication $A \supset B$.

[3] In fact, refinement logic without the `magic` rule is co-PSPACE complete, which is considered a much more difficult problem than the co-NP complete validity problem of classical propositional logic.

We also could view the goal G as a *task* that needs to be fulfilled and the rules tell us how to solve that task. "to solve $A \wedge B$ you must solve both A and B", etc. A third view looks at G as an *event* that happens, assuming that the events in the list H have occurred.

All these views are very much influenced by a computer scientists view of logic. Formulas are not just truths, but tied to constructions. And it is the construction that interests us.

Refinement logic without the magic rule is highly constructive. It doesn't just tell us that something is true but gives us the precise reasons why. It shows us how to construct a solution. In fact, our Nuprl proof system, which uses this refinement logic, is capable of extracting this construction from a proof as executable algorithm.

Although refinement logic without `magic` is not complete with respect to propositional logic as defined in Smullyan's book (so-called *classical* logic), it represents a very well known logic, called *constructive*, *intuitionistic*, or *computational* logic, which is relevant for reasoning about programs.

The magic rule destroys constructivity, computability, and often even an intuitive understanding, because it states that one of two alternatives, P or not P, must be true. We don't know which of the two, but we are assured that there is no third alternative, so one must be the case. Using this postulate makes it very easy to prove certain statements since one doesn't have to worry how to provide the necessary evidence. But for a computer scientist, this postulate is very unsatisfactory.

Let me give you one example of a proof that uses this law in a very questionable, but absolutely correct way:

> *We want to prove that there are two irrational numbers x and y such that x^y is rational.*
>
> Consider $a = \sqrt{2}^{\sqrt{2}}$. Then a is either rational or not. In the first case choose $x = y = \sqrt{2}$, in the other choose $x = a$ and $y = \sqrt{2}$.
> That completes the proof but we still don't know what x and y are. We have been given two alternatives and told that one of the two is our solution.

This indicates that using the law of excluded middle often goes against our intuition because it gives a very abstract view of truth that is not required to provide any specific information about the choices that have to be made.

12.5 Decidability

Refinement logic is not only a complete and correct method for constructing proofs but also helps us *decide* whether a formula is true or false. In fact, for any sequent $H \vdash G$ it is possible to decide whether it is *valid* (i.e.the corresponding implication is a tautology) or can be falsified by an assignment of values to its variables.

Theorem 12.6 (Decidability of Refinement Logic)
$\forall S{:}SEQUENT.\ valid(S) \vee \exists v_0{:}Var_S \to \mathbb{B}.\ SValue(S,v_0)=f$

James Caldwell has given a formal and constructive proof of this theorem in his PhD thesis. The proof implicitly constructs a tableau that either proves the validity of the sequent or provides a counterexample for it. Since we can convert a tableau into a Gentzen proof and that one into Refinement Logic, we know that Refinement Logic is decidable.