Program Verification: Lecture 7

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Local Confluence

Call a rewrite theory (Σ, B, R) locally confluent iff whenever we have $t \longrightarrow_{R/B} u$ and $t \longrightarrow_{R/B} v$, then $u \downarrow_{R/B} v$.

Exercise. Prove that if (Σ, B, R) is terminating and locally confluent, then it is confluent

Hint: use well-founded induction (see *STACS*) on the well-founded relation $\longrightarrow_{R/B}$.

Checking Confluence: the Church-Rosser Checker

The Maude Church-Roser Checker, tries, under the assumption of termination, to check the confluence property by checking local confluence.

Given a B-terminating rewrite theory $(\Sigma, B, \vec{E})$ (with Σ B-preregular) as a input, two things can happen:

1. If $(\Sigma, B, \vec{E})$ is both confluent and sort-decreasing, the tool will respond confirming both properties.
2. Otherwise, the tool will provide counterexamples to either confluence or sort-decreasingness.

In Case (2), $(\Sigma, B, \vec{E})$, although not confluent, may still be ground confluent, i.e., be OK for execution. More reasoning will be needed to see if ground confluence holds or not.

Checking Confluence: the Church-Rosser Checker (II)

In case the check fails, the proof obligations returned can be very useful for further analysis, either to establish ground confluence, or to find a conterexample.

The Church-Rosser Checker is part of the Maude Formal Environment. It extends Full Maude, and checks the confluence of Full Maude functional modules (assuming termination).

The module to be checked, say FOO, should have been declared in Maude. We then give to the Church-Rosser Checker the command,

```
Maude> (check Church-Rosser FOO .)
```

enclosed in parentheses (followed by carriage return).

Checking Confluence: the Church-Rosser Checker (III)

We can illustrate the use of the Church-Rosser Checker with our running example of natural number addition.

```
fmod NAT-MIXFIX is
 sort Natural .
 op 0 : -> Natural [ctor] .
 op s_ : Natural -> Natural [ctor] .
 op _+_ : Natural Natural -> Natural .
 vars N M : Natural .
 eq N + 0 = N .
 eq N + s M = s(N + M) .
 endfm
```

Checking Confluence: the Church-Rosser Checker (IV)

Assuming a separate proof of termination, we can check `NAT-MIXFIX` confluent and sort-decreasing by giving the command:

```
 (check Church-Rosser NAT-MIXFIX .)
```

```
Church-Rosser checking of NAT-MIXFIX
Checking solution :
The specification is Church-Rosser .
```

Ground Confluent but not Confluent

Sometimes a module will not pass the check, not because there is any real problem with its equations, but simply because it is ground confluent but not confluent.

In such a case, the tool will return a set of critical pairs as proof obligations. Such critical pairs are equations $t = t'$ such that:

- $E \cup B \vdash \ t = t'$, but
- the joinability property $t \downarrow_{\vec{E}/B} t'$ fails.

But, as we shall see, these pairs are sufficient, as proof obligations, to establish ground confluence. That is, if we can show $\overline{\theta}(t) \downarrow_{\vec{E}/B} \overline{\theta}(t')$ for each ground substitution θ, then E is indeed ground confluent (assuming termination).

Ground Confluent but not Confluent (II)

We can illustrate ground confluence with the following module,

```
fmod ANOTHER-NAT is
  sorts Zero Natural .
  subsort Zero < Natural .
  op 0 : -> Zero .
  op s_ : Natural -> Natural .
  ops (_+_) (_*_) : Natural Natural -> Natural [comm] .
  vars N M : Natural .
  eq 0 + N = N .
  eq s N + M = s (N + M) .
  eq 0 * N = 0 .
  eq s N * M = M + (N * M) .
endfm
```

Ground Confluent but not Confluent (II)

```
(check Church-Rosser ANOTHER-NAT .)

Church-Rosser checking of  ANOTHER-NAT
Checking solution :
var N : Natural .
var N@ : Natural .
cp s ( N + ( N@ + ( N * N@ ) ) ) = s ( N@ + ( N + ( N * N@ ) ) )
rewrites: 1368 in 0ms cpu (10ms real) (~ rewrites/second)
```

Ground Confluent but not Confluent (II)

Where does this critical pair come from? It comes from applying the equation

```
 eq s N * M = M + (N * M) .
```

modulo commutativity to the term `s N * s M` in two different ways yielding terms that, after further simplification,

```
s M + (N * s M) = s(M + (N + (N * M)))
```

```
s N + (M * s N) = s(N + (M + (N * M)))
```

cannot be further simplified, and therefore cannot be joined, showing that the equations are not confluent. However, every ground instance can be joined.

Ground Confluent but not Confluent (III)

What can we do in such a situation? One of four things:

1. use the critical pair as useful information to transform the equations into equivalent equations that are confluent and pass the test; or

2. transform the theory, not by changing the equations, but by adding some more axioms (here, adding `assoc` to $+$ will work); or

3. prove an inductive theorem about the rewriting relation $\longrightarrow_E$ itself, not about equality!, showing that for each ground instance the pair can be joined; or

4. find a conterexample disproving ground confluence.

Ground Confluent but not Confluent (IV)

In our example, alternative (1) yields a transformed module, by realizing that the equation

```
eq s N * M = M + (N * M) .
```

is in a sense too general, since, the case `M = 0` is covered by the other equations for addition and multiplication. Therefore, we can assume `M = s M'`, and replace the above equation by the more specialized equation,

```
eq s N * s M = s((N + M) + (N * M)) .
```

to get the confluent transformed module,

Ground Confluent but not Confluent (V)

```
fmod CONFLUENT-NAT is
  sorts Zero Natural .
  subsort Zero < Natural .
  op 0 : -> Zero .
  op s_ : Natural -> Natural .
  ops (_+_) (_*_) : Natural Natural -> Natural [comm] .
  vars N M : Natural .
  eq 0 + N = N .
  eq s N + M = s (N + M) .
  eq 0 * N = 0 .
  eq s N * s M = s((N + M) + (N * M)) .
endfm
```

Ground Confluent but not Confluent (VI)

```
(check Church-Rosser CONFLUENT-NAT .)
```

```
Church-Rosser checking of CONFLUENT-NAT
Checking solution :
The specification is Church-Rosser .
```

Justification of the Church-Rosser Checker

The justification will be somewhat incomplete, since it will be resticted to term rewriting systems $(\Sigma, \vec{E})$ (i.e., $B = \emptyset$). A full account of all the issues involved, covering both the rewriting modulo B case and conditional rewrite rules, can be found in:

F. Durán and J. Meseguer, "On the Church-Rosser and Coherence Properties of Conditional Order-Sorted Rewrite Theories," *JLAP*, 81, 816–850, 2012. Tech Report version available online at the UIUC IDEALS Repository: `http://hdl.handle.net/2142/17384`

Justification of the Church-Rosser Checker (II)

The Church-Rosser Checker does two things:

- check that the equations are sort-decreasing; and
- check confluence (assuming termination) by checking local confluence for all possible critical pairs.

Checking sort-decreasingness is relatively easy to do, since, as explained in Lecture 6, assuming Σ is preregular, it reduces to checking for each rewrite rule $t \to t'$ that

$$ls(t\rho) \geq ls(t'\rho)$$

for each variable specializations ρ. This is easy, since if Σ is finite there is only a finite number of such specializations.

Justification of the Church-Rosser Checker (III)

Consider, for example, that we want to check the sort-decreasingness of an equation, with `I` of sort `Integer`,

```
eq I + 0 = I .
```

in a module `INTEGER` with two declarations of addition,

```
op _+_ : Natural Natural -> Natural .
op _+_ : Integer Integer -> Integer .
```

and with subsorts `NzInteger`, `NzNatural`, and `Natural`. Then, to check that the equation is sort decreasing, it is enough to check that the sort of the lefthand side is greater or equal to that of the righthand side for the original equation, and for the equations obtained replacing `I` by a variable in each of the subsorts.

Justification of the Church-Rosser Checker (IV)

Why should we check sort-decreasingness as well as (local) confluence? Because, besides being an important property, lack of sort decreasingness can also cause lack of confluence.

The rewrite rules shown in Lecture 6,
$\vec{E} = \{c \to d, f(f(x : C)) \to f(x : C)\}$ with $c : C$, $d : D$, and $C < D$, have lefthand sides with no symbols in comon. As we shall see, this is the best possible situation for confluence, since then there are no critical pairs.

However, $\vec{E}$ is not confluent. Indeed, we can rewrite $f(f(c))$ to both $f(d)$ and $f(f(d))$, which cannot be further rewritten.

Justification of the Church-Rosser Checker (V)

So, the main questions remaining are:

- what is a critical pair? and
- why is checking joinability of critical pairs sufficient for checking confluence under the termination (and the already checked sort-decreasingness) assumptions?

Justification of the Church-Rosser Checker (VI)

As mentioned earlier in the lecture, a set E of equations that is locally confluent and terminating is confluent.

Therefore, under the termination assumption, all we need to do is to convince ourselves that, given a term t, and given two one-step simplifications $t \longrightarrow_E t'$, and $t \longrightarrow_E t''$, we always can join t' and t''. That is, we always have, $t' \downarrow_E t''$.

The crucial point, then, is to analyze where in t do the rewrites $t \longrightarrow_E t'$, and $t \longrightarrow_E t''$ happen. For this we need to talk about positions in a term.

Term Positions and Subterm Occurences

Each Σ-term can be viewed as a tree in the obvious way. Each position in the tree can be denoted by a string of natural numbers, indicating the path that we must follow to go down in the tree and reach the position.

At each level, the corresponding number in the string indicates the argument position on which we must go down, to finally reach the desired position. For example, the term $f(h(d), q(b, a), g(a, k(c)))$ has the subterm $k(c)$ at position 3.2.

Given a Σ-term t and a position p we denote by t_p the subterm occuring at that position; thus, $f(h(d), q(b, a), g(a, k(c)))_{3.2} = k(c)$.

Notation for Term Decomposition at a Position

Given a position $p \in (\mathbf{N} - \{0\})^*$ in a term t we denote by $t[]_p$ the context obtained by placing a hole $[]$ at position p. For example, $f(h(d), q(b, a), g(a, k(c)))[]_{3.2} = f(h(d), q(b, a), g(a, []))$.

Therefore, if p is a position in t, we obtain a context-subterm decomposition of t as the pair $(t[]_p, t_p)$, For example, $f(h(d), q(b, a), g(a, k(c)))$ decomposes at 3.2 as the contex-subterm pair $(f(h(d), q(b, a), g(a, [])), k(c))$

Given a context $t[]_p$ and a term u, the result of replacing the hole by u, that is, the term $(t[]_p)[u]$ is abbreviated to $t[u]_p$. Of course, if $u = t_p$ we have the identity $t = t[t_p]_p$.

Justification of the Church-Rosser Checker (VII)

Recall that a simplification step with E must happen at a given position p in t. Therefore, for $t \longrightarrow_E t'$, and $t \longrightarrow_E t''$ we must have two positions, p and q in t and two oriented equations $u = v$ and $u' = v'$ in E with substitutions θ and μ such that:

- $t = t[u\theta]_p = t[u'\mu]_q$;
- $t' = t[v\theta]_p$;
- $t'' = t[v'\mu]_q$.

all now hinges upon where p and q are located in t.

Justification of the Church-Rosser Checker (VIII)

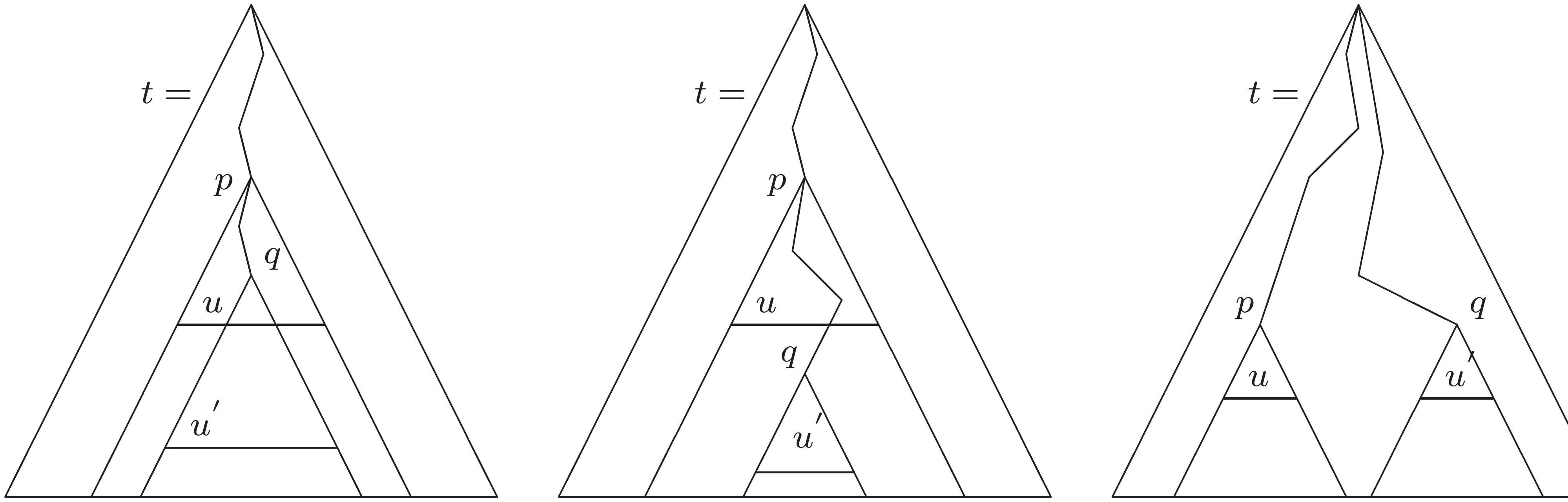
There are essentially three possibilities (see the picture):

1. **nested simplification with overlap**: there is a path r such that $q = p.r$ (or $p = q.r$, but this case is symmetric) and r is a nonvariable position in u;

2. **nested simplification without overlap**: there is a path r such that $q = p.r$, but r is not a nonvariable position in u;

3. **sideways simplification**: there isn't such an r at all.

Three Possibilities for p and q



Justification of the Church-Rosser Checker (IX)

In the sideways case, where neither $q = p.r$, nor $p = q.r$, the positions are totally independent, in the sense that we have:

$$t = (t[u\theta]_p)[u'\mu]_q = (t[u'\mu]_q)[u\theta]_p.$$

Therefore, we have:

- $t' = t[v\theta]_p = (t[u'\mu]_q)[v\theta]_p = (t[v\theta]_p)[u'\mu]_q$; and

- $t'' = t[v'\mu]_q = (t[u\theta]_p)[v'\mu]_q = (t[v'\mu]_q)[u\theta]_p$.

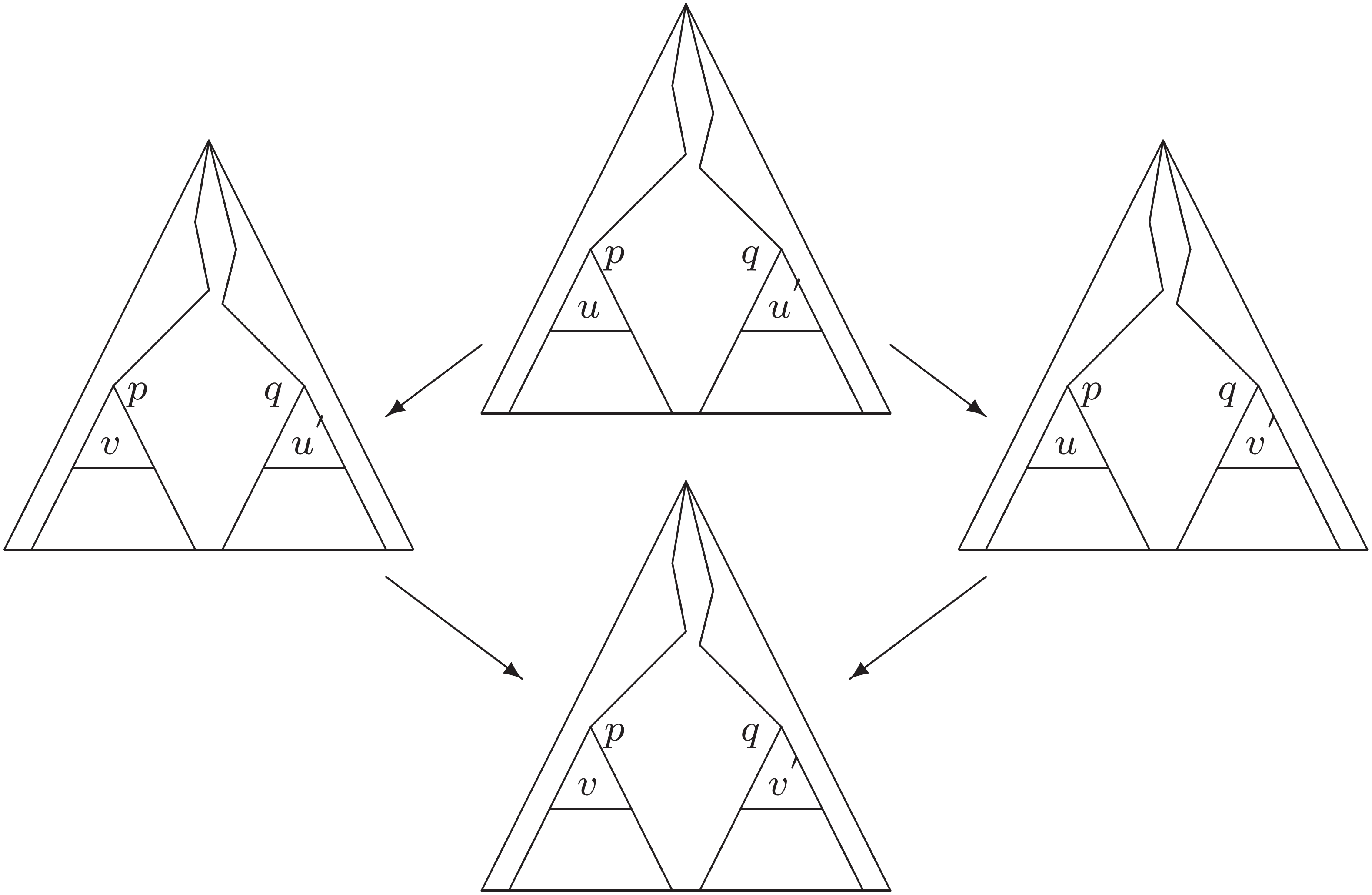
Justification of the Church-Rosser Checker (X)

Therefore we have a term w of the form (see the picture):

$$w = t'[v'\mu]_q = (t[v\theta]_p)[v'\mu]_q = (t[v'\mu]_q)[v\theta]_p = t''[v\theta]_p.$$

and therefore, $t' \longrightarrow_E w$, and $t'' \longrightarrow_E w$.

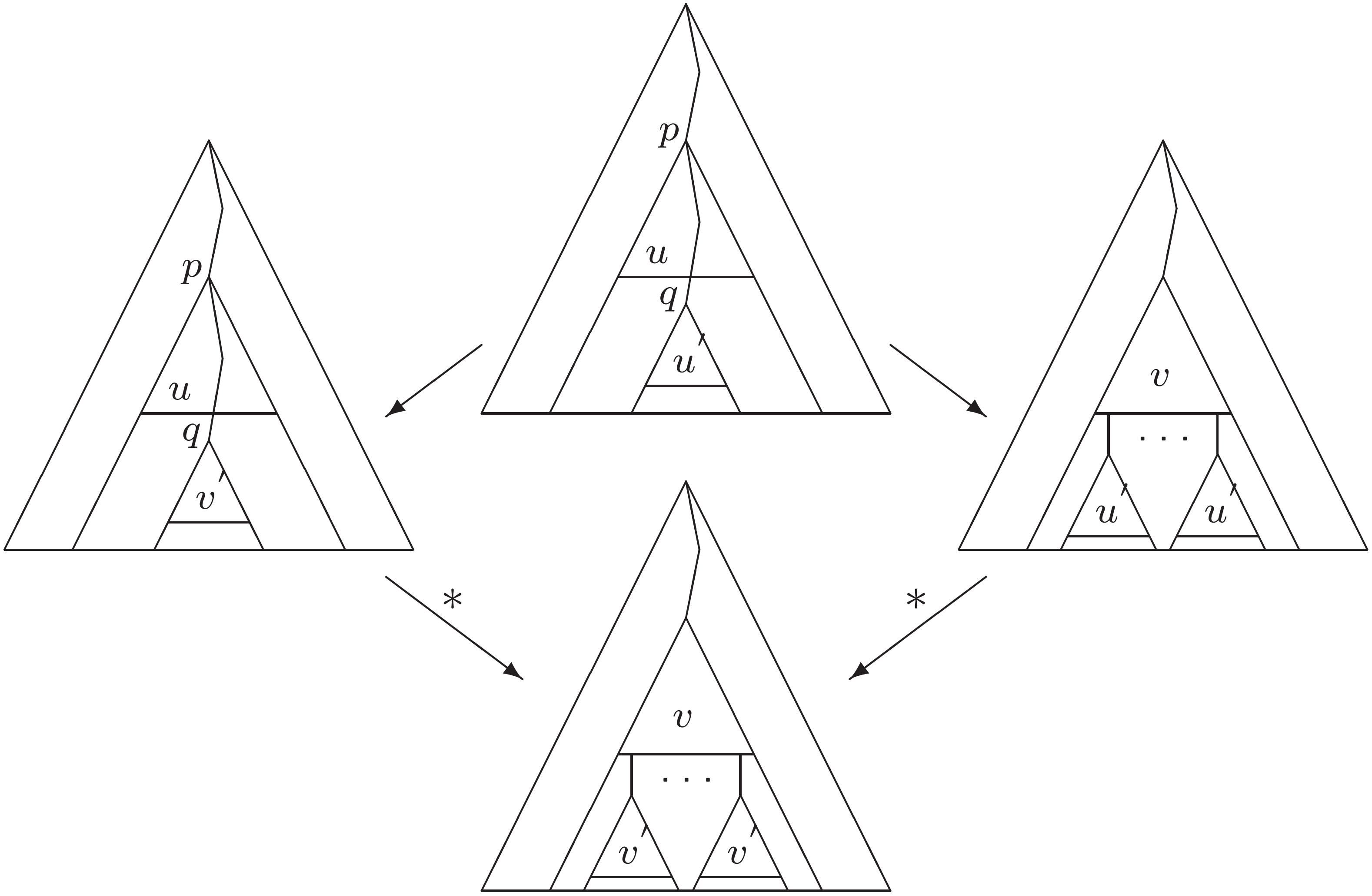
Sideways Simplification



Justification of the Church-Rosser Checker (XI)

The case of nested simplification without overlap is also always joinable. The detailed proof is left as an exercise; but note that, since a single variable in u may occur several times in v, the occurrence of u' underneath u may be copied several times by the simplification with the equation $u = v$. This means that to reach a common w from t' several steps of simplification may be needed (see the picture).

Nested Simplification without Overlap



Nested Simplifications with Overlap

Therefore, we have reduced the confluence property (assuming termination) to the joinablilty problem for nested simplifications with overlap in which we have equations $u = v$ and $u' = v'$ in E (**Note**: $u' = v'$ could be the same equation $u = v$ considered twice!) and should consider terms t with positions p and $p.r$, and a substitution $\alpha = \theta \uplus \rho$ such that:

1. $t_p = u\alpha$;

2. r is a nonvariable position in u, and $t_{p.r} = u_r\alpha = u'\alpha$.

This formulation of the overlap situation assumes that the variables in u and u' are disjoint, or have been made so by renaming their variables, even in the case when the equation $u = v$ is considered twice (self-overlap).

Context-Free Nested Simplifications with Overlap

The joinabiliy problem for nested simplifications with overlap then consists in showing,

$$(*) \quad t[v\alpha]_p \downarrow_E (t[u\alpha[v'\alpha]_r]_p).$$

Our next reduction of the problem comes from the observation that the context t in which all this happens is irrelevant. That is, we can reduce the problem to that of checking joinability for all context-free nested simplifications with overlap of the form,

$$(\flat) \quad v\alpha \downarrow_E u\alpha[v'\alpha]_r.$$

Context-Free Nested Simplifications with Overlap (II)

Of course, $(*) \Rightarrow (\flat)$, but we also have $(\flat) \Rightarrow (*)$, because of the following,

Context Lemma: Let $t = t[u]_p \in T_\Sigma(X)$, and suppose that we have, $u \overset{*}{\longrightarrow}_E v$. Then we have, $t[u]_p \overset{*}{\longrightarrow}_E t[v]_p$.

Proof: It is obviously enough to check it for one step $(\longrightarrow_E)$, that is, for $u \longrightarrow_E v$. But by sort-decreasingness of $\longrightarrow_E$ we then have a well-formed term $t[v]_p \in T_\Sigma(X)$, where if the rewriting of u happened at, say, position r, then the rewriting of $t = t[u]_p$ now happens at position $p.r$ and yields, $t[u]_p \longrightarrow_E t[v]_p$. q.e.d.

Justification of the Church-Rosser Checker (XII)

Therefore, we have so far reduced the problem of confluence (under the termination assumption) to the considerably simpler problem of joinability of context-free nested simplifications with overlap, and the question still remains,

What is a critical pair? Stay tuned!