Coinductive Proof Nets

Doctoraalscriptie voor de studie Cognitieve Kunstmatige Intelligentie

Laura Korte
Laura.Korte@phil.uu.nl

Begeleider: Dr. V. van Oostrom

June 21, 2002

'Ignorance of Axioms', the Lecturer continued, 'is a great drawback in life. It wastes so much time to have to say them over and over again. For instance, take the Axiom, *"Nothing is greater than itself"*; that is, *"Nothing can contain itself"*. How often do you hear people say "He was so excited, he was quite unable to contain himself." Why of *course* he was unable! The *excitement* had nothing to do with it!'

***Sylvie and Bruno Concluded* - Lewis Carroll**

Contents

Acknowledgments

First of all, I would like to give a major thank you to my supervisor Vincent van Oostrom, for making proof theory look like fun, for his seemingly infinite knowledge about rewriting and for the time and effort he put into this project. I would also like to thank Michael Moortgat, Michele Abrusci and Richard Moot for reading my thesis and providing feedback. And last, but certainly not least, my parents for their love and support. Many thanks for being there.

1 Introduction

Ever since the concept Artificial Intelligence was first introduced, logic has played a major role in its research and is still one of the first courses to be taught in every AI-study. The intuition that our thoughts can be formalized using logic is as appealing as ever.

However, AI-research has made a lot of progress since the introduction of Classical Logic. Many refinements have been proposed and many far more expressive[1] kinds of logic are now available: non-monotonic logic, modal logic, linear logic, etc. They each try to fix one of the many deficiencies of Classical Logic with regard to representing human reasoning. One of those deficiencies is the inability to reason (or prove a formula) with limited resources. The logic that offers a solution to this is Linear Logic. We will see more of this logic later on.

Another striking deficiency of classical logic, when it comes to simulating human reasoning, is the inability to represent infinite objects, whereas people have the ability to reason about infinite objects like the set of all natural numbers. For this purpose, we will in this thesis investigate the so-called *Coinductive Proof Nets*, a coinductive version of the already well known Inductive Proof Nets. The main difference between the two is that the former handles both finite and infinite objects, while the latter is restricted to finite objects. Showing what effect enriching Inductive Proof Nets with infinite objects has on the properties of the system will be the main theme of this thesis.

But what about proof nets? What are they and why would one want to use these nets? The answer to this question is *proof identification*[2]. In sequent calculus, the proof system used to automatically prove CL-formulas, one often gets several distinct proofs for one and the same formula. But are they really distinct? It turns out that sometimes they are but quite often, they are not. Quite often they are in fact the exact same proof modulo the order in which the rules are applied. Proof nets offer an elegant way to identify these identical proofs: each proof is mapped to a proof net and proofs that are identical (modulo rule order) are mapped to the same net.

After the motivation in **Section 2** and the preliminaries in **Section 3**, we will give a short introduction to (type-free) inductive proof nets in **Section 4** and take a look at the properties of proof net reduction. A proof of confluence for directed proof net reduction can also be found in this section.

In **Section 5** we turn to (type-free) coinductive proof nets. We will give a formal definition and take a look at the similarities and differences between inductive and coinductive proof net reduction. The confluence proof from the previous section will also be extended to coinductive proof net reduction.

In **Section 6** we will investigate typed proof nets and typed proof net reduction, strong normalization in particular. In **Section 7**, the coinductive version of these typed proof nets will be introduced and once again we will look at

[1]While they are not more expressive in the sense that they are more powerful than Classical Logic, these logics do make expressing certain kinds of expressions a lot easier and more elegant.

[2]note that this identification only fully works for the multiplicative fragment of linear logic.

the similarities and differences between typed inductive and typed coinductive proof net reduction. Furthermore a proof of head normalization for typed coinductive proof nets will be presented. Finally, in **Section 8**, the conclusions, we will look back at what we have accomplished and at future work.

2 Motivation

2.1 Theoretical Motivation

The main reason for the introduction of the coinductive proof nets discussed in this thesis, is the existence of the coinductive λ-calculus (see [Joa01]). From [Dan90] [Reg92] we know that a correspondence between λ-calculus and proof nets holds, so if there exists an infinite version of the λ-calculus, it is interesting to ask ourselves the question whether or not there is a useful infinite version of proof nets as well.

The relation between proof nets and λ-calculus is one of implementation: an arbitrary λ-term can be implemented in a proof net. And the reason we want λ-terms to be implemented in proof nets is a very simple one: it allows a more efficient reduction of λ-terms, because in proof net reduction, duplication is controled. While β-reduction of λ-calculus can create an arbitrary number of copies of a term[3], proof net reduction can create only one (per step).

2.2 Applications

This correspondence between proof nets and λ-terms also demonstrates the connection between proof nets and artificial intelligence. Proof nets can be used in every single one of the many applications within the field of artificial intelligence that makes use of the λ-calculus. One example of such an application is the use of λ-terms to represent semantics in natural language processing, and if typed, proof nets are ideal for representing sentences in categorial grammar. There is even some research which supports the claim that humans use proof-net-like structures to process natural language! The reader is referred to [Joh98] and [Mor00] for details of this research project. See also [Moo02] and [GR96] for more information about the use of proof nets in natural language analysis in general.

Furthermore, because of the correspondence between proof nets and λ-terms, proof nets present an efficient tool to implement functional programming languages like Haskell or Lisp [AG98]. They are after all just a fancy version of λ-calculus. That is, every Haskell statement is in fact a λ-term, enriched with some extra objects like `Bool` and `Char`. For example:

```
myfunction        :: a -> b -> a
myfunction a b     =  a
```

This Haskell statement corresponds to the λ-term $\lambda a.\lambda b.a$.

Now in lazy languages like Haskell, one can also do computations with infinite objects like the list of all natural numbers or the list of all strings that can be generated using a DCG[4] like:

$$\mathrm{S} \rightarrow \mathrm{aSb} \mid \epsilon$$

[3]For example: β-reduction of the λ-term $(\lambda x.xxxxxxxxxxy)f$ creates 9 additional copies of f

[4]BNF in computer science

These examples justify our introduction of coinductive proof nets, which are able to represent infinite objects (λ-terms) and computations over infinite objects very elegantly.

3 Preliminaries

In this section we will give an overview of the theory, lemma's and definitions used throughout this thesis. For a complete introduction to rewriting one can refer to [Ter02].

3.1 Abstract Rewriting Systems

An *abstract rewriting system (ARS)* is a quadruple $\langle A, \Phi, \mathsf{src}, \mathsf{tgt} \rangle$ with A a set of objects, Φ a set of steps and $\mathsf{src}, \mathsf{tgt} : \Phi \to A$ the source and target functions, respectively.

3.2 Normalization

We will now define the notion of normalization and a few other notions closely related to normalization (from [Ter02]). Throughout this text and also in the other sections of this thesis, we will use $\twoheadrightarrow$ for the transitive, reflexive closure of $\to$.

Normal Form $a \in A$ is in *normal form* if there exists no b such that $a \to b$.

Weakly Normalizing $a \in A$ is *weakly normalizing* if $a \twoheadrightarrow b$ for some normal form $b \in A$. The rewrite relation $\to$ is *weakly normalizing* (WN) if every $a \in A$ is *weakly normalizing*.

Strongly Normalizing $a \in A$ is *strongly normalizing* if every reduction sequence starting from a is finite. The rewrite relation $\to$ is *strongly normalizing* (SN) if every $a \in A$ is *strongly normalizing*.

Terminating See *Strongly Normalizing*.

Furthermore, we sometimes want to infer from SN or WN for one ARS, SN or WN for another ARS. This can be done by two lemma's called the *Projection Lemma* and the *Lifting Lemma*.

Lemma 3.1 (Projection) *If you have an ARS $\mathcal{M} = \langle M, \to_M, src_M, tgt_M \rangle$, another ARS $\mathcal{N}, \langle N, \to_N, src_M, tgt_M \rangle$ for which $\to_N$ is SN, a mapping $\mathcal{MAP}$ which maps every $m \in M$ to a certain $n \in N$ and the following scheme:*

$$\begin{tikzcd}
m \arrow[r] \arrow[d, maps to, "\mathcal{MAP}"] & m' \arrow[d, maps to, dashed, "\mathcal{MAP}"] \\
n \arrow[r, dashed] & n'
\end{tikzcd}$$

The first ARS ($\mathcal{M}$), can be mapped onto the second ($\mathcal{N}$).

Corollary 3.1 *As a corollary of Lemma 3.1, SN for $\to_M$ may be concluded, because if there would be an infinite sequence of $\to_M$-steps, there would also be an infinite sequence of $\to_N$-steps, which contract.icts our assumption that $\to_N$ is SN.*

Lemma 3.2 (Lifting) *If you have an ARS $\mathcal{M} = \langle M, \rightarrow_M, src_M, tgt_M \rangle$, another ARS $\mathcal{N}, \langle N, \rightarrow_N, src_N, tgt_N \rangle$ for which $\rightarrow_N$ in WN, a mapping $\mathcal{MAP}$ which maps every $n \in N$ to a certain $m \in M$ and the following scheme:*

$$\begin{tikzcd}
m \arrow[r, dashed] \arrow[d, mapsto, "\mathcal{MAP}"] & m' \arrow[d, mapsto, dashed, "\mathcal{MAP}"] \\
n \arrow[r] & n'
\end{tikzcd}$$

The second ARS ($\mathcal{N}$), can be lifted to the second ($\mathcal{M}$).

Corollary 3.2 *As a corollary of Lemma 3.2, WN for $\rightarrow_M$ may be concluded, but only if it can be proven that if there is a step from a certain $m \in M$ and m can be mapped to a certain $n \in N$, there is a step from n as well.*

Now if there is a sequence of $\rightarrow_N$-steps leading to normal form (which there is because of WN), there is also a sequence of $\rightarrow_M$-steps leading to normal form.

3.3 Confluence

We will now define the notion of confluence and a few other notions closely related to confluence (from [Ter02]):

Diamond Property $a \in A$ has the *diamond property* if $\forall b, c \in A((c \leftarrow a \rightarrow b) \Rightarrow \exists d \in A(c \rightarrow d \leftarrow b))$. The rewrite relation $\rightarrow$ has the *diamond property* (DP) if every $a \in A$ has the *diamond property*.

Local Confluence $a \in A$ is *locally confluent* if $\forall b, c \in A((c \leftarrow a \rightarrow b) \Rightarrow \exists d \in A(c \twoheadrightarrow d \twoheadleftarrow b))$. The rewrite relation $\rightarrow$ is *locally confluent* if every $a \in A$ is *locally confluent*.

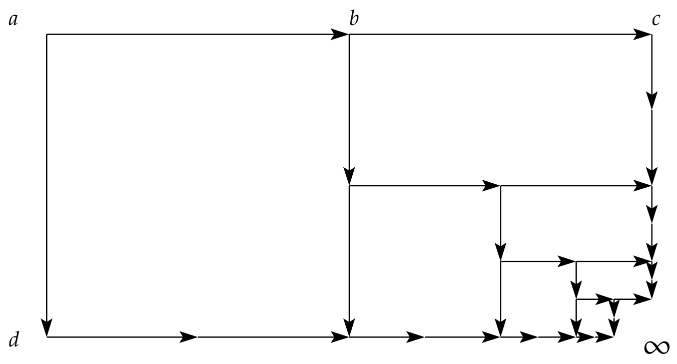
Weak Church-Rosser See *Local Confluence*. Weak Church-Rosser is abbreviated as WCR.

Confluence $a \in A$ is *confluent* if $\forall b, c \in A((c \twoheadleftarrow a \twoheadrightarrow b) \Rightarrow \exists d \in A(c \twoheadrightarrow d \twoheadleftarrow b))$. The rewrite relation $\rightarrow$ is *confluent* if every $a \in A$ is *confluent*.

Church-Rosser See *Confluence*. Church-Rosser is abbreviated as CR.

3.3.1 Newman's Lemma

A counter example against the proposition that local confluence implies confluence is shown in the following figure:



We see that an infinite reduction may be possible in which case d and c will never reduce to the same reduct. However, if we can somehow prove that such an infinite reduction does not exist (strong normalization) we also know that the steps in the figure above cannot decrease infinitely, which would give us confluence! This proposition is called *Newman's Lemma* and states that if an ARS is both SN and WCR, it is also CR.

Lemma 3.3 (Newman) *SN and WCR $\Rightarrow$ CR*

For a proof of Newman's Lemma one can refer to [Ter02].

3.4 Linear Graph

Since proof nets are a special kind of linear graphs, we will first give a definition of a linear graph as can be found in [Oos01].

Definition 3.1 (Linear Graph) *A linear graph signature is a set of symbols σ such that every symbol has a source and a target arity. Both these arities are natural numbers. A linear graph over σ is a pair $G = \langle V, E\rangle$ such that:*

- *V is a set of nodes such that with every node $v \in V$ a symbol in σ is associated. This symbol is denoted by $G(v)$. The arity of a node is the arity of its symbol. If the node v has arity n, m then we say that v has n source and m target ports.*
- *E is a set of edges $((v_1, i), (v_2, j))$, such that i an j are source and target ports of respectively $v_1 \in V$ and $v_2 \in V$*

The arity of a linear graph is the pair $\langle n, m\rangle$, such that n is the number of unconnected source ports and m is the number of unconnected target ports. A linear graph is closed *if the arity is $\langle 0, 0\rangle$.*

3.5 Graph Rewriting

Since this thesis is concerned not with term rewriting systems, but with graph rewriting systems (GRS's), we will give a (very) short introduction to graph rewriting. See [Plu98] for more information about (term) graph rewriting systems.

Definition 3.2 (Graph Rewriting Rule) *A graph rewriting rule $L \to R$ consists of a graph* L, *a graph* R *and a mapping from the vertices of* L *to the vertices of* R.

Now in applying a rewrite rule R to a graph G, a subgraph matching the left-hand side of R will be replaced by the left-hand side of R resulting in a graph G'.

Definition 3.3 (Graph Rewriting System (GRS)) *A graph rewriting system is a finite set of graph rewriting rules.*

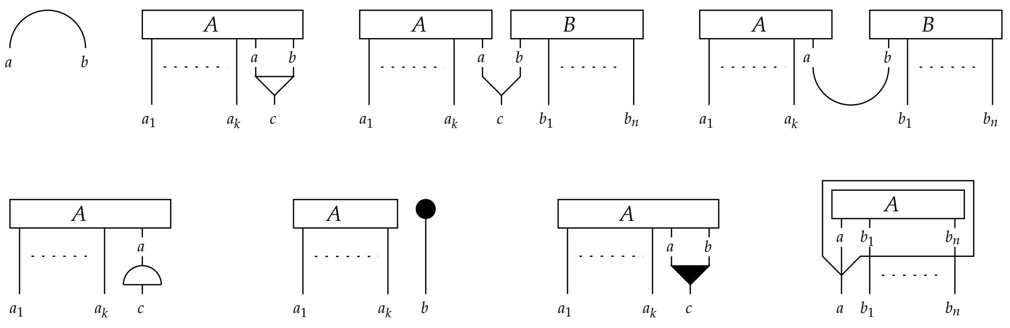
4 Proof Nets

4.1 Introduction

In this first section, we will give a formal definition of proof nets and proof net reduction. Furthermore, we will prove CR for type-free proof nets and give a counterexample against SN of proof net reduction.

4.2 Definition PN

Definition 4.1 (Inductive Proof Net) *If A and B are inductive proof nets, then so are:*



From left to right, the connectives are called axiom, par, tensor, cut *and on the second row* dereliction, weakening, fan *(which is also called* contraction*) and* box *(which is also called* promotion*).*

The first row is called the *multiplicative* fragment and the second the *exponential* fragment.

Inductive proof nets are only a means to define the objects we are truly interested in: proof nets. Proof nets are the graphs associated with inductive proof nets. Typically, one proof net can be associated with several inductive proof nets.

Definition 4.2 (Proof Net) *The proof net associated with an inductive proof net P is the graph $\mathbb{G}(P)$. The signature of this graph consists of the symbols $\blacktriangledown : 2,1$; ◖ $: 1,1$; $\bullet : 0,1$; $\triangledown : 2,1$; $\vee : 2,1$; $\cap : 0,2$; $\cup : 2,0$ and $\square(Q) : 0,n$, where n is the number of ports of Q.*

fan: *If the proof net N is associated with the inductive proof net A, then the proof net associated with its par-disjunction is N to which a $\blacktriangledown$-node and two edges, both going from (a different) a port of N to $\blacktriangledown$, have been added.*

box: *If the proof net N is associated with the inductive proof net A, then the proof net associated with its box is a $\square(N)$-node with the ports of N.*

dereliction: *If the proof net N is associated with the inductive proof net A, then the proof net associated with its dereliction is N to which a ◖ -node and one edge, going from a port of N to ◖ , has been added.*

weakening: *If the proof net N is associated with the inductive proof net A, then the proof net associated with its weakening is N to which a •-node has been added.*

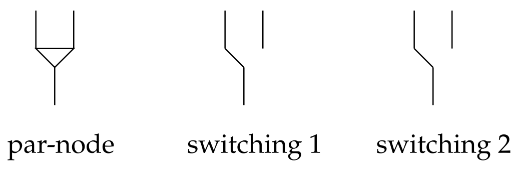
par: *If the proof net N is associated with the inductive proof net A, then the proof net associated with its par-disjunction is N to which a $\triangledown$-node and two edges, both going from (a different) a port of N to $\triangledown$, have been added.*

tensor: *If the proof nets N and M are associated with the inductive proof nets A and B, then the proof net associated with their tensor-union is the (disjunct) union of N and M to which a $\vee$-node and two edges, one going from a port of N to $\vee$ and one going from a port of M to $\vee$, have been added.*

axiom: *With an axiom we associate the node $\cap$*

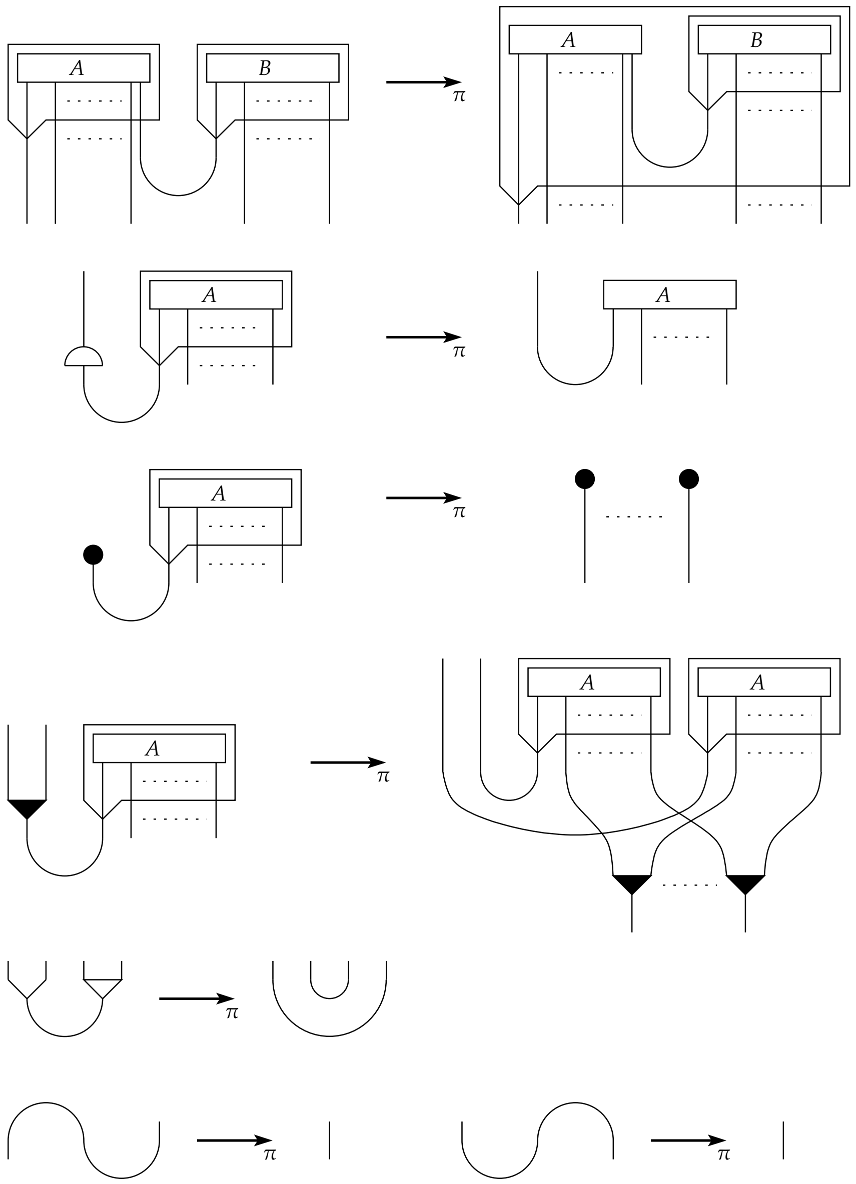
cut: *If the proof nets N and M are associated with the inductive proof nets A and B, then the proof net associated with their cut is the (disjunct) union of N and M to which a $\cup$-node and two edges, one going from a port of N to $\cup$ and one going from a port of M to $\cup$, have been added.*

Apart from the inductive definition we have just seen, proof nets are also frequently defined as geometrical objects. In this case, the so-called *switching criterion* serves as a correctness criterion. That is, if every switching of a proof net *P* is a connected, acyclic graph (a tree) *P* is well-formed. The word *switching* comes from the idea to use par-nodes as if they were switches:



Note that this correctness criterion can only be used for the multiplicative fragment of proof nets. It is obvious that rules like weakening and addition interfere with the restriction that every switching has to be connected. See [DR89] for more information about correctness criteria. Throughout the rest of this thesis we will only use the inductive definition.

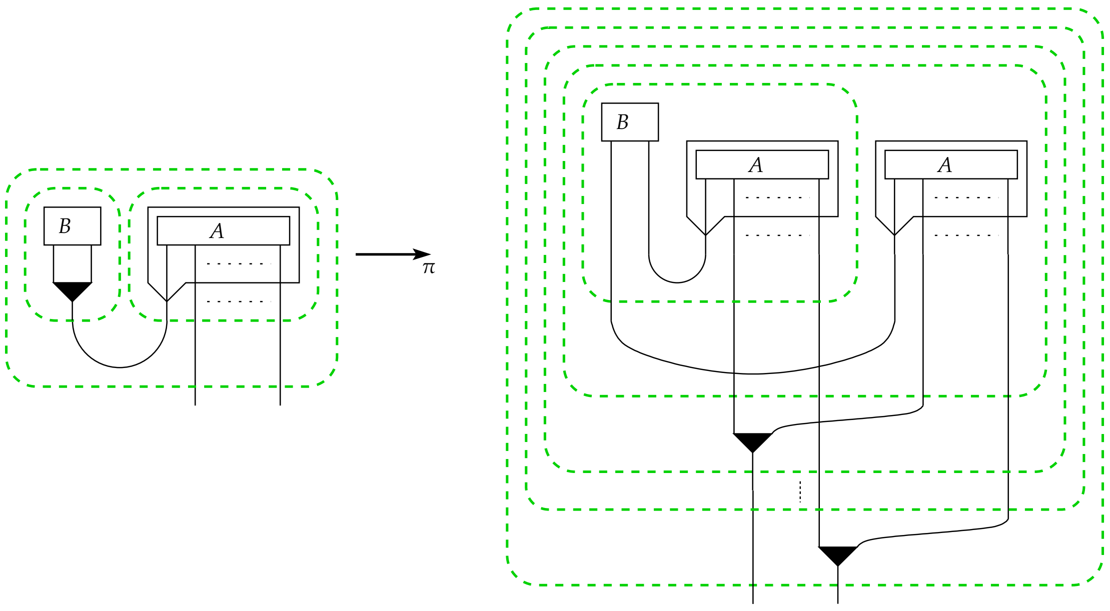
Definition 4.3 (Proof Net Reduction $\longrightarrow_\pi$)



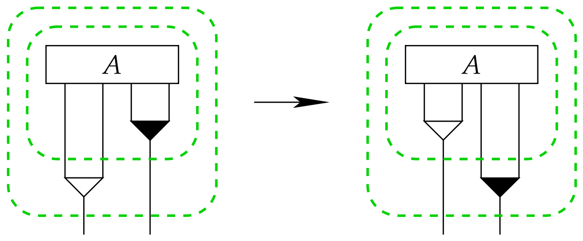
where A_{rhs} and B_{rhs} are descendents of respectively A_{lhs} and B_{lhs}. The rules are called (in the order they are presented above): box–box, box–dereliction, box–weakening, box–fan, tensor–par, axiom–cut *and* cut–axiom.

Of course we would like the result of applying a $\longrightarrow_{\pi}$-rule to be a proof net again, therefore we will prove the subject reduction property (SR) for $\longrightarrow_{\pi}$.

Lemma 4.1 (SR for $\longrightarrow_{\pi}$) *We need show for every left-hand-side of a $\longrightarrow_{\pi}$-rule, that if it is a well-formed proof net, its right-hand-side is too. Since this is rather obvious,* *we will only show this for the box–fan rule:*



Green boxes indicate well-formed components of the proof net. There is only one problem we might encounter in constructing a scheme such as the one shown above: we might have associated the wrong inductive structure with the proof net shown on the left-hand side of the $\longrightarrow_\pi$-rule, for we know that more than one inductive proof net can be associated with a proof net. However, if a fan-node is indeed connected to a box, one can always re-arrange the green boxes to fit the left-hand side of the $\longrightarrow_\pi$-rule shown above, using rules like the following:



The same goes for the rest of the $\longrightarrow_\pi$-rules.

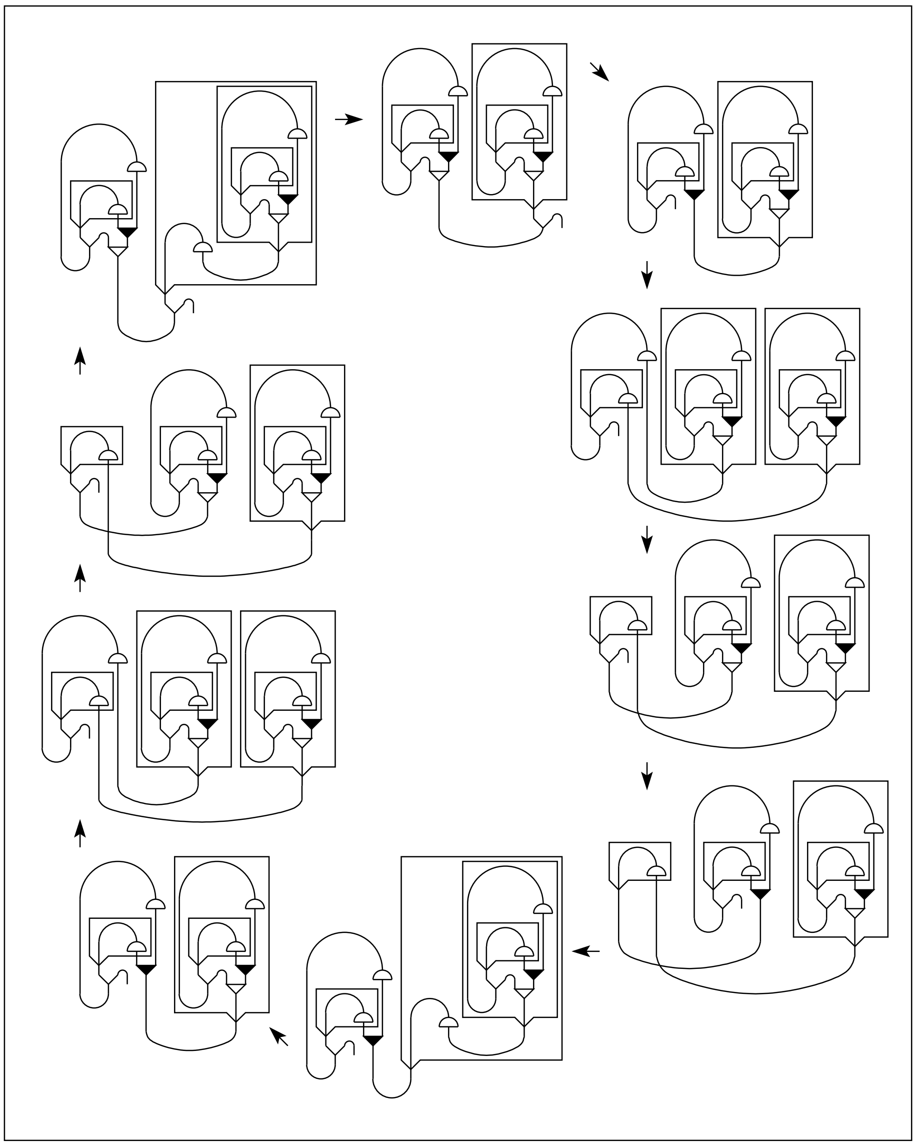
□

4.3 Confluence for Proof Nets

4.3.1 Introduction

For every ARS, there are two important properties one would like the system to possess: *strong normalization* and *confluence*. So naturally, these will be the properties we will try to establish for our proof net reduction[5]. However, not every ARS has both or even one these properties and unfortunately, proof net reduction is one those systems that lack stong normalization. Which is by the way a result we could have expected, considering the fact that the λ-calculus, an ARS without strong normalization, can be simulated using proof nets (see [Oos01] for the correspondence between proof nets and λ-terms). We can therefore use a λ-term with an infinite rewriting sequence $((\lambda x.xx)(\lambda x.xx))$ as a counterexample against strong normalization for proof net reduction:

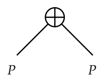
[5]Note that for the multiplicative fragment, it is extremely easy to prove CR and SN: the ARS for the multiplicative fragment is weakly orthogonal and the number of nodes of the proof net reduces with every rewrite step.



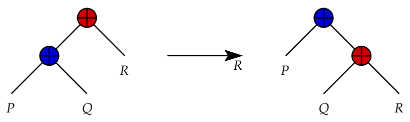
Confluence on the other hand looks promising. Even though we do not have the diamond property a priori, it can be proven using so called *developments* [Mel02] [Ter02]. To show how these developments work and what our proof strategy will be, we will first present a proof of confluence for ⊕-trees after which we will give a similar proof of confluence for proof net reduction.

4.3.2 Confluence for ⊕-Trees

This proof deals with the so-called '⊕-trees'. These trees can be seen as graphical representations of the ordinary +-operator. Our goal will be to prove associativity of ⊕-trees to be confluent. First we will give a formal definition of ⊕-trees and the associativity rewrite rule.

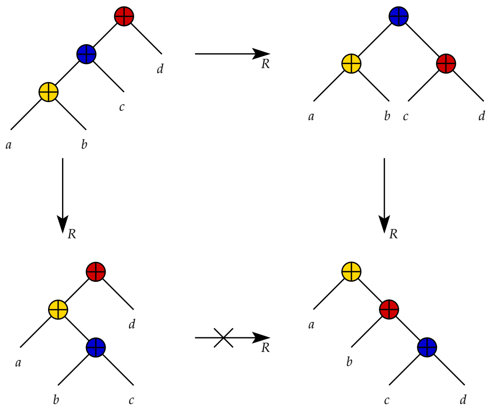
Definition 4.4 (⊕-Tree) *P* ::= *Int* | 

Definition 4.5 (Rewrite Relation $\rightarrow_R$)



Here, the red and blue ⊕'s on the right-hand-side are descendants of (respectively) the red and blue ⊕'s on the left-hand-side. Furthermore P_{rhs}, Q_{rhs} and R_{rhs} are descendants of respectively P_{lhs}, Q_{lhs} and R_{lhs}.

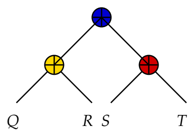
First we always try if a rewrite relation has the diamond property, because it is so easy to derive CR from DP. Unfortunately, our rewrite relation does not have this diamond property, as can be illustrated by the following figure:



Since we do not have the diamond property a priori, we will need to find a notion of parallel step, which contracts several redexes at once. This notion will be the *development of a cluster*. Intuitively, one marks an arbitrary number of redexes, the union of which will be called the *cluster*, and then contracts all of the redexes in this cluster. So a development can be seen as the concatenation of a number of steps modulo the order in which they are applied. However, there is still some work that needs to be done if the redexes in the cluster overlap. For this purpose, a definition of the *residue of a cluster* will be given after which the development of a cluster *C* can be defined as the concatenation of the contraction of a redex in *C* and the development of the residue of *C*.

Definition 4.6 (Cluster) *A cluster is a connected component of ⊕'s. In figures a set of ⊕'s encircled with a green line will denote a cluster.*

By 'connected' we mean the transitive, reflexive, symmetric closure of 'directly below':

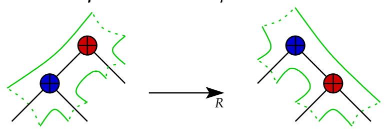


Both the yellow and the red $\oplus$ are directly below the blue one. In the pictures we will use, the $\oplus$'s inside an area encircled by a green line denote a cluster. A line consisting of green dots denotes a possible extension of the cluster.

Definition 4.7 (Residue) *The residue of a cluster C after a rewrite-step ψ:*

1. *No overlap:* $C - \psi = \mathsf{desc}(C)$
   where $\mathsf{desc}(C)$ *is the descendant of* C*.*

2. (a) *subsumption :* $C - \psi$ *:*

   (b) *partial overlap:*
   if C is a cluster and ψ is a step then $(C \cup \psi)$ *is a cluster again:*
   $C - \psi = (C \cup \psi) - \psi$ *(like* (2a)*)*

Definition 4.8 (Development) *A development of a cluster C is defined as a step $\psi \in C$ followed by the development of $C - \psi$*

Notation: *$a \overset{C}{\multimap} b$ denotes the development of the cluster C in a, b being the result of a after the development.*

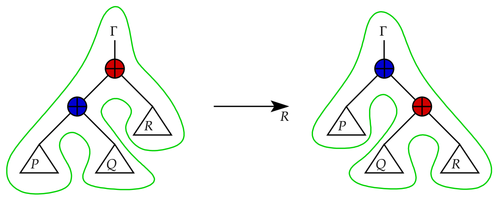
To establish confluence for developments we will first prove strong normalization and weak Church-Rosser, after which we will combine the two, to form a confluence proof by Newman's Lemma.

Lemma 4.2 (Developments are Strongly Normalizing) *In order to prove that developments are strongly normalizing, we will first define a notion of weight for a cluster, after which we will proceed by showing that this weight decreases with every rewrite step.*

Definition 4.9 (Weight of a Cluster) *The weight $\mathcal{G}$ of a cluster C is defined as the sum of the weight of every $\oplus$-symbol in the cluster.*

Definition 4.10 (Weight of a $\oplus$-symbol) *The weight g of a $\oplus$-symbol in a cluster C is the number of left-branches in the cluster C it is embedded in.*

Rule 1



$$\boxed{\begin{array}{lcll} \mathcal{G}(LHS) & = & \mathcal{G}(\Gamma) & + \\ & & \mathcal{G}(P) + (\boxed{\mathbf{2}} + n) * Pl(P) & + \\ & & \mathcal{G}(Q) + (1+n) * Pl(Q) & + \\ & & \mathcal{G}(R) + n * Pl(R) & + \\ & & \boxed{\mathbf{1}} + n + n & \end{array}} > \boxed{\begin{array}{lcll} \mathcal{G}(RHS) & = & \mathcal{G}(\Gamma) & + \\ & & \mathcal{G}(P) + (1+n) * Pl(P) & + \\ & & \mathcal{G}(Q) + (1+n) * Pl(Q) & + \\ & & \mathcal{G}(R) + n * Pl(R) & + \\ & & n + n & \end{array}}$$

Where n = the number of left branches in Γ the top $\oplus$ is embedded in, and $Pl(P)$ is the number of $\oplus$'s in P.

□

Note that $\rightarrow_R$ itself is also SN, for we could extend our notion of 'weight of a cluster' to 'weight of a $\oplus$-tree' and then give a proof like the one for Lemma 4.2 (Developments are Strongly Normalizing).

Lemma 4.3 (Developments have the Weak Church-Rosser property)

1. *No overlap:*

$$\begin{tikzcd} & {} \arrow[ld, "\psi"'] \arrow[rd, "\phi"] & \\ {} \arrow[rd, "\phi"'] & & {} \arrow[ld, "\psi"] \\ & {} & \end{tikzcd}$$

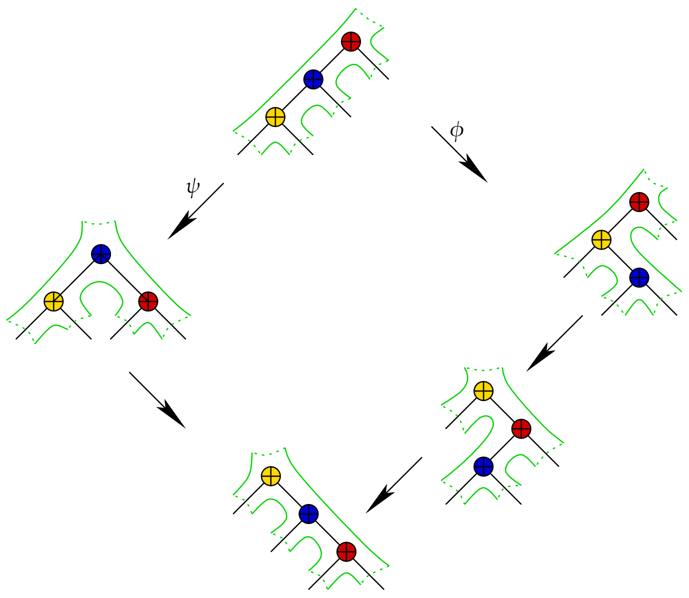
if there is no overlap, (the descendant of) the redex set ψ still exists after contracting the redex set ϕ and the other way around.

2. (a) $\psi = \phi$:

$$\begin{tikzcd} & {} \arrow[ld, "\psi"'] \arrow[rd, "\phi"] & \\ {} \arrow[rd, two heads, "0"'] & & {} \arrow[ld, two heads, "0"] \\ & {} & \end{tikzcd}$$

if the redex sets are the same, we don't have to do any steps to arrive at a similar reduct.

   (b) *Critical pair:*



□

Lemma 4.4 (Developments have the Church-Rosser property)
By Newman's lemma: SN and WCR ⇒ CR. We proved the first condition (Strongly Normalizing) in Lemma 4.2 and the second in Lemma 4.3, so we may conclude that developments have in fact the Church-Rosser property.

□

A similar proof can be found in [Mel02]. See also [Mac71]. Now that we have finally shown that developments have the Church-Rosser property (or are confluent), it remains to be proven that these developments are indeed an apt notion of parallel $\rightarrow_R$-step, for the goal we set ourselves was not to prove that developments are confluent, but to prove confluence for associativity of ⊕!

Lemma 4.5 ($\rightarrow_R \subseteq \multimap\!\!\!\rightarrow \subseteq \rightarrow_R^*$) *Since every single $\rightarrow_R$-step can be defined as the development of a cluster containing exactly one redex, the first subsumption is a fact. The second subsumption is true by the definition of the development of a cluster, for $\multimap\!\!\!\rightarrow$ is in fact nothing more than the concatenation of a finite number of $\rightarrow_R$-steps. Finite, because $\rightarrow_R$ is SN.*

□

Proof 4.1 (Associativity of ⊕ has the Church-Rosser property)
We have already proven that developments have the Church-Rosser property in Lemma 4.4 and since $\rightarrow_R \subseteq \multimap\!\!\!\rightarrow \subseteq \rightarrow_R^*$ by Lemma 4.5, we may conclude that the transitive reflexive closure of $\multimap\!\!\!\rightarrow$ is equal to the transitive reflexive closure of $\rightarrow_R$: $\multimap\!\!\!\rightarrow^* = \rightarrow_R$. We may now also derive Church-Rosser for $\rightarrow_R$ from the proof of Church-Rosser for $\multimap\!\!\!\rightarrow$.

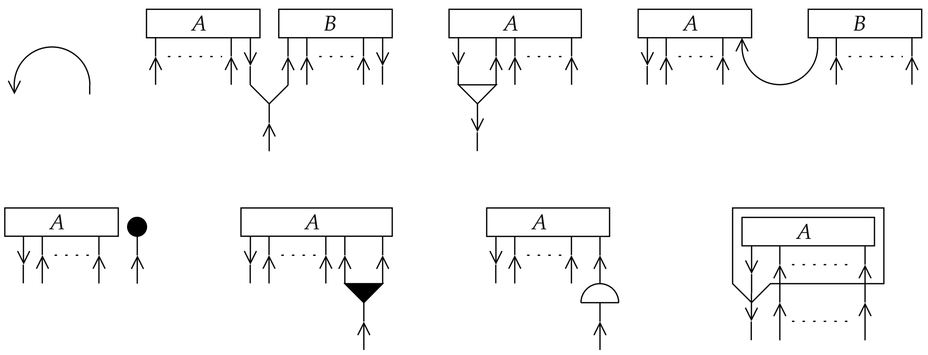
□

It is a fact that there are more ways to prove CR for associativity of $\oplus$ and that we did not choose the easiest one. We could have left out the entire concept of *developments* and instead prove that the associativity rule itself is strongly normalizing! This would have saved us a lot of time, but then again, we only included this proof to demonstrate how we are going to prove CR for proof net reduction in the next section, which justifies our somewhat inefficient proof of CR for associativity of $\oplus$.

4.3.3 Confluence for DPN

For reasons that will become clear later on, we will prove confluence for *directed* proof nets instead of for ordinary proof nets.

Definition 4.11 (Directed Proof Net) *A directed proofnet is an inductive proofnet of which exactly one port is labelled output port. All other ports are labelled input ports:*



Ports with an arrow going into the proof nets A or B are input ports, and the ones with an arrow going out of the proof nets A or B are the output ports.

In the previous proof we have already encountered the strategy we will be using for the proof of confluence for proof net reduction. In this proof too, clusters and developments of clusters will be used to define an apt notion of parallel reduction step. However, an important difference between $\oplus$-clusters and proof net clusters is that while the former have to be connected, the latter do not have such a restriction. Furthermore, whereas in the case of $\oplus$-trees developments took care of the entire rewrite relation $\rightarrow_R$, this is not possible for proof net reduction $\longrightarrow_\pi$, because we would lack the essential property of SN (see the introduction of Section 4.3.1). What we will do instead, is partition the proof net reduction relation $\longrightarrow_\pi$ into the sub-relations $\longrightarrow_\omega$ and $\longrightarrow_{(\psi,\sim)}$. The former will be a strongly normalizing subset containing box–dereliction, box–fan, box–weakening and box–box in which we will define developments. The latter will contain the remaining rules: tensor–par, cut–axiom and axiom–cut. From now on the tensor–par rule will be called the ψ-rule and the cut–axiom and axiom–cut rules will be called the $\sim$-rules.

Now to prove that $\longrightarrow_{\pi}$ is confluent, we will first prove that developments defined on $\longrightarrow_{\varpi}$ and a certain notion of parallel $(\psi, \sim)$-step are both confluent independently of each other, after which we will combine these two proofs into a proof of confluence for $\longrightarrow_{\pi}$.

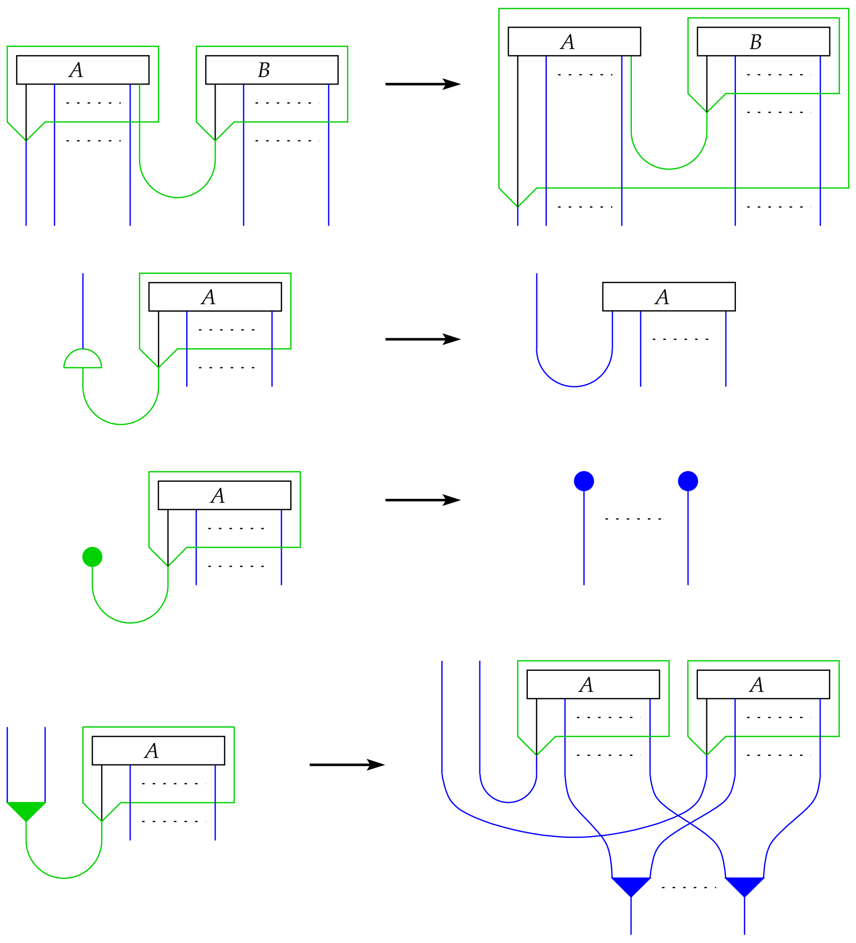
But according to the proof idea just defined, we first need a proof that developments defined on $\longrightarrow_{\varpi}$ have the CR property. Such a proof will be given below and will closely resemble the proof of CR for developments defined on $\oplus$-trees (Lemma 4.4).

Definition 4.12 (Inductive Cluster) *An inductive cluster is a subgraph, containing only box, fan, dereliction, weakening and cut nodes, of the graph of a proofnet. In figures, if a node is green, it is part of a cluster. If it is blue, whether it is part of a cluster depends on whether or not the node is the descendant of a green node.*

Definition 4.13 (Residue) *The residue of a cluster C after a rewrite-step ψ:*

1. *No overlap:* $C - \psi = \mathsf{desc}(C)$
   where $\mathsf{desc}(C)$ *is the descendant of C.*

2. (a) *subsumption :* $C - \psi$:



(b) *partial overlap:*
if C is a cluster and ψ is a step then $(C \cup \psi)$ *is a cluster again:*
$C - \psi = (C \cup \psi) - \psi$ *(like* (2a)*)*

Definition 4.14 (Development) *A development of a cluster C is defined as a step $\psi \in C$ followed by the development of $C - \psi$*

Notation: *$a \overset{C}{-\!\!\circ\!\!\!\rightarrow} b$ denotes the development of the cluster C in a, b being the result of a after the development.*

As we already saw in Proof 4.1, we will eventually have to prove SN for our inductive developments. Proving SN always involves defining a measure which decreases with every rewrite step. Our measure will be an ordered triple $\langle \mathsf{fd}, \mathsf{bd}, n \rangle$, which we will explained in detail later on. We chose to use a triple, because it allows one to reason about several measures instead of one. Now to prove that $\langle \mathsf{fd}, \mathsf{bd}, n \rangle$ decreases, we have to prove one of the following for every rewrite step:

- $\mathsf{fd}_{lhs} > \mathsf{fd}_{rhs}$
- $\mathsf{fd}_{lhs} = \mathsf{fd}_{rhs}$ and $\mathsf{bd}_{lhs} > \mathsf{bd}_{rhs}$
- $\mathsf{fd}_{lhs} = \mathsf{fd}_{rhs}$ and $\mathsf{bd}_{lhs} = \mathsf{bd}_{rhs}$ and $n_{lhs} > n_{rhs}$

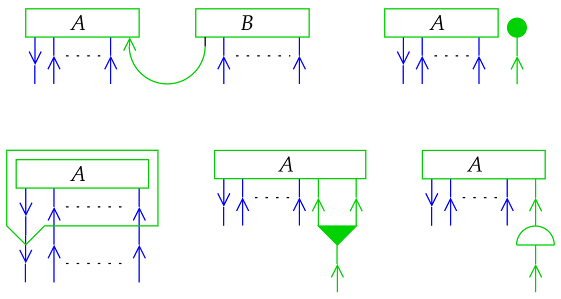
where *lhs* stands for 'left-hand-side' and *rhs* stands for 'right-hand-side'. The idea behind the first measure fd, which stands for *fan depth*, is that in applying box–fan, the fans are being pushed through the box. It remains to be proven that this can only be done finitely many times.

This is where the arrows of a directed proof net come in handy! Our aim will be to show that the arrows define some sort of ordering (*behind*) of the nodes which decides whether or not a certain node can interact with another one. But first we will have to show that inductive clusters do not contain or produce cycles, or such an ordering would be useless.

The idea behind the first lemma (Lemma 4.6) is that directed inductive clusters do not contain cycles, because they have no *ceiling*. By *ceiling*, we mean a link in the proofnet where an upward arrow turns to go downwards. Now as long as a directed inductive cluster does not contain such links, a cycle can never be formed, because in trying to form one, you always need at least one 'ceiling link'. An axiom link is the only link that would qualify as a ceiling in our definition of proof nets, and since directed inductive cluster do not contain axiom links, they don't have a ceiling either. Hence, they do not contain cycles. A formal proof of this idea can be found below:

Lemma 4.6 (A directed inductive cluster does not contain cycles)
Obviously, if there is no connected subset of the cluster C, which contains a cycle, then C itself does not contain a cycle. It remains to be proven that there can be no such connected subset. We will prove this by induction on the construction of a connected subset:

- *The box, fan, dereliction, weakening and cut nodes do not contain a cycle.*
- *If A and B are directed inductive clusters, which do not contain a cycle, then neither do the following inductive clusters:*

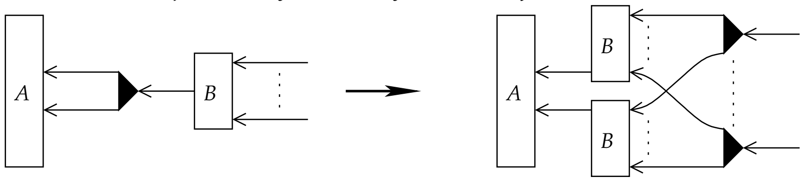


□

Lemma 4.7 (Inductive clusters without cycles have the Subject Reduction property) *We will show that an inductive cluster without cycles, still does not contain such a cycle after a reduction step.*

box–dereliction: *Trivial, the paths do not change.*

box–fan: *The fan is merely being 'pushed through' the box (as shown in the picture below) which explains why no new cycles can be formed.*



box–box: *Trivial, the paths do not change.*

box–weakening: *Trivial, the paths are cut off.*

□

Now using the arrows of a directed proof net and the fact that inductive clusters neither contain nor produce any cycles, we are ready to define an ordering on boxes:

Definition 4.15 (behind in PN) *A box* $\mathbf{B_1}$ *is behind a node* $\mathbf{N}$ *if:*

- *one can reach the node* $\mathbf{N}$ *by following the arrows going out of the output port of the box* $\mathbf{B_1}$ *and passing only fan-, dereliction-, cut- and weakening nodes.*
- *the box* $\mathbf{B_1}$ *is inside a box* $\mathbf{B_2}$ *which is behind the node* $\mathbf{N}$
- *the box* $\mathbf{B_1}$ *is behind a box* $\mathbf{B_2}$ *which is behind the node* $\mathbf{N}$ *(transitivity).*

Note that one can not go through tensor-, par- or axiom nodes, because they are not part of the cluster.

Now since the fans are being pushed further and further along a path (i.e. behind more boxes) each time we apply a box–fan rule, and since there are no infinite paths because of the lack of cycles, this process will eventually halt.

The idea behind the second measure of our triple $\langle \mathsf{fd}, \mathsf{bd}, n \rangle$, which stands for *box depth*, is that the box–box rule pushes boxes into one another and that this too, can only be done finitely many times. For this purpose, we define the *level of a node*:

Definition 4.16 (Level of Node)
The level of a node N is recursively defined as:

- $\mathsf{bl}(N) = 1$
  if N is not inside a(nother) box

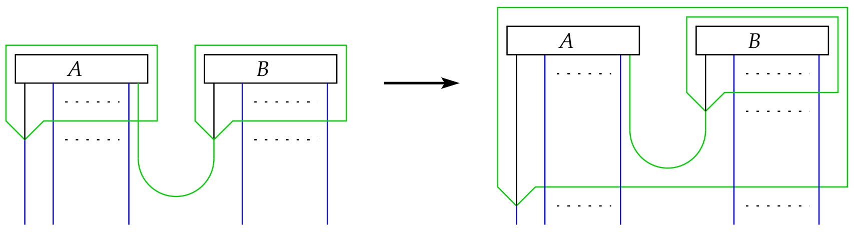
- $\mathsf{bl}(N) = \mathsf{bl}(B) + 1$
  if N is inside box B

Now for a proof net containing n boxes, this measure can never be more than $\sum_{i=1}^{n} i$. A decreasing measure can therefore be defined on a proof net as the maximum of bd minus the level of every box in the proof net. The third and final measure of our triple $\langle \mathsf{fd}, \mathsf{bd}, n \rangle$ is the number of boxes n in a proof net. The rewrite rules box–weakening and box–dereliction both remove a box-node, which decreases this n. We will now formalize this proof of SN for developments.

Lemma 4.8 (Developments are Strongly Normalizing) *In order to prove that developments are strongly normalizing, we will first define a notion of weight for an inductive cluster, after which we can proceed by showing that this weight decreases with every rewrite step.*

Definition 4.17 *The weight $\mathcal{G}$ of a cluster* $\mathbf{C}$ *is a triple $\langle \mathsf{fd}, \mathsf{bd}, n \rangle$, where* fd *is the multiset $\biguplus_{i=1}^{n} \mathsf{fn}(i)$, $\mathsf{fn}(x)$ is the number of fan-nodes box x is behind,* bd *is $\sum_{i=1}^{n} i - \sum_{i=1}^{n} \mathsf{lv}(i)$,* $\mathsf{lv}(x)$ *is the level of box x and n is the number of boxes in the cluster.*

Case 1



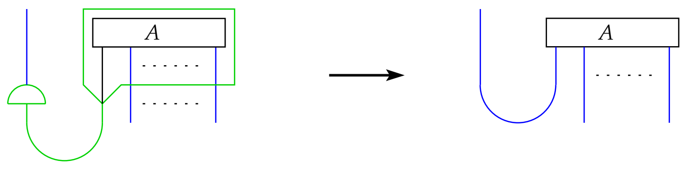
$$\boxed{\begin{array}{lcl} \mathsf{fd}_{lhs} & = & \biguplus_{i=1}^{m} \mathsf{fn}(i) \\ \mathsf{bd}_{lhs} & = & \sum_{i=1}^{m} i - \sum_{i=1}^{m} \mathsf{lv}(i) \end{array}}$$

$$\mathsf{fd}_{rhs} = \biguplus_{i=1}^{m} \mathsf{fn}(i) = \biguplus_{i=1}^{m} \mathsf{fn}(i) = \mathsf{fd}_{lhs} \qquad \Rightarrow \qquad \mathsf{fd}_{rhs} = \mathsf{fd}_{rlhs}$$

$$\mathsf{bd}_{rhs} = \sum_{i=1}^{m} i - ((\sum_{i=1}^{m} \mathsf{lv}(i)) + 1) < \sum_{i=1}^{m} i - \sum_{i=1}^{m} \mathsf{lv}(i) = \mathsf{bd}_{lhs} \qquad \Rightarrow \qquad \mathsf{bd}_{rhs} < \mathsf{bd}_{lhs}$$

$$\left.\begin{array}{lcl} \mathsf{fd}_{lhs} & = & \mathsf{fd}_{rhs} \\ \mathsf{bd}_{lhs} & > & \mathsf{bd}_{rhs} \end{array}\right\} \Rightarrow \langle \mathsf{fd}_{lhs}, \mathsf{bd}_{lhs}, n_{lhs} \rangle > \langle \mathsf{fd}_{rhs}, \mathsf{bd}_{rhs}, n_{rhs} \rangle$$

Case 2

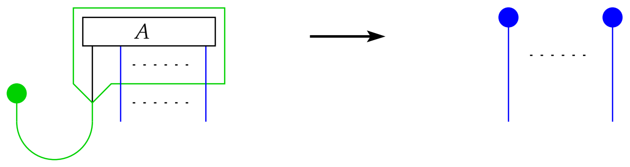


$$\boxed{\begin{array}{lcl} \mathsf{fd}_{lhs} & = & \biguplus_{i=1}^{m+1} \mathsf{fn}(i) \\ \mathsf{bd}_{lhs} & = & \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \mathsf{lv}(i) \\ n_{lhs} & = & m+1 \end{array}}$$

$$\begin{array}{lcl} \mathsf{fd}_{rhs} = \biguplus_{i=1}^{m} \mathsf{fn}(i) \leq \biguplus_{i=1}^{m+1} \mathsf{fn}(i) = \mathsf{fd}_{lhs} & \Rightarrow & \mathsf{fd}_{rhs} \leq \mathsf{fd}_{rlhs} \\ \mathsf{bd}_{rhs} = \sum_{i=1}^{m} i - \sum_{i=1}^{m} \mathsf{lv}(i) \leq \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \mathsf{lv}(i) = \mathsf{bd}_{lhs} & \Rightarrow & \mathsf{bd}_{rhs} \leq \mathsf{bd}_{lhs} \\ n_{rhs} = m < m+1 = n_{lhs} & \Rightarrow & n_{rhs} < n_{lhs} \end{array}$$

$$\left.\begin{array}{lcl} \mathsf{fd}_{lhs} & \geq & \mathsf{fd}_{rhs} \\ \mathsf{bd}_{lhs} & \geq & \mathsf{bd}_{rhs} \\ n_{lhs} & > & n_{rhs} \end{array}\right\} \Rightarrow \langle \mathsf{fd}_{lhs}, \mathsf{bd}_{lhs}, n_{lhs} \rangle > \langle \mathsf{fd}_{rhs}, \mathsf{bd}_{rhs}, n_{rhs} \rangle$$

Case 3

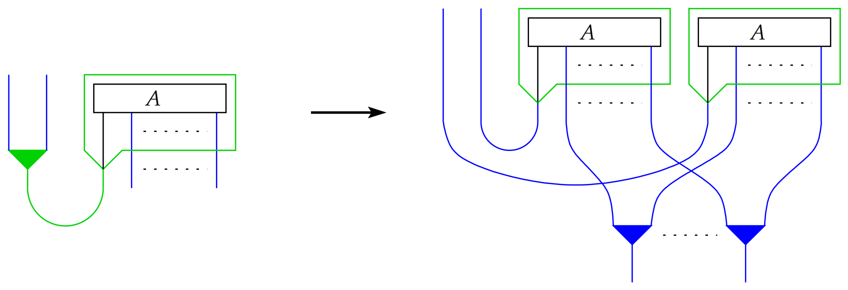


$$\boxed{\begin{array}{lcl} \mathsf{fd}_{lhs} & = & \biguplus_{i=1}^{m+1} \mathsf{fn}(i) \\ \mathsf{bd}_{lhs} & = & \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \mathsf{lv}(i) \\ n_{lhs} & = & m+1 \end{array}}$$

$$\begin{array}{lcl} \mathsf{fd}_{rhs} = \biguplus_{i=1}^{m} \mathsf{fn}(i) \leq \biguplus_{i=1}^{m+1} \mathsf{fn}(i) = \mathsf{fd}_{lhs} & \Rightarrow & \mathsf{fd}_{rhs} \leq \mathsf{fd}_{rlhs} \\ \mathsf{bd}_{rhs} = \sum_{i=1}^{m} i - \sum_{i=1}^{m} \mathsf{lv}(i) \leq \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \mathsf{lv}(i) = \mathsf{bd}_{lhs} & \Rightarrow & \mathsf{bd}_{rhs} \leq \mathsf{bd}_{lhs} \\ n_{rhs} = m < m+1 = n_{lhs} & \Rightarrow & n_{rhs} < n_{lhs} \end{array}$$

$$\left.\begin{array}{lcl} \mathsf{fd}_{lhs} & \geq & \mathsf{fd}_{rhs} \\ \mathsf{bd}_{lhs} & \geq & \mathsf{bd}_{rhs} \\ n_{lhs} & > & n_{rhs} \end{array}\right\} \Rightarrow \langle \mathsf{fd}_{lhs}, \mathsf{bd}_{lhs}, n_{lhs} \rangle > \langle \mathsf{fd}_{rhs}, \mathsf{bd}_{rhs}, n_{rhs} \rangle$$

Case 4



$$\boxed{\mathsf{fd}_{lhs} \quad = \quad \biguplus_{i=1}^{m_{in}} \mathsf{fn}(i) \uplus \biguplus_{i=1}^{m_{out}} \mathsf{fn}(i) \uplus \mathsf{fn}(b)}$$

$$\mathsf{fd}_{rhs} = \biguplus_{i=1}^{m_{in}}(\mathsf{fn}(i)-1) \uplus \biguplus_{i=1}^{m_{in}}(\mathsf{fn}(i)-1) \uplus \biguplus_{i=1}^{m_{out}} \mathsf{fn}(i) \uplus (\mathsf{fn}(b)-1) \uplus (\mathsf{fn}(b)-1) < \biguplus_{i=1}^{m_{in}} \mathsf{fn}(i) \uplus \biguplus_{i=1}^{m_{out}} \mathsf{fn}(i) \uplus \mathsf{fn}(b) \quad \Rightarrow \quad \mathsf{fd}_{rhs} < \mathsf{fd}_{rlhs}$$

$$\left.\mathsf{fd}_{lhs} \quad > \quad \mathsf{fd}_{rhs}\right\} \quad \Rightarrow \quad \langle \mathsf{fd}_{lhs}, \mathsf{bd}_{lhs}, n_{lhs}\rangle > \langle \mathsf{fd}_{rhs}, \mathsf{bd}_{rhs}, n_{rhs}\rangle$$

□

In Proof 4.1 we had to prove, besides strong normalization, that developments are WCR to be able to use Newman's Lemma. That is exactly what we have to do next for the purpose of establishing CR for developments.

Lemma 4.9 (Developments have the Weak Church-Rosser property)

1. *No overlap:*

$$\begin{tikzcd} & \cdot \arrow[dl, "\psi"'] \arrow[dr, "\phi"] & \\ \cdot \arrow[dr, "\phi"'] & & \cdot \arrow[dl, "\psi"] \\ & \cdot & \end{tikzcd}$$

   or worst case (if the redex of ψ is duplicated by ϕ):

$$\begin{tikzcd} & \cdot \arrow[dl, "\psi"'] \arrow[dr, "\phi"] & & \\ \cdot \arrow[dr, "\phi"'] & & \cdot \arrow[d, "\psi"] & \\ & \cdot & \cdot \arrow[l, "\psi"] & \end{tikzcd}$$

   If there is no overlap, (the descendant of) the redex set ψ still exists after contracting the redex set ϕ and the other way around. However, ψ may have been duplicated by ϕ, in which case we need to contract both descendants.

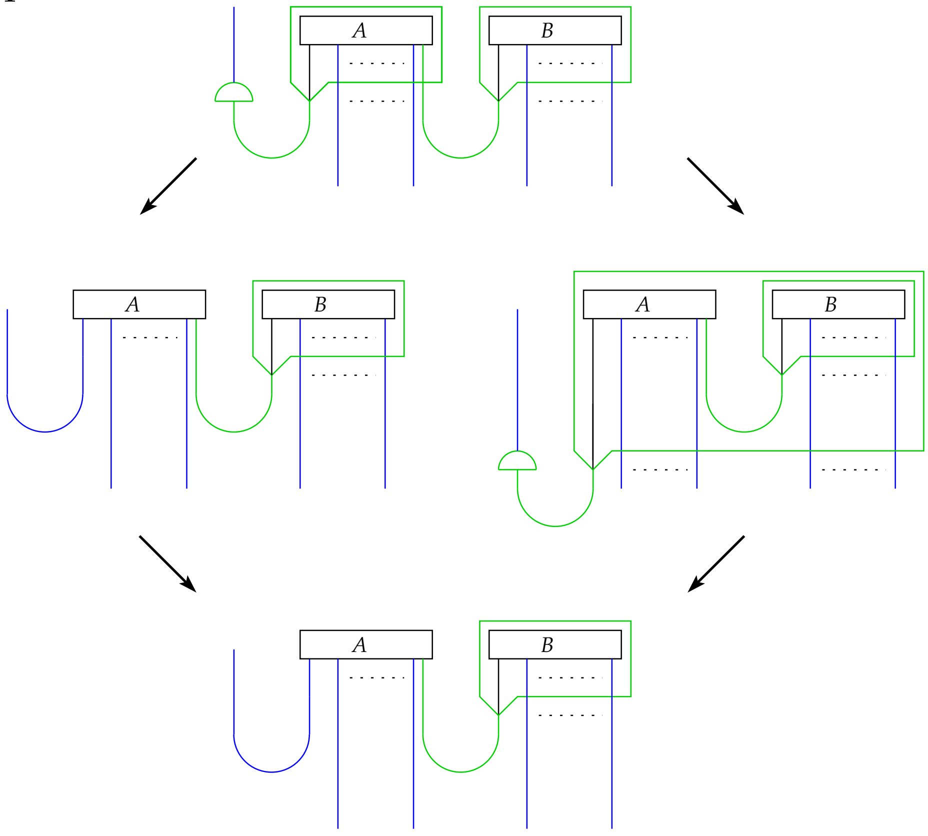
2. (a) $\psi = \phi$

$$\begin{tikzcd} & \cdot \arrow[dl, "\psi"'] \arrow[dr, "\phi"] & \\ \cdot \arrow[dr, "0"'] & & \cdot \arrow[dl, "0"] \\ & \cdot & \end{tikzcd}$$

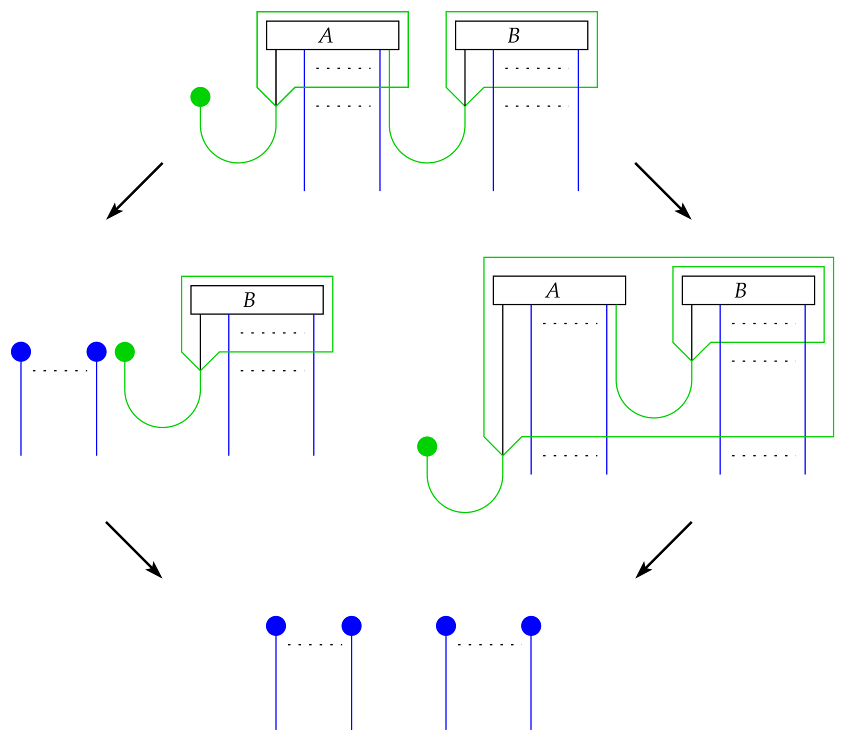
   if the redex sets are the same, we don't have to do any steps to arrive at a similar reduct.

(b) Critical pairs:

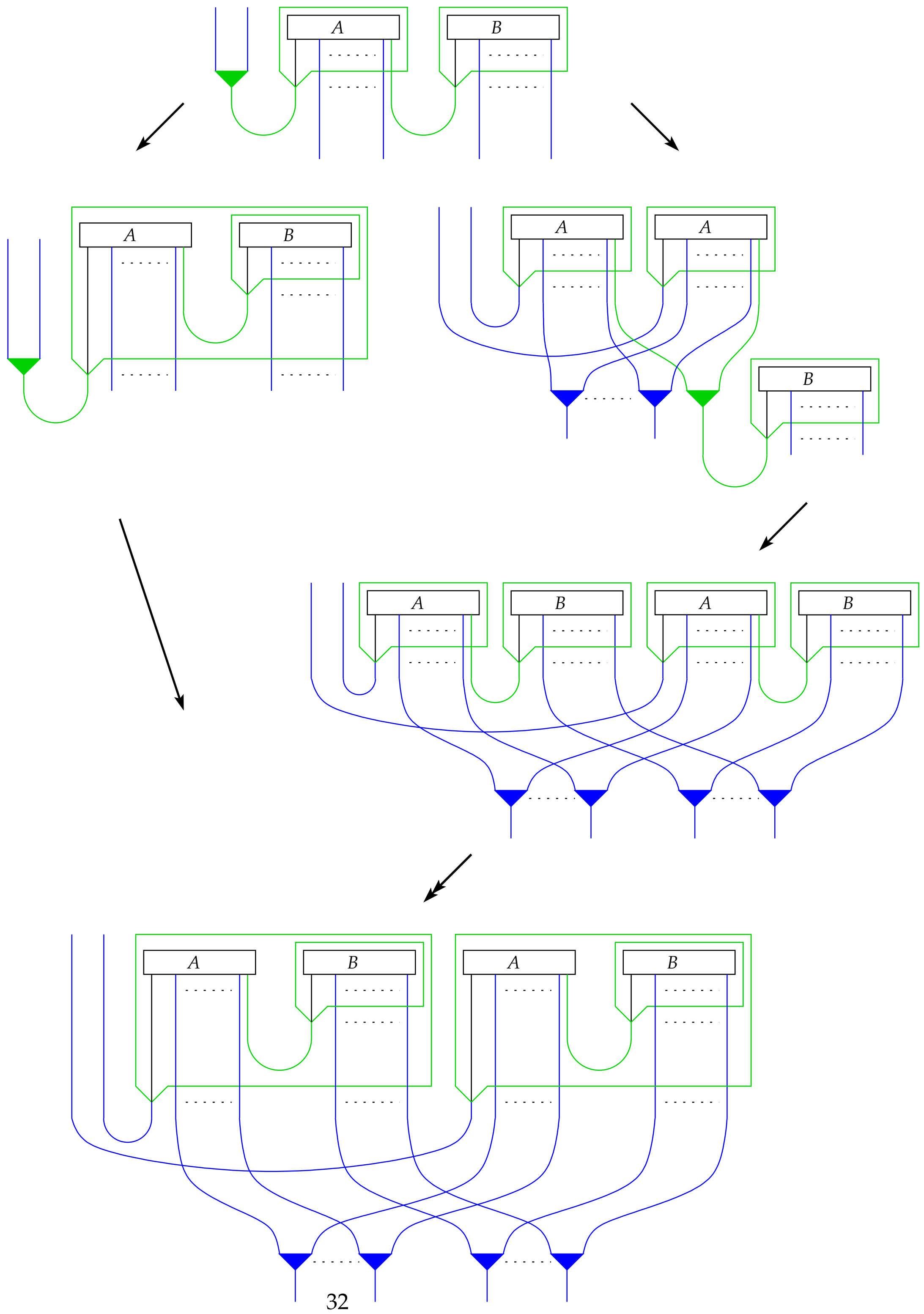
Case 1



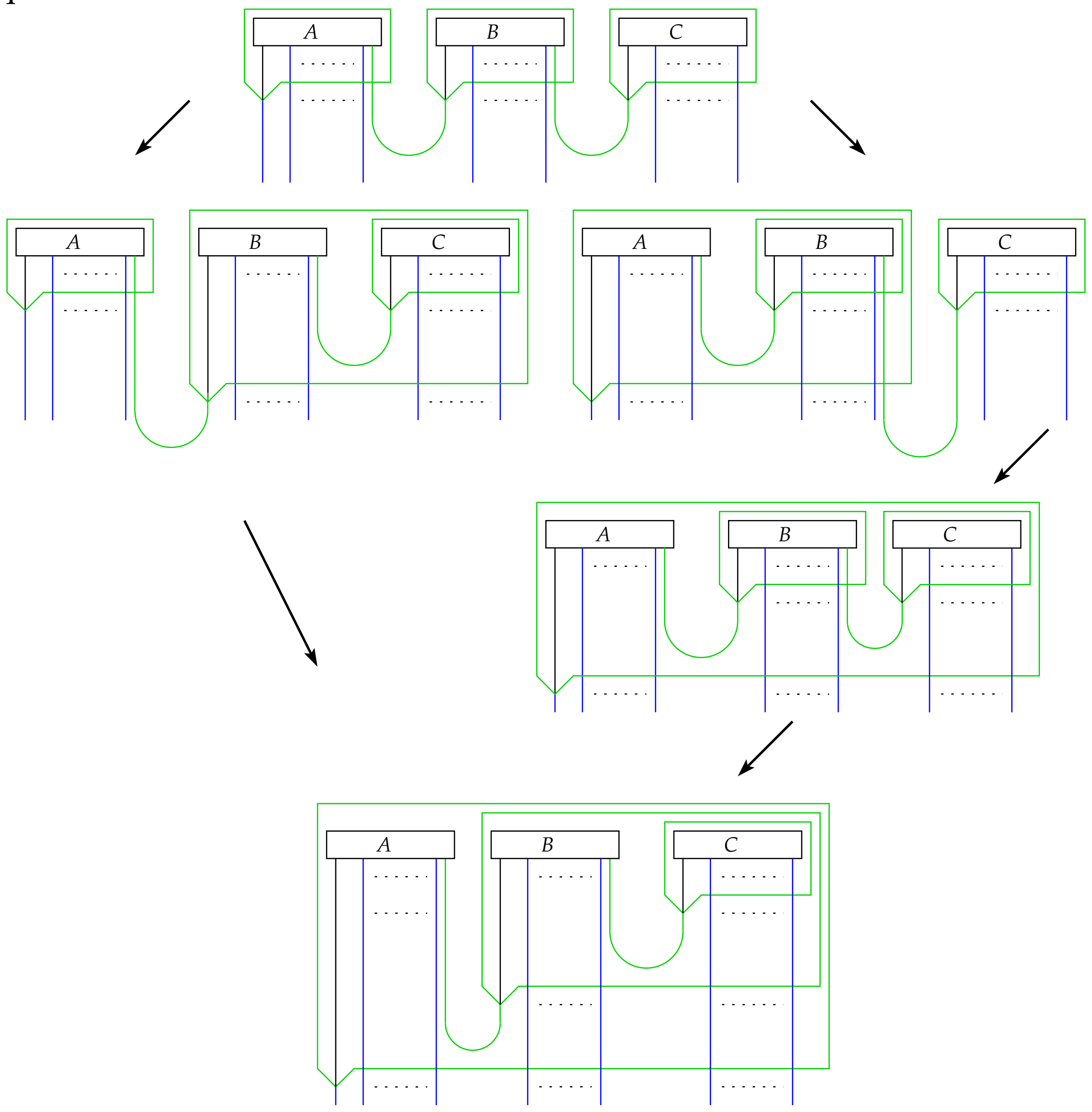
Case 2



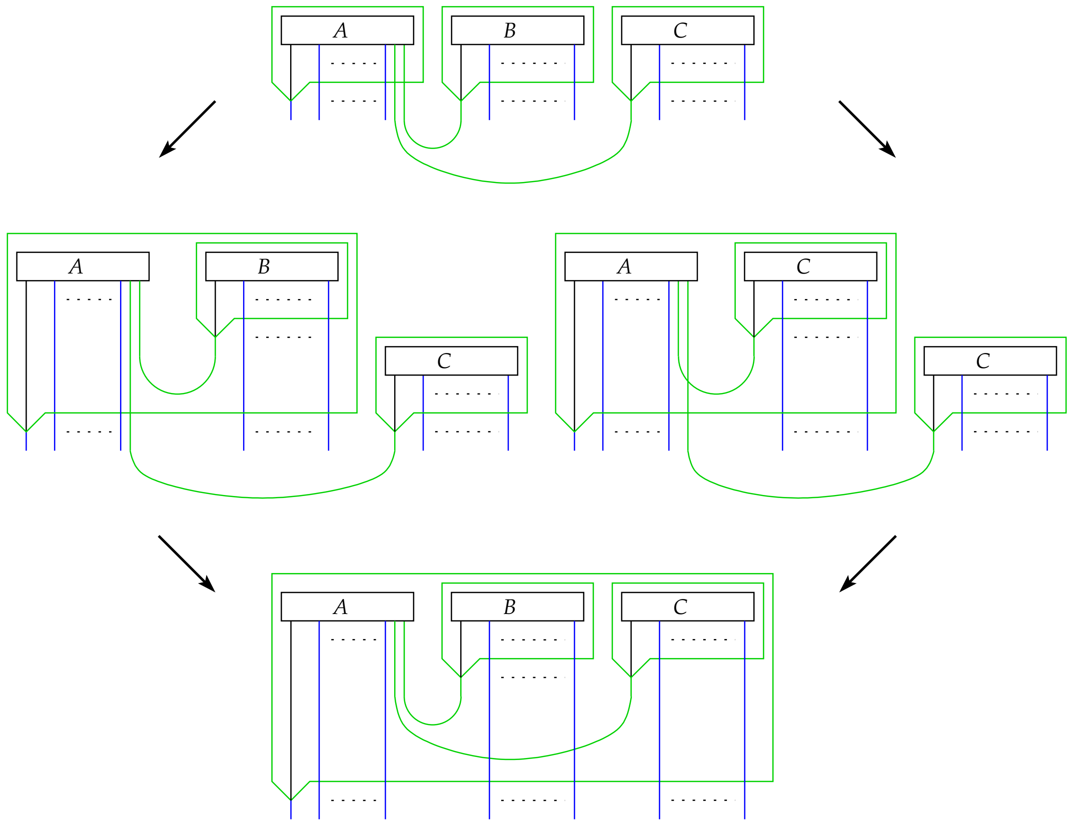
Case 3


A
B
A
B
A
A
B
A
B
A
B
A
B
A
B

Case 4



Case 5



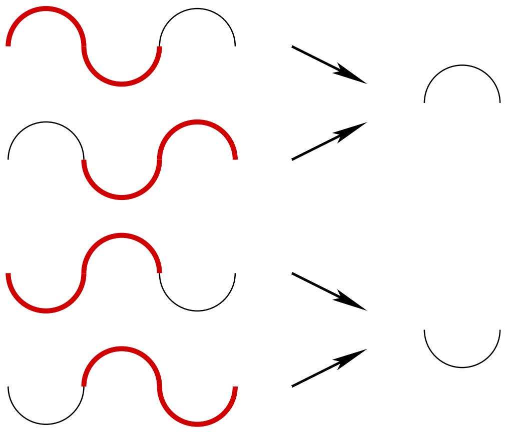
□

Lemma 4.10 (Developments have the Church-Rosser property) *By Newman's lemma: SN and WCR $\Rightarrow$ CR. We proved the first condition (Strongly Normalizing) in Lemma 4.8 and the second in Lemma 4.9, so we may conclude that developments have in fact the Church-Rosser property.*

□

We have now established that developments are CR, but since developments deal with only a few of the proof net reduction rules, we are still far from our initial goal: CR for proof net reduction. Therefore, we will proceed by defining a notion of parallel proof step ($\multimap\!\!\!\!\bullet\!\!\rightarrow$) for the rest of the rules.

Definition 4.18 (Parallel $(\psi, \sim)$-step: $\multimap\!\!\!\!\bullet\!\!\rightarrow$) *Since ψ-steps are orthogonal and do not overlap with $\sim$-steps, one can contract an arbitrary number of ψ-redexes at the same time. $\sim$-steps however are not orthogonal, they are weakly orthogonal: $\sim$-steps overlap with other $\sim$-steps, but contraction of either one of the redexes of their critical pairs, results in the exact same contractum:*



So even though $\sim$-steps are not orthogonal, we can still contract an arbitary number of $\sim$-steps at the same time. These observations result in the following definition of a parallel $(\psi, \sim)$-step ($\mathrel{-\!\!\bullet\!\!\!\rightarrow}$):

$$\boxed{\begin{array}{c} a \mathrel{-\!\!\bullet\!\!\!\rightarrow} b \\ \textit{iff} \\ \textit{the reduction of an arbitrary number of} \\ (\psi,\sim)\textit{-redexes in } a \textit{ at the same time,} \\ \textit{results in } b. \end{array}}$$

In general the union of two rewrite systems that are CR, is not CR. Take for example the two abstract rewriting systems R_1 and R_2 and their union $R_1 \cup R_2$:

R_1	$a \to b$	CR
R_2	$a \to c$	CR
$R_1 \cup R_2$	$a \to b$ $a \to c$	$\times$

So we will explicitly have to prove that the union of our two notions of parallel steps ($\mathrel{-\!\!\circ\!\!\!\rightarrow} \cup \mathrel{-\!\!\bullet\!\!\!\rightarrow}$) is indeed CR.

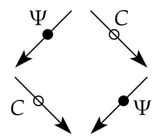
Lemma 4.11 (($\mathrel{-\!\!\circ\!\!\!\rightarrow} \cup \mathrel{-\!\!\bullet\!\!\!\rightarrow}$) represented by $\mathrel{-\!\!\!\ast\!\!\!\rightarrow}$ has the Diamond Property) *Cases:*

1. *Two parallel $(\psi, \sim)$-steps:*

$$\begin{tikzcd} & {} \arrow[dl, "\Psi"'] \arrow[dr, "\Phi"] & \\ {} \arrow[dr, "\Phi - \Psi"'] & & {} \arrow[dl, "\Psi - \Phi"] \\ & {} & \end{tikzcd}$$

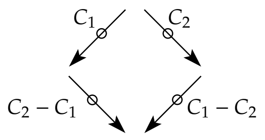
because $\mathrel{-\!\!\bullet\!\!\!\rightarrow}$ is weakly orthogonal.

2. *One development and one parallel $(\psi, \sim)$-step:*



because parallel $(\psi, \sim)$-steps do not overlap with developments. The only thing that might go wrong is when a $(\psi, \sim)$-redex in Ψ is being duplicated by a box–fan redex in C. But by definition of a parallel $(\psi, \sim)$-step, one can contract an arbitrary number of $(\psi, \sim)$-redexes at the same time, in particular both instances of the duplicated $(\psi, \sim)$-redex in Ψ.

3. *Two developments:*



by Lemma 4.10.

□

Lemma 4.12 (($\xrightarrow{\circ} \cup \xrightarrow{\bullet}$) represented by $\twoheadrightarrow$ is CR) *We proved that $\twoheadrightarrow$ has the diamond property in Lemma 4.11 and since DP $\Rightarrow$ CR, $\twoheadrightarrow$ has the Church-Rosser property.*

□

Lemma 4.13 ($\longrightarrow_\pi \subseteq \twoheadrightarrow \subseteq \longrightarrow_\pi^*$) *A single $\longrightarrow_\pi$-step is either the development of a cluster containing exactly one redex, or a parallel $(\psi, \sim)$-step contracting exactly one redex, and since $\twoheadrightarrow$ is defined as the union of all developments and all parallel $(\psi, \sim)$-steps, the first subsumption ($\longrightarrow_\pi \subseteq \twoheadrightarrow$) is a fact. Now for the second subsumption: a $\twoheadrightarrow$-step s_1 is either the development of a cluster C, or a parallel $(\psi, \sim)$-step contracting a set of redexes R:*

- *if s_1 is the development of a cluster C, then s_1 is by definition a $\longrightarrow_\pi^*$-step, for developments are defined as the concatenation of a (finite) number of $\longrightarrow_\pi$-steps.*
- *if s_1 is a parallel $(\psi, \sim)$-step contracting a set of redexes R, s_1 can be simulated by a $\longrightarrow_\pi^*$-step: since none of the redexes in R overlap, one is not obliged to contract them at the same time, but could also choose to contract them one by one in which case the concatenation of all these steps would be a $\longrightarrow_\pi^*$-step.*

□

Lemma 4.14 (DPN-reduction ($\longrightarrow_\pi$) has the Church-Rosser property) *We have already proven that $\twoheadrightarrow$ (or $\xrightarrow{\circ} \cup \xrightarrow{\bullet}$) has the Church-Rosser property in Lemma 4.12 and since $\longrightarrow_\pi \subseteq \twoheadrightarrow \subseteq \longrightarrow_\pi^*$ by Lemma 4.13, we may conclude that the transitive reflexive closure of $\twoheadrightarrow$ is equal to the transitive reflexive closure of $\longrightarrow_\pi$: $\twoheadrightarrow^* = \longrightarrow_\pi^*$. We may now also derive Church-Rosser for $\longrightarrow_\pi$ from the proof of Church-Rosser for $\twoheadrightarrow$.*

□

5 Coinductive Proof Nets

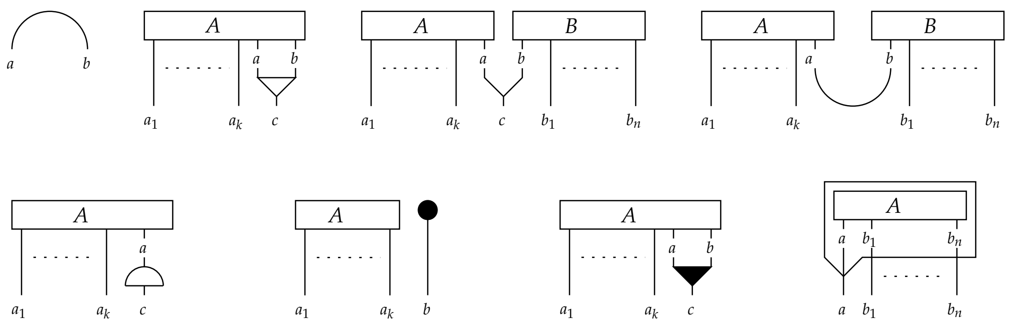
5.1 Coinduction

Coinduction is a relatively new research area, which was introduced to allow reasoning not only about finite, but also about infinite objects such as infinite lists. For an introduction to coinduction and coalgebras one is refered to [Jac97].

Definition 5.1 (Coinduction) *Coinduction is the use of finality for coalgebras.*

5.2 Definition Coinductive Proof Nets

Definition 5.2 (Coinductive Proof Net) *If C is a coinductive proof net, then it is of one of the following forms:*



where A or A and B are coinductive proof nets again with their sets of port defined as:

axiom: $\mathsf{prt}(C) = \{a, b\}$ *for some a and b*

par: $\mathsf{prt}(C) \cup \{a, b\} = \mathsf{prt}(A) \cup \{c\}$

tensor $\mathsf{prt}(C) \cup \{a, b\} = \mathsf{prt}(A) \cup \mathsf{prt}(B) \cup \{c\}$

cut: $\mathsf{prt}(C) \cup \{a, b\} = \mathsf{prt}(A) \cup \mathsf{prt}(B)$

dereliction: $\mathsf{prt}(C) \cup \{a\} = \mathsf{prt}(A) \cup \{c\}$

weakening: $\mathsf{prt}(C) = \mathsf{prt}(A) \cup \{b\}$

fan: $\mathsf{prt}(C) \cup \{a, b\} = \mathsf{prt}(A) \cup \{c\}$

box: $\mathsf{prt}(C) = \mathsf{prt}(A)$

where $\mathsf{prt}(x)$ *is a function from proof nets to sets of ports.*

Definition 5.3 (Proof Net) *The proof net associated with a coinductive proof net P is the graph* $\mathbb{G}(P)$. *The signature of this graph consists of the symbols* $\blacktriangledown : 2,1$ *;* ◖ $: 1,1$ *;* $\bullet : 0,1$ *;* $\triangledown : 2,1$ *;* $\vee : 2,1$ *;* $\cap : 0,2$ *;* $\cup : 2,0$ *and* $\square(Q) : 0, n$, *where n is the number of ports of Q.*

fan: *If the proof net* $\langle V \cup v_i, E \cup \{((v_j, n), (v_i, 1)), ((v_k, m), (v_i, 2))\}\rangle$ *where* $\mathbb{G}(v_i) = \blacktriangledown$*, is associated with the coinductive proof net* $\mathsf{fan}_{v_i}(A)$*, then the proof net associated with* A *is* $\langle V, E\rangle$*.*

box: *If the proof net* $\langle \{v_i\}, \emptyset\rangle$ *where* $\mathbb{G}(v_i) = \square(Q)$*, is associated with the coinductive proof net* $\mathsf{box}_{v_i}(A)$*, then the proof net associated with* A *is* Q*.*

dereliction: *If the proof net* $\langle V \cup v_i, E \cup \{((v_j, n), (v_i, 1))\}\rangle$ *where* $\mathbb{G}(v_i) =$◖ *, is associated with the coinductive proof net* $\mathsf{der}_{v_i}(A)$*, then the proof net associated with* A *is* $\langle V, E\rangle$*.*

weakening: *If the proof net* $\langle V \cup v_i, E\rangle$ *where* $\mathbb{G}(v_i) =$◖ *, is associated with the coinductive proof net* $\mathsf{weak}_{v_i}(A)$*, then the proof net associated with* A *is* $\langle V, E\rangle$*.*

par: *If the proof net* $\langle V \cup v_i, E \cup \{((v_j, n), (v_i, 1)), ((v_k, m), (v_i, 2))\}\rangle$ *where* $\mathbb{G}(v_i) = \triangledown$*, is associated with the coinductive proof net* $\mathsf{par}_{v_i}(A)$*, then the proof net associated with* A *is* $\langle V, E\rangle$*.*

tensor: *If the proof net* $\langle V \cup v_i, E \cup \{\langle 0, ((v_j, n), (v_i, 1))\rangle, \langle 1, ((v_k, m), (v_i, 2))\rangle\}\rangle$ *where* $\mathbb{G}(v_i) = \vee$*, is associated with the coinductive proof net* $\mathsf{tensor}_{v_i}(A, B)$*, then the proof net associated with* A *is* $\langle \pi_A V, \pi_A E\rangle$ *and* B *is* $\langle \pi_B V, \pi_B E\rangle$*.*

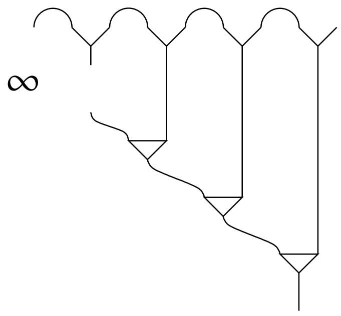
axiom: *The proof net* $\langle \{v_i\}, \emptyset\rangle$*,where* $\mathbb{G}(v_i) = \triangledown$*, is associated with the coinductive proof net* axiom

cut: *If the proof net* $\langle V \cup v_i, E \cup \{\langle 0, ((v_j, n), (v_i, 1))\rangle, \langle 1, ((v_k, m), (v_i, 2))\rangle\}\rangle$ *where* $\mathbb{G}(v_i) = \cup$*, is associated with the coinductive proof net* $\mathsf{cut}_{v_i}(A, B)$*, then the proof net associated with* A *is* $\langle \pi_A V, \pi_A E\rangle$ *and* B *is* $\langle \pi_B V, \pi_B E\rangle$*.*

where $\mathsf{fan}_{v_i}(A)$ *is the coinductive destruction step which removes a* fan *with label* v_i*. The other functions work in a similar way. Furthermore,* $\pi_A(C) = \{a \mid \langle 0, a\rangle \in C\}$ *and* $\pi_B(C) = \{b \mid \langle 1, b\rangle \in C\}$*.*

In Section 4 we saw that that there was an alternative way to define proof nets: the geometrical way. The main difference being that instead of an inductive well-formedness criterion, a switching criterion was used. However, this switching criterion could only be used for the multiplicative fragment of proof nets, because rules like weakening and addition would interfere with connectedness.

Now of course we would like to have a switching criterion for coinductive proof nets as well, but is there such a criterion? It is easy to see that (at least a priori) this is not the case. Just like the rules weakening and addition, the infinite objects of the set of coinductive proof nets, clash with connectedness:



The infinite λ-term corresponding to this infinite proofnet is: $\lambda f_1.\lambda f_2.\lambda f_3.\ldots f_1(f_2(f_3(\ldots)))$
We will now turn to the coinductive version of proof net reduction:

Definition 5.4 (Coinductive Proof Net Reduction $\longrightarrow_{\pi}^{co}$) *See Definition 4.3:* $\longrightarrow_{\pi}^{co} = \longrightarrow_{\pi}$

5.3 Confluence

Just like we did for inductive proof nets, we will try to prove the two important properties one would like every ARS to have: SN and CR. The first one, strong normalization, did not hold for inductive proof nets and since $PN \subseteq CPN$ it will not hold for coinductive proof nets either. However, the question whether or not coinductive proof net reduction has the CR property can not be answered positively just by the fact that inductive proof nets have that property. Take for instance this example from the infinitary λ-calculus:

$$\begin{tikzcd}
{[\lambda f.fffff\ldots]([\lambda x.x]g)} \arrow[r] \arrow[d] & {[\lambda f.fffff\ldots]g} \arrow[d] \\
{([\lambda x.x]g)([\lambda x.x]g)([\lambda x.x]g)([\lambda x.x]g)([\lambda x.x]g)\ldots} \arrow[r, "\infty"] & ggggg\ldots
\end{tikzcd}$$

So even though we have already seen that inductive proof nets are confluent, we still don't know if their coinductive counterpart is CR too or if we lose confluence in extending the set of finite proofnets with infinite ones. Luckily, this is not the case. Since we have constructed the proof of CR for inductive proof nets in such a way that it does not depend on the fact whether or not a proof net is defined inductively or coinductively, the proof of CR for inductive proof nets is also a valid proof of CR for coinductive proof nets!

6 Typed Proof Nets

6.1 Introduction

Just like there is a typed version of the type-free λ-calculus, there is also a typed version of (type-free) proof nets. Linear Logic rather than Intuitionistic Logic will be used to define these types for reasons that will become clear in the following section.

6.2 Linear Logic

Linear Logic (LL) as first described by Girard [Gir87] [Gir95] is a refinement of Classical Logic (CL) which distinguishes itself from other logics by resource sensitivity. This means that provability in LL depends on the number of occurrences of each formula. The need for such a logic is illustrated by the following example:

A	=	Lewis possessing a copy of *Alice in Wonderland*
B	=	Charles possessing a copy of *Alice in Wonderland*
C	=	Rev. Dodgson possessing a copy of *Alice in Wonderland*
D	=	Lewis wants to give *Alice in Wonderland* to Charles
E	=	Lewis wants to give *Alice in Wonderland* to Rev. Dodgson

Furthermore, the following propositions hold:

- A
- D
- E
- $(A \wedge D) \rightarrow B$
- $(A \wedge E) \rightarrow C$

Now in the real world, Lewis would have a problem, because he has only one copy of *Alice in Wonderland* and two people he wants to give it to. But according to CL, Lewis is a happy man, for he can give both Charles and Reverend Dodgson a copy of *Alice in Wonderland*: we just use the proposition A twice. Once to infer B and once to infer C.

So maybe we did something wrong in defining the two implications. Could we perhaps use $(A \wedge D) \rightarrow B \wedge \neg A$ instead of $(A \wedge D) \rightarrow B$? While this new implication does seem to grasp what happens, the meaning CL assigns to it is far from the one we would expect. According to CL, from the new implication combined with $A \wedge B$ (which holds because of our assumptions) we can derive anything, since it is equivalent to $\perp$.

Linear Logic offers an elegant solution to this problem, because using the same proposition twice is not allowed.

As said before, LL is obtained by refining CL. Let us first take a look at CL's structural rules. It is obvious that two of these, namely *weakening* and *contraction*, do not belong in a resource sensitive logic. Therefore, they will be banished as structural rules (but will have to return later on in some other form, since we don't want to lose any of the expressive power of Classical Logic). Furthermore, four new binary connectives, '$\otimes$' (tensor), '$\parr$' (par), '$\oplus$' (plus), '&' (with), will replace the two classical ones, two additional unary connectives, '!' (of course) and it's dual '?' (why not), are introduced and classical negation will be replaced by '$^\perp$' (nil).

Throughout the rest of the text *par* and it's dual *tensor* will be refered to as the multiplicatives or context-insensitive connectives, and *plus* and *with* will be refered to as the additives or context-sensitive connectives. *Of course* and *why not* will be called the exponentials. In this thesis we will only use the multiplicatives and the exponentials.

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}(LC) \quad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}(RC)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}(LW) \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}(RW)$$

6.2.1 Sequent Calculus

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A}(ofcourse) \quad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}(weakening)$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}(dereliction) \quad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}(contraction)$$

$$\frac{\vdash \Gamma, A \quad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}(times) \quad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}(par)$$

$$\frac{}{\vdash A, A^\perp}(identity) \quad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}(cut)$$

and for the additives:

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \& B}(with) \quad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}(leftplus) \quad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}(rightplus)$$

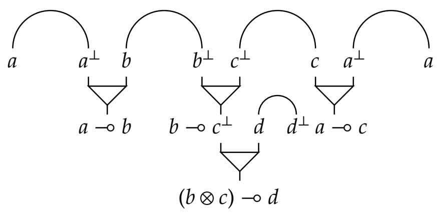
Proof nets give us an elegant tool to identify multiplicative LL-proofs that are essentially the same (modulo the order in which the rules are applied) by mapping two identical LL-proofs to one and the same proof net. An example of such an identification is given by the proof net of the sequent $a, a, a \multimap b, a \multimap c, (b \otimes c) \multimap d \vdash d$. Both:

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$a \vdash a$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$a \vdash a$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$b \vdash b$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$c \vdash c$}
\RightLabel{$(\otimes R)$}
\BinaryInfC{$b, c \vdash b \otimes c$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$b, c, (b \otimes c) \multimap d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$a, b, a \multimap c, (b \otimes c) \multimap d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$a, a, a \multimap b, a \multimap c, (b \otimes c) \multimap d \vdash d$}
\end{prooftree}
$$

and:

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$a \vdash a$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$a \vdash a$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$b \vdash b$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$c \vdash c$}
\RightLabel{$(\otimes R)$}
\BinaryInfC{$b, c \vdash b \otimes c$}
\AxiomC{}
\RightLabel{(Ax)}
\UnaryInfC{$d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$b, c, (b \otimes c) \multimap d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$a, a \multimap b, c, (b \otimes c) \multimap d \vdash d$}
\RightLabel{$(\multimap L)$}
\BinaryInfC{$a, a, a \multimap b, a \multimap c, (b \otimes c) \multimap d \vdash d$}
\end{prooftree}
$$

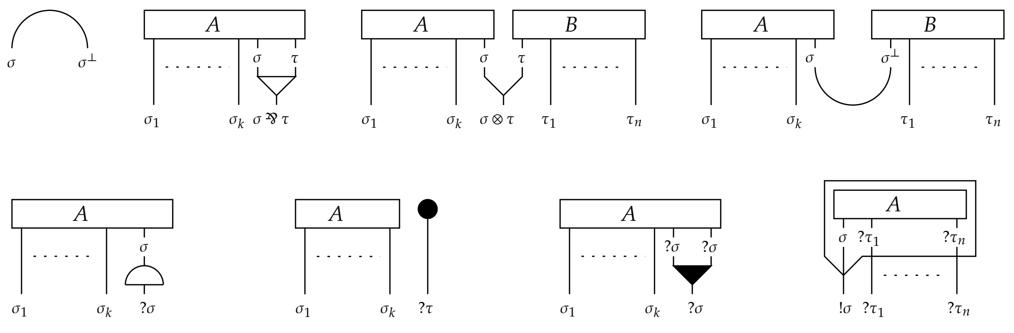
will be mapped to the following proof net:



6.3 Definition TPN

Definition 6.1 (Typed Inductive Proof Net) *A typed inductive proofnet is a directed proofnet for which a type is assigned to every port of the net. C : σ means that the typed proofnet C has a global output port labeled σ.*

If A and B are typed inductive proof nets, then so are:



Definition 6.2 (Typed Proof Nets) *The typed proof net associated with an typed inductive proof net P is the graph $\mathbb{G}(P)$. This graph is constructed in a way similar to the one in Definition 4.2 only now every port is assigned a label (type).*

6.4 SN for Typed Proof Nets

Proving that typed proof net reduction is strongly normalizing is difficult, because it seems to have both increasing and decreasing rewriting rules. For example, taking the number of boxes of a proof net as a measure would not work, since box–weakening deletes boxes, while box–fan duplicates them.

For this reason we will split the $\longrightarrow_{\pi^t}$-ARS in two: the first system will contain only the weakening rule and the second every non-weakening rule. Now to prove that $\longrightarrow_{\pi^t}$ is SN, we will prove that the two abstract rewriting systems resulting from this division are both SN. Furthermore we will have to prove that their union $\longrightarrow_{\pi^t}$ is still SN, because this is not the case in general. Take for example the two abstract rewriting systems R_1 and R_2:

R_1	$a \to b$	SN
R_2	$b \to a$	SN
$R_1 \cup R_2$	$a \to b$ $b \to a$	×

Independently, they are both strongly normalizing, but their union is obviously not. To prove that the union of our two abstract rewriting systems $\to_w$ and $\longrightarrow_{\pi^t_{-w}}$ is indeed SN, we will use the following lemma:

Lemma 6.1 (Postponing) *Let $\to = (\to_1 \cup \to_2)$ and if $a \to_1 b \to_2 c$ then $a \to_2 d \twoheadrightarrow c$ (which is equivalent to $(\to_1 \circ \to_2) \subseteq (\to_2 \circ \twoheadrightarrow)$) then $\to$ is SN iff $\to_1$ and $\to_2$ are both SN.*

The 'only if' direction of the iff is trivial, since $\to_1 \subseteq \to$ and $\to_2 \subseteq \to$. To prove the 'if' direction, we will show that if $\to$ permits an infinite sequence, then there is a reduction $a \to_2 b$ such that b permits an infinite reduction sequence again. This would prove that $\to_2$ is not SN, which contradicts one of our assumptions.

1st step is $\to_2$: *Trivial.*

1st step is $\to_1$: *Since $\to_1$ is SN by assumption, the remaining reduction sequence will be of the form $\twoheadrightarrow_1 \circ \to_2 \circ R$ and because from our assumption that $(\to_1 \circ \to_2) \subseteq (\to_2 \circ \twoheadrightarrow)$ we can derive $(\twoheadrightarrow_1 \circ \to_2) \subseteq (\to_2 \circ \twoheadrightarrow)$ by induction on the length of $\twoheadrightarrow_1$, we can create an infinite reduction.*

□

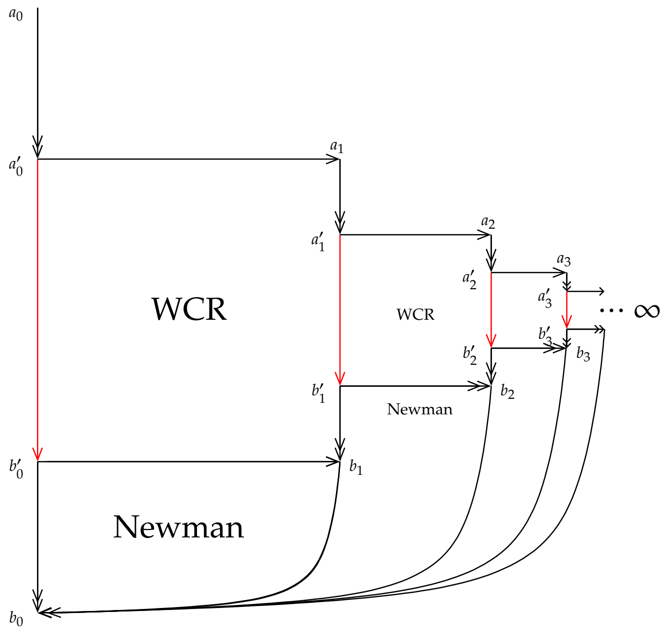
So our aim is now to prove three things:

1. SN for $\longrightarrow_{\pi^t_{-w}}$
2. that $\to_w$-steps can always be postponed

First, we will give a proof of SN for $\longrightarrow_{\pi^t_{-w}}$, using the following lemma which can be found in [Oos01]. The idea is that if an object keeps growing and there is some sort of limit to how big such an object can become, the process will eventually stop. We will use the notion *eventually increasing*:

Definition 6.3 (Eventually Increasing) *An ARS is eventually increasing if there is a function f to $\mathbb{N}$ such that if $a \rightarrow b$ then $f(a) \leq f(b)$, and $\rightarrow \cap =_f$ is SN.*

Lemma 6.2 (WCR, WN, eventually increasing $\Rightarrow$ SN) *An ARS which is WCR, WN and eventually increasing, is SN. We will prove this using the following scheme:*

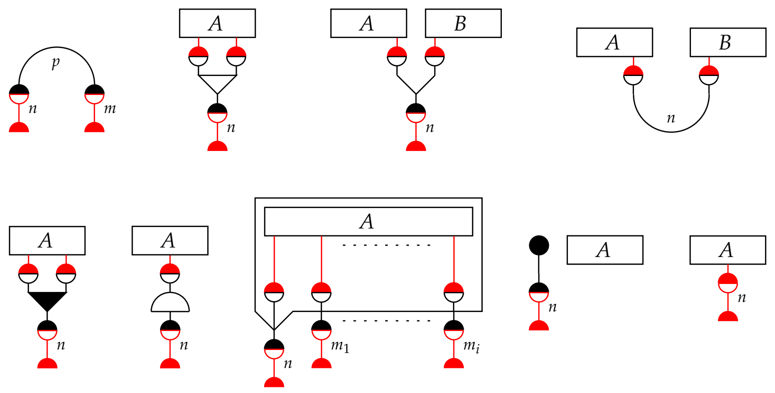


Let R be an unsafe reduction $R : a_0 \twoheadrightarrow b_0$ *to normalform. The reduction is* unsafe *in the sense that there is another reduction $S : a_0 \twoheadrightarrow a_1 \rightarrow \ldots\infty$. Note that for every unsafe reduction to normal form there is a critical step, the contractum of which is SN. In the picture above, the critical steps are red. Now using WCR repeatedly, we can construct an infinite reduction sequence $a_0 \twoheadrightarrow a_1 \rightarrow \ldots\infty$ such that $f(a_i) \leq f(b_0)$, which leads to a contradiction because of well-foundedness of f. We may now conclude that there are no* unsafe reductions.

□

For the purpose of proving that $\longrightarrow_{\pi^t_{-w}}$ is eventually increasing, we will introduce a more sophistocated kind of an inductive proof net, called an inductive memory net. The introduction of these memory nets is necessary, because $\longrightarrow_{\pi^t_{-w}}$ still has both increasing and decreasing rules. For instance, if we would like to define the size of a typed proof net as the sum of the size of every cut node, we would find that most rules are decreasing except for the box–fan rule, which duplicates an arbitrary number of cuts. We could also take the sum of the weight of every node of a proof net as a measure, but this approach is bound to fail as well, since in the ~-rules nodes inevitably disappear. We will see that the memory net equivalent of $\longrightarrow_{\pi^t_{-w}}$, which will be called $\rightarrow_m$, does not have any decreasing rules. The name 'memory net' is used because unlike ordinary inductive proof nets, memory nets 'remember' how many increasing rules have been applied. That is, they store integers that are raised by the increasing rules in their so-called storage links. We will now give a formal definition of a memory net:

Definition 6.4 (Inductive Memory Nets) *If m, n and p are integers and A and B are inductive memory nets, then so are the following nets:*

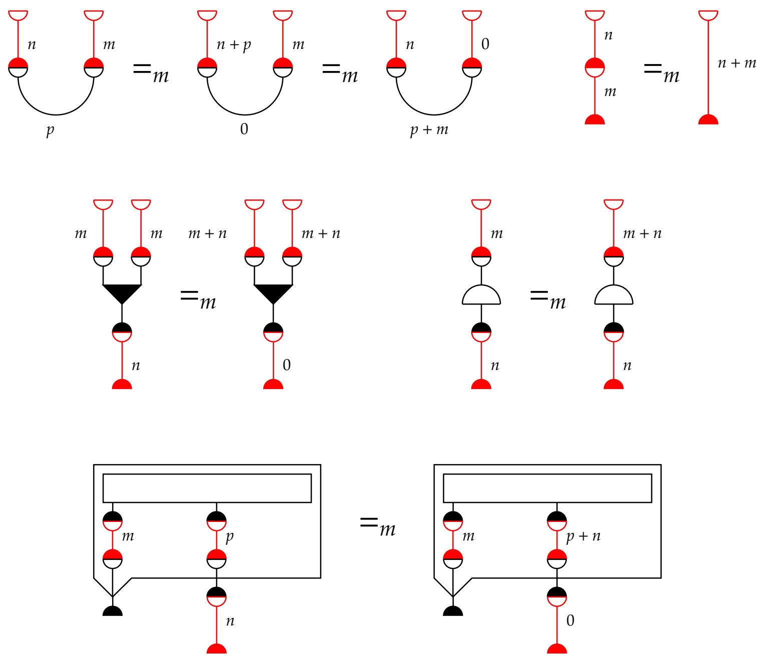


We see that memory nets are in fact ordinary proof nets of which the cut and axiom nodes are labeled with an integer and every link has been replaced by a number of what we will call *storage links* (the red objects in the picture above). Just like the cut and axiom links, storage links are labelled with an integer. They can be seen as representations of little pieces of a proof net link, resulting from splitting the edges of an ordinary proof net. However, some storage links do *not* represent part of an ordinary proof net link: for the storage links that are not connected through their lower port their is no proof net link equivalent. To be able to distiguish between these two kinds of storage links, we introduce the notion of *global port*:

Definition 6.5 (Global Ports) *A port will be call 'global' if it is not connected to another port.*

We will also define a notion of equivalence on the memory nets. The idea behind this equivalence relation is that intuitively it doesn't matter in what memory-node the integers are being stored, just as long as the total amount being stored remains the same.

Definition 6.6 (Memory Equivalence ($=_m$))



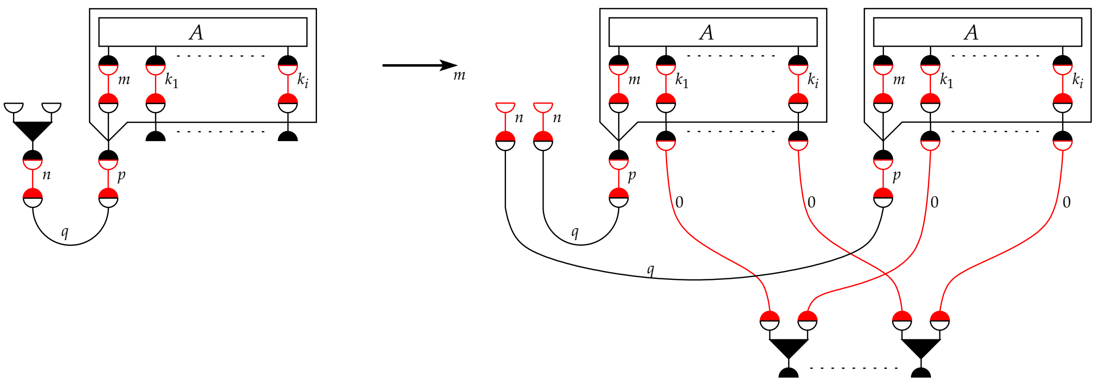
n m p $=_m$ $n+p$ m 0 $=_m$ n 0 $p+m$
n m $=_m$ $n+m$
m m n $=_m$ $m+n$ $m+n$ 0
m n $=_m$ $m+n$ n
m p n $=_m$ m $p+n$ 0

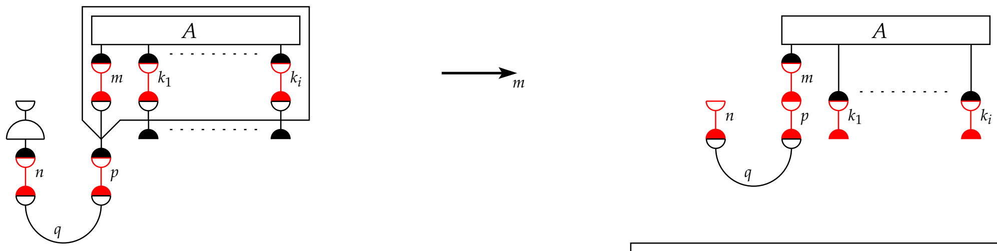
Obviously, we will demand that if one of these equivalence rules is applied to a memory net the result is still a memory net. In other words, we need to prove the subject reduction property for $=_m$:

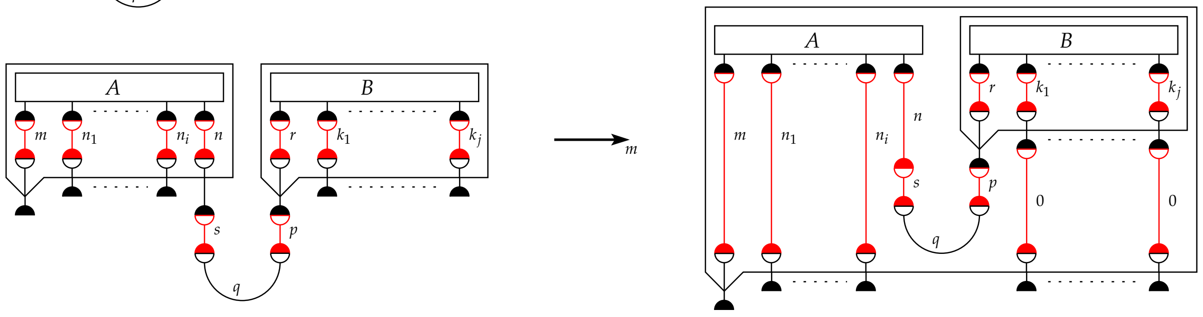
Lemma 6.3 (SR for $=_m$) *Since $=_m$ never deletes storage links, but only merges/splits them (deletion would give us trouble at the global ports) and moves around their labels, the result applying a $=_m$-rule to a memory net is still a memory net.*

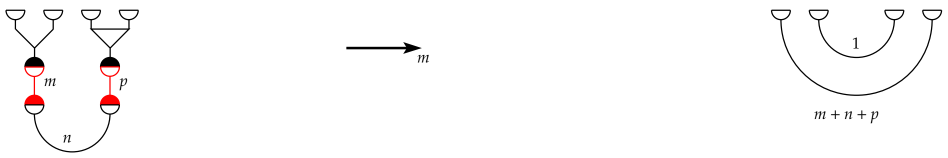
□

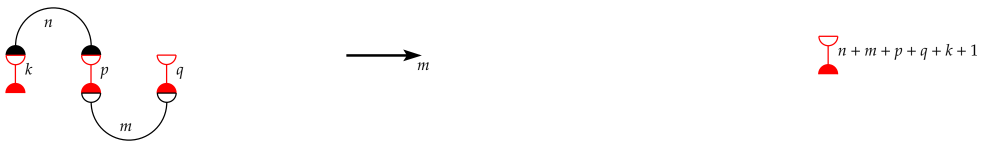
Definition 6.7 (Memory Net Reduction ($\to_m$))

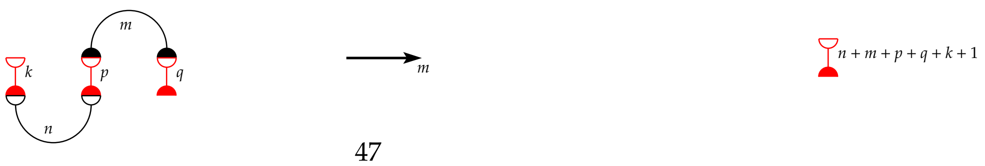

A
m
k_1
k_i
n
p
q
m
A
A
m
k_1
k_i
n
n
p
p
q
q
0
0
0
0


A
m
k_1
k_i
n
p
q
m
A
m
p
n
q
k_1
k_i


A
m
n_1
n_i
n
B
r
k_1
k_j
s
p
q
m
A
m
n_1
n_i
n
B
r
k_1
k_j
s
p
q
0
0

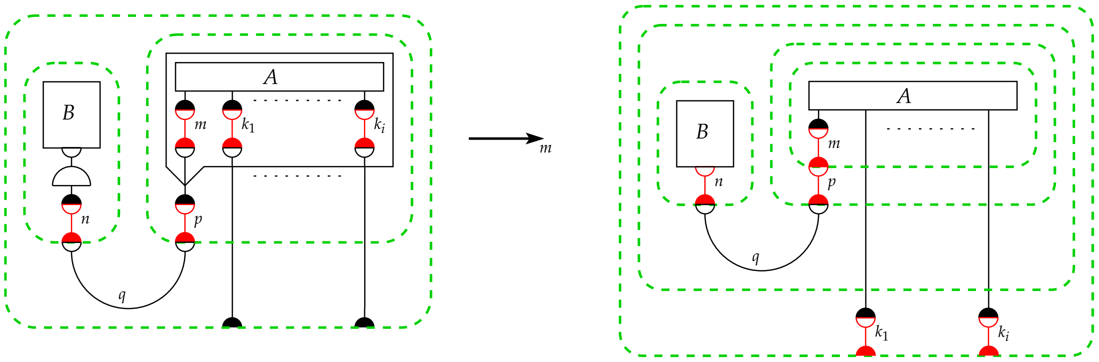

m
p
n
m
1
$m + n + p$


n
k
p
q
m
m
$n + m + p + q + k + 1$

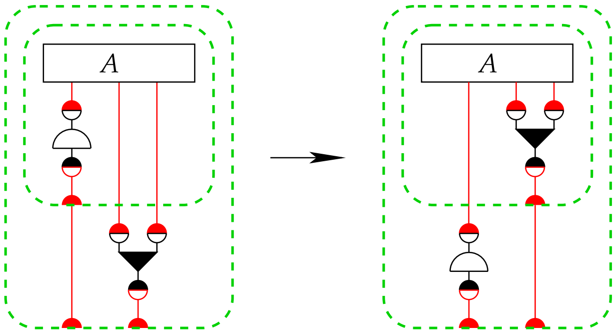

m
k
p
q
n
m
$n + m + p + q + k + 1$

We do not only want the subject reduction property for $=_m$, but also for $\to_m$: if a $\to_m$-rule is applied to a memory net then the result should be a memory net again:

Lemma 6.4 (SR for $\to_m$) *We need show for every left-hand-side of a $\to_m$-rule, that if it is a well-formed memory net, its right-hand-side is too. Since this is rather obvious, we will only show this for the box–dereliction rule:*



Green boxes indicate well-formed components of the proof net. Like we saw in Lemma 4.1, there is only one problem we might encounter in constructing a scheme such as the one shown above: we might have associated the wrong inductive structure with the proof net shown on the left-hand side of the $\to_m$-rule, for we know that more than one inductive memory net can be associated with a memory net. However, if a dereliction-node is indeed connected to a box through a number of storage links and a cut link, one can always re-arrange the green boxes to fit the left-hand side of the $\to_m$-rule shown above, using rules like the following:



The same goes for the rest of the $\to_m$-rules.

□

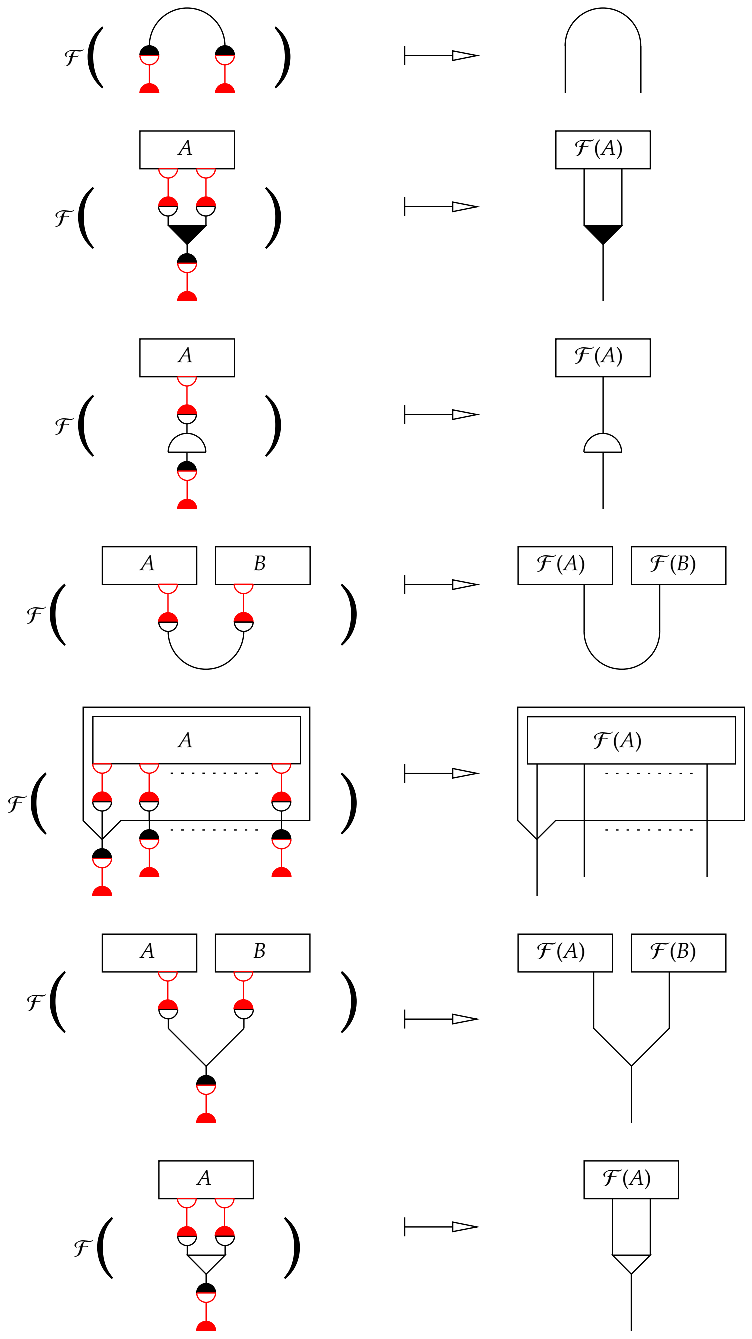
Of course we want to want some kind of correspondence between these newly defined memory nets and our original proof nets. This correspondence will be shown by proving two important lemmas (the Lifting Lemma and the Projection Lemma) on the mapping $\mathcal{M}$, which maps a memory net in memory-NF to its corresponding proof net. The idea here is that we want to transform a memory net to some sort of standard, on which we will be able to apply the mapping. This standard will be called the memory-NF of a memory net. Memory Nets in memory-NF will have a unique storage link at every non-storage-link-port (i.e. every port which is not the port of a storage link), so that they can be 'broken down' by the mapping. This mapping will, in transforming a memory net into an ordinary proof net, delete the storage links attached to the translated link.

Definition 6.8 (memory-NF of a Memory Net) *A memory net M is in memory-NF iff*

- *for every two nodes of M which are connected by a path of storage links, this path has length 2 and*
- *there is exactly one storage link at every global output and input port of M*

This memory-NF can be reached using $=_m$.

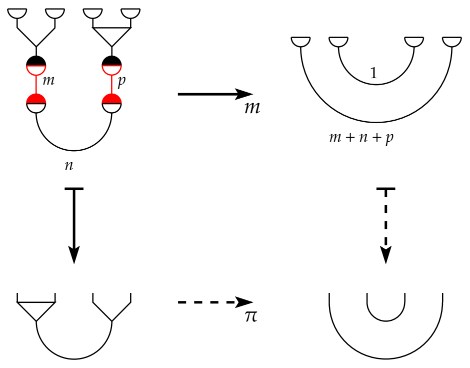
Definition 6.9 (Mapping $\mathcal{M}$) *The function $\mathcal{M}$ first brings a memory net to memory-NF. Then all the storage links connected to a global port are thrown away and all the other ports are mapped to proof net links, by means of the following function $\mathcal{F}$:*



Lemma 6.5 (Projection)

$$\begin{tikzcd}
m \arrow[r] \arrow[d, mapsto, "\mathcal{MAP}"] & m' \arrow[d, mapsto, dashed, "\mathcal{MAP}"] \\
n \arrow[r, dashed] & n'
\end{tikzcd}$$

Since the storage links are merely a specification of ordinary links every memory net can be mapped to a proof net by means of the mapping $\mathcal{M}$, which removes every storage link from a memory net. Furthermore, every $\rightarrow_m$ rule has a $\longrightarrow_{\pi^t_{-w}}$-equivalent, so it suffices to show that for every $\rightarrow_m$-rule, its left-hand-side can be mapped to the left-hand-side of a certain $\longrightarrow_{\pi^t_{-w}}$-rule P and that its right-hand-side side can be mapped to the right-hand-side of P. Since this is fairly obvious, we only show such a map for one of the $\rightarrow_m$-rules:

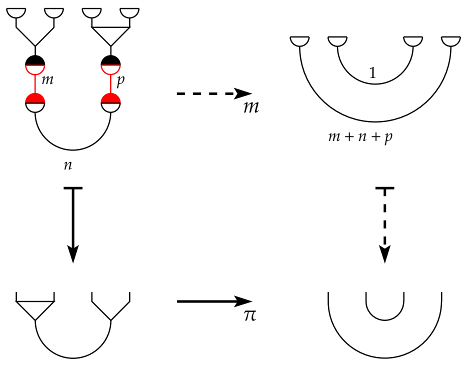


□

Lemma 6.6 (Lifting modulo $=_m$)

$$\begin{tikzcd}
m \arrow[r, dashed] \arrow[d, mapsto, "\mathcal{MAP}"] & m' \arrow[d, mapsto, dashed, "\mathcal{MAP}"] \\
n \arrow[r] & n'
\end{tikzcd}$$

Using the mapping $\mathcal{M}$ again, we will show there is an $\rightarrow_m$-equivalent of every $\longrightarrow_{\pi^t_{-w}}$-rule, which means we will prove that for every $\rightarrow_m$-rule M, if its left-hand-side can be mapped to the left-hand-side of a $\longrightarrow_{\pi^t_{-w}}$-rule P, the right-hand-side of the $\rightarrow_m$-rule M can be mapped to the right-hand-side of the $\longrightarrow_{\pi^t_{-w}}$-rule P. Since this is fairly obvious, we only show such a map for one of the $\longrightarrow_{\pi^t_{-w}}$-rules:



□

According to Lemma 6.2, the first property we have to prove in order to get SN for $\to_m$ is WCR. However, since an equivalence relation ($=_m$) is defined on the memory nets the $\to_m$-rules are applied to, proving WCR for $\to_m$ will do us no good. Instead, we need to prove WCR for $\to_m$ modulo $=_m$. But first a definition of what it is exactly that we mean by 'weakly confluent modulo' will be given.

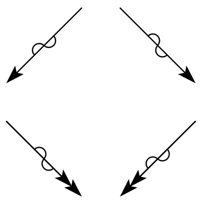
In an ARS that is extended by a set of equations, we will typically have rewriting sequences of the form:

$$a \sim a' \to b \sim b' \to c \sim c' \to \ldots$$

where $\sim$ is some equivalence relation. So in this more general case, we also find that trying to establish local confluence for our rewrite relation (which in this case is $\to$) would be useless. Rather, we would like to have local confluence for the relation defined by $\sim \circ \to \circ \sim$.

Definition 6.10 ($\rightsquigarrow$) *The $\rightsquigarrow$-arrow is defined as $\sim \circ \to \circ \sim$, where $\circ$ is relation composition*[6]*. In our case, $\rightsquigarrow_m$ will be: $=_m \circ \to_m \circ =_m$.*

So to prove local confluence for $\to_m$ modulo $=_m$ we will have to show that: $(C \;{}_m\!\leftsquigarrow A \rightsquigarrow_m B) \Rightarrow (C \rightsquigarrow_m D \;{}_m\!\leftsquigarrow B)$:



We will do this with the help of the following scheme:

[6] $a \to_1 \circ \to_2 b$ iff $a \to_1 b' \to_2 b$

$$\begin{tikzcd}
 & & & & P \arrow[dl, dash, squiggly] \arrow[dr, dash, squiggly] & & & & \\
 & & & a \arrow[dll] \arrow[dr, squiggly] & \text{def} & b \arrow[dl, squiggly] \arrow[drr] & & & \\
 & a' \arrow[ddr, dotted, squiggly] & \text{L 6.7} & & c \arrow[dl, dashed] \arrow[dr, dashed] & & \text{L 6.7} & b' \arrow[ddl, dotted, squiggly] & \\
 & & & c'' \arrow[dl, dotted, squiggly] \arrow[dr] & \text{L 6.8} & c' \arrow[dl] \arrow[dr, dotted, squiggly] & & & \\
Q \arrow[uur, dash, squiggly] \arrow[rr, dotted, squiggly] & \text{def} & d & & S & & e & \text{def} & R \arrow[ll, dotted, squiggly] \arrow[uul, dash, squiggly]
\end{tikzcd}$$

Here $\rightsquigarrow$ is a function using (an oriented version of $=_m$) which rewrites a memory net to its USNF, with USNF defined as below:

Definition 6.11 (USNF of a memory net) *A memory net is in USNF if every non-zero integer is pushed forward in the direction of the arrows as far as the $=_m$-equivalence allows.*

So we could say that two memory nets are the same modulo $=_m$ if they have the same USNF.

Lemma 6.7 (($B\,{}_{\mathfrak{m}}\!\leftarrow A \rightsquigarrow A'$) $\Rightarrow$ ($B \rightsquigarrow C \leftrightsquigarrow \circ\ {}_{\mathfrak{m}}\!\leftarrow A'$)) *The idea here is that we would like to apply both $\rightarrow_m$-steps and normalization arbitrarily, that is, it shouldn't matter whether we choose to first normalize a memory net and then apply a $\rightarrow_m$-step or the other way around.*

Since normalization is defined as 'pushing every integer forward as far as possible' and since

- *no rule of $\rightarrow_m$ blocks a path*
- *no unlawful duplication (duplication which would not be taken care of by unsharing) is done by any of the $\rightarrow_m$-rules*
- *by how much the net is increased does not depend on the weight of any link*

we can safely assume that we won't have any trouble normalizing a net after applying a $\rightarrow_m$-step. However a path can *be opened by the tensor-par rule, which will allow the integers that we trapped between the tensor and the par to travel to the end of the newly created path, so additional normalization may be necessary if we choose do normalization first.*

$\square$

Lemma 6.8 (($d\ {}_m\!\!\leftarrow c \rightarrow_m e$) $\Rightarrow$ ($d \twoheadrightarrow_m S\ {}_m\!\!\twoheadleftarrow e$)) *We will now prove the final part of our WCR-mod proof: if to a certain memory net two $\rightarrow_m$-rules are applied, the two resulting memory nets can be rewritten into the same memory net again. Note that for the sake of readability, we have left out quite a few of the obvious labels of the critical pair pictures.*

1. *No overlap:*

$$\begin{tikzcd}
& {} \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
{} \arrow[dr, "m_2"'] & & {} \arrow[dl, "m_1"] \\
& {} &
\end{tikzcd}$$

or worst case (if the redex of m_1 is duplicated by m_2):

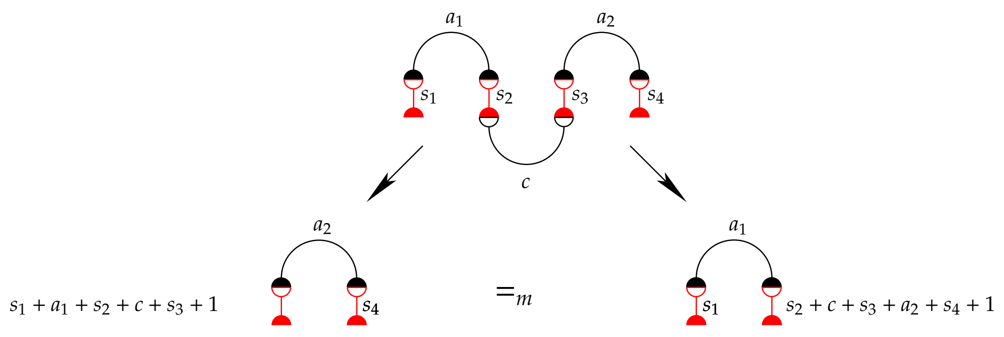
$$\begin{tikzcd}
& {} \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
{} \arrow[dr, "m_2"'] & & {} \arrow[dl, twoheadrightarrow, "m_1", "m_1"'] \\
& {} &
\end{tikzcd}$$

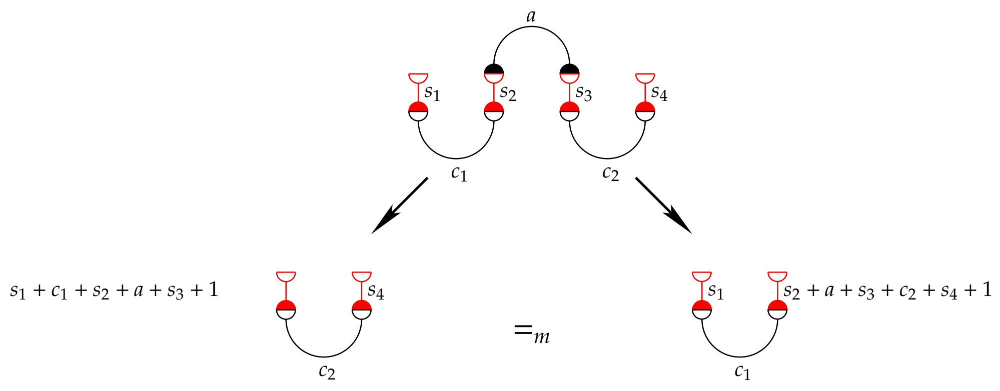
2. (a) $m_1 = m_2$:

$$\begin{tikzcd}
& {} \arrow[dl, "m_1"'] \arrow[dr, "m_2"] & \\
{} \arrow[dr, "0"'] & & {} \arrow[dl, "0"] \\
& {} &
\end{tikzcd}$$

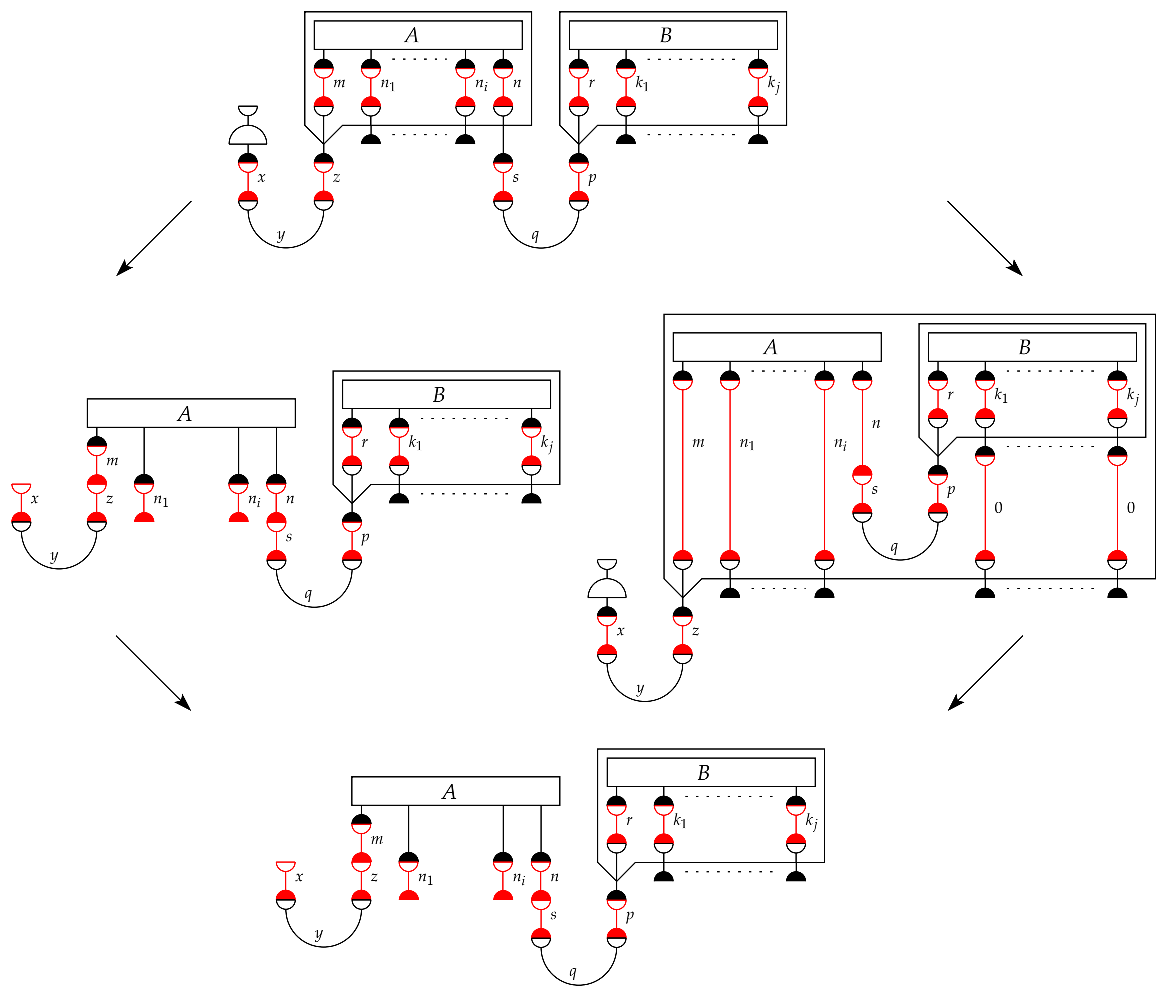
(b) *Critical pairs:*

Case 1

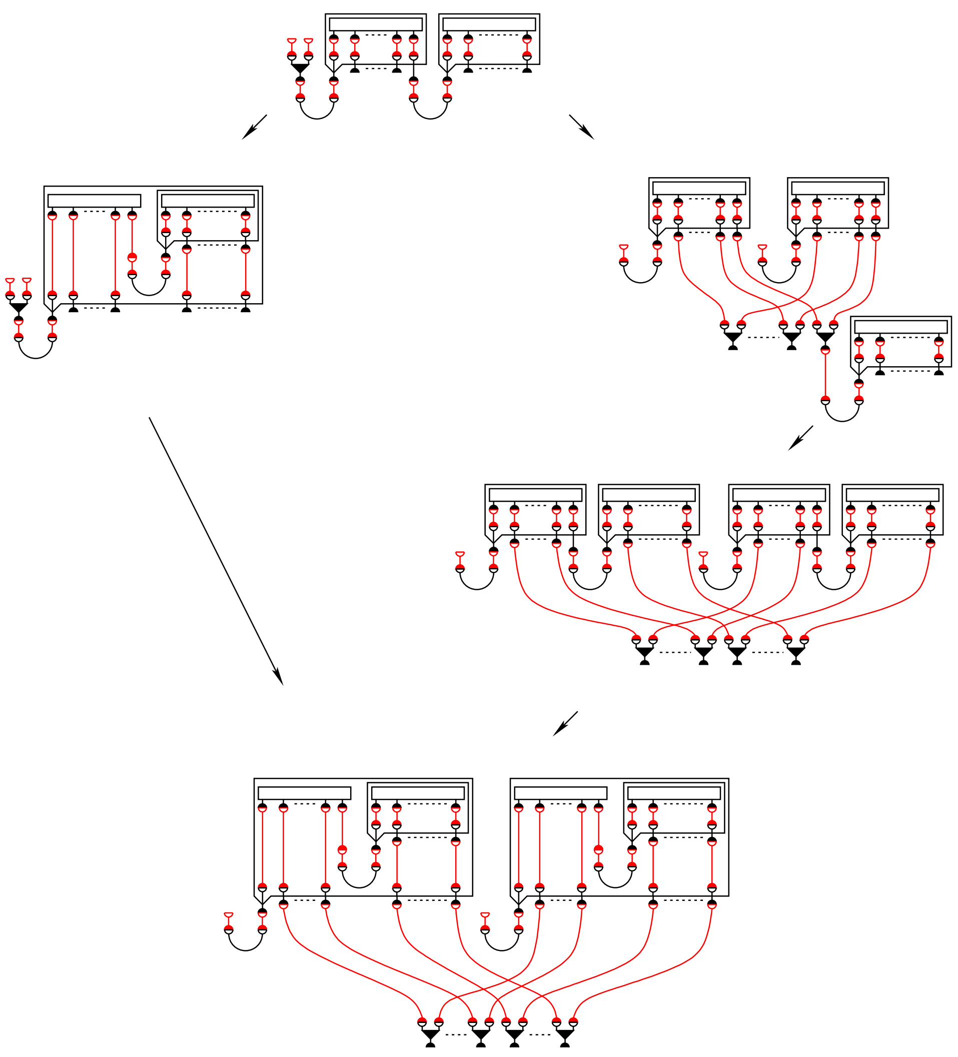

a_1
a_2
s_1
s_2
s_3
s_4
c
a_2
s_4
$s_1 + a_1 + s_2 + c + s_3 + 1$
$=_m$
a_1
s_1
$s_2 + c + s_3 + a_2 + s_4 + 1$


a
s_1
s_2
s_3
s_4
c_1
c_2
$s_1 + c_1 + s_2 + a + s_3 + 1$
s_4
c_2
$=_m$
s_1
$s_2 + a + s_3 + c_2 + s_4 + 1$
c_1

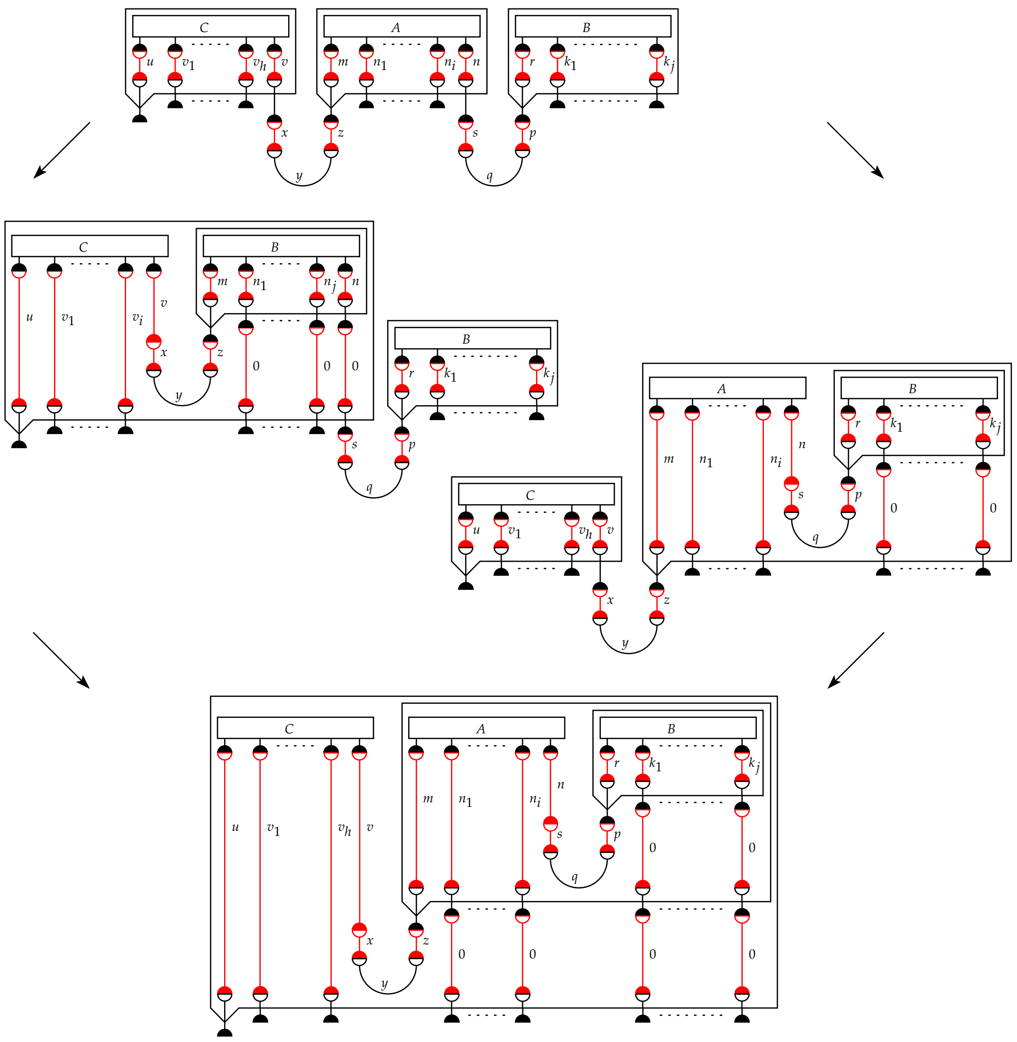
Case 2



Case 3



Case 4



□

Lemma 6.9 (($c' \rightsquigarrow e \leftsquigarrow R) \Rightarrow R =_m c'$) *By definition of $\rightsquigarrow$: two memory nets are equal modulo $=_m$ if they have the same USNF.*

□

Lemma 6.10 ($\rightarrow_m$ is weakly confluent modulo $=_m$) *By Lemma 6.8 and Lemma 6.7 the following scheme[7] proves WCR modulo $=_m$ for $\rightarrow_m$:*

[7]this is the exact same scheme as the one that appeared on page 53

$$\begin{tikzcd}
 & & & & P \arrow[dl, squiggly, no head] \arrow[dr, squiggly, no head] & & & & \\
 & & & a \arrow[dll] \arrow[dr, squiggly] & \text{def} & b \arrow[dl, squiggly] \arrow[drr] & & & \\
 & a' \arrow[ddr, dotted, squiggly] & \text{L 6.7} & & c \arrow[dl, dashed] \arrow[dr, dashed] & & \text{L 6.7} & b' \arrow[ddl, dotted, squiggly] & \\
 & \text{def} & & c'' \arrow[dl, dotted, squiggly] \arrow[dr, Rightarrow] & \text{L 6.8} & c' \arrow[dl, Rightarrow] \arrow[dr, dotted, squiggly] & & & \text{def} \\
Q \arrow[uur, squiggly, no head] \arrow[rr, dotted, squiggly] & & d & & S & & e & & R \arrow[uul, squiggly, no head] \arrow[ll, dotted, squiggly]
\end{tikzcd}$$

□

Besides WCR, which we proved in Lemma 6.10, we needed to prove two more properties of $\rightarrow_m$ in order to derive SN for $\rightarrow_m$. One of these was that $\rightarrow_m$ is eventually increasing. A proof of this proposition is given below.

Definition 6.12 (Weight $\mathcal{S}$ of a Memory Net in USNF) *The weight of a memory net M in USNF is the sum of all the labels of M.*

Definition 6.13 ($=_\mathcal{S}$-step) *A step x is an $=_\mathcal{S}$-step iff $x \in (\rightarrow_m \cap =_\mathcal{S})$. This means that box–fan, box–dereliction and box–box the x-rules are the $=_\mathcal{S}$-steps, because the weight $\mathcal{S}$ of a memory net M does not change when one of these rules is applied.*

Lemma 6.11 (($\rightarrow_m \cap =_\mathcal{S}$) is Strongly Normalizing) *We have already seen that developments of directed inductive clusters of $\longrightarrow_\varpi$ are strongly normalizing (Lemma 4.8) and that there is a proof net equivalent of every memory net rule (Projection – Lemma 6.5). Now if we take a closer look, we see that the $=_\mathcal{S}$-rules are exactly those memory net rules that have a $\longrightarrow_\varpi$-equivalent, so the Projection Lemma allows us to conclude that ($\rightarrow_m \cap =_\mathcal{S}$) is strongly normalizing as well.*

□

Lemma 6.12 ($\longrightarrow_{\pi^t_{-w}}$ is eventually increasing) *To prove that $\longrightarrow_{\pi^t_{-w}}$ is eventually increasing, we need a measure which eventually increases and a proof that the $=_m$-rules are SN. The latter has already been proven in Lemma 6.11 and we will show that the weight $\mathcal{S}$ of a memory net is an apt measure which eventually increases:*

1. *$=_\mathcal{S}$-steps:*

   box–dereliction: $\mathcal{S}(rhs) = \mathcal{S}(lhs)$

box–fan: $\mathcal{S}(rhs) = \mathcal{S}(lhs)$

box–box: $\mathcal{S}(rhs) = \mathcal{S}(lhs)$

2. *$(\psi, \sim)$-steps:*

   tensor–par: $\mathcal{S}(lhs) + 2 \leq \mathcal{S}(rhs) \Rightarrow \mathcal{S}(lhs) < \mathcal{S}(rhs)$

   axiom–cut: $\mathcal{S}(lhs) + 1 \leq \mathcal{S}(rhs) \Rightarrow \mathcal{S}(lhs) < \mathcal{S}(rhs)$

   cut–axiom: $\mathcal{S}(lhs) + 1 \leq \mathcal{S}(rhs) \Rightarrow \mathcal{S}(lhs) < \mathcal{S}(rhs)$

□

Now the last property we had to prove was WN for $\rightarrow_m$. We will show this by presenting a reduction strategy, which always leads to the normal form of a proof net.

Definition 6.14 ($=_\mathcal{S}$-normalform) *A (typed) proof net is in $=_\mathcal{S}$-NF iff it does not contain any $=_\mathcal{S}$-redexes.*

Definition 6.15 *The weight $\mathcal{G}'$ of a typed proof net is the sum of the weight* g *of the types of all cut-nodes. The weight* g *of a type:*

- $\mathsf{g}(a) = 1$ *if $a \in VAR$*
- $\mathsf{g}(a^\perp) = 1$ *if $a \in VAR$*
- $\mathsf{g}(\sigma \otimes \tau) = \mathsf{g}(\sigma) + 1 + \mathsf{g}(\tau)$
- $\mathsf{g}(\sigma \parr \tau) = \mathsf{g}(\sigma) + 1 + \mathsf{g}(\tau)$
- $\mathsf{g}(!\sigma) = 1 + \mathsf{g}(\sigma)$
- $\mathsf{g}(?\sigma) = 1 + \mathsf{g}(\sigma)$

Lemma 6.13 ($\longrightarrow_{\pi^t_{-w}}$ is Weakly Normalizing) *To prove that $\longrightarrow_{\pi^t_{-w}}$ is indeed Weakly Normalizing, we will present a reduction strategy $\mathcal{RS}$ which reduces every proof net P in $=_\mathcal{S}$-NF to its normal form N:*

1. *Contract a cut-node at the highest level l, i.e. a cut link which is nested inside the largest number of boxes: $P \rightarrow Q$*
2. *Reduce Q to $=_\mathcal{S}$-NF: $Q \twoheadrightarrow_{=_\mathcal{S}} Q'$*
3. *If Q is not in its normal form N: $\mathcal{RS}(Q)$*

Obviously, the weight of the typed proof net decreases over the first step, because we contract one of the following redexes:

tensor-par: $\mathcal{G}'(Q) = \mathcal{G}'(P) - 1 \Rightarrow \mathcal{G}'(P) > \mathcal{G}'(Q)$

axiom-cut: $\mathcal{G}'(Q) = \mathcal{G}'(P) - 1 \Rightarrow \mathcal{G}'(P) > \mathcal{G}'(Q)$

cut-axiom: $\mathcal{G}'(Q) = \mathcal{G}'(P) - 1 \Rightarrow \mathcal{G}'(P) > \mathcal{G}'(Q)$

After the first step of $\mathcal{RS}$ is executed, new x-redexes may have been formed at level l. However, contracting these redexes until $=_{\mathcal{S}}$-NF is reached again (step 2), never results in the duplication of cut-nodes. This fact becomes clear if once we realize that a cut at level i can only duplicate a cut at level i+1, but in selecting the cut-node at the highest *level for contraction (step 1) we eliminate the possibility that any cut exists beyond level i.*

If no new $=_{\mathcal{S}}$-redexes have been formed after the first step, Q′ like Q is in $=_{\mathcal{S}}$-NF and can be used as input for a recursive call to $\mathcal{RS}$.

$\square$

Lemma 6.14 ($\rightarrow_m$ is SN) *By Lemma 6.2, we may conclude from Lemma 6.10, Lemma 6.12 and Lemma 6.13 combined with the Lifting Lemma 6.6 that $\rightarrow_m$ is indeed SN.*

$\square$

We have now proven the first of the three goals we set ourselves at the beginning of this proof of SN for $\longrightarrow_{\pi^t}$. We will now proceed with the second one: $\rightarrow_w$ can always be postponed.

Lemma 6.15 (Postponing Weakening) *Applying a weakening step can always be postponed: $(\rightarrow_w \circ \longrightarrow_{\pi^t_{-w}}) \subseteq (\longrightarrow_{\pi^t_{-w}} \circ \twoheadrightarrow_{\pi^t})$. This easy to see, because the weakening rule can only create weakening redexes.*

$\square$

Proof 6.1 ($\longrightarrow_{\pi^t}$ is SN) *By Lemma 6.14 and the Lifting Lemma 6.6 we know $\longrightarrow_{\pi^t_{-w}}$ is SN. Furthermore the weakening rule is obviously SN*[8] *and can always be postponed until after all the $\longrightarrow_{\pi^t_{-w}}$-steps have been done (Lemma 6.15). So by proving Lemma 6.1, we have also proven SN for $\longrightarrow_{\pi^t}$.*

$\square$

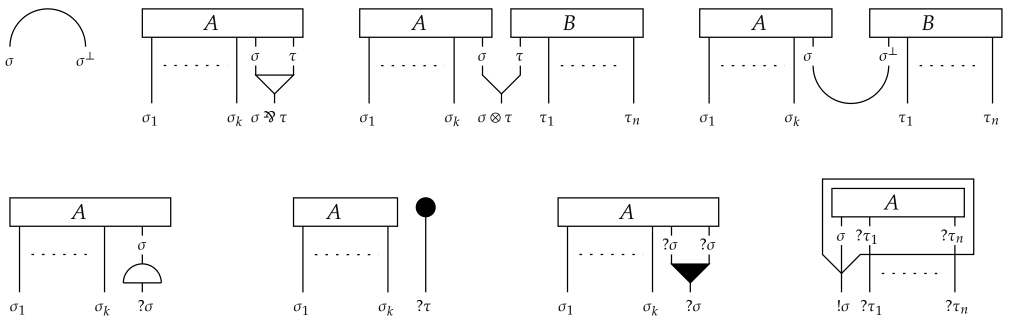
[8]every weakening step deletes a node

7 Typed Coinductive Proof Nets

7.1 Definition TCPN

Definition 7.1 (Coinductive Typed Proof Net) *A typed coinductive proofnet is a directed coinductive proofnet for which a type is assigned to every port of the net.* $\Gamma : \sigma$ *means that the typed coinductive proofnet* Γ *has a global output port labeled* σ*.*

If C is a coinductive proof net. then it is of one of the following forms:



σ $\sigma^\perp$
A σ τ σ_1 σ_k $\sigma ⅋ \tau$
A B σ τ σ_1 σ_k $\sigma \otimes \tau$ τ_1 τ_n
A B σ $\sigma^\perp$ σ_1 σ_k τ_1 τ_n
A σ σ_1 σ_k $?\sigma$
A σ_1 σ_k $?\tau$
A $?\sigma$ $?\sigma$ σ_1 σ_k $?\sigma$
A σ $?\tau_1$ $?\tau_n$ $!\sigma$ $?\tau_1$ $?\tau_n$

where A or A and B are coinductive proof nets again with their sets of ports defined as:

axiom: $\mathsf{prt}(C) = \{a : \sigma, b : \sigma^\perp\}$ *for some* $a : \sigma$ *and* $b : \sigma^\perp$

par: $\mathsf{prt}(C) \cup \{a : \sigma, b : \tau\} = \mathsf{prt}(A) \cup \{c : \sigma ⅋ \tau\}$

tensor $\mathsf{prt}(C) \cup \{a : \sigma, b : \tau\} = \mathsf{prt}(A) \cup \mathsf{prt}(B) \cup \{c : \sigma \otimes \tau\}$

cut: $\mathsf{prt}(C) \cup \{a : \sigma, b : \sigma^\perp\} = \mathsf{prt}(A) \cup \mathsf{prt}(B)$

dereliction: $\mathsf{prt}(C) \cup \{a : \sigma\} = \mathsf{prt}(A) \cup \{c :?\sigma\}$

weakening: $\mathsf{prt}(C) = \mathsf{prt}(A) \cup \{b :?\tau\}$

fan: $\mathsf{prt}(C) \cup \{a :?\sigma, b :?\sigma\} = \mathsf{prt}(A) \cup \{c\}$

box: $\mathsf{prt}(C) = \mathsf{prt}(A)$*, where every input port of C with type* $?\tau$ *will have type* $?\tau$ *in A and the output port of C with type* $!\sigma$ *will have type* σ *in A.*

where $\mathsf{prt}(x)$ *is a function from proof nets to sets of ports.*

Definition 7.2 (Typed Proof Nets) *The typed proof net associated with an typed inductive proof net P is the graph* $\mathbb{G}(P)$*. This graph is constructed in a way similar to the one in Definition 4.2 only now every port is assigned a label (type).*

Definition 7.3 (Typed Coinductive Proof Net)

7.2 Properties of TCPN

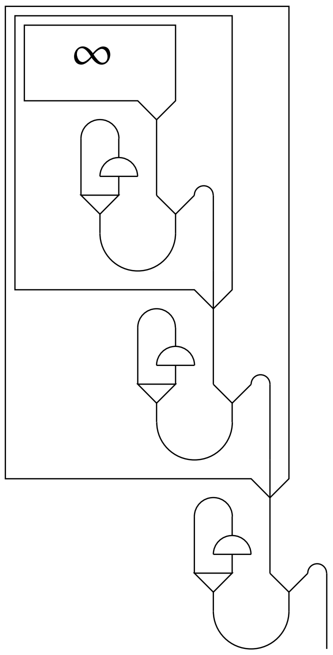
Now what about the properties of these typed coinductive proof nets? They are of course CR, because *TCPN* ⊆ *CPN* and we already knew that *CPN* is CR. But for the inductive equivalent of typed coinductive proof nets we found an additional property: strong normalization or termination. Unfortunately, in extending the set of finite proof nets with infinite ones, we also created a few proof nets with infinitely many redexes. It should be obvious that these proof nets will not be stongly normalizing and therefore typed coinductive proof net reduction $\longrightarrow^{co}_{\pi^t}$ can not be strongly normalizing either.

However, there exists a counterpart of strong normalization which is able to handle infinite objects: head normalization (HN). The idea behind head normalization is that one should only contract those redexes that are necessary. This approach should lead to the head normal form of an object; it should return stable information which is not subject to change anymore even though not all redexes are contracted yet.

But we will see that HN too, is not a property of $\longrightarrow^{co}_{\pi^t}$. Our counterexample comes from the coinductive λ-calculus [Joa01]:

$$CE = (\lambda x.x)(\lambda x.x)(\lambda x.x)(\lambda x.x)(\lambda x.x)(\lambda x.x)\ldots\infty$$

We see that whatever reduction we apply to *CE*, the result will be *CE* again even though this λ-term is perfectly typable. Now since every typed λ-term can be mapped onto a typed proof net, we can constuct a proof net counterpart of this example:



8 Conclusions

In this thesis we investigated (co)inductive proof nets. We have presented a proof of confluence for both inductive and coinductive proof net reduction ($\longrightarrow_{\pi}$ and $\longrightarrow_{\pi}^{co}$) and a proof of strong normalization for typed inductive proof net reduction $\longrightarrow_{\pi^t}$. Counterexamples were given for strong normalization of type-free inductive proof net reduction $\longrightarrow_{\pi}$, for strong normalization of typed coinductive proof net reduction $\longrightarrow_{\pi^t}^{co}$ and for head normalization of typed coinductive proof net reduction $\longrightarrow_{\pi^t}^{co}$.

So even though we established confluence for typed coinductive proof net reduction, the rewrite system $\mathcal{CPN} = (CP, \longrightarrow_{\pi^t}^{co})$ is far from ideal. We would like to have at least some control on the termination of a rewriting sequence. Therefore future work includes exploring the possibility of a standardization procedure, which would always reduce a proof net to its (head) normal form, given that the proof net has one.

Futhermore, future work could include finding (if there is one) an apt switching criterion for the exponential fragment, the additive fragment and of course for the coinductive version of proof nets.

And in a broader perspective, one could think of investigating the cognitive claims about proof nets made in [Joh98] and [Mor00]. Maybe these claims can even be extended and teach us something not only about the processes involved in analyzing natural language, but also about the processes involved in reasoning . . .

References

[Abr90] Samson Abramsky, *The lazy lambda-calculus*, In D. Turner, editor, Declarative Programming, Academic Press, 1990.

[AG98] A. Asperti and S. Guerrini, *The optimal Implementation of Functional Programming Languages*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1998.

[Bet00] Inge Bethke, Jan Willem Klop, Roel de Vrijer, *Descendants and Origins in Term Rewriting*, Information and Computation 159, 2000.

[DR89] Vincent Danos and Laurent Régnier, *The structure of multiplicatives*, Archive for Mathematical logic 28:181-203, 1989.

[Dan90] Vincent Danos, *La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du lambda-calcul)*, PhD thesis Université Paris VII, 1990.

[Gir87] Jean-Yves Girard, *Linear Logic*, Theoretical Computer Science 50, 1987.

[Gir95] Jean-Yves Girard, *Linear Logic, Its Syntax and Semantics*, Advances in Linear Logic (Proc. of the Workshop on Linear Logic, Cornell University, June 1993), Cambridge University Press, 1995.

[GR96] Philippe de Groote and Christian Retoré, *On the Semantic Readings of Proof Nets*, Formal Grammar, FoLLI, 1996.

[Jac97] Bart Jacobs and Jan Rutten, *A Tutorial on (Co)Algebras and (Co)Induction*, Bulletin of the European Association for Theoretical Computer Science 62, 1997.

[Joa01] Felix Joachimski, *Reduction Properties of* Π *IE-Systems*, PhD Thesis LMU München, 2001.

[Joh98] Mark Johnson, *Proof nets and the complexity of processing center embedded constructions*, JoLLI 7:433-447, 1998.

[Joi93] Jean-Baptiste Joinet, *Etude de la normalisation du calcul des séquents classique à travers la logique linéaire*, PhD thesis Université Paris VII, 1993.

[Laf95] Yves Lafont, *From Proof-Nets to Interaction Nets*, Advances in Linear Logic, Cambridge University Press, 1995.

[Mac71] Saunders MacLane, *Categories for the Working Mathematician*, Springer-Verlag 1971.

[Mel02] Paul-André Melliès, *Residual Theory Revisited*, In Proceedings of Conference on Rewriting Techniques and Applications, Copenhague, 2002 (to appear).

[Mid97] Aart Middeldorp, *Call by Need Computations to Root-Stable Form*, Symposium on Principles of Programming Languages, 1997.

[Moo02] Richard Moot, *Proof Nets for Linguistic Analysis*, PhD Thesis Utrecht University 2002.

[Mor00] Glyn Morrill, *Incremental processing and acceptability*, Computational Linguistics 26(3):319-338, 2000.

[Ken99] Richard Kennaway, Vincent van Oostrom, Fer-Jan de Vries, *Meaningless Terms in Rewriting*, Journal of Functional and Logic Programming 1, 1999.

[Oos01] Vincent van Oostrom, *Net-calculus*, `http://www.phil.uu.nl/~oostrom/`, 2001.

[Pey87] Simon L. Peyton Jones, *The Implementation of Functional Programming Languages*, 1987.

[Plu98] Detlef Plump, *Term graph rewriting*, Handbook of Graph Grammars and Computing by Graph Transformation, volume 2, World Scientific, 1998.

[vR96] Femke van Raamsdonk, *Confluence and Normalisation for Higher-Order Rewriting*, PhD thesis Vrije Universiteit, Amsterdam, 1996.

[Reg92] Laurent Régnier, *Lambda-calcul et résaux*, PhD thesis Université Paris VII, 1992.

[Sor98] Morten Heine B. Sorensen and Pawel Urzyczyn, *Lectures on the Curry-Howard Isomorphism*, DIKU-rapport 98/14, Technical Report, Department of Computer Science, University of Copenhagen, 1998.

[Ter02] Terese, *Term Rewriting Systems*, Cambridge University Press, 2002 (to appear).