

Coinductive Proof Nets

Laura Korte¹ and Vincent van Oostrom²

¹ University of Edinburgh, Edinburgh, United Kingdom,
Laura.Korte at ed.ac.uk,
WWW home page: <http://homepages.inf.ed.ac.uk/s0235396/>

² Universiteit Utrecht, Utrecht, The Netherlands,
Vincent.vanOostrom at phil.uu.nl,
WWW home page: <http://www.phil.uu.nl/~oostrom/>

Abstract. In this article we will introduce the coinductive counterpart of inductively defined proof nets – coinductive proof nets – and investigate their properties. We will prove confluence by constructing a new confluence proof for inductive proof nets, which doubles as a confluence proof for coinductive proof nets. We will also present a new strong normalisation proof for typed inductive proof nets, and a counter example against both strong normalisation and head normalisation for typed coinductive proof nets.

1 Introduction

Proof Nets were first introduced by Girard as part of his definition of linear logic [[Gir87],[Gir95]]. Linear logic of course, is best known for its implications on resource-awareness, because all resources are dealt with explicitly. In Girard’s original theory, typed lambda terms can be decomposed into proof nets. These proof nets, like their accompanying logic, handle resources explicitly. In practice, this means that resource duplication is controlled.

Later, Danos and Régnier recognised a relation between proof nets and the untyped lambda calculus in [Dan90] and [Reg92]. The relation they found, is one of implementation: any lambda term can be implemented by a proof net. The fact that proof nets allows a more efficient reduction of lambda terms, than beta-reduction, is a compelling argument in favour of this implementation. The reason proof nets allow a more efficient reduction, is because duplication is controlled. In particular, plain beta reduction can create an arbitrary number of copies of a term (e.g. $(\lambda x.xxxx)f$ creates four additional copies of f), whereas proof net reduction can create only one (per step).

All of the above research was conducted in the inductive domain, discussing only proof nets of finite size. Recently, Joachimski investigated the coinductive lambda calculus in [Joa01]. His research together with the known relation between proof nets and the lambda calculus, was the main incentive for the authors to investigate the properties of coinductive proof nets.

Conventions and Preliminaries In this paper, we will use the following conventions. First of all, we will use ‘confluence’, ‘Church–Rosser Property’ and ‘CR’ interchangeably. The same holds for ‘Subject Reduction Property’, ‘SR’ and ‘Preservation’. Furthermore, we will abbreviate ‘Strongly Normalisation’ as ‘SN’ and ‘Weak Church–Rosser Property’ as ‘WCR’. For a brief description of these terms, please refer to Appendix A. Finally, for the confluence proof in this article, we will make use of Newman’s Lemma, which states that SN and WCR implies CR. A proof of Newman’s Lemma can be found in Terese [Ter03].

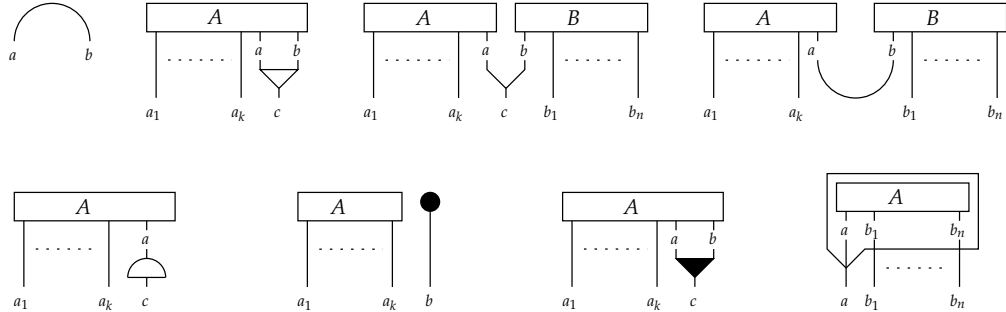
Outline. In section 2, we discuss inductively defined proof nets and their properties. It includes a confluence proof, and a strong normalisation proof. In section 3, we discuss coinductively defined proof nets and their properties. It includes a confluence proof, and a counter example against strong normalisation and head normalisation. Section 4 contains conclusions and future work.

2 Inductive Proof Nets

We present the inductive definition of proof nets (see [Laf95]) and the linear graphs they define.

2.1 Definition

Definition 1 (Inductive Proof Net (IPN)). *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. Together they are known as MELL: Multiplicative Exponential Linear Logic. Note our system does not include the *additive* fragment (\oplus and $\&$). Our motivation for investigating coinductive proof nets stems from the implementation relation that holds between the lambda calculus and proof nets.

In particular: the multiplicative exponential fragment of proof nets, because all lambda terms can be implemented in this particular fragment.

Note that ‘net-calculus’ is actually a better name for proof nets the way they are defined above, because no type (or proposition) has been associated with the term (or proof) yet. However, we will continue to use ‘proof nets’ (typed and untyped) in the remainder of this paper, to emphasize the close relationship between proof nets and the lambda calculus.

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

Definition 2 (Proof Net). *The proof net associated with an inductive proof net P is the linear graph $G(P)$. The signature of this linear graph consists of the symbols $\blacktriangledown : 2, 1$; $\sqsupset : 1, 1$; $\bullet : 0, 1$; $\nabla : 2, 1$; $\vee : 2, 1$; $\cap : 0, 2$; $\cup : 2, 0$ 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 \sqsupset -node and one edge, going from a port of N to \sqsupset , 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 \bullet -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 ∇ -node and two edges, both going from (a different) a port of N to ∇ , have been added.*

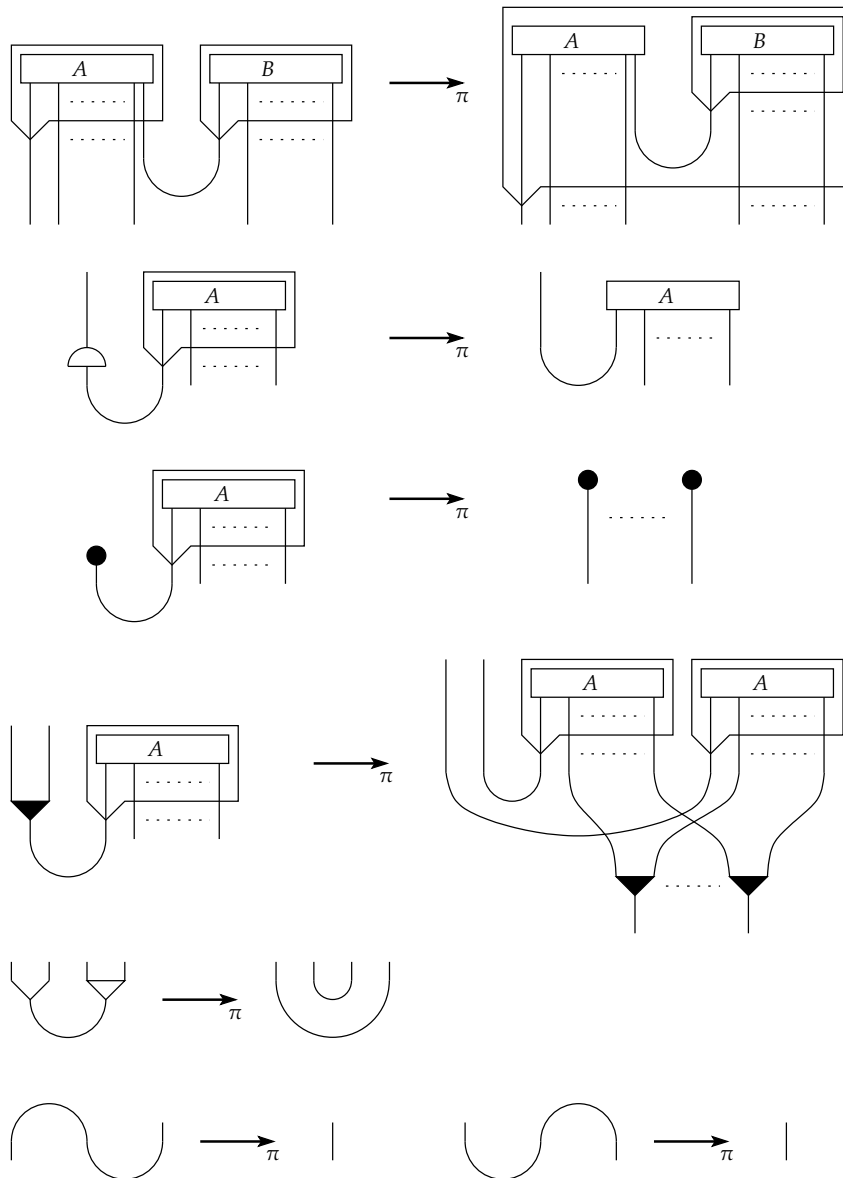
tensor: *If the proof nets N and M are associated with the inductive proof nets A and B , then the proof net associated with their tensor-union is the (disjunct) union of N and M to which a \vee -node and two edges, one going from a port of N to \vee and one going from a port of M to \vee , have been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

We present a rewrite relation on proof nets \longrightarrow_{π} , i.e. it is a rewrite relation defined on linear graphs.

Definition 3 (Proof Net Reduction \longrightarrow_{π}).



where A_{rhs} and B_{rhs} are descendants 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.

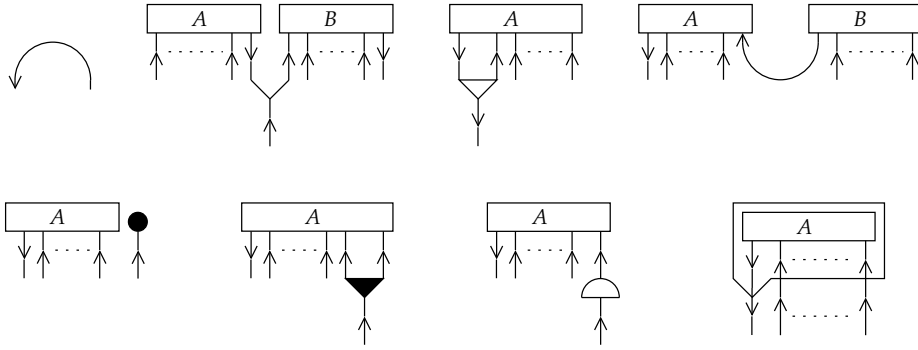
2.2 Properties

Confluence? The inductively defined lambda calculus is confluent, so naturally, we would like the proof nets that implement it to have this same property.

Proposition 4. *Proof Net Reduction (\longrightarrow_{π}) has the Church–Rosser property.*

Due to space limitations, the proofs of most lemmas can be found in Appendix B. These lemmas are marked with (*). For reasons that will become clear later on, we will prove confluence for *directed* proof nets instead of for ordinary proof nets.

Definition 5 (Directed Proof Net). *A directed proof net is an inductive proof net 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 confluence proof we are presenting, *clusters* and *developments of clusters* (see [Ter03]) will be used to define an apt notion of multi-step reduction. Unfortunately, one of the requirements of a confluence proof using developments is SN for those developments. This implies that we cannot define our developments on the entire rewrite relation \longrightarrow_{π} , because it includes both increasing and decreasing rules. Therefore, our proof strategy consists of the following:

First we will partition the proof net reduction relation \longrightarrow_{π} into the sub-relations \longrightarrow_{ω} and $\longrightarrow_{(\psi, \sim)}$. The former will be a strongly normalizing subset containing box–dereliction, box–fan, box–weakening and box–box for which we will define developments. The latter will contain the remaining rules: tensor–par, cut–axiom and axiom–cut. In the remainder of this paper, we will refer to the tensor–par rule as the ψ -rule, and to the cut–axiom and axiom–cut rules as the \sim -rules.

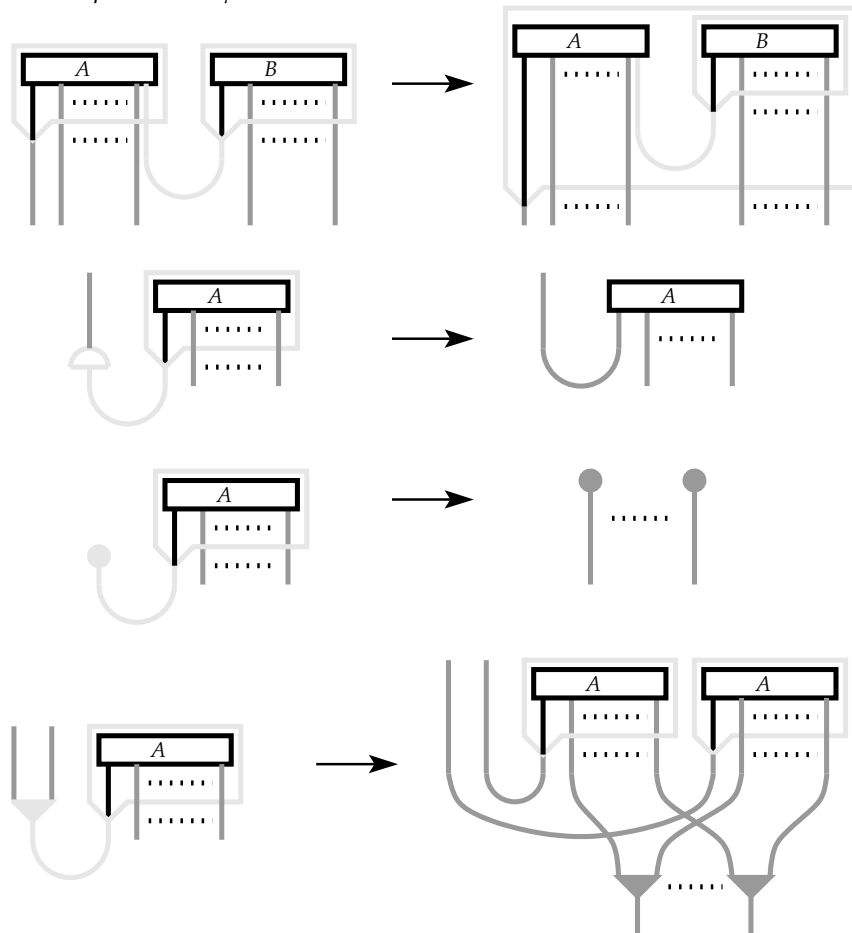
In order to prove that \longrightarrow_{π} is confluent, we will first prove independently, that both developments defined on \longrightarrow_{ω} , and a our notion of multi- (ψ, \sim) -step reduction – as will be defined later – are confluent. We will then combine these two separate proofs into a single proof of confluence for \longrightarrow_{π} .

According to the proof strategy we just outlined, our first objective is to prove confluence for developments defined on \longrightarrow_{ω} . After the definitions of *inductive cluster*, *residue* and *development*, a proof follows.

Definition 6 (Inductive Cluster). An inductive cluster is a subgraph, containing only box, fan, dereliction, weakening and cut nodes, of the graph of a proof net. In the figures, if a node is light grey, it is part of a cluster. If it is dark grey, whether or not it is part of a cluster depends on whether or not the node is the descendant of a light grey node.

Definition 7 (Residue). The residue of a cluster C after a rewrite-step ψ :

1. No overlap: $C - \psi = \text{desc}(C)$
where $\text{desc}(C)$ 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 8 (Development). A development of a cluster C is defined as a step $\psi \in C$ followed by the development of $C - \psi$

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

Next, we will need to prove SN for inductive developments. Our decreasing measure will be a lexicographic product of three orders: $\langle \text{fd}, \text{bd}, n \rangle$, which will be explained in more detail later. A triple was chosen, because it allows us to reason about several measures of interest, instead of just one. In order to prove that $\langle \text{fd}, \text{bd}, n \rangle$ decreases, we need prove one of the following for every rewrite step:

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

where lhs stands for ‘left-hand side’ and rhs stands for ‘right-hand side’. The first measure fd , which is short for *fan depth*, expresses the idea that in applying the box–fan rule, the fans involved are being pushed through the box. We will prove 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 an order (*behind*) on the nodes which decides whether or not a certain node can interact with another one. However, if inductive cluster can contain or produce cycles, such an order would be useless. For this purpose, we prove the following lemma:

Lemma 9 (*). *A directed inductive cluster does not contain cycles.*

Lemma 10 (*). *Inductive clusters without cycles have the Subject Reduction property.*

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 11 (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.

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 $\langle \text{fd}, \text{bd}, n \rangle$, 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 12 (Level of Node).

The level of a node N is recursively defined as:

- $\text{bl}(N) = 1$
if N is not inside a(nother) box
- $\text{bl}(N) = \text{bl}(B) + 1$
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 $\langle \text{fd}, \text{bd}, n \rangle$ 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 13 (*). *Developments are Strongly Normalizing*

In order to apply *Newman's Lemma* to prove that developments have the CR property, we also need to prove that developments have the WCR property.

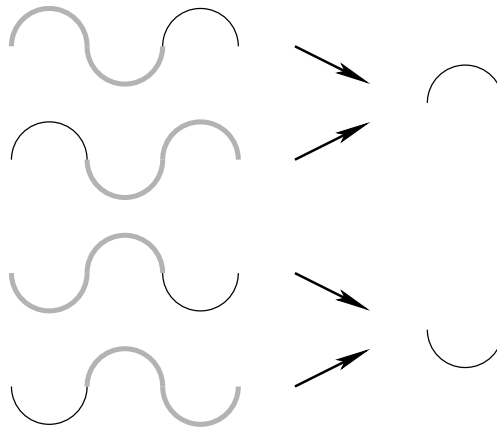
Lemma 14. *Developments have the Weak Church-Rosser property*

We are now in a position to prove confluence for developments.

Lemma 15 (*). *Developments have 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 multi-step proof reduction (\dashrightarrow) for the rest of the rules.

Definition 16 (Multi-step reduction (ψ, \sim)-step: \dashrightarrow). *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 arbitrary number of \sim -steps at the same time. These observations result in the following definition of a multi- (ψ, \sim) -step (\dashrightarrow):

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

Next, we need to prove that the union of our two notions of multi-step reductions ($\dashrightarrow \cup \dashrightarrow$) is CR as well. In order to do so, we will first prove that this union has the diamond property.

Lemma 17 (*). $(\dashrightarrow \cup \dashrightarrow)$ represented by \dashrightarrow has the Diamond Property.

Lemma 18 (*). $(\dashrightarrow \cup \dashrightarrow)$ represented by \dashrightarrow is CR .

For our purpose of proving confluence, we need one more lemma ³:

Lemma 19 (*). $\longrightarrow_{\pi} \subseteq \dashrightarrow \subseteq \longrightarrow_{\pi}^*$

Now we have arrived at the point where we can prove Proposition 4 confluence for inductive proof nets.

Proof (Proposition 4). We have already proven that \dashrightarrow (or $\dashrightarrow \cup \dashrightarrow$) has the Church–Rosser property in Lemma 18 and since $\longrightarrow_{\pi} \subseteq \dashrightarrow \subseteq \longrightarrow_{\pi}^*$ by Lemma 19, we may conclude that the transitive reflexive closure of \dashrightarrow is equal to the transitive reflexive closure of \longrightarrow_{π} : $\dashrightarrow^* = \longrightarrow_{\pi}^*$. We may now also derive Church–Rosser for \longrightarrow_{π} from the proof of Church–Rosser for \dashrightarrow . \square

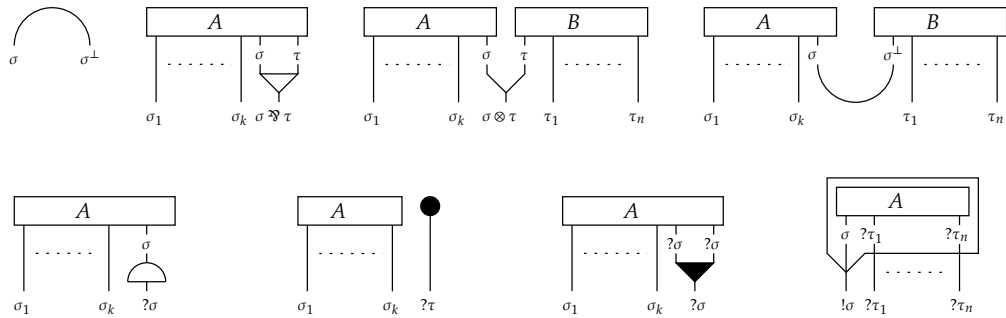
Strong Normalisation? Obviously, untyped proof net reduction cannot be strongly normalising, because untyped lambda reduction is not.

However, proof nets can be typed with linear logic formulas, using the following definition:

Definition 20 (Typed Inductive Proof Net (TIPN)). A typed inductive proof net is a directed proof net for which a type is assigned to every port of the net. $C : \sigma$ means that the typed proof net C has a global output port labelled σ .

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

³ Exercise 1.3.1 in [Ter03]



Typed inductive proof nets define the class of typed proof nets in the following manner:

Definition 21 (Typed Proof Nets). *The typed proof net associated with an typed inductive proof net P is the linear graph $G(P)$. This linear graph is constructed in a way similar to the one in Definition 2 only now every port is assigned a label (type).*

A proof of strong normalisation for typed inductive proof nets can be found in Appendix C.

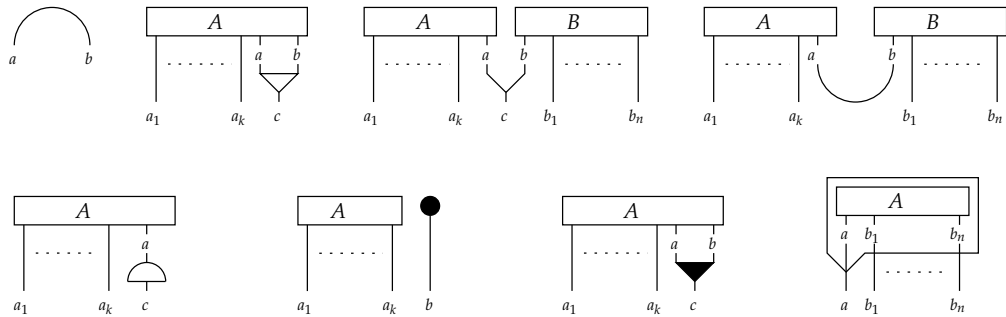
Proposition 22. *Typed Proof Net Reduction \longrightarrow_{π^i} is SN.*

Proof. See Appendix C. □

3 Coinductive Proof Nets

3.1 Definition

Definition 23 (Coinductive Proof Net (CPN)). *If C is a coinductive proof net, then it is of one of the following forms ⁴:*



⁴ Note that the picture used in this definition is identical to the one used in Definition 1. This corresponds to the intuition that coinductive proof nets differ from inductive proof nets only in the direction we read the grammar: it is the difference between composition and decomposition.

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

axiom: $\text{prt}(C) = \{a, b\}$ for some a and b
par: $\text{prt}(C) \cup \{a, b\} = \text{prt}(A) \cup \{c\}$
tensor $\text{prt}(C) \cup \{a, b\} = \text{prt}(A) \cup \text{prt}(B) \cup \{c\}$
cut: $\text{prt}(C) \cup \{a, b\} = \text{prt}(A) \cup \text{prt}(B)$
dereliction: $\text{prt}(C) \cup \{a\} = \text{prt}(A) \cup \{c\}$
weakening: $\text{prt}(C) = \text{prt}(A) \cup \{b\}$
fan: $\text{prt}(C) \cup \{a, b\} = \text{prt}(A) \cup \{c\}$
box: $\text{prt}(C) = \text{prt}(A)$

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

Like inductive proof nets, coinductive proof nets are merely a means to define proof nets. A definition of the class of proof nets defined by the coinductive definition follows.

Definition 24 (Proof Net). The proof net associated with a coinductive proof net P is the linear graph $\mathbb{G}(P)$. The signature of this linear graph consists of the symbols $\blacktriangledown : 2, 1$; $\circlearrowleft : 1, 1$; $\bullet : 0, 1$; $\nabla : 2, 1$; $\vee : 2, 1$; $\cap : 0, 2$; $\cup : 2, 0$ 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 $\text{fan}_{v_i}(A)$, 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 $\text{box}_{v_i}(A)$, 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 $\mathbb{G}(v_i) = \circlearrowleft$, is associated with the coinductive proof net $\text{der}_{v_i}(A)$, 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 $\mathbb{G}(v_i) = \bullet$, is associated with the coinductive proof net $\text{weak}_{v_i}(A)$, 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 $\mathbb{G}(v_i) = \nabla$, is associated with the coinductive proof net $\text{par}_{v_i}(A)$, then the proof net associated with A is $\langle V, E \rangle$.

tensor: If the proof net $\langle V \cup v_i, E \cup \{0, ((v_j, n), (v_i, 1)), 1, ((v_k, m), (v_i, 2))\} \rangle$ where $\mathbb{G}(v_i) = \vee$, is associated with the coinductive proof net $\text{tensor}_{v_i}(A, B)$, 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 $\mathbb{G}(v_i) = \nabla$, is associated with the coinductive proof net axiom

cut: If the proof net $\langle V \cup v_i, E \cup \{0, ((v_j, n), (v_i, 1)), 1, ((v_k, m), (v_i, 2))\} \rangle$ where $\mathbb{G}(v_i) = \cup$, is associated with the coinductive proof net $\text{cut}_{v_i}(A, B)$, 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 $\text{fan}_{v_i}(A)$ 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 $\pi_B(C) = \{b \mid \langle 1, b \rangle \in C\}$.

The reduction relation on proof nets as defined in Definition 3 remains the same.

3.2 Properties

CPNs are a superset of the IPNs, which implies that any property we proved for the latter, may not hold for the former. We investigate this further.

Confluence? Confluence for the inductively defined lambda calculus can be proven using a notion of multi-step reduction. This proof does not hold for the coinductively defined lambda calculus, because it depends on the fact that there are only finitely many of these multi-steps. In the coinductive lambda calculus there can be infinitely many, as the following example shows:

$$\begin{array}{ccc}
 [\lambda f.f f f f f \dots](\lambda x.x]g) & \rightarrow & [\lambda f.f f f f f \dots]g \\
 \downarrow & & \downarrow \\
 (\lambda x.x]g)(\lambda x.x]g)(\lambda x.x]g)(\lambda x.x]g)(\lambda x.x]g)\dots \rightarrow^\infty & & g g g g g \dots
 \end{array}$$

If we choose the path of innermost reductions first, we end up at a normal form in exactly two steps. If on the other hand, we had chosen the path of outermost reductions first instead, we would have ended up with an infinite amount of redexes as a result of the initial reduction.

However, in our proof of confluence for IPNs, we did not at any point use their finiteness. So the confluence proof we presented for IPNs, is indeed also a confluence proof for CPNs.

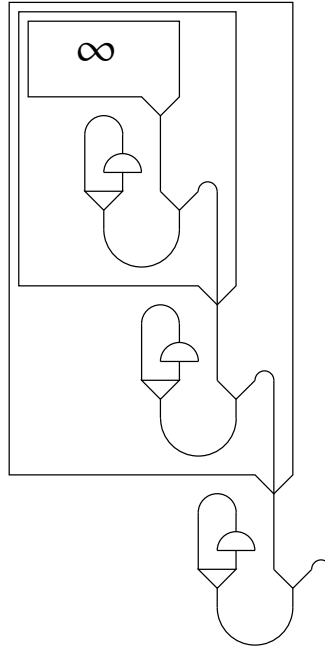
Proposition 25. *Coinductive Proof Net Reduction (\rightarrow_π^{co}) has the Church–Rosser property.*

Proof. See the confluence proof in Section 2.2. □

Strong Normalisation? Unfortunately, we cannot apply the same trick to our strong normalisation proof for TIPNs, for the simple reason that it *does* use their finiteness. In particular, we are unable to prove weak normalisation for TIPN reduction without weakening (Lemma 54), due to the existence of Typed Coinductive Proof Nets (TCPNs) with infinitely many redexes

Proposition 26. *TCPN is not strongly normalizing*

Proof. Consider the following proof net:



Reduction will never terminate for this proof net, yet it is perfectly typable. It represents the (typable) lambda term CE , which also has an infinite amount of redexes:

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

□

A result like this, is of course to be expected from a system which contains infinite objects. More worrisome is the fact that the proof net for representing CE , also seems to be a counter example for strong normalisation's counterpart in the infinite domain: head normalisation (HN).

Proposition 27. *TCPN is not head normalizing.*

Proof. Consider the proof net shown of Proposition 26 once more. Regardless of the redex we contract, it will never yield any stable information. □

On the other hand, the lambda term CE does not have a head normal form either. In the lambda calculus, terms like CE are referred to as *meaningless* [Ken99].

4 Conclusions

We have shown that both inductive and coinductive proof net reduction is confluent, and that typed inductive proof net reduction is strongly normalizing. We

also proved that typed coinductive proof net reduction is not strongly normalising or head normalising. These conclusions are summarised in the following two tables.

	CR	SN
IPN	✓	✗
TIPN	✓	✓

	CR	SN	HN
CPN	✓	✗	✗
TCPN	✓	✗	✗

Properties of Inductive Proof Nets Properties of Coinductive Proof Nets

Future Work. Our normalisation results for coinductive proof nets were somewhat disappointing, in that we have thus far been unable find an alternative for the strong normalisation we proved for typed inductive proof net reduction. Therefore future work includes exploring the possibility of a standardisation procedure for typed coinductive proof net reduction. Such a procedure would always reduce a proof net to its (head) normal form, provided that it has one, and would give us at least some control on the termination of a proof nets reduction sequence. However, in order to define such a procedure, we will first need answer questions like: 1) is there an appropriate notion of head normal form for proof nets, and 2) if so, what is it?

Acknowledgements. The research for this paper was carried out at the University of Utrecht as part of the MSc thesis of Laura Korte, supervised by Vincent van Oostrom. Laura Korte is currently supported by a studentship from the Laboratory for Foundations of Computer Science of the University of Edinburgh. Vincent van Oostrom is a lecturer at the University of Utrecht.

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, Copenhagen, 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, *Eventually Increasing*, <http://www.phil.uu.nl/~oostrom/publication/rewriting.html>, 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 réseaux*, 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.
- [Ter03] Terese, *Term Rewriting Systems*, Cambridge Tracts in Theoretical Computer Science, Vol. 55, Cambridge University Press, 2003.

A Preliminaries

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

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*.

Weak Church–Rosser $a \in A$ is *locally confluent* if $\forall b, c \in A((c \leftarrow a \rightarrow b) \Rightarrow \exists d \in A(c \twoheadrightarrow d \leftarrow b))$. The rewrite relation \rightarrow is *locally confluent* if every $a \in A$ is *locally confluent*.

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

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 and 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 $\langle 0, 0 \rangle$.

Graph Rewriting Rule A graph rewriting rule $L \rightarrow R$ consists of a graph L , a graph R and a mapping from the vertices of L to the vertices of R . 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' .

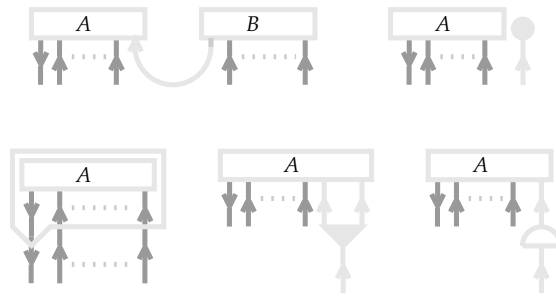
Graph Rewriting System (GRS) A graph rewriting system is a finite set of graph rewriting rules.

B Proofs of Lemmas used in the Confluence Proof for (Co)Inductive Proof Nets

Proof (Lemma 9). The general idea is that directed inductive clusters do not contain cycles, because they have no *ceiling*. By *ceiling*, we mean a link in the proof net where an upward arrow turns to go downwards. 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 follows.

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:

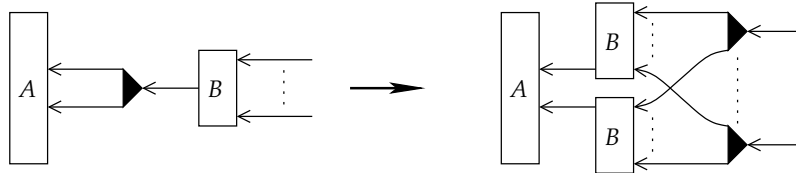


□

Proof (Lemma 10). 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.

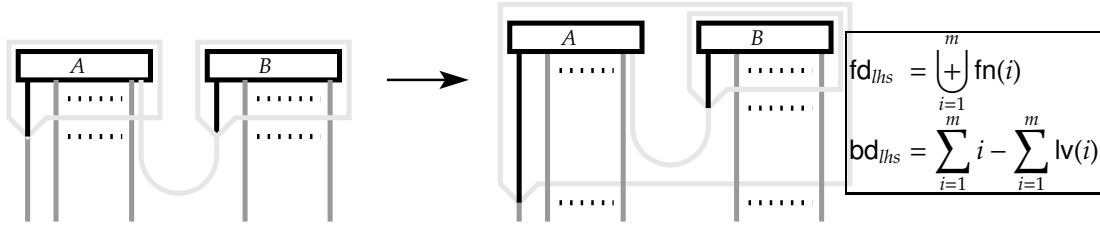
box-weakening: Trivial, the paths are cut off.

□

Proof (Lemma 13). 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 28. The weight \mathcal{G} of a cluster C is a triple $\langle \text{fd}, \text{bd}, n \rangle$, where fd is the multiset $\biguplus_{i=1}^n \text{fn}(i)$, $\text{fn}(x)$ is the number of fan-nodes box x is behind, bd is $\sum_{i=1}^n i - \sum_{i=1}^n \text{lv}(i)$, $\text{lv}(x)$ is the level of box x and n is the number of boxes in the cluster.

Case 1

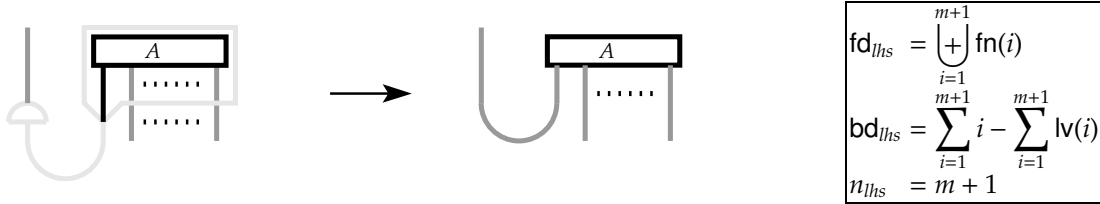


$$\text{fd}_{rhs} = \biguplus_{i=1}^m \text{fn}(i) = \biguplus_{i=1}^m \text{fn}(i) = \text{fd}_{lhs} \quad \Rightarrow \text{fd}_{rhs} = \text{fd}_{lhs}$$

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

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

Case 2



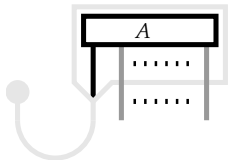
$$\text{fd}_{rhs} = \biguplus_{i=1}^m \text{fn}(i) \leq \biguplus_{i=1}^{m+1} \text{fn}(i) = \text{fd}_{lhs} \quad \Rightarrow \text{fd}_{rhs} \leq \text{fd}_{lhs}$$

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

$$n_{rhs} = m < m + 1 = n_{lhs} \quad \Rightarrow n_{rhs} < n_{lhs}$$

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

Case 3

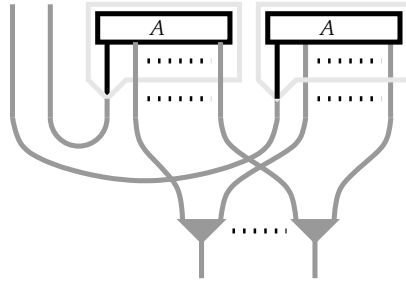
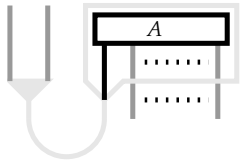


$$\begin{array}{l} \text{fd}_{lhs} = \bigcup_{i=1}^{m+1} \text{fn}(i) \\ \text{bd}_{lhs} = \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \text{lv}(i) \\ n_{lhs} = m + 1 \end{array}$$

$$\begin{array}{l} \text{fd}_{rhs} = \bigcup_{i=1}^m \text{fn}(i) \leq \bigcup_{i=1}^{m+1} \text{fn}(i) = \text{fd}_{lhs} \Rightarrow \text{fd}_{rhs} \leq \text{fd}_{lhs} \\ \text{bd}_{rhs} = \sum_{i=1}^m i - \sum_{i=1}^m \text{lv}(i) \leq \sum_{i=1}^{m+1} i - \sum_{i=1}^{m+1} \text{lv}(i) = \text{bd}_{lhs} \Rightarrow \text{bd}_{rhs} \leq \text{bd}_{lhs} \\ n_{rhs} = m < m + 1 = n_{lhs} \Rightarrow n_{rhs} < n_{lhs} \end{array}$$

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

Case 4



$$\text{fd}_{lhs} = \bigcup_{i=1}^{m_{in}} \text{fn}(i) \uplus \bigcup_{i=1}^{m_{out}} \text{fn}(i) \uplus \text{fn}(b)$$

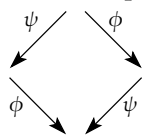
$$\text{fd}_{rhs} = \bigcup_{i=1}^{m_{in}} (\text{fn}(i) - 1) \uplus \bigcup_{i=1}^{m_{in}} (\text{fn}(i) - 1) \uplus \bigcup_{i=1}^{m_{out}} \text{fn}(i) \uplus (\text{fn}(b) - 1) \uplus (\text{fn}(b) - 1) < \bigcup_{i=1}^{m_{in}} \text{fn}(i) \uplus \bigcup_{i=1}^{m_{out}} \text{fn}(i) \uplus \text{fn}(b) \Rightarrow \text{fd}_{rhs} < \text{fd}_{lhs}$$

$$\left. \begin{array}{l} \text{fd}_{lhs} > \text{fd}_{rhs} \end{array} \right\} \Rightarrow \langle \text{fd}_{lhs}, \text{bd}_{lhs}, n_{lhs} \rangle > \langle \text{fd}_{rhs}, \text{bd}_{rhs}, n_{rhs} \rangle$$

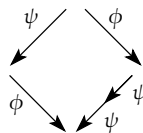
□

Proof (Lemma 14).

1. No overlap:

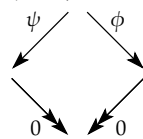


or worst case (if the redex of ψ is duplicated by ϕ):



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.

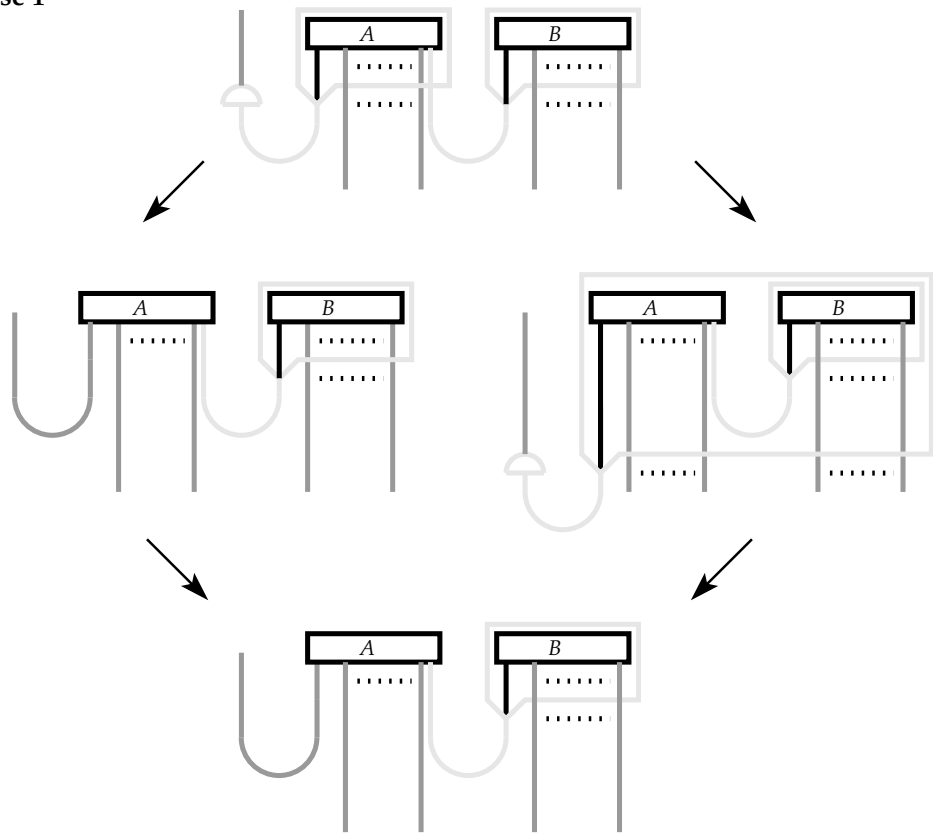
2. (a) $\psi = \phi$



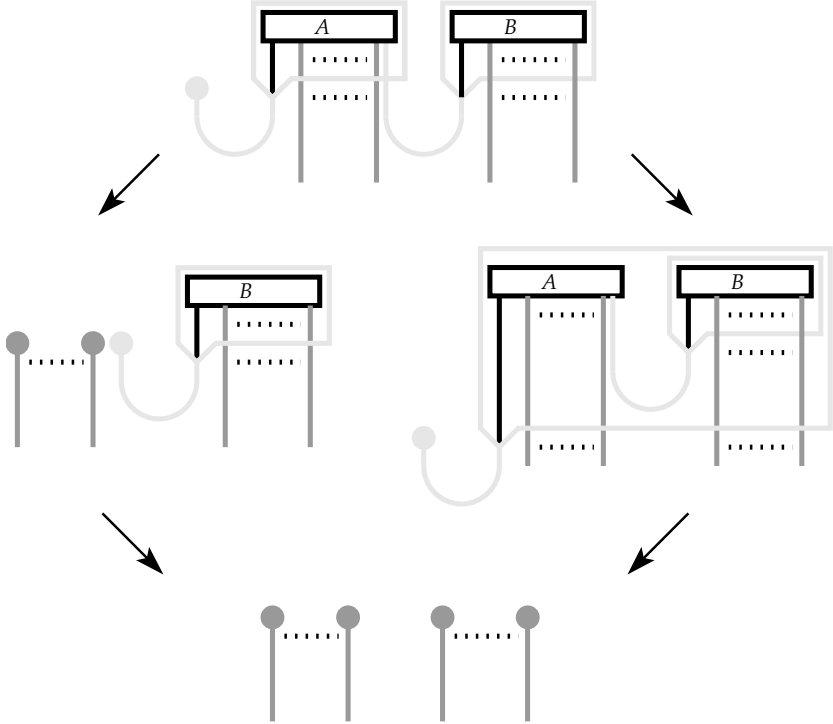
if the redex sets are the same, we don't have to do any steps to arrive at a similar reduct.

(b) Critical pairs:

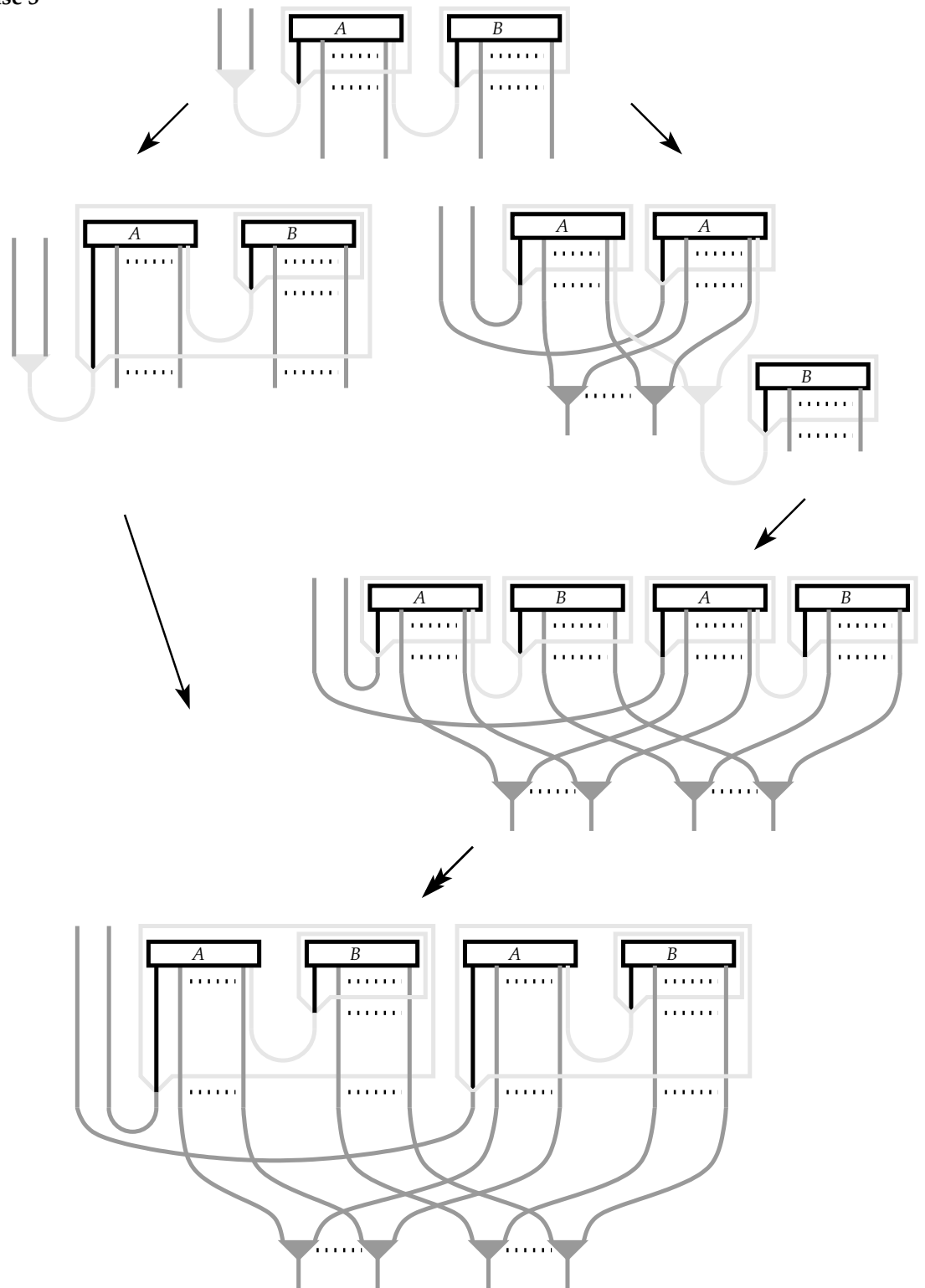
Case 1



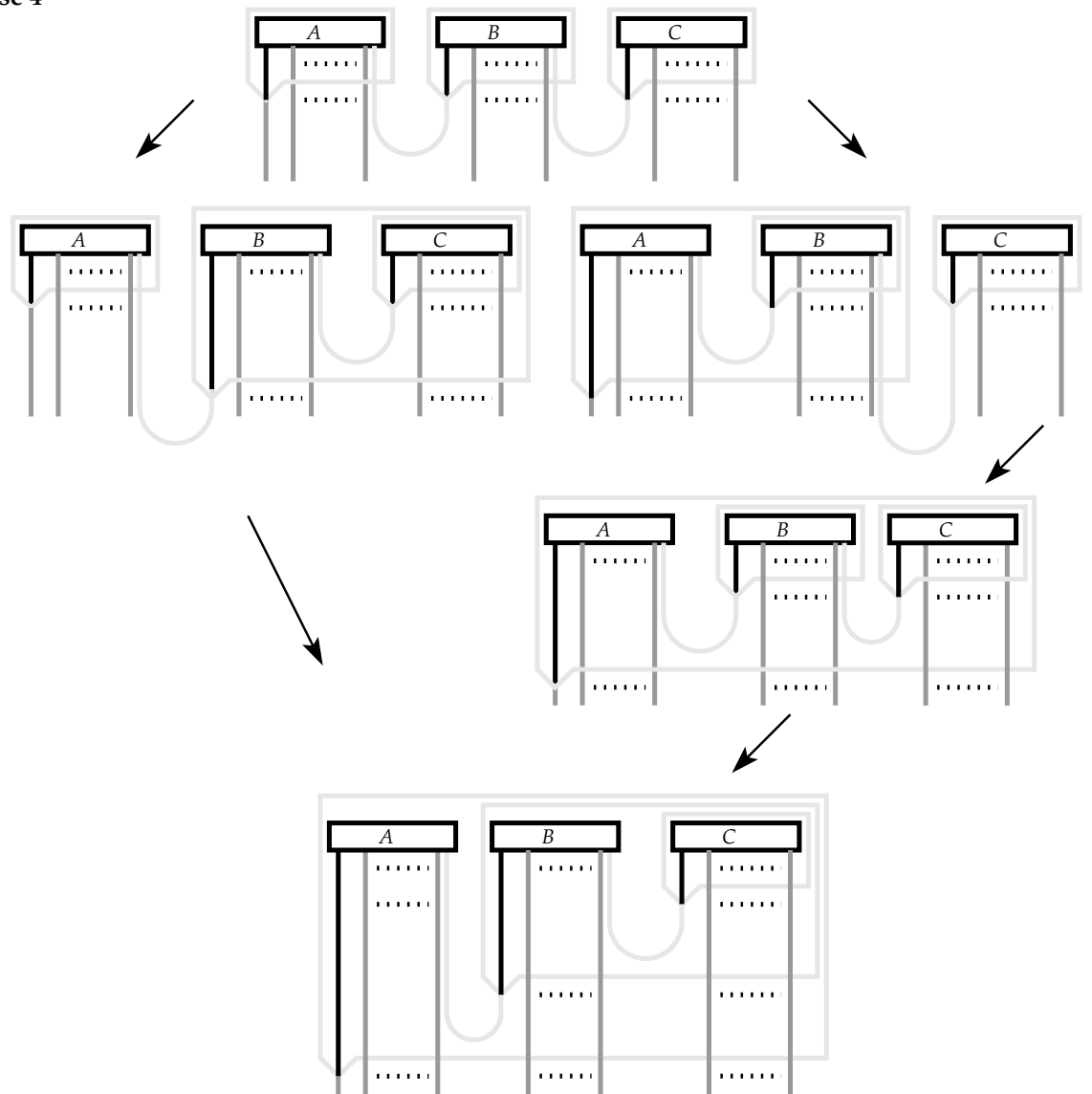
Case 2



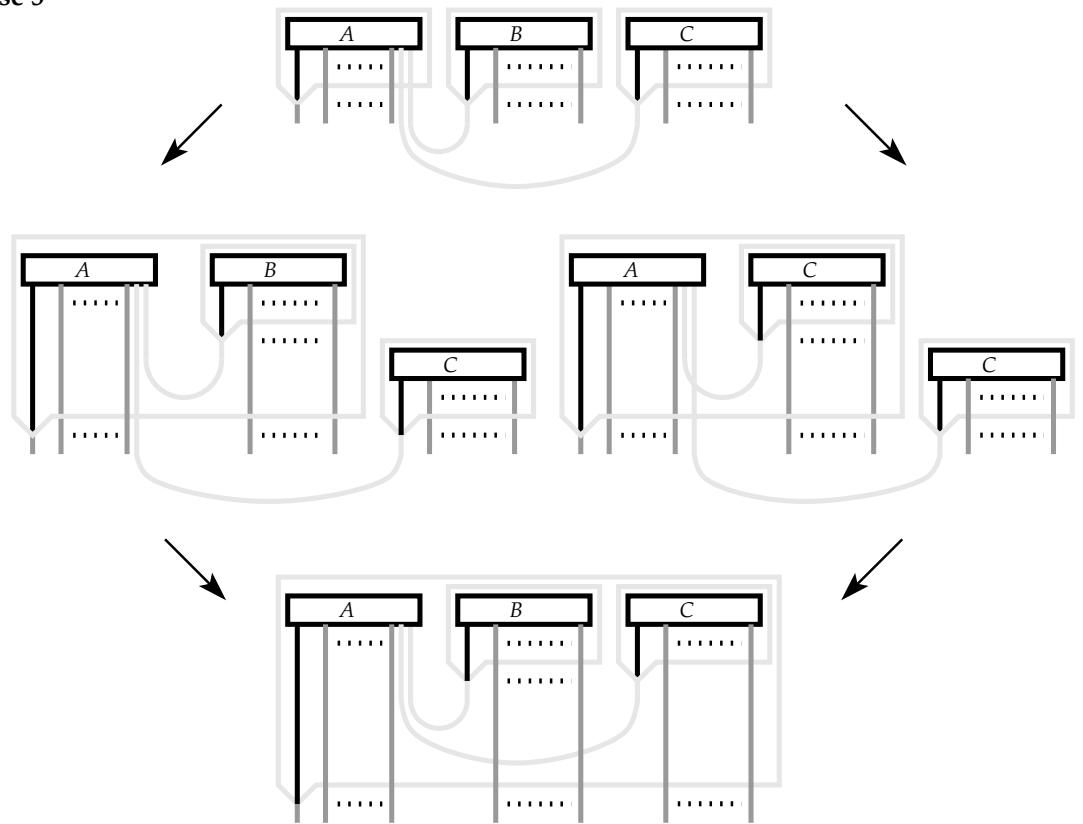
Case 3



Case 4



Case 5

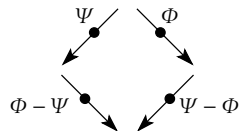


□

Proof (Lemma 15). By Newman's lemma: SN and WCR \Rightarrow CR. We proved the first condition (Strongly Normalizing) in Lemma 13 and the second in Lemma 14, so we may conclude that developments have in fact the Church-Rosser property. □

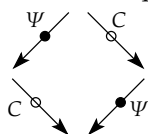
Proof (Lemma 17). Cases:

1. Two multi- (ψ, \sim) -steps:



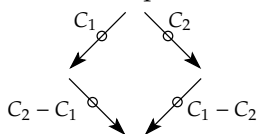
because $\dashv\rightarrow$ is weakly orthogonal.

2. One development and one multi- (ψ, \sim) -step:



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

3. Two developments:



by Lemma 15.

□

Proof (Lemma 18). We proved that \rightarrow^* has the diamond property in Lemma 17 and since $DP \Rightarrow CR$, \rightarrow^* has the Church-Rosser property. □

Proof (Lemma 19). A single \rightarrow_π -step is either the development of a cluster containing exactly one redex, or a multi- (ψ, \sim) -step contracting exactly one redex, and since \rightarrow^* is defined as the union of all developments and all multi- (ψ, \sim) -steps, the first subsumption ($\rightarrow_\pi \subseteq \rightarrow^*$) is a fact. Now for the second subsumption: a \rightarrow^* -step s_1 is either the development of a cluster C , or a multi- (ψ, \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 \rightarrow_π^* -step, for developments are defined as the concatenation of a (finite) number of \rightarrow_π -steps.
- if s_1 is a multi- (ψ, \sim) -step contracting a set of redexes R , s_1 can be simulated by a \rightarrow_π^* -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 \rightarrow_π^* -step.

□

C Strong Normalisation Proof for Typed Inductive Proof Nets

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

Because of this complication, we will split the \longrightarrow_{π^t} -ARS in two: the first system (\longrightarrow_w) will contain only the weakening rule and the second ($\longrightarrow_{\pi^t-w}$) every non-weakening rule. In order to prove that \longrightarrow_{π^t} is SN, we will first show that both abstract rewriting systems resulting from this division are SN. We will then have to prove that their union \longrightarrow_{π^t} is also SN. For this purpose, we will use the following lemma:

Lemma 29 (Postponing). *Let $\rightarrow = (\rightarrow_1 \cup \rightarrow_2)$ and if $a \rightarrow_1 b \rightarrow_2 c$ then $a \rightarrow_2 d \rightarrow c$ (which is equivalent to $(\rightarrow_1 \circ \rightarrow_2) \subseteq (\rightarrow_2 \circ \rightarrow)$) then \rightarrow is SN iff \rightarrow_1 and \rightarrow_2 are both SN.*

Proof (Lemma 29). The ‘only if’ direction of the iff is trivial, since $\rightarrow_1 \subseteq \rightarrow$ and $\rightarrow_2 \subseteq \rightarrow$. To prove the ‘if’ direction, we will show that if \rightarrow permits an infinite sequence, then there is a reduction $a \rightarrow_2 b$ such that b permits an infinite reduction sequence again. This would prove that \rightarrow_2 is not SN, which contradicts one of our assumptions.

1st step is \rightarrow_2 : Trivial.

1st step is \rightarrow_1 : Since \rightarrow_1 is SN by assumption, the remaining reduction sequence will be of the form $\rightarrow_1 \circ \rightarrow_2 \circ R$ and because from our assumption that $(\rightarrow_1 \circ \rightarrow_2) \subseteq (\rightarrow_2 \circ \rightarrow)$ we can derive $(\rightarrow_1 \circ \rightarrow_2) \subseteq (\rightarrow_2 \circ \rightarrow)$ by induction on the length of \rightarrow_1 , we can create an infinite reduction. □

Our aim now, is to prove the following three things:

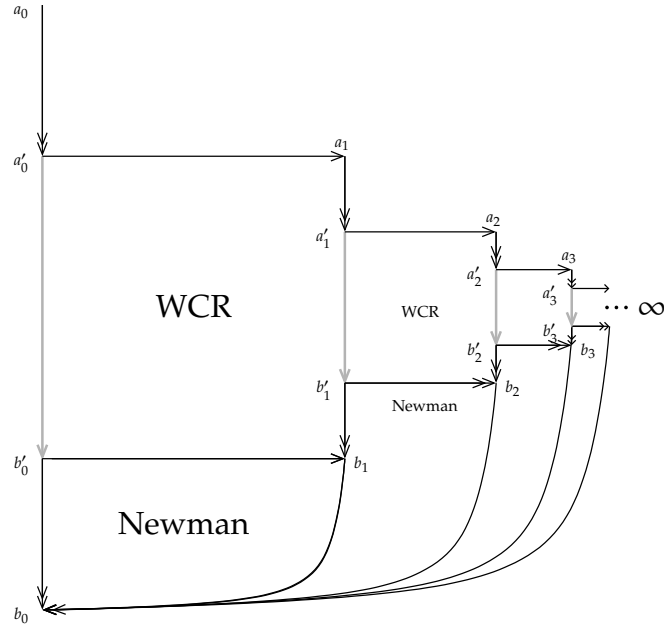
1. SN for $\longrightarrow_{\pi^t-w}$
2. that \rightarrow_w -steps can always be postponed
3. SN for \rightarrow_w

First, we will give a proof of SN for $\longrightarrow_{\pi^t-w}$, using the following lemma which can be found in [Oos01]. The general 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 30 (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 31 (WCR, WN, eventually increasing \Rightarrow SN). *An ARS which is WCR, WN and eventually increasing, is SN.*

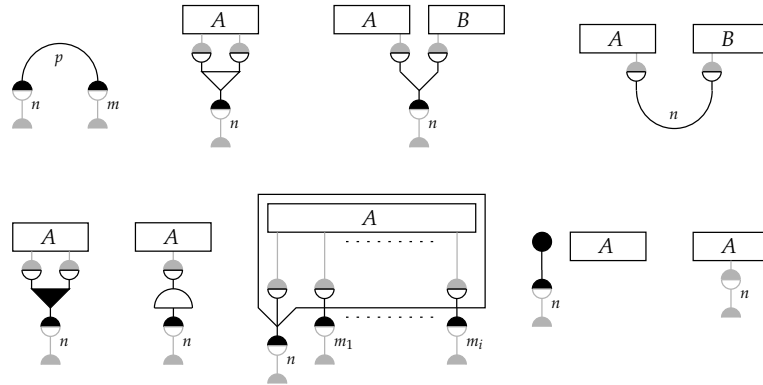
Proof (Lemma 31). We will prove this using the following scheme:



Let R be an *unsafe reduction* $R : a_0 \twoheadrightarrow b_0$ to normal form. The reduction is *unsafe* in the sense that there is another reduction $S : a_0 \rightarrow a_1 \rightarrow \dots \infty$. 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 grey. Now using WCR repeatedly, we can construct an infinite reduction sequence $a_0 \twoheadrightarrow a_1 \rightarrow \dots \infty$ 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*. \square

For the purpose of proving that $\twoheadrightarrow_{\pi_{-w}^t}$ is eventually increasing, we will introduce a more sophisticated kind of inductive proof nets, called inductive memory nets. The introduction of these memory nets is necessary, because $\twoheadrightarrow_{\pi_{-w}^t}$ 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 \sim -rules nodes inevitably disappear. We will see that the memory net equivalent of $\twoheadrightarrow_{\pi_{-w}^t}$, which will be called \twoheadrightarrow_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 32 (Inductive Memory Nets). If m , n and p are integers and A and B are inductive memory nets, then so are the following nets:

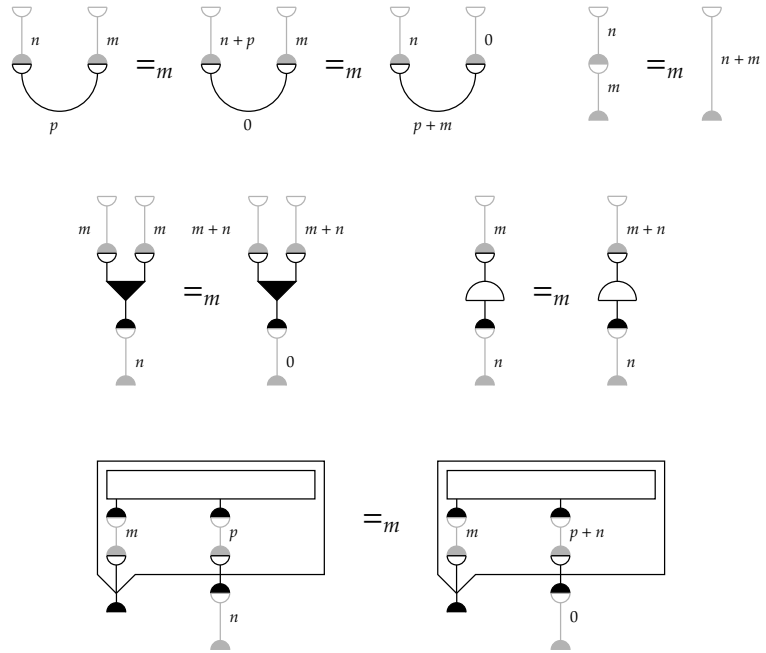


We see that memory nets are in fact ordinary proof nets of which the cut and axiom nodes are labelled with an integer and every link has been replaced by a number of what we will call *storage links* (the grey objects in the picture above). Just like the cut and axiom links, storage links are labelled 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 there is no proof net link equivalent. To be able to distinguish between these two kinds of storage links, we introduce the notion of *global port*:

Definition 33 (Global Ports). A port will be called '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 34 (Memory Equivalence ($=_m$)).



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 35. *SR for $=_m$*

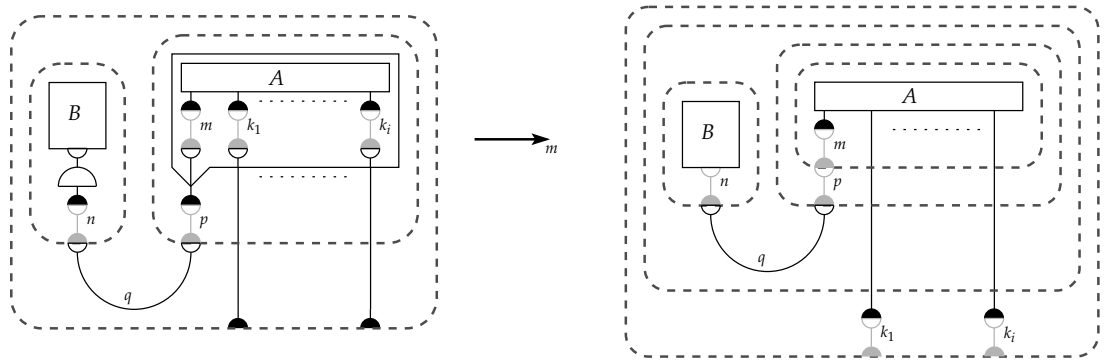
Proof (Lemma 35). Since $=_m$ never deletes storage links, but only merges/splits them (deletion would give us trouble at the global ports) and moves around their labels, the result applying a $=_m$ -rule to a memory net is still a memory net. \square

Definition 36 (Memory Net Reduction (\rightarrow_m)).

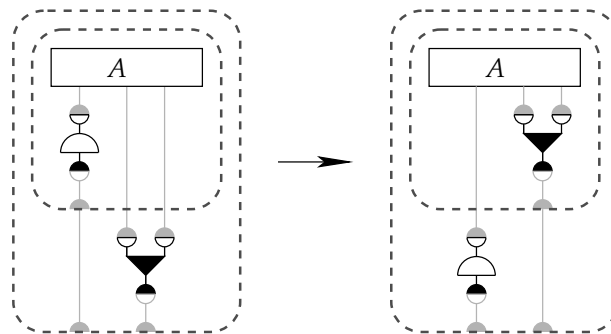
We do not only want the subject reduction property for $=_m$, but also for \rightarrow_m : if a \rightarrow_m -rule is applied to a memory net then the result should be a memory net again:

Lemma 37. *SR for \rightarrow_m*

Proof (Lemma 37). We need show for every left-hand side of a \rightarrow_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:



Dashed 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 \rightarrow_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 dashed boxes to fit the left-hand side of the \rightarrow_m -rule shown above, using rules like the following:



The same goes for the rest of the \rightarrow_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

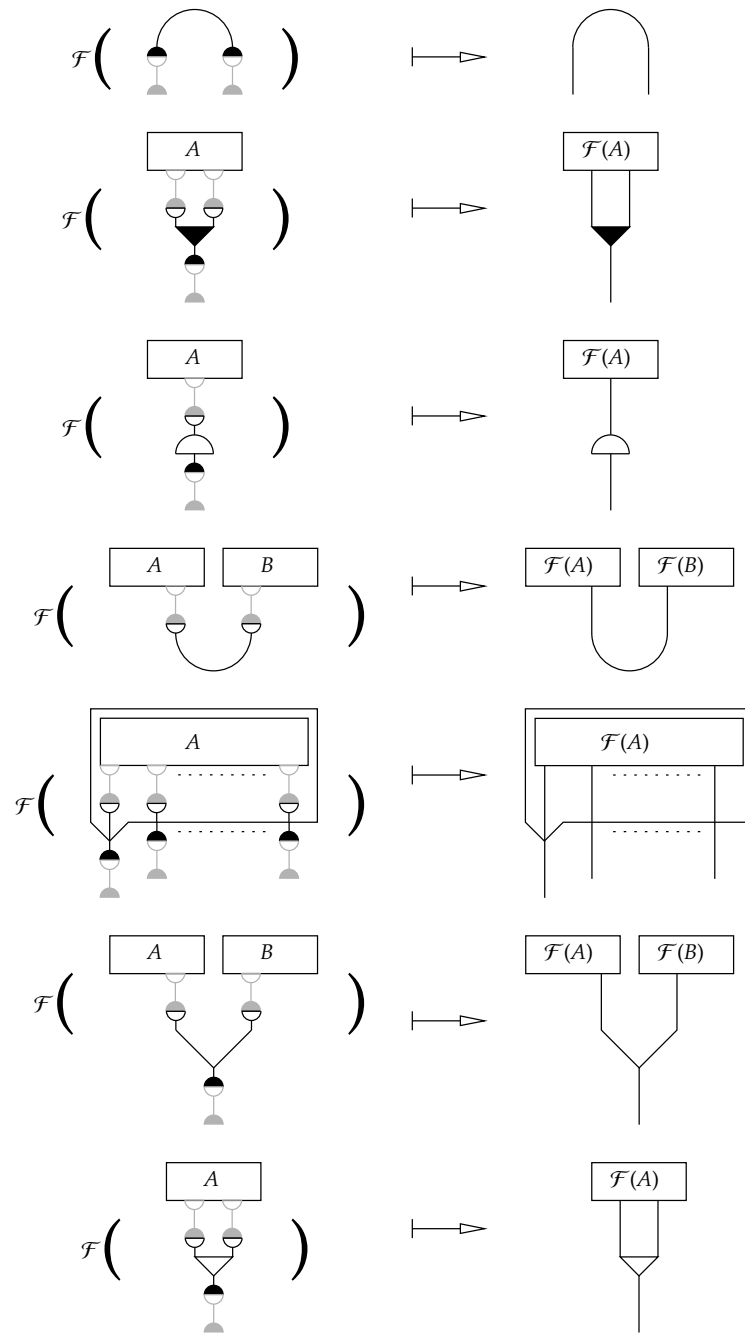
Projection Lemma) on the mapping \mathcal{M} , which maps a memory net in memory-NF to its corresponding proof net. The idea here is that we want to transform a memory net to some sort of standard, on which we will be able to apply the mapping. This standard will be called the memory-NF of a memory net. 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 38 (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 $=_m$.

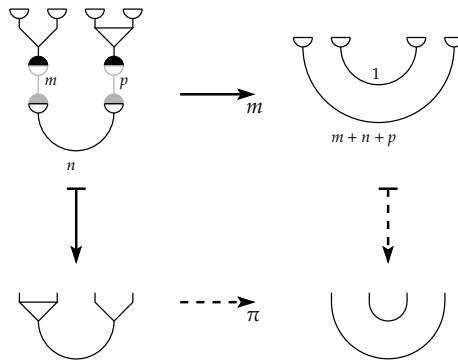
Definition 39 (Mapping \mathcal{M}). *The function \mathcal{M} first brings a memory net to memory-NF. Then all the storage links connected to a global port are thrown away and all the other ports are mapped to proof net links, by means of the following function \mathcal{F} :*



Lemma 40 (Projection).

$$\begin{array}{ccc}
 m & \longrightarrow & m' \\
 \downarrow \mathcal{M}\mathcal{AP} & & \downarrow \mathcal{M}\mathcal{AP} \\
 n & \dashrightarrow & n'
 \end{array}$$

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 $\rightarrow_{\pi_{-w}^t}$ -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 $\rightarrow_{\pi_{-w}^t}$ -rule P and that its right-hand 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:

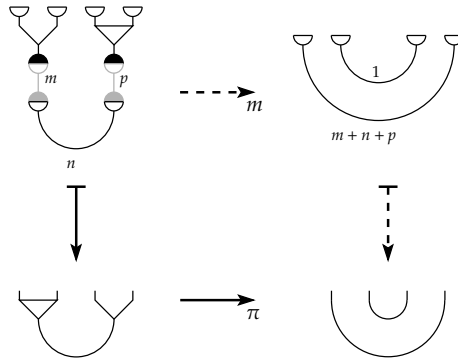


□

Lemma 41 (Lifting modulo $=_m$).

$$\begin{array}{ccc}
 m & \dashrightarrow & m' \\
 \downarrow \mathcal{M}\mathcal{AP} & & \downarrow \mathcal{M}\mathcal{AP} \\
 n & \longrightarrow & n'
 \end{array}$$

Using the mapping \mathcal{M} again, we will show there is an \rightarrow_m -equivalent of every $\rightarrow_{\pi_{-w}^t}$ -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 $\rightarrow_{\pi_{-w}^t}$ -rule P , the right-hand side of the \rightarrow_m -rule M can be mapped to the right-hand side of the $\rightarrow_{\pi_{-w}^t}$ -rule P . Since this is fairly obvious, we only show such a map for one of the $\rightarrow_{\pi_{-w}^t}$ -rules:



□

According to Lemma 31, the first property we have to prove in order to get SN for \rightarrow_m is WCR. However, since an equivalence relation ($=_m$) is defined on the memory nets the \rightarrow_m -rules are applied to, proving WCR for \rightarrow_m will do us no good. Instead, we need to prove WCR for \rightarrow_m modulo $=_m$. But first a definition of what it is exactly that we mean by ‘weakly confluent modulo’ will be given.

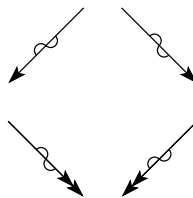
In an ARS that is extended by a set of equations, we will typically have rewriting sequences of the form:

$$a \sim a' \rightarrow b \sim b' \rightarrow c \sim c' \rightarrow \dots$$

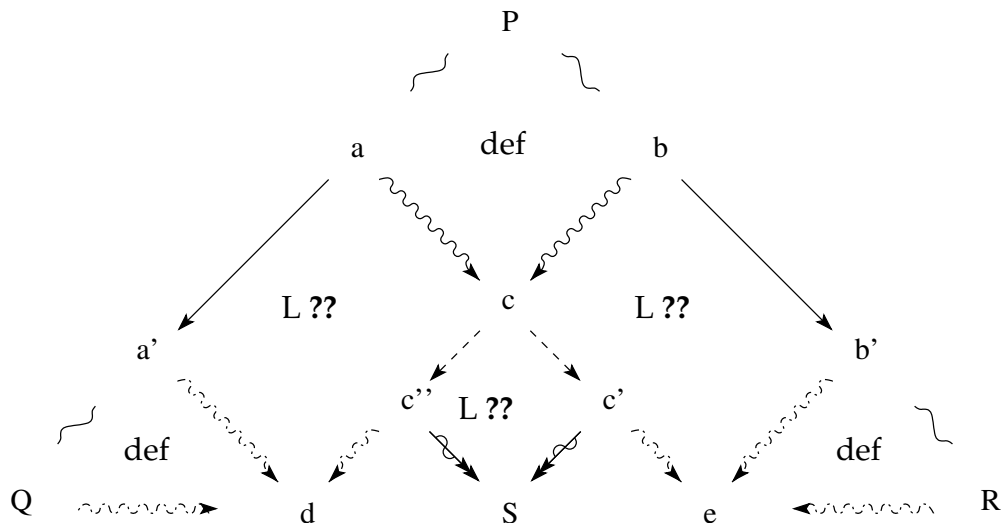
where \sim is some equivalence relation. So in this more general case, we also find that trying to establish local confluence for our rewrite relation (which in this case is \rightarrow) would be useless. Rather, we would like to have local confluence for the relation defined by $\sim \circ \rightarrow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

Definition 42 (\twoheadrightarrow). The \twoheadrightarrow -arrow is defined as $\sim \circ \rightarrow \circ \sim$, where \circ is relation composition. In our case, \twoheadrightarrow_m will be: $=_m \circ \rightarrow_m \circ =_m$.

So to prove local confluence for \rightarrow_m modulo $=_m$ we will have to show that: $(C \twoheadrightarrow_m \leftarrow A \twoheadrightarrow_m B) \Rightarrow (C \twoheadrightarrow_m D \twoheadrightarrow_m \leftarrow B)$:



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



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 43 (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 44 ($(B_m \leftarrow A \rightsquigarrow A') \Rightarrow (B \rightsquigarrow C \leftarrow \circ_m \leftarrow A')$). 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

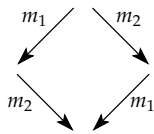
- no rule of \rightarrow_m blocks a path
- no unlawful duplication (duplication which would not be taken care of by unsharing) is done by any of the \rightarrow_m -rules
- by how much the net is increased does not depend on the weight of any link

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 to normalize first.

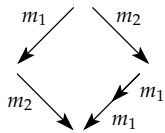
□

Lemma 45 $((d \xrightarrow{m} c \xrightarrow{m} e) \Rightarrow (d \xrightarrow{m} S \xrightarrow{m} e))$. 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.

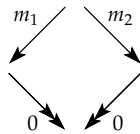
1. No overlap:



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

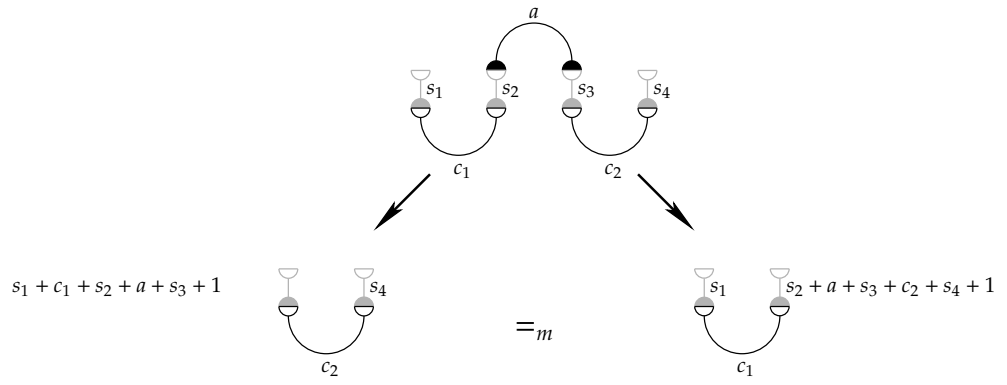
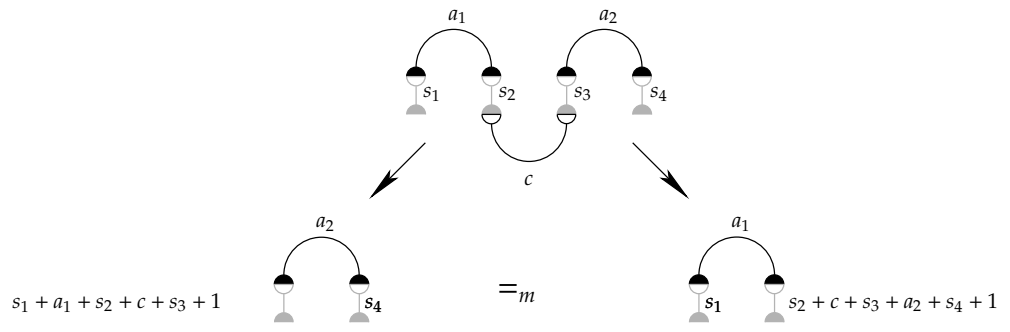


2. (a) $m_1 = m_2$:

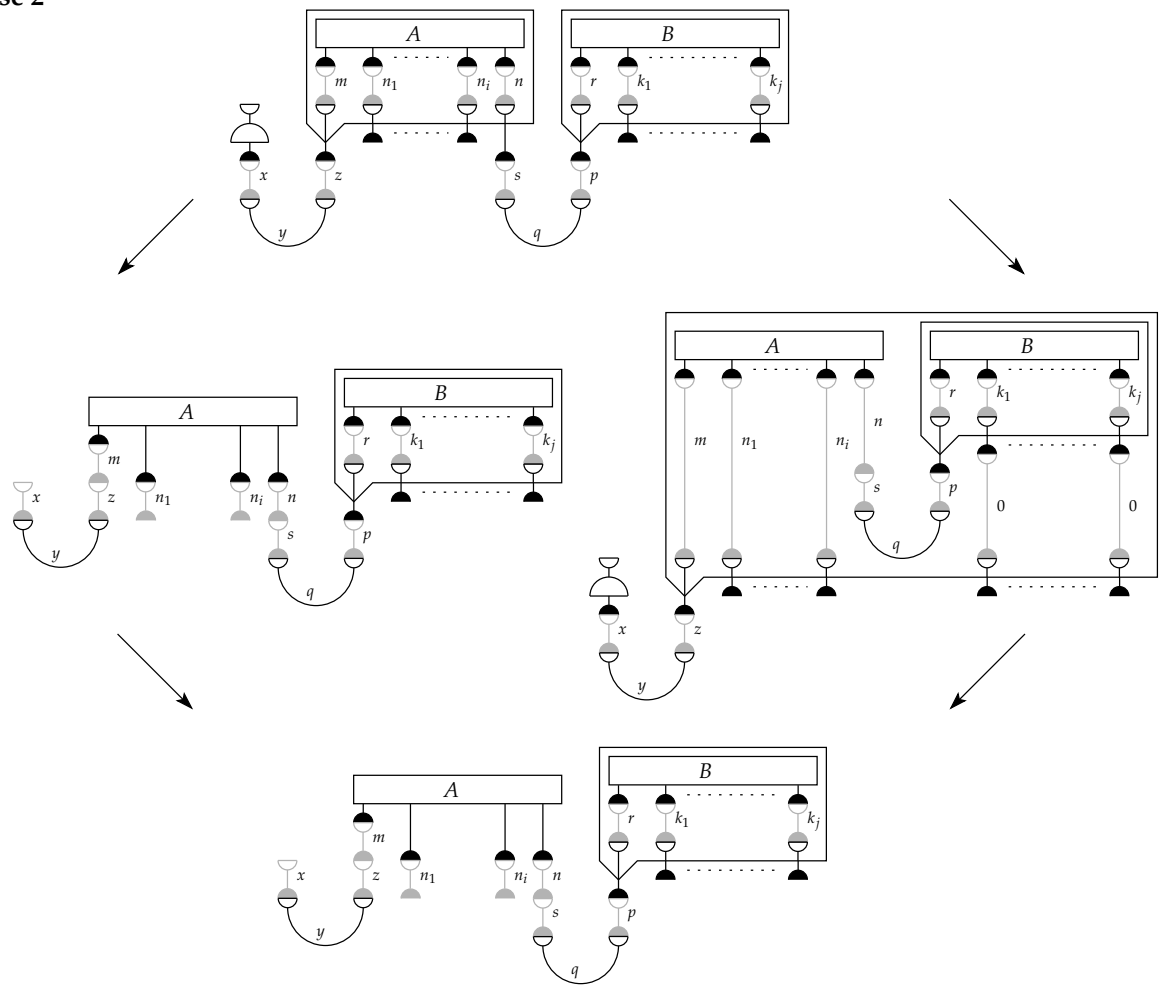


(b) Critical pairs:

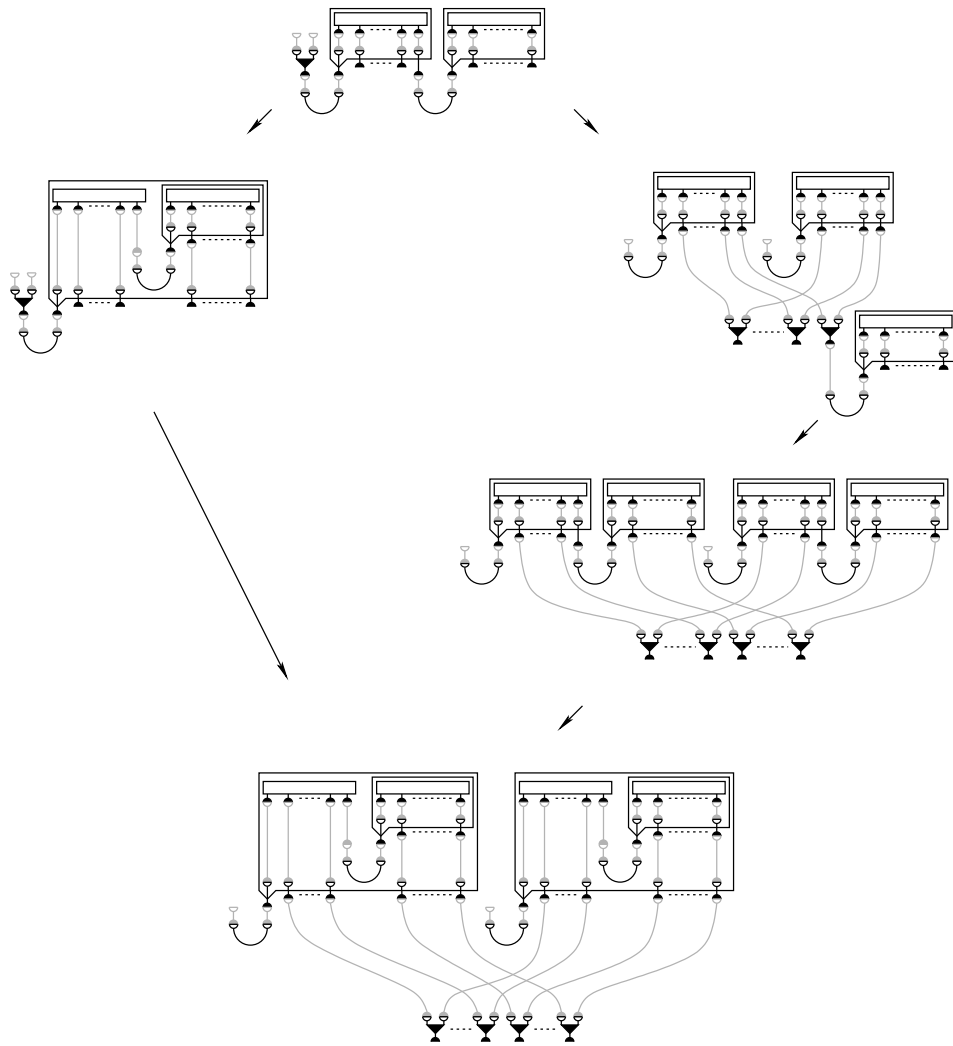
Case 1



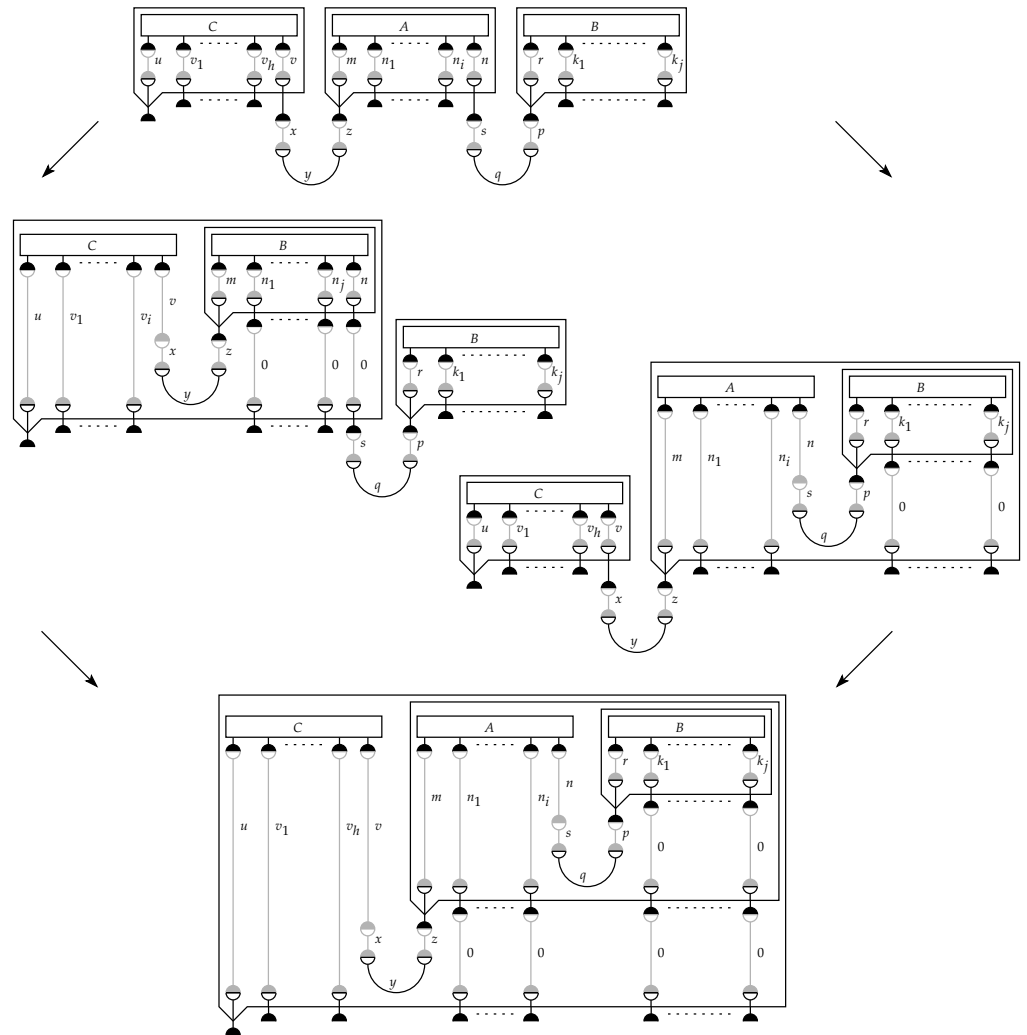
Case 2



Case 3



Case 4

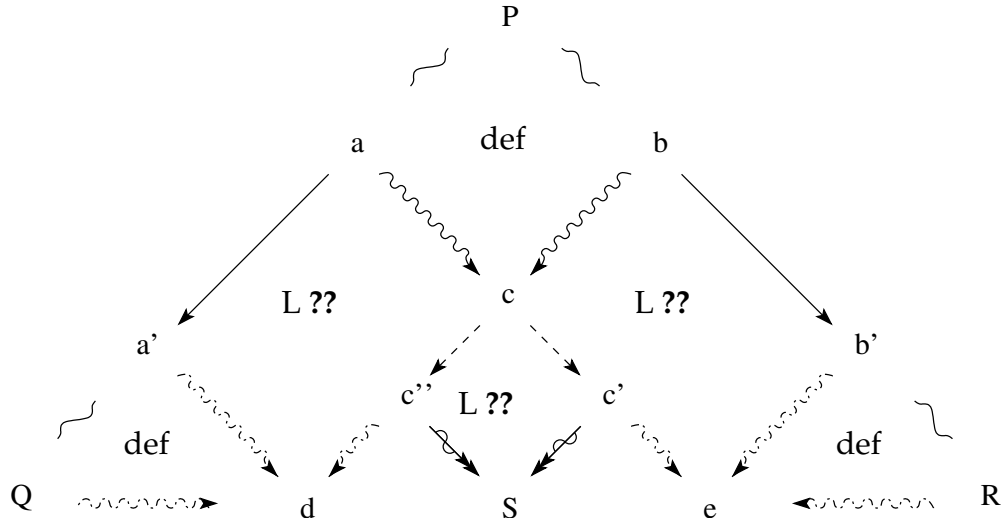


□

Lemma 46 $((c' \rightsquigarrow e \leftarrow R) \Rightarrow R =_m c')$. By definition of \rightsquigarrow : two memory nets are equal modulo $=_m$ if they have the same USNF.

□

Lemma 47 (\rightarrow_m is weakly confluent modulo $=_m$). By Lemma 45 and Lemma 44 the following scheme⁵ proves WCR modulo $=_m$ for \rightarrow_m :



Besides WCR, which we proved in Lemma 47, 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 48 (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 49 ($=_S$ -step). A step x is an $=_S$ -step iff $x \in (\rightarrow_m \cap =_S)$. This means that box-fan, box-dereliction and box-box the x -rules are the $=_S$ -steps, because the weight \mathcal{S} of a memory net M does not change when one of these rules is applied.

Lemma 50 ($(\rightarrow_m \cap =_S)$ is Strongly Normalizing). We have already seen that developments of directed inductive clusters of \rightarrow_ω are strongly normalizing (Lemma 13) and that there is a proof net equivalent of every memory net rule (Projection – Lemma 40). Now if we take a closer look, we see that the $=_S$ -rules are exactly those memory net rules that have a \rightarrow_ω -equivalent, so the Projection Lemma allows us to conclude that $(\rightarrow_m \cap =_S)$ is strongly normalizing as well.

Lemma 51 ($\rightarrow_{\pi_{-w}^t}$ is eventually increasing). To prove that $\rightarrow_{\pi_{-w}^t}$ 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 50 and we will show that the weight \mathcal{S} of a memory net is an apt measure which eventually increases:

⁵ this is the exact same scheme as the one that appeared on page 37

1. $=_S$ -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. (ψ, \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 52 ($=_S$ -normal form). A (typed) proof net is in $=_S$ -NF iff it does not contain any $=_S$ -redexes.

Definition 53. The weight \mathcal{G}' of a typed proof net is the sum of the weight \mathbf{g} of the types of all cut-nodes. The weight \mathbf{g} of a type:

- $\mathbf{g}(a) = 1$ if $a \in \text{VAR}$
- $\mathbf{g}(a^\perp) = 1$ if $a \in \text{VAR}$
- $\mathbf{g}(\sigma \otimes \tau) = \mathbf{g}(\sigma) + 1 + \mathbf{g}(\tau)$
- $\mathbf{g}(\sigma \wp \tau) = \mathbf{g}(\sigma) + 1 + \mathbf{g}(\tau)$
- $\mathbf{g}(!\sigma) = 1 + \mathbf{g}(\sigma)$
- $\mathbf{g}(?\sigma) = 1 + \mathbf{g}(\sigma)$

Lemma 54 ($\rightarrow_{\pi_w^t}$ is Weakly Normalizing). To prove that $\rightarrow_{\pi_w^t}$ is indeed Weakly Normalizing, we will present a reduction strategy \mathcal{RS} which reduces every proof net P in $=_S$ -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 $=_S$ -NF: $Q \rightarrow_{=S} Q'$
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 $=_S$ -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 $=_S$ -redexes have been formed after the first step, Q' like Q is in $=_S$ -NF and can be used as input for a recursive call to \mathcal{RS} .

□

Lemma 55 (\rightarrow_m is SN). *By Lemma 31, we may conclude from Lemma 47, Lemma 51 and Lemma 54 combined with the Lifting Lemma 41 that \rightarrow_m is indeed SN.*

□

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

Lemma 56 (Postponing Weakening). *Applying a weakening step can always be postponed: $(\rightarrow_w \circ \rightarrow_{\pi_{-w}^t}) \subseteq (\rightarrow_{\pi_{-w}^t} \circ \rightarrow_{\pi^t})$. This is easy to see, because the weakening rule can only create weakening redexes.*

□

Proof (Typed Proof Net Reduction \rightarrow_{π^t} is SN). By Lemma 55 and the Lifting Lemma 41 we know $\rightarrow_{\pi_{-w}^t}$ is SN. Furthermore the weakening rule is obviously SN⁶ and can always be postponed until after all the $\rightarrow_{\pi_{-w}^t}$ -steps have been done (Lemma 56). So by proving Lemma 29, we have also proven SN for \rightarrow_{π^t} .

□

⁶ every weakening step deletes a node