

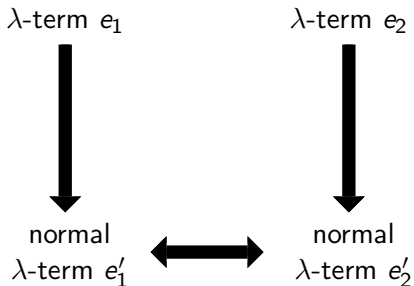
# Proof Nets for Intuitionistic Logic

Final Talk for the Diploma Thesis  
of Matthias Horbach

Saarland University  
Programming Systems Lab

October 12, 2006

# A New Notion of Proof / Program Equivalence





# Outline

- 1 Proof Theory
  - History of Proof Theory
  - Intuitionistic Logic
- 2 Proofs in Intuitionistic Logic
  - The Simply Typed Lambda-Calculus
  - Proof Nets
- 3 Equivalence of Proofs
  - Equality of Lambda-Terms
  - Equality of Proof Nets
- 4 Conclusions

# Proof Theory

## Historical Background

- Before around 1920 proofs were just plain text.
- Hour of birth of proof theory:  
Hilbert's Program to formalize all of mathematics
- Goals of proof theory: Given a logic,
  - 1 find formal proof systems and
  - 2 identify equal proofs.

# Proof Theory

## Importance for Computer Science

- The same questions affect programming:
  - 1 find programming paradigms and
  - 2 identify equal programs.
- Known notions of program equivalence:

Programs are equivalent,

  - if they take arguments of the same type and return objects of the same type.
  - if they compute the same function using the same algorithm, in the sense that the programs are equal modulo inlining of subprocedures.
  - $\vdots$
  - if they are syntactically equal.
- We will see: functional programs can be regarded as proofs in intuitionistic logic.

# What is Intuitionistic Logic?

- Starting point: classical propositional logic.  
Formulas consist of propositional variables ( $a, b$ ) and boolean connectives ( $\neg, \rightarrow, \wedge, \vee$ ).
- Criticism (e.g. by Heyting):  
Is “ $i = 5$ , if  $A$  is true, and  $i = 4$ , if  $A$  is false” a well-formed definition?
- Similar problem in programming:  
“ $i = 5$ , if program  $P$  terminates, and  $i = 4$ , if  $P$  does not terminate”
- Proposed solution: restrict classical reasoning by excluding the *tertium non datur* principle.
- This yields *intuitionistic logic*, the logical framework for functional programming.
- We will (for now) only consider the purely implicative fragment!

# What is Intuitionistic Logic?

- Starting point: classical propositional logic.  
Formulas consist of propositional variables ( $a, b$ ) and boolean connectives ( $\neg, \rightarrow, \wedge, \vee$ ).
- Criticism (e.g. by Heyting):  
Is “ $i = 5$ , if  $A$  is true, and  $i = 4$ , if  $A$  is false” a well-formed definition?
- Similar problem in programming:  
“ $i = 5$ , if program  $P$  terminates, and  $i = 4$ , if  $P$  does not terminate”
- Proposed solution: restrict classical reasoning by excluding the *tertium non datur* principle.
- This yields *intuitionistic logic*, the logical framework for functional programming.
- We will (for now) only consider the purely implicative fragment!



- 1 Proof Theory
  - History of Proof Theory
  - Intuitionistic Logic
- 2 Proofs in Intuitionistic Logic
  - The Simply Typed Lambda-Calculus
  - Proof Nets
- 3 Equivalence of Proofs
  - Equality of Lambda-Terms
  - Equality of Proof Nets
- 4 Conclusions

# The Simply Typed $\lambda$ -Calculus

## The Reference Proof System

- Functional programming is about modeling functions.
- Syntax of  $\lambda$ -terms (Church 1936), i.e. of programs:

$$e ::= v \mid \lambda v.e \mid e e$$

Additionally annotate the type of every variable and allow only well-typed applications.

- Curry-Howard-Correspondence:
  - Read types as formulas.
  - A purely implicational formula is intuitionistically valid, if and only if it corresponds to the type of a closed  $\lambda$ -term.
- Example:

$\lambda$ -term	$\rightsquigarrow$	type	$\Leftarrow$	formula
$\lambda x.x$	$\rightsquigarrow$	$a \rightarrow a$	$\Leftarrow$	$a \rightarrow a$

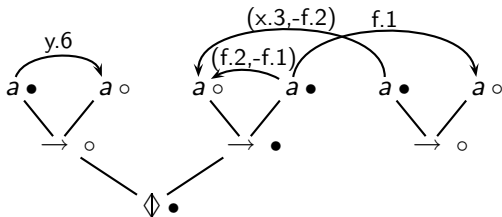
# Proof Nets

## Why Proof Nets?

- Invention of proof nets: Girard (1986)
- He wanted:
  - a proof system for linear logic
  - parallelity, compactness and minimal syntax
  - capturing the “essence” of a proof
- He believed all these goals to be brought together in proof nets.
- Proof nets for classical logic:  
Lamarche and Straßburger (2005).
- Now: Proof nets for intuitionistic logic.

# Proof Nets

## The Shape of Intuitionistic Nets



- Nets are a graphical proof structure, consisting of:
  - a tree coding the formula we want to prove
  - some special trees (*cuts*) modeling modularity of proofs
  - (labeled) links between leaves of all these trees
- Nodes are polarized to indicate negative (●) and positive (○) contexts.
- Links have to connect negative and positive atoms.

# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f (f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f (f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$f$

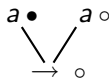


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$f, f$



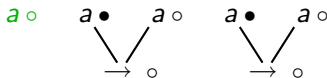


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$f, f, x$

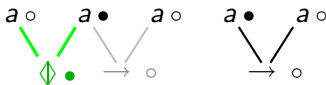


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$f, (f x)$

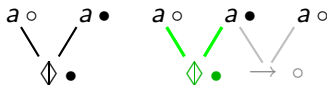


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$f(f x)$



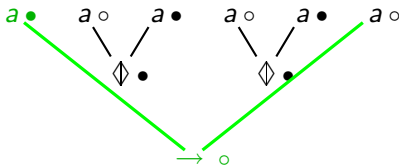


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

$\lambda x.f(f x)$



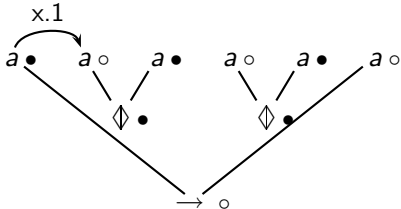


# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

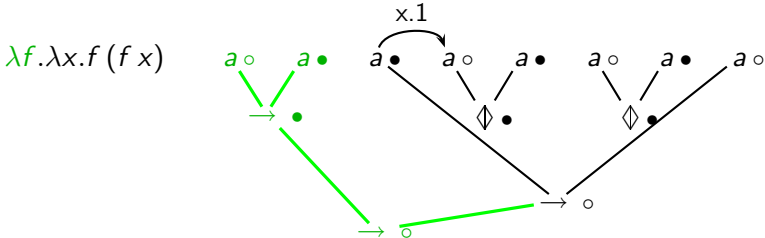
$\lambda f.\lambda x.f(f x)$



# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .

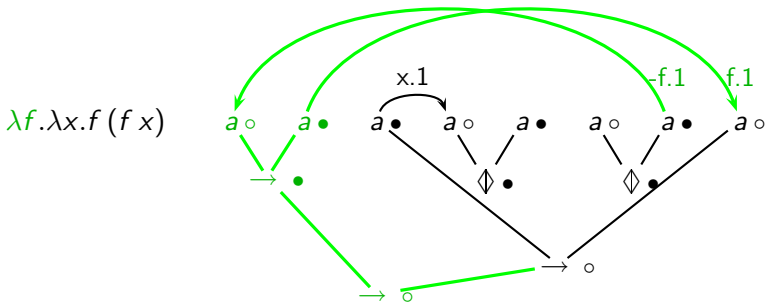




# Proof Nets

## Nets and $\lambda$ -Terms

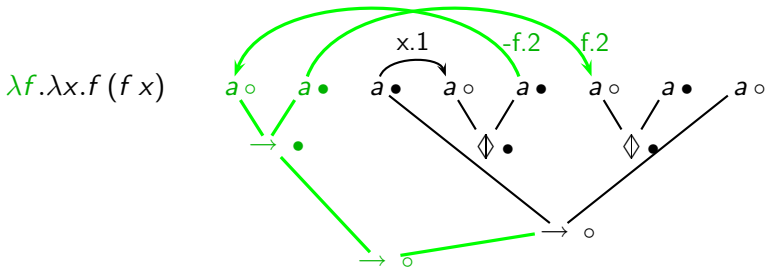
- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .



# Proof Nets

## Nets and $\lambda$ -Terms

- Nets extend the idea of functional programs:  
There is a translation from  $\lambda$ -terms to nets.
- We translate  $\lambda f.\lambda x.f(f x)$ , where  $x : a$  and  $f : a \rightarrow a$ .





# Proof Nets

## Properties of Proof Nets

Question: What kinds of properties distinguish proof nets?

▶ skip one

# Proof Nets

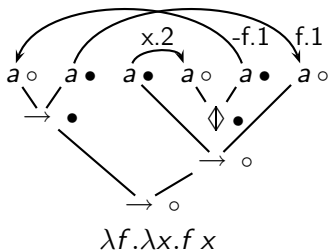
## Properties of Proof Nets — Classical Correctness

### Definition

A *conjunctive pruning* of a net is obtained by deleting one subtree of each  $\rightarrow^\bullet$  node and each  $\diamond^\bullet$ -node (and the node itself).

A net is *classically correct*, if every conjunctive pruning contains at least one link.

Example:



# Proof Nets

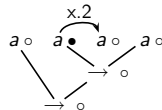
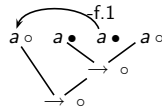
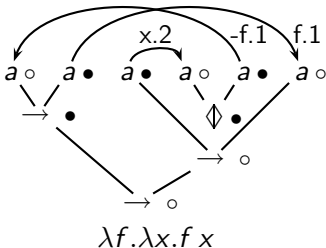
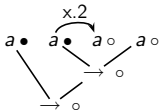
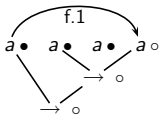
## Properties of Proof Nets — Classical Correctness

### Definition

A *conjunctive pruning* of a net is obtained by deleting one subtree of each  $\rightarrow^\bullet$  node and each  $\diamond^\bullet$ -node (and the node itself).

A net is *classically correct*, if every conjunctive pruning contains at least one link.

Example:



# Proof Nets

## Properties of Proof Nets — Classical Correctness

### Theorem

*All proof nets are classically correct.*

# Proof Nets

## Properties of Proof Nets — Classical Correctness

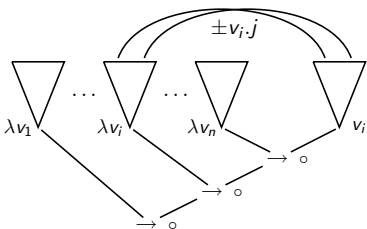
### Theorem

*All proof nets are classically correct.*

Proof idea:

*case 1:* The proof net corresponds to an application-free term:

$$e = \lambda v_1. \dots \lambda v_n. v_i$$





# Proof Nets

## Properties of Proof Nets — Classical Correctness

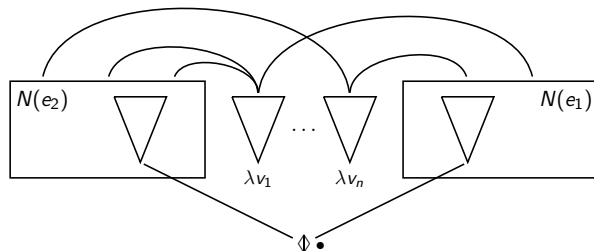
### Theorem

*All proof nets are classically correct.*

Proof idea:

case 2: The proof net corresponds to a term with applications:

$$e = \lambda v_1. \dots \lambda v_n. e_1 e_2$$



# Proof Nets

## Properties of Proof Nets — Classical Correctness

### Theorem

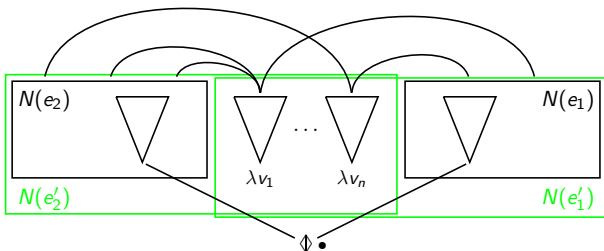
*All proof nets are classically correct.*

Proof idea:

case 2: The proof net corresponds to a term with applications:

$$e = \lambda v_1. \dots \lambda v_n. e_1 e_2$$

Consider  $e'_i = \lambda v_1. \dots \lambda v_n. e_i$ .



# Proof Nets

## Properties of Proof Nets — Paths

- Cuts model which term is used as input to which other term.
- Links model which variable occurrences are affected by the instantiation of which binder.
- In combination, they model the information flow through a term.

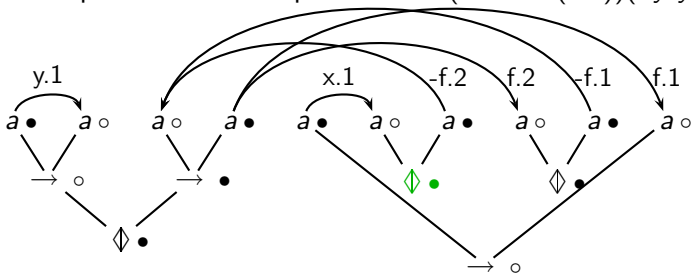
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$ :



### Theorem

*The number of paths in each proof net is finite.*

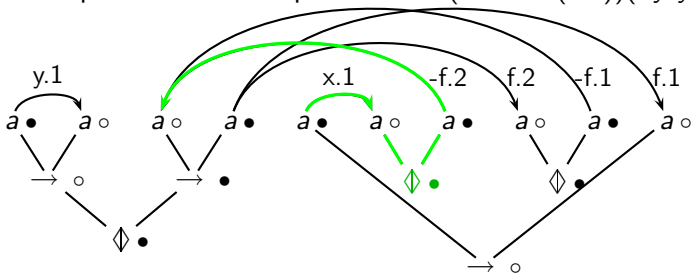
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$ :



### Theorem

*The number of paths in each proof net is finite.*

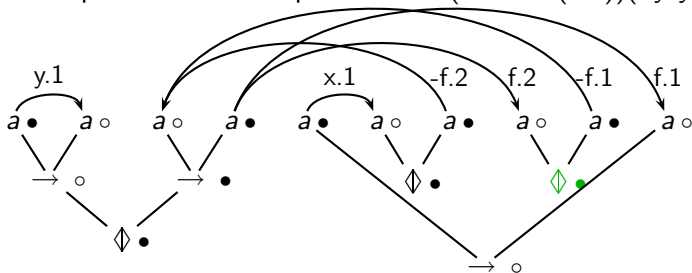
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$ :



### Theorem

*The number of paths in each proof net is finite.*

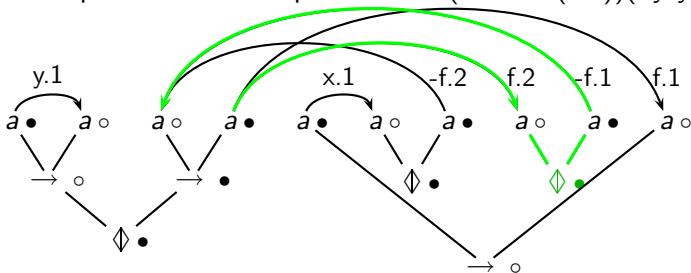
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$ :



### Theorem

*The number of paths in each proof net is finite.*

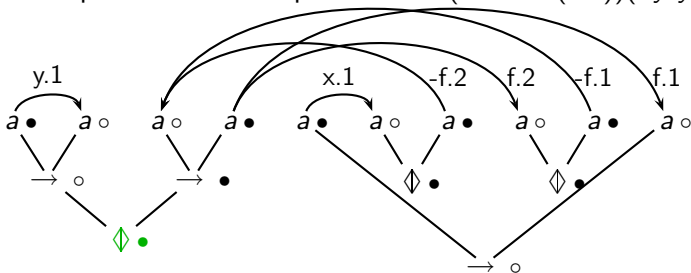
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$ :



### Theorem

*The number of paths in each proof net is finite.*



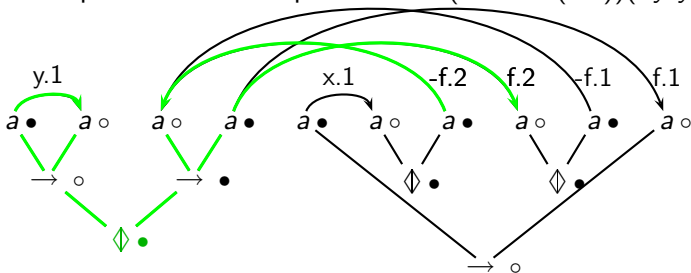
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$ :



### Theorem

*The number of paths in each proof net is finite.*

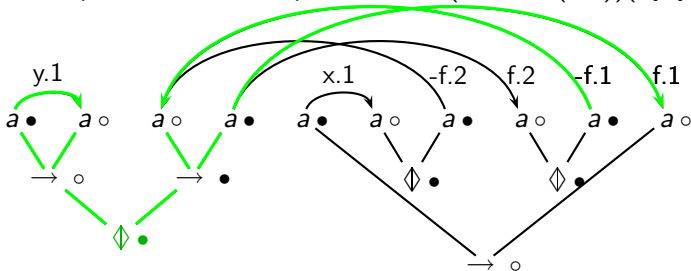
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$ :



### Theorem

*The number of paths in each proof net is finite.*

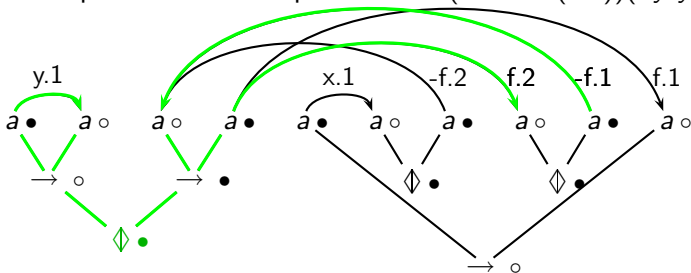
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$ :



### Theorem

*The number of paths in each proof net is finite.*

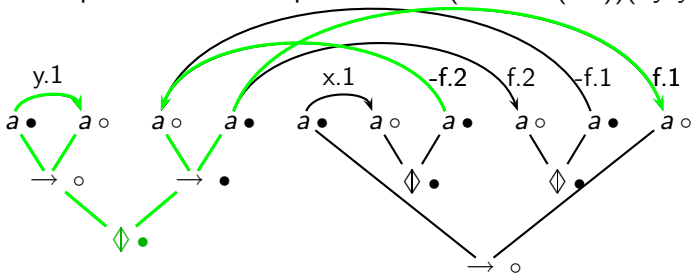
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$ :



### Theorem

*The number of paths in each proof net is finite.*

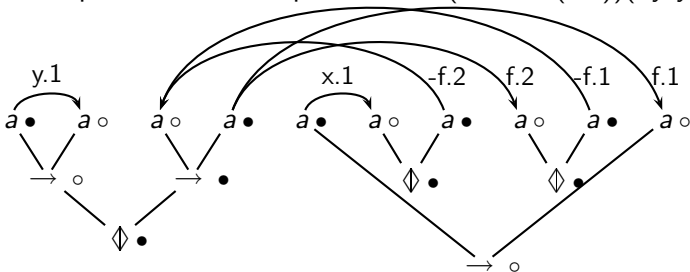
# Proof Nets

## Properties of Proof Nets — Paths

### Definition

Path = series of links that are connected by cuts  
+ a well-formedness condition

- Example: Paths in the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$ :



### Theorem

*The number of paths in each proof net is finite.*

# Proof Nets

## Properties of Proof Nets — Ramification

- Paths model information/program flow
  - Parts of a program may be visited several times during one run.
  - The result of a program is determined by exactly one sequence of operations.
- Analog for proof nets:
  - Nodes may be connected by several paths.
  - But: This does not hold for *output nodes*!

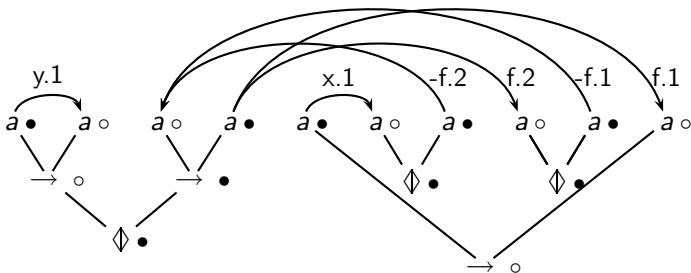
### Theorem

*Proof nets are unramified, i.e. output nodes can be reached by exactly one (maximal) path.*

# Proof Nets

## Properties of Proof Nets — Ramification

- Example 1: Double application:

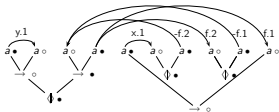


only path: x.1, -f.2, y.1, f.2, f.1, y.1, -f.1

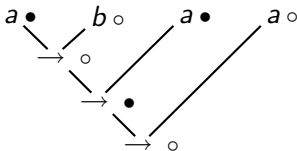
# Proof Nets

## Properties of Proof Nets — Ramification

- Example 1: Double application:



- Example 2: Pierce's law:

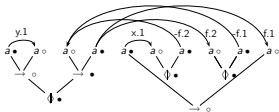




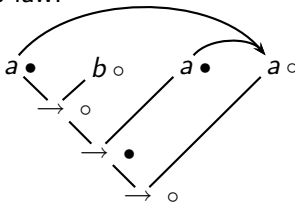
# Proof Nets

## Properties of Proof Nets — Ramification

- Example 1: Double application:



- Example 2: Pierce's law:



- 1 Proof Theory
  - History of Proof Theory
  - Intuitionistic Logic
  
- 2 Proofs in Intuitionistic Logic
  - The Simply Typed Lambda-Calculus
  - Proof Nets
  
- 3 Equivalence of Proofs
  - Equality of Lambda-Terms
  - Equality of Proof Nets
  
- 4 Conclusions

# Normalization of $\lambda$ -Terms

When are two programs in the  $\lambda$ -calculus equal?

- $\beta\eta$ -reduction is terminating and confluent.
- Two programs are considered equal, if their  $\beta\eta$ -normal forms agree.
- Example ( $id := \lambda y.y$ ):

$$(\lambda f.\lambda x.f (f x)) id \rightsquigarrow \lambda x.id (id x) \rightsquigarrow^* \lambda x.x$$

# Normalization of Proof Nets

- Idea behind the equality of proof nets is also:  
Two proof nets are equal, if they can be reduced to the same normal form.
- In the  $\lambda$ -calculus, a normal form is reached by the evaluation (= elimination) of applications.
- In proof nets, applications correspond to cuts.
- This gives the following idea:
  - Nets are in normal form, if they are cut-free.
  - We need a cut elimination procedure for nets.
- Every net that can be reached from a proof net by a sequence of cut eliminations will also be called *proof net*.

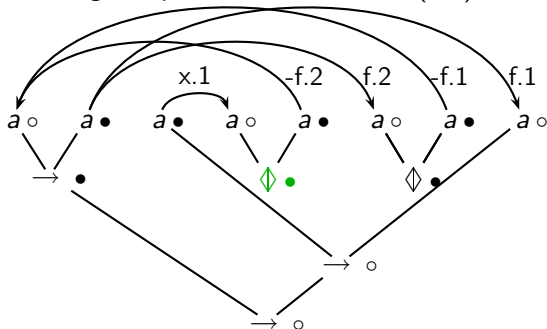
# The Cut Elimination Procedure

- To eliminate a cut,
  - 1 throw it away and
  - 2 replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f (f x)$

▶ complex example

# The Cut Elimination Procedure

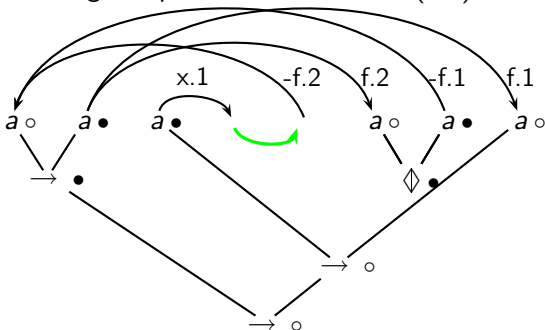
- To eliminate a cut,
  - 1 throw it away and
  - 2 replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f(f x)$



▶ complex example

# The Cut Elimination Procedure

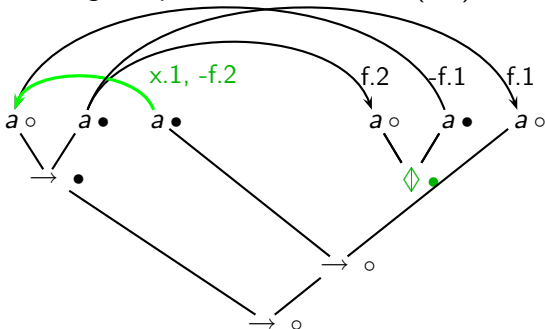
- To eliminate a cut,
  - throw it away and
  - replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f(f x)$



▶ complex example

# The Cut Elimination Procedure

- To eliminate a cut,
  - 1 throw it away and
  - 2 replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f(f x)$

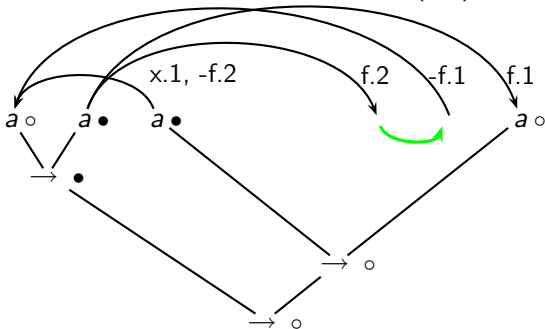


▶ complex example



# The Cut Elimination Procedure

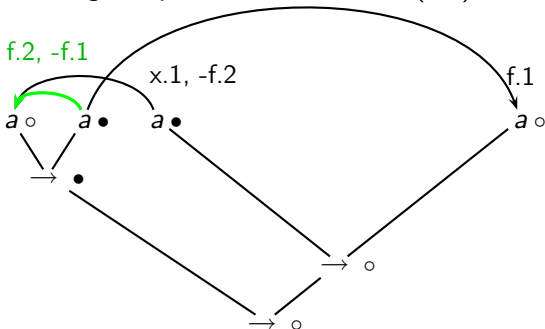
- To eliminate a cut,
  - 1 throw it away and
  - 2 replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f (f x)$



▶ complex example

# The Cut Elimination Procedure

- To eliminate a cut,
  - 1 throw it away and
  - 2 replace links by paths through the cut.
- Example: Reducing the proof net of  $\lambda f.\lambda x.f (f x)$



▶ complex example



- 1 Proof Theory
  - History of Proof Theory
  - Intuitionistic Logic
  
- 2 Proofs in Intuitionistic Logic
  - The Simply Typed Lambda-Calculus
  - Proof Nets
  
- 3 Equivalence of Proofs
  - Equality of Lambda-Terms
  - Equality of Proof Nets
  
- 4 Conclusions

# Properties of Cut Elimination

## Theorem

*It is decidable (up to link labels), whether a given net is a proof net.*

## Theorem

*Cut elimination transforms nets (proof nets) into nets (proof nets).  
Cut elimination is terminating and confluent.*

## Corollary

*Proof nets and cut elimination form a proof system for intuitionistic logic, where equality of proofs is decidable.*

# Properties of Cut Elimination

## Theorem

*Normal forms in the  $\lambda$ -calculus and in any proof net calculus cannot coincide.*

## Theorem

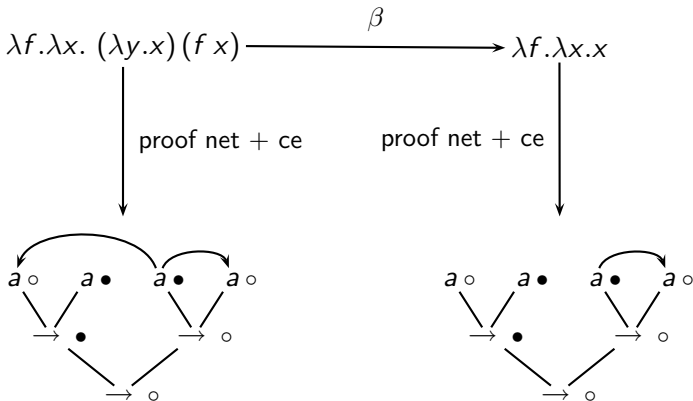
*In many cases, this proof system “refines” the system of  $\lambda$ -terms and  $\beta\eta$ -reduction:*

- *Each  $\eta$ -step corresponds to one step of cut elimination.*
- *Each linear  $\beta$ -step corresponds to one step of cut elimination.*
- *Each  $\beta$ -step with closed argument corresponds to an unchanged normal form.*
- *Each weakening step corresponds to the deletion of links in the normal form.*

# Properties of Proof Nets and Cut Elimination

## Exemplary Advantages of Proof Nets

- Proof nets are more fine-grained than  $\lambda$ -terms and preserve some modularity information:



# Properties of Proof Nets and Cut Elimination

## Exemplary Advantages of Proof Nets

- Proof nets are often more space- and time-efficient:

The  $\beta$ -normal form of

$$\lambda x. \lambda z. (\lambda y. z y y)^{n+1} x$$

- has a size exponential in  $n$  and
- is reached after at most exponentially many reductions,

but the corresponding normal proof net

- has only linearly many links and
- can be computed in linear time.



# Properties of Proof Nets and Cut Elimination

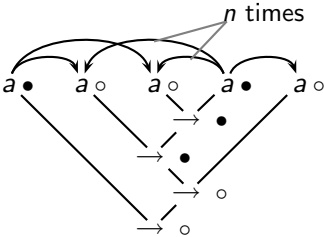
## Exemplary Advantages of Proof Nets

- Proof nets are often more space- and time-efficient:  
The  $\beta$ -normal form of

$$\lambda x. \lambda z. (\lambda y. z y y)^{n+1} x$$

- has a size exponential in  $n$  and
- is reached after at most exponentially many reductions,

but the corresponding normal proof net



- has only linearly many links and
- can be computed in linear time.

# Properties of Proof Nets and Cut Elimination

## Scaling

### Sums and Products

#### Theorem

*The translation of  $\lambda$ -terms into proof nets can be extended to sums and products.*

*All theorems (except unramification) remain valid.*

#### Theorem




*Each reduction step of sum- or product terms corresponds to the deletion of links in the normal form.*

### Universal Types




#### Theorem

*A proof net for a formula  $A$  gives rise to proof nets for every instance  $A\sigma$ .*

# References I

-  Jean Yves Girard.  
Linear logic.  
*Theoretical Computer Science*, 50(1):1–101, 1987.
-  François Lamarche and Lutz Straßburger.  
Naming proofs in classical propositional logic.  
In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications, TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 246–261. Springer-Verlag, 2005.
-  Lutz Straßburger.  
From deep inference to proof nets.  
In Paola Bruscoli François Lamarche and Charles Stewart, editors, *Structures and Deduction*, pages 2–18. Satellite Workshop of ICALP05, 2005.

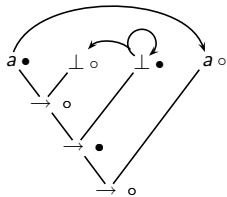
## References II

-  Vincent Danos and Laurant Regnier.  
The structure of multiplicatives.  
*Archives of Mathematical Logic*, 28:181–203, 1989.
-  François Lamarche.  
Proof nets for intuitionistic linear logic I: Essential nets.  
Technical report, Imperial College, London, 1995.
-  François Lamarche and Christian Retoré.  
Proof nets for the Lambek calculus — an overview.  
In Michele Abrusci and Claudia Casadio, editors, *Proofs and Linguistic Categories*, volume 46, pages 241–262. Cooperativa Libreria Universitaria Editrice Bologna, 1996.

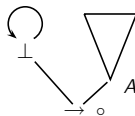
Thank you for your attention!

# Interesting Nets

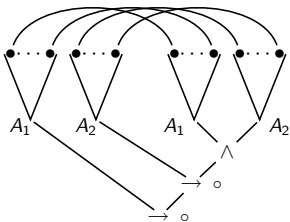
dneg



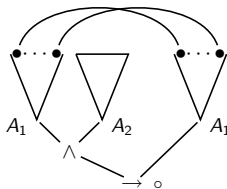
null



pair



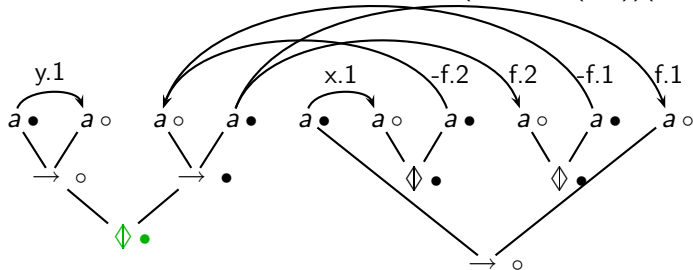
$\pi_1$



# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$

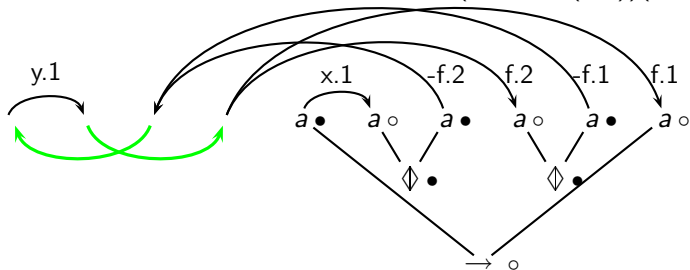


◀ back

# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f. \lambda x. f(f x))(\lambda y. y)$



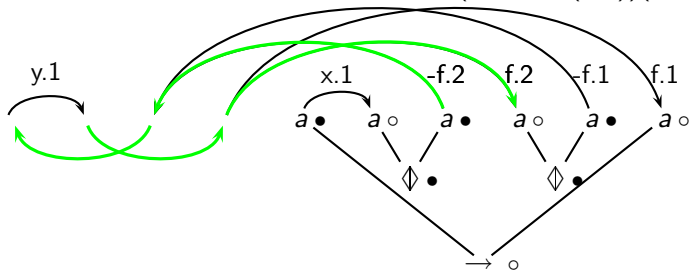
← back



# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$

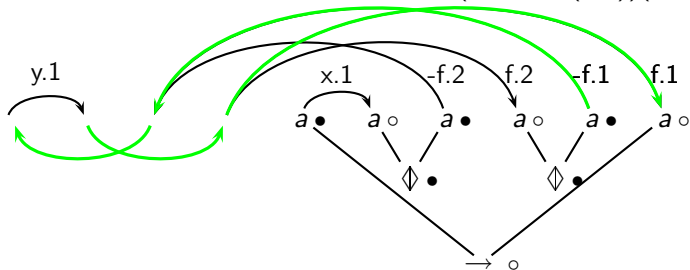


◀ back

# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$

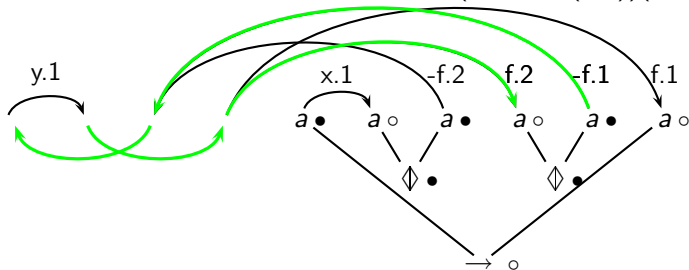


◀ back

# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$

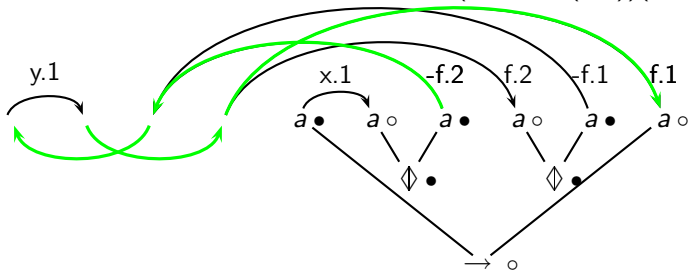


◀ back

# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$

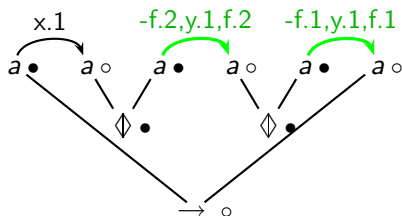


◀ back

# The Cut Elimination Procedure

## A Complex Reduction Step

- Example: Reducing the proof net of  $(\lambda f.\lambda x.f(f x))(\lambda y.y)$



◀ back