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

\[
\lambda\text{-term } e_1 \quad \xrightarrow{\text{normal}} \quad \lambda\text{-term } e_1' \quad \xleftrightarrow{\text{normal}} \quad \lambda\text{-term } e_2' \quad \xrightarrow{\text{normal}} \quad \lambda\text{-term } e_2
\]
A New Notion of Proof / Program Equivalence

λ-term $e_1$ \quad \quad λ-term $e_2$

proof net $p_1$ \quad \quad proof net $p_2$

normal $\lambda$-term $e'_1$ \quad \quad normal $\lambda$-term $e'_2$

normal $\lambda$-term $e'_1$ \quad \quad normal $\lambda$-term $e'_2$

equivalence $=$ inlining + modular structure

normal proof net $p'_1$ \quad \quad normal proof net $p'_2$
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.
- 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 \((\neg, \rightarrow, \land, \lor)\).

- Criticism (e.g. by Heyting): Is “\(i = 5, \text{ if } A \text{ is true, and } i = 4, \text{ if } A \text{ is false}\)” a well-formed definition?

- Similar problem in programming: “\(i = 5, \text{ if program } P \text{ terminates, and } i = 4, \text{ if } P \text{ does not terminate}\)”

- Proposed solution: restrict classical reasoning by excluding the \textit{tertium non datur} principle.

- This yields \textit{intuitionistic logic}, the logical framework for functional programming.

- We will (for now) only consider the purely implicational fragment!
What is Intuitionistic Logic?

- Starting point: classical propositional logic. Formulas consist of propositional variables ($a, b$) and boolean connectives ($\neg, \to, \land, \lor$).

- 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 implicational 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
Functional programming is about modeling functions.

Syntax of λ-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 λ-term.

Example:

<table>
<thead>
<tr>
<th>λ-term</th>
<th>type</th>
<th>formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>λx.x</td>
<td>a → a</td>
<td>a → a</td>
</tr>
</tbody>
</table>
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 \( f : a \to a \).
Proof Nets
Nets and λ-Terms

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

```
\[ f, f, x \quad \text{to} \quad a \circ \quad a \bullet \quad a \circ \quad a \bullet \quad a \circ \]
```

\[ \rightarrow \circ \quad \rightarrow \circ \]
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 $f : a \to a$. 

$$f (f x)$$
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 $f : a \to a$. 

$$\lambda x. f \,(f \,x) \quad \text{and} \quad a \bullet \quad a \bullet \quad a \bullet \quad a \bullet \quad a \bullet \quad a \bullet$$
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 $f : a \to a$. 

\[ \lambda x. f\ (f\ x) \]
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 $f : a \rightarrow a$. 

\[
\lambda x. f (f x)
\]
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 λ-Terms

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

\[
\lambda f. \lambda x. f \; (f \; x)
\]
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 $f : a \to a$. 

\[ \lambda f. \lambda x. f \ (f \ x) \]
Proof Nets
Nets and λ-Terms

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

This translation function is “almost injective”.
- Nets emerging from closed λ-terms are called proof nets.
Question: What kinds of properties distinguish proof nets?
**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:**

$$\lambda f.\lambda x. f\ x$$
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.
**Theorem**

*All proof nets are classically correct.*

Proof idea:

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

\[ e = \lambda v_1 \ldots \lambda v_n . v_i \]
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:

\[ e = \lambda v_1 \ldots \lambda v_n . e_1 e_2 \]
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:

\[ e = \lambda v_1 \ldots \lambda v_n \cdot e_1 \; e_2 \]

Consider \( e_i' = \lambda v_1 \ldots \lambda v_n \cdot e_i \).
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 \((\lambda f.\lambda x.f(f x))(\lambda y.y)\):

```
\begin{align*}
\bullet & \rightarrow a \circ \rightarrow \circ \\
\bullet & \rightarrow a \circ \rightarrow \bullet \\
\diamond & \rightarrow \circ \\
\end{align*}
```

**Theorem**

The number of paths in each proof net is finite.
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 \((\lambda f.\lambda x.f(f\,x))(\lambda y.y)\):

```
 a • a o
   → o
     ◊
   → ◊
```

**Theorem**

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

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

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

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 \((\lambda f. \lambda x. f(fx))(\lambda y.y)\):

\[
\text{a} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{a}
\]

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 $(\lambda f.\lambda x.f(fx))(\lambda y.y)$:

- Theorem

  *The number of paths in each proof net is finite.*
Definition
Path = series of links that are connected by cuts
   + a well-formedness condition

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

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.*
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 ($id := \lambda y.y$):

$$ (\lambda f. \lambda x.f (f x)) \ id \leadsto \lambda x.id \ (id \ x) \leadsto^* \lambda x.x $$
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 \( \lambda f. \lambda x. f \,(f \,x) \)
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 $\lambda f.\lambda x.f(f\,x)$
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 $\lambda f. \lambda x. f (f x)$
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 $\lambda f. \lambda x. f(fx)$

Diagram:

```
   x.1, -f.2
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
   a  a  a
   →   →   →
```
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 $\lambda f. \lambda x. f (f x)$
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 $\lambda f. \lambda x. f(fx)$

```
 a ◦
  v
 a •
```

```
 x.1, -f.2
 ↕
 a •
```

```
 f.2, -f.1
 ↕
 a •
```

```
 f.1
 ↕
 a ◦
```

⇒ complex example
<table>
<thead>
<tr>
<th>Proof Theory</th>
<th>Proofs in Intuitionistic Logic</th>
<th>Equivalence of Proofs</th>
<th>Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
**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 λ-terms and preserve some modularity information:

\[ \lambda f. \lambda x. (\lambda y.x)(f \ x) \xrightarrow{\beta} \lambda f. \lambda x.x \]

Proof net + ce

\[ \xrightarrow{\beta} \]

\[ \xrightarrow{\text{proof net + ce}} \]

\[ \xrightarrow{a \circ a \bullet a \bullet a \circ} \]

\[ \xrightarrow{\text{proof net + ce}} \]

\[ \xrightarrow{\beta} \]

\[ \xrightarrow{a \circ a \bullet a \bullet a \circ} \]

\[ \xrightarrow{\beta} \]

\[ \xrightarrow{a \circ a \bullet a \bullet a \circ} \]
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.
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

![Proof Net Diagram]

- 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 λ-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_σ.*
Jean Yves Girard.
Linear logic.

François Lamarche and Lutz Straßburger.
Naming proofs in classical propositional logic.

Lutz Straßburger.
From deep inference to proof nets.
Vincent Danos and Laurant Regnier.
The structure of multiplicatives.

François Lamarche.
Proof nets for intuitionistic linear logic I: Essential nets.

François Lamarche and Christian Retoré.
Proof nets for the Lambek calculus — an overview.
Thank you for your attention!
Interesting Nets

dneg

null

pair

$\pi_1$
Example: Reducing the proof net of \((\lambda f.\lambda x.f \,(f \,x))(\lambda y.y)\)
Example: Reducing the proof net of \((\lambda f.\lambda x.f (f x))(\lambda y.y)\)
Example: Reducing the proof net of \((\lambda f.\lambda x.f (f x))(\lambda y.y)\)
Example: Reducing the proof net of \((\lambda f.\lambda x.f(f x))(\lambda y.y)\)
Example: Reducing the proof net of \((\lambda f.\lambda x.f(fx))(\lambda y.y)\)
Example: Reducing the proof net of $(\lambda f. \lambda x. f (f x))(\lambda y. y)$
Example: Reducing the proof net of \((\lambda f. \lambda x. f (f x))(\lambda y. y)\)