Light Logics and Implicit Computational Complexity

Damiano Mazza
CNRS, UMR 7030, Laboratoire d'Informatique de Paris Nord
Université Paris 13, Sorbonne Paris Cité

Logoi Summer School
Turin, 31 August 2013

Implicit computational complexity

- From clocks to certificates:

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
    else:
       print '"];'
       children = []
       for  in n, childenumerate(ast[1:]):
          children.append(dotwrite(child))
       print ,'    %s -> {' % nodename
       for  in :namechildren
          print '%s' % name,
```

vs.

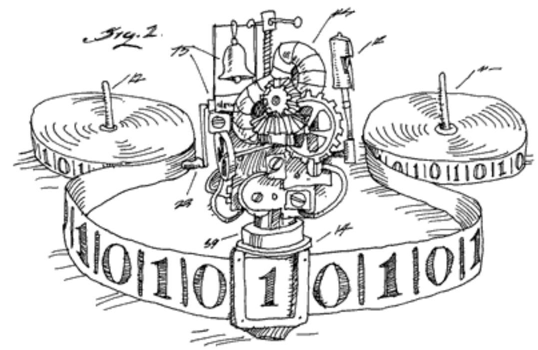

```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print '    %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
    else:
       print '"];'
       children = []
       for  in n, childenumerate(ast[1:]):
          children.append(dotwrite(child))
       print ,'    %s -> {' % nodename
       for  in :namechildren
          print '%s' % name,
```


- Ultimate goal: understanding why a program has a given complexity.

- *E.g.*: What does a polytime program look like?

An analogy: termination



- What does a terminating program look like?

- Subsumes an undecidable problem, OK, but it doesn't mean we can't:

  1. non-trivially characterize termination ($e.g.$ intersection types);
  2. find *decidable* criteria isolating an interesting subset of terminating programs ($e.g.$ simple types, `ML` polymorphism);
  3. find programming languages whose programs *intrinsically* terminate and which nevertheless have reasonable expressive power ($e.g.$ primitive recursive functions).

The polytime side of the analogy

Some of the things we will see:

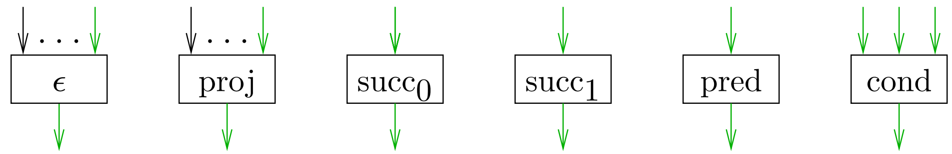
1. $d\ell\mathrm{PCF}$ by Dal Lago and Gaboardi (2011).

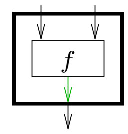
2. **DLAL** by Baillot and Terui (2004), STA (Gaboardi and Ronchi 2007), quasi-interpretations (Bonfante, Marion, Moyen 2007), . . .

3. λ-calculi based on light logics (Girard 1998, Lafont 2003), ramification and predicative recursion (Leivant 1991, Bellantoni and Cook 1992, Leivant and Marion 1993, Bellantoni, Niggl, Schwichtenberg 2000, . . .)

Example: Leivant via Bellantoni-Cook

- Idea: strings are both *data* (safe) and *recursion templates* (normal).

- Basic functions: 

- Closed under (well-sorted) composition, *lifting* 

and *predicative recursion on notation*:

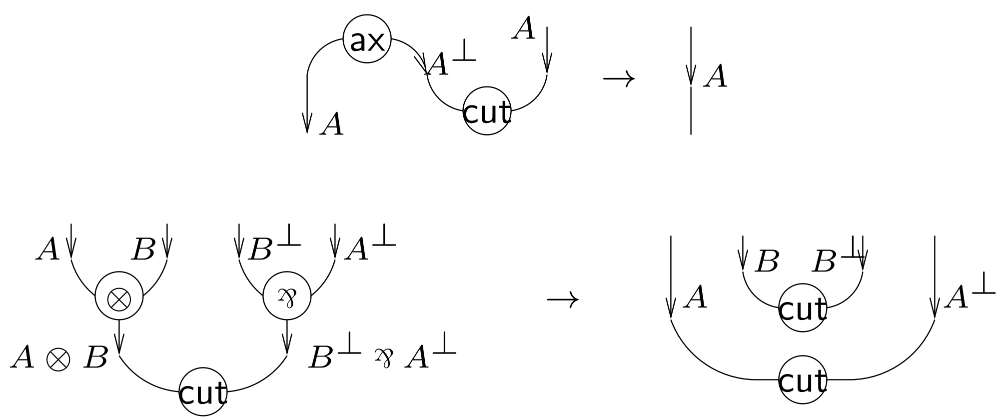
$$\mathrm{rec}[f_0, f_1, g](\epsilon, \vec{x}; \vec{a}) = g(\vec{x}; \vec{a})$$

$$\mathrm{rec}[f_0, f_1, g](zi, \vec{x}; \vec{a}) = f_i(z, \vec{x}; \vec{a}, \mathrm{rec}[f_0, f_1, g](z, \vec{x}; \vec{a}))$$

- Polytime functions: normal inputs to safe output.

A linear (and trivial) example: the affine λ-calculus

- Remember cut-elimination in multiplicative linear logic:



- Number of steps bounded by the size of the initial proof net.

- Affine λ-calculus: $t, u ::= x \mid \lambda x.t \mid tu \text{ s.t. } \mathrm{fv}(t) \cap \mathrm{fv}(u) = \emptyset$.

A parenthesis: the complexity/ies of $\mathbf{MLL}$

- Cut-elimination ($i.e.$, given two $\mathbf{MLL}$ proof nets, do they have the same cut-free form?) is $\mathbf{P}$-complete (Mairson and Terui 2003). We have basically seen that it is in $\mathbf{P}$; hardness is shown by encoding Boolean circuits in $\mathbf{MLL}$ proof nets.

- Interestingly, $\mathbf{MLL}$ cut-elimination *with atomic axioms* is in $\mathbf{L}$. The algorithm uses the geometry of interaction! What's hiding behind η?

- Correctness ($i.e.$, is an $\mathbf{MLL}$ proof structure a proof net?) is $\mathbf{NL}$-complete (de Naurois and Mogbil 2009). There is a correctness criterion the verification of which subsumes reachability.

- Provability ($i.e.$, is the $\mathbf{MLL}$ formula A provable?) is $\mathbf{NP}$-complete. Can you see why it is in $\mathbf{NP}$?

Naive set theory

- Terms: $t, u ::= x \mid \{x \mid A\}$
- Formulas: $A, B ::= t \in u \mid t \notin u \mid A \wedge B \mid A \vee B \mid \forall x.A \mid \exists x.A$
- One-sided classical sequent calculus $(\mathbf{LK})$, plus

$$\frac{\vdash \Gamma, A[t/x]}{\vdash \Gamma, t \in \{x \mid A\}} \in \qquad\qquad \frac{\vdash \Gamma, \neg A[t/x]}{\vdash \Gamma, t \notin \{x \mid A\}} \notin$$

- Standard cut-elimination rules, plus the obvious one for membership.

Russel's antinomy

- Define:

$$M := x \notin x,$$
$$r := \{x \mid M\},$$
$$R := \neg M[r/x] = r \in r$$

- We have

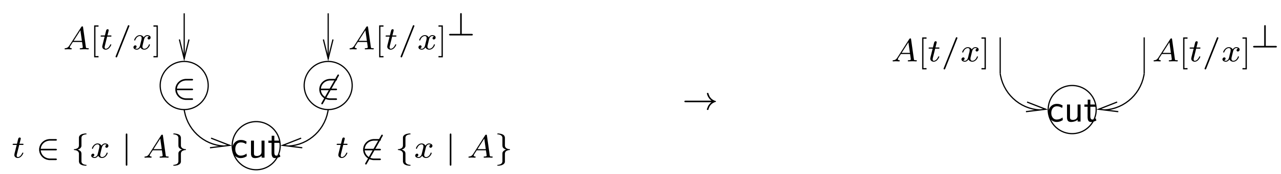
$$\dfrac{\dfrac{\dfrac{}{\vdash \neg R, R}}{\vdash \neg R, \neg R}\,\notin}{\vdash \neg R} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \neg R, R}}{\vdash R, R}\,\in}{\vdash R}$$
$$\overline{\qquad\qquad\vdash\qquad\qquad}$$

- As a consequence, cut-elimination does not terminate.

Naive set theory in MLL

- Terms: $t, u ::= x \mid \{x \mid A\}$
- Formulas: $A, B ::= t \in u \mid t \notin u \mid A \otimes B \mid A \parr B \mid \forall x.A \mid \exists x.A$
- Usual multiplicative proof nets (without units), plus

- Usual multiplicative cut-elimination rules, plus



No contraction, no contradiction

- Define M, r and R as before. It is still true that R is equivalent to $R^\perp$ ($i.e.$, $(R \multimap R^\perp) \otimes (R^\perp \multimap R)$ is derivable).

- However, the empty sequent is no longer derivable!

Why?

No contraction, no contradiction

- Define M, r and R as before. It is still true that R is equivalent to $R^\perp$ ($i.e.$, $(R \multimap R^\perp) \otimes (R^\perp \multimap R)$ is derivable).

- However, the empty sequent is no longer derivable!

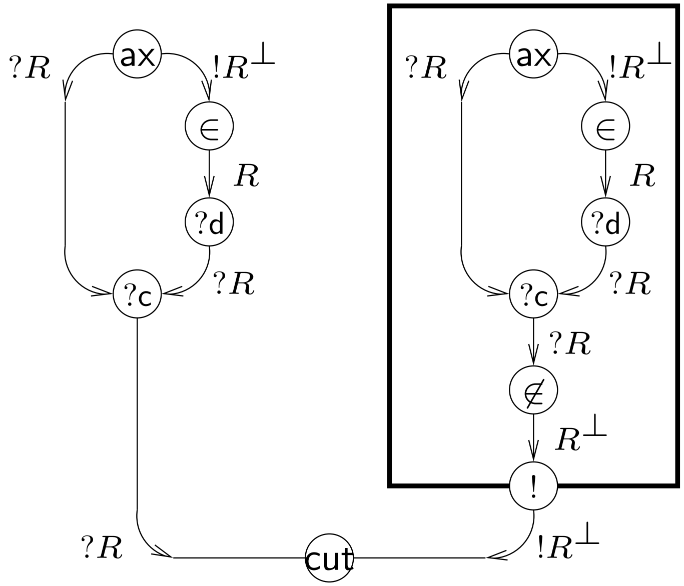
- Because cut-elimination holds by the usual argument:

size-decrease $+$ preservation of correctness

- Girard's insight: the key is *untyped* cut-elimination, $i.e.$, a cut-elimination proof not relying on formulas.

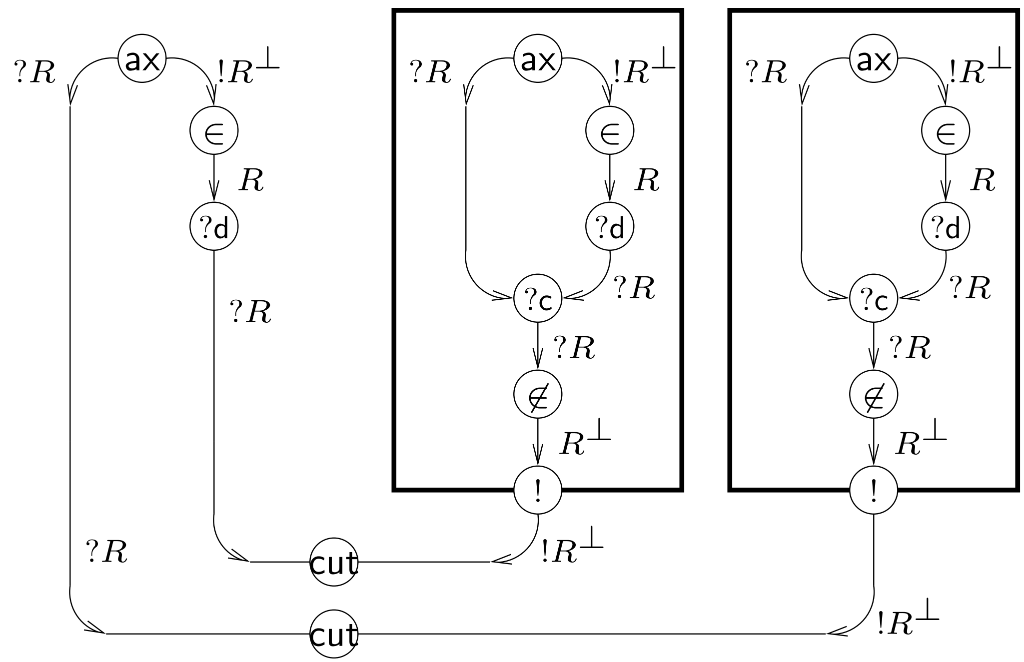
Russel's antinomy in MELL

- Remember the translation of classical negation in linear logic: R is equivalent to $!R^{\perp}$.



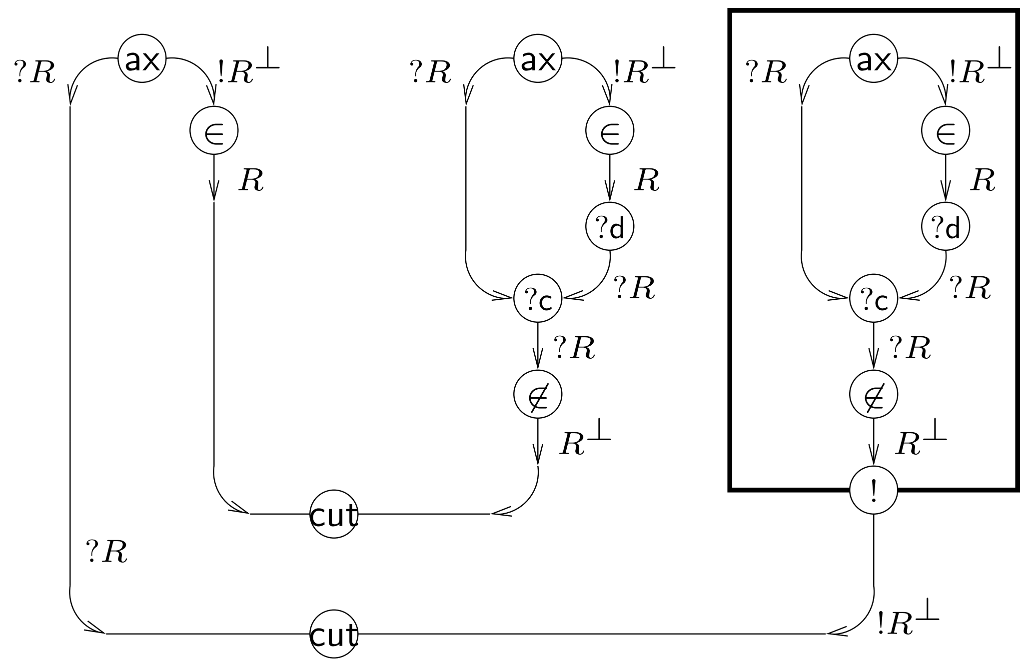
Russel's antinomy in MELL

- Remember the translation of classical negation in linear logic:
  R is equivalent to $!R^{\perp}$.



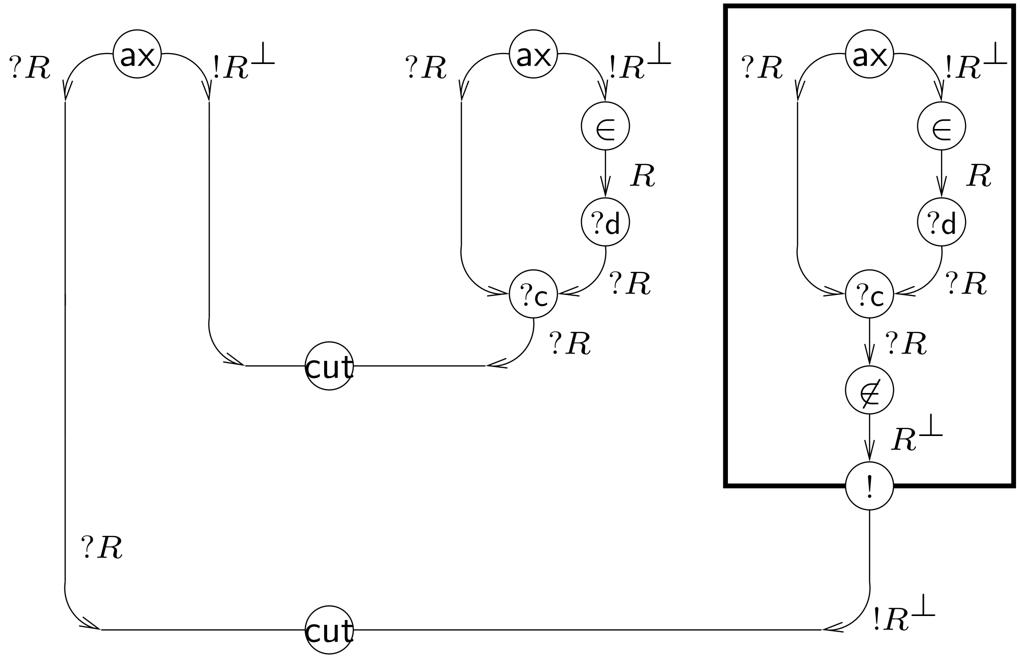
Russel's antinomy in MELL

- Remember the translation of classical negation in linear logic: R is equivalent to $!R^\perp$.



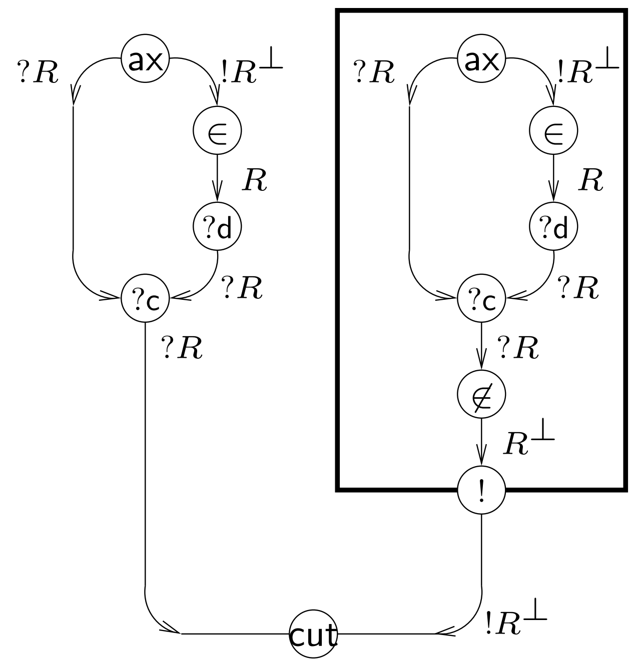
Russel's antinomy in $\mathbf{MELL}$

- Remember the translation of classical negation in linear logic: R is equivalent to $!R^\perp$.



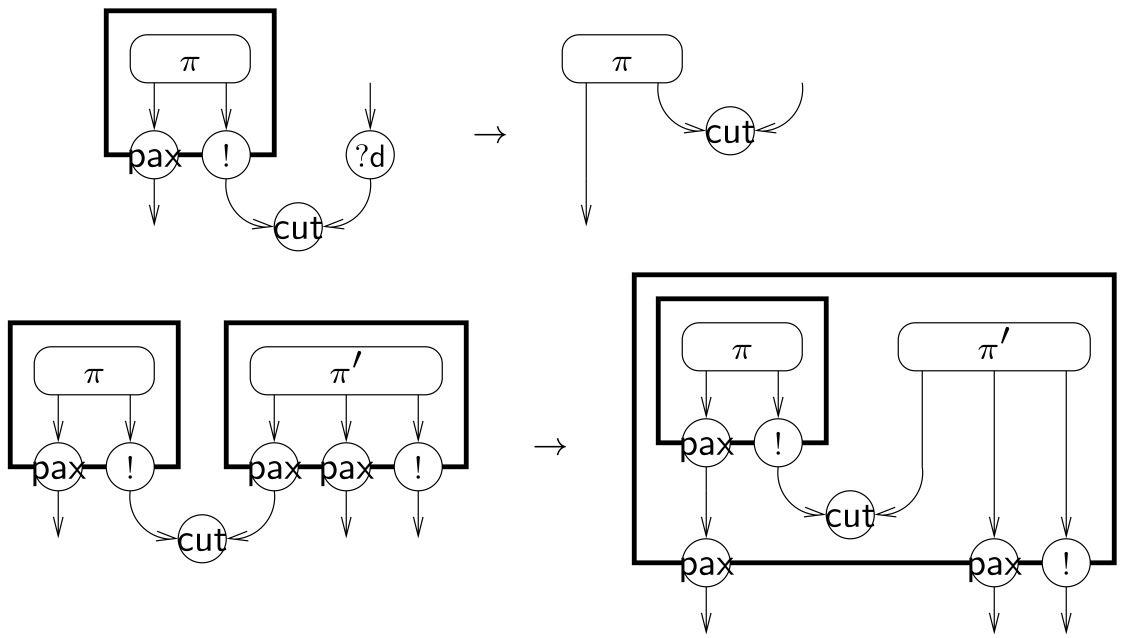
Russel's antinomy in MELL

- Remember the translation of classical negation in linear logic: R is equivalent to $!R^{\perp}$.



Opening boxes, boxing boxes

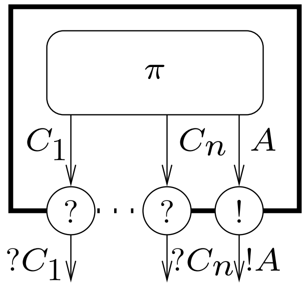
- Remember the *depth* of a proof net: it is the maximum number of boxes nested one into the other. It is altered by two cut-elimination steps:



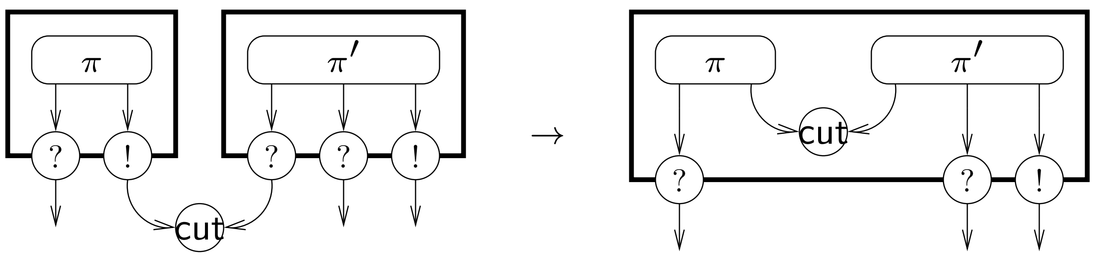
- Depth-changing is needed in Russel's antinomy!

ELL: functorial boxes

- We eliminate $?$d links, and replace boxes with *functorial boxes*:



- Cut-elimination does not alter the depth:



Untyped cut-elimination

- We consider 3 cut-elimination steps: axiom, multiplicative, exponential (contraction/weakening + functorial box).

- Let $|\pi|_i$ be the size of the proof net π at depth $i \geq 0$, and let $|\pi|_i = 0$ for all $i < 0$. If π has depth d, we define

$$\begin{array}{rccc} \alpha_\pi : & \mathbb{N} & \to & \mathbb{N} \\ & n & \mapsto & |\pi|_{d-n} \end{array}$$

- We see α_π as an ordinal $< \omega^\omega$ and verify that

$$\pi \to \pi' \qquad \text{implies} \qquad \alpha_\pi > \alpha_{\pi'}.$$

- Correctness is preserved, so we have (untyped) cut-elimination!

Quantifying the runtime

- When we operate at depth i, nothing happens at depth $j < i$. So, if π has depth d and normal form π', we may go "depth by depth":

$$\pi = \pi_0 \to^* \pi_1 \to^* \pi_2 \to^* \cdots \to^* \pi_n \to^* \pi_{n+1} = \pi', \qquad n \leq d$$

  - The length of $\pi_i \to^* \pi_{i+1}$ is bounded by $|\pi_i|$ (the size of π_i);
  - cut-elimination steps at most square the size of proof nets, so

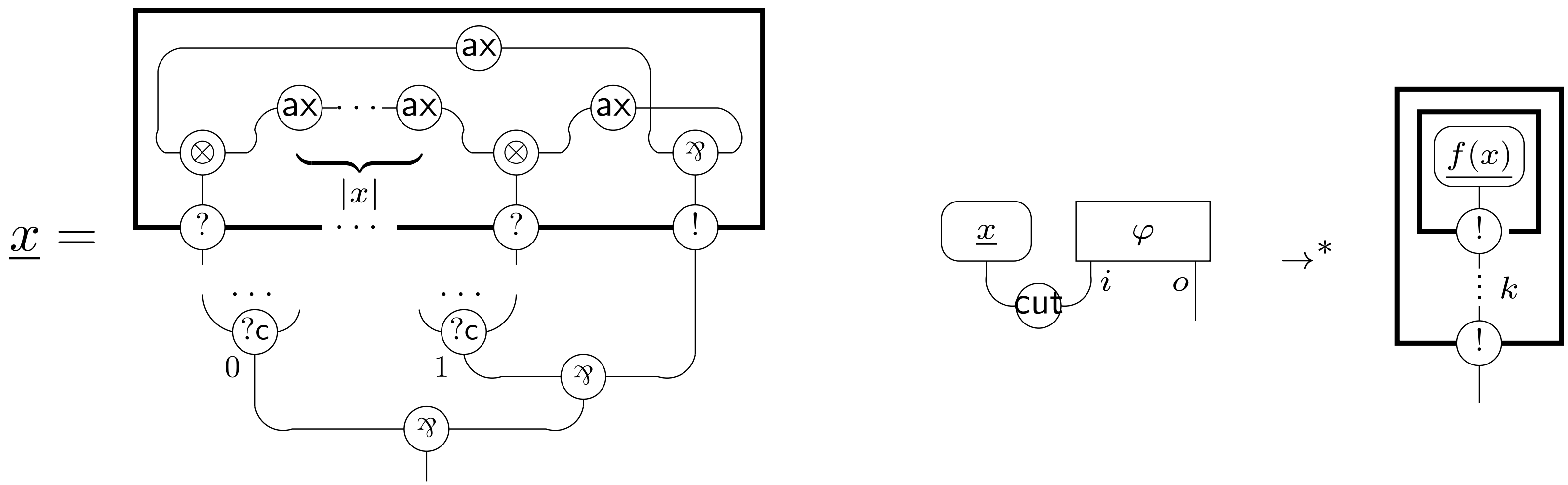
$$|\pi_{i+1}| \leq |\pi_i|^{2^{|\pi_i|}} \leq 2^{2^{2^{|\pi_i|}}} = 2_3^{|\pi_i|}$$

- Therefore, the total runtime is bounded by

$$\sum_{i=0}^{n} |\pi_i| \leq \sum_{i=0}^{n} 2_{3i}^{|\pi|} \leq (n+1) 2_{3n}^{|\pi|} \leq (d+1) 2_{3d}^{|\pi|}$$

Representing functions on strings

- A function $f : \{0,1\}^* \to \{0,1\}^*$ is *representable* in $\mathbf{ELL}$ if there are $k \geq 0$ and a proof net φ with two conclusions, i and o, such that



- In fact, we are using Church strings: $\lambda f_0.\lambda f_1.\lambda z.f_{i_1}(\ldots f_{i_n} z \ldots)$.

A characterization of elementary functions

- Let π be the proof net obtained by cutting φ with $\underline{x}$ on i.
  - $|\pi| = \Theta(|x|)$;
  - the depth of π does not depend on x.

- Cut-elimination on Turing machines has only a polynomial slowdown. Hence, all functions representable in $\mathbf{ELL}$ are elementary.

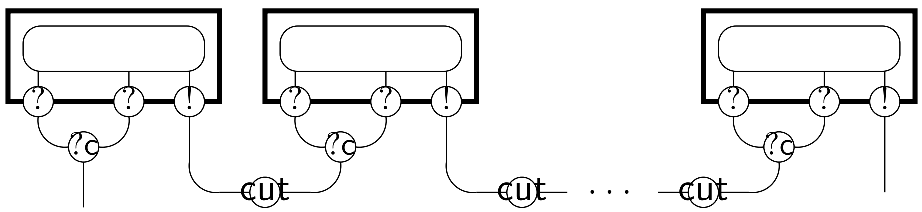
- Conversely, one may show that every elementary function may be represented in $\mathbf{ELL}$. Furthermore, we may restrict to intuitionistic second-order typable proof nets, of type $\mathbf{S} \vdash !^k\mathbf{S}$ for some $k \geq 0$, where

$$\mathbf{S} := \forall X.!(X \multimap X) \multimap !(X \multimap X) \multimap !(X \multimap X),$$

  which is a decoration of the system F type of Church binary strings.

LLL: forbidding exponential chains

- The exponential blow-up in the normalization of **ELL** is essentially due to configurations such as the following:



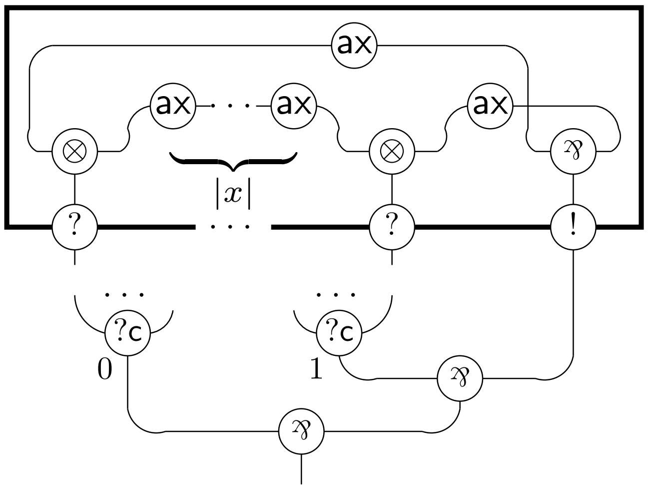
- **LLL** is defined by restricting to boxes with at most one auxiliary door.

- The total arity of contractions at depth i *does not increase* during cut-elimination at depth i. Therefore, $|\pi_{i+1}| \leq |\pi_i|^2$, and we get

$$\text{runtime} \leq \sum_{i=0}^{n} |\pi_i| \leq \sum_{i=0}^{n} |\pi|^{2^i} \leq (n+1)|\pi|^{2^n} \leq (d+1)|\pi|^{2^d}$$

A problem of expressiveness

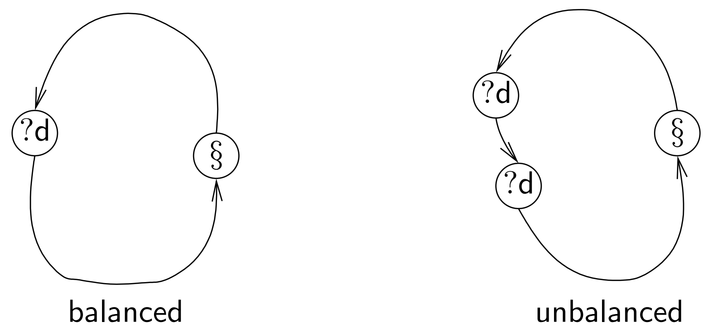
- Recall the representation of binary strings in $\mathbf{ELL}$:



- In $\mathbf{LLL}$, this only works for strings of length at most $1\ldots$

The paragraph

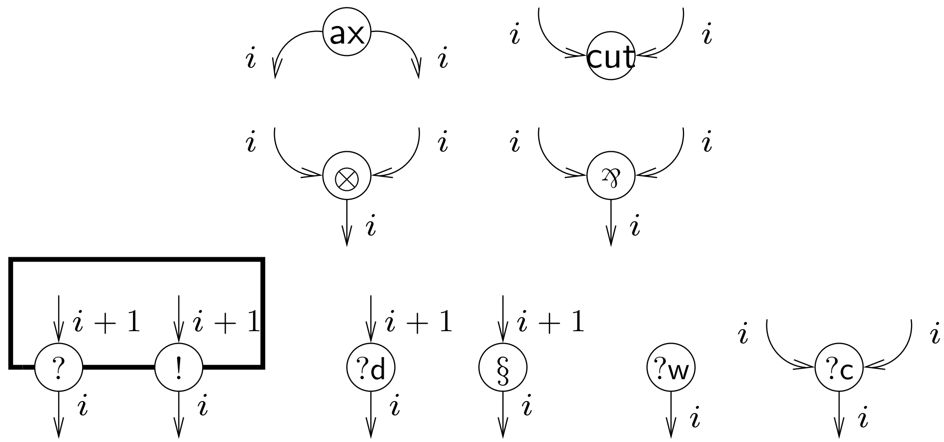
- We re-introduce $?\mathrm{d}$ links, plus a new unary link §.
- We define *balanced cycles* (ignoring switches!!!) by counting the number of $?\mathrm{d}$ and § links crossed going "up" and "down":



- A proof net with § links is *balanced* if all of its cycles are balanced (cycles are allowed to jump between conclusions).

Levels

- A proof net is balanced iff there exists a labelling of its links in $\mathbb{N}$ s.t.



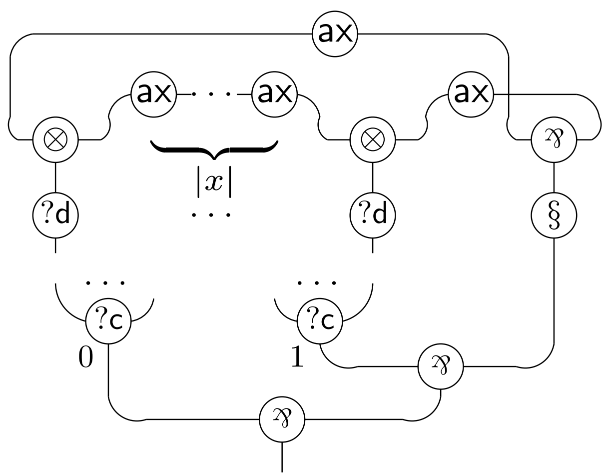
and all conclusions have the same label (Baillot and M. 2010, Boudes, M. and Tortora de Falco 2013).

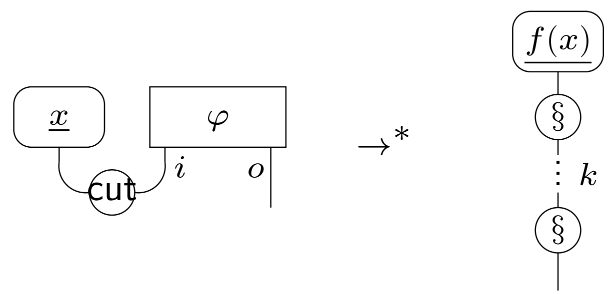
- This integer is the *level* of a link. It behaves very much like the depth.

Representing functions in $\mathbf{LLL}$

- A function $f : \{0,1\}^* \to \{0,1\}^*$ is *representable* in $\mathbf{LLL}$ if there are $k \geq 0$ and a proof net φ with two conclusions, i and o, such that

$\underline{x} =$





- We are using again Church strings, but with a different decoration.

A characterization of polytime functions

- Let π be the proof net obtained by cutting φ with $\underline{x}$ on i.
  - $|\pi| = \Theta(|x|)$;
  - the level of π does not depend on x.

- Cut-elimination on Turing machines has only a polynomial slowdown. Hence, all functions representable in $\mathbf{LLL}$ are polytime.

- Conversely, one may show that every polytime function may be represented in $\mathbf{LLL}$. Furthermore, we may restrict to intuitionistic second-order typable proof nets, of type $\mathbf{S}' \vdash \S^k\mathbf{S}'$ for some $k \geq 0$, where

$$\mathbf{S}' := \forall X.!(X \multimap X) \multimap !(X \multimap X) \multimap \S(X \multimap X),$$

  which is another decoration of the system F type of Church binary strings.

A word on the completeness proofs

- For $\mathbf{ELL}$, it is possible to use recursive-theoretic characterizations of elementary functions ($e.g.$ Danos and Joinet (2003) use Kalmar's: elementary functions contain constants, projections, addition, multiplication, equality test and are closed under composition, bounded sums and bounded products).

- For $\mathbf{LLL}$, it would be nice to use Bellantoni and Cook's characterization, but it doesn't work. So we do things "manually" (Girard 1998):
  - we show that $\mathbf{LLL}$ can encode one step of computation of arbitrary Turing machines;
  - we show that polynomials (on unary integers) are representable in $\mathbf{LLL}$;
  - so we have Turing machines with polynomial clocks, and we are done.

A word on representations

- Observe that the type of binary strings, both in $\mathbf{ELL}$ and $\mathbf{LLL}$, is *not* what you would obtain by applying Girard's (CbN) translation of intuitionistic logic into linear logic:

$$\forall X.!(!X \multimap X) \multimap !(!X \multimap X) \multimap !X \multimap X.$$

- In fact, let $\mathbf{PN}_\lambda$ denote the set of all $\mathbf{MELL}$ proof nets which are CbN translations of some λ-term, and let $\mathbf{PN}_{\mathbf{ELL}}$ be the set of $\mathbf{ELL}$ proof nets (embedded in $\mathbf{MELL}$ in the obvious way). Then

$$\mathbf{PN}_\lambda \cap \mathbf{PN}_{\mathbf{ELL}} = \emptyset.$$

- Moreover, there is no such thing as polarized $\mathbf{ELL}$, $\mathbf{LLL}$, etc.

Soft linear logic

- Replace the usual exponential rules of sequent $\mathbf{LL}$ calculus

$$\frac{\vdash ?\Gamma, A}{\vdash ?\Gamma, !A} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}$$

with

$$\frac{\vdash \Gamma, A}{\vdash ?\Gamma, !A}\ \text{functorial promotion} \qquad \frac{\vdash \Gamma, \overbrace{A, \ldots, A}^{n}}{\vdash \Gamma, ?A}\ \text{multiplexing}$$

- Untyped cut-elimination in $\mathcal{O}(s^d)$ steps (size s, depth d), with a marvelously simple proof. Entails polytime soundness.

- By contrast, proving polytime completeness is tricky. As a programming language, $\mathbf{SLL}$ is far from user friendly. . .

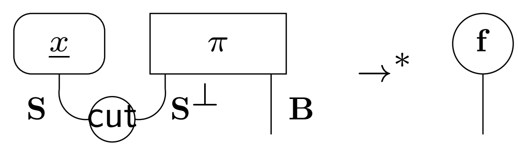
A word on diagonalization

- How do we separate primitive recursive sets from recursive sets? Diagonalization, of course:

$$\{x \in \{0,1\}^* \mid \exists P \text{ prim. rec. } x = \ulcorner P \urcorner \text{ and } P(x) = \text{false}\} \in \mathbf{R} \setminus \mathbf{PR}$$

- What happens if we diagonalize $\mathbf{P}$? The set

$$\Bigg\{ x \in \{0,1\}^* \mid \exists \pi \in \mathbf{SLL}.\ x = \ulcorner \pi \urcorner \text{ and}$$



$$\Bigg\}$$

cannot be in $\mathbf{P}$ by construction. Can you show an upper bound to its complexity? (Following the recursion-theory analogy, it should be in $\mathbf{NP} \cap \mathbf{coNP}$, but probably it is not even in $\mathbf{PSPACE}$. . .).

Dual light affine logic

- Recall how, in $\mathbf{LLL}$, we may actually restrict to intuitionistic, second-order typed proof nets, $i.e.$, λ-terms. The following type system is due to Baillot and Terui (2004), using Barber and Plotkin (1997):

$$\frac{}{\Theta; x : A \vdash x : A}$$

$$\frac{\Theta; \Gamma, x : A \vdash t : B}{\Theta; \Gamma \vdash \lambda x.t : A \multimap B} \qquad \frac{\Theta; \Gamma \vdash t : A \multimap B \quad \Theta; \Delta \vdash u : A}{\Theta; \Gamma, \Delta \vdash tu : B}$$

$$\frac{\Theta, z : A; \Gamma \vdash t : B}{\Theta; \Gamma \vdash \lambda x.t : A \Rightarrow B} \qquad \frac{\Theta; \Gamma \vdash t : A \Rightarrow B \quad ; x : C \vdash u : A}{\Theta \cup z : C; \Gamma \vdash tu : B}$$

$$\frac{; \Gamma, \Delta \vdash t : A}{\Gamma; \S\Delta \vdash t : \S A} \qquad \frac{\Theta; \Delta \vdash u : \S A \quad \Theta; \Gamma, x : \S A \vdash t : B}{\Theta; \Gamma, \Delta \vdash t[u/x] : B}$$

Dual light affine logic

$$\frac{\Theta; \Gamma \vdash t : A}{\Theta; \Gamma \vdash t : \forall X.A} \qquad \frac{\Theta; \Gamma \vdash t : \forall X.A}{\Theta; \Gamma \vdash t : A[B/X]}$$

Theorem. *The functions definable by λ-terms of type $\mathbf{S}' \multimap \S^k \mathbf{S}'$ in $\mathbf{DLAL}$ are exactly the polytime functions.*

- However, there's an issue of *intensional expressiveness*: although every polytime function $f : \{0,1\}^* \to \{0,1\}^*$ admits a $\mathbf{DLAL}$-typable λ-term t computing it, t is most likely to be very contrived, $i.e.$, it may look nothing like the λ-term you would write to compute f.

- A system with similar properties, STA (Soft Type Assignment), based on affine $\mathbf{SLL}$ instead of $\mathbf{LLL}$, was introduced by Gaboardi and Ronchi Della Rocca (2007). It suffers from a similar problem.

The sub-elementary hierarchy within $\mathbf{ELL}$

- Baillot (2011) has shown how, using fixpoints in (affine) $\mathbf{ELL}$ types, one may obtain the following characterization (we stipulate 0-$\mathbf{EXP}$=$\mathbf{P}$):

  Theorem. n-$\mathbf{EXP}$ *(with* $n \geq 0$*) is the class of languages decidable by* $\mathbf{ELL}$ *proof nets of type* $!\mathbf{S} \vdash !^{2+n}\mathbf{B}$*, where* $\mathbf{B} = \forall X.X \multimap X \multimap X$*.*

- Later, Laurent has shown how to obtain the same characterization in the untyped framework (which was our choice in this lecture).

- The idea is that, to know the value of a Boolean (the “answer”) one may stop normalizing at the depth where the Boolean is.

The categorical perspective

- Quick recap on categorical models of $\mathbf{MELL}$:
  - a $*$-autonomous category $(\mathcal{L}, \otimes, 1, \perp)$;
  - a monoidal comonad $(!, \mathsf{dig}, \mathsf{der})$ on $\mathcal{L}$...
  - ... such that every $!A$ is a commutative comonoid;
  - (and the free !-coalgebra and the comonoid structure interact nicely).

- A model of $\mathbf{ELL}$ drops the condition that ! is a comonad. The ! functor of $\mathbf{LLL}$ further drops the monoidality requirement.

- Paradox: although being a model of $\mathbf{ELL}$ is "easier", in practice it is hard to find a *strict* one! In fact, the simplest way to model exponentials is to construct $!A$ as the free commutative comonoid, which automatically yields a model of $\mathbf{MELL}$ (a *Lafont category*, as most practical models).

Objects with involutions

- Let $\mathcal{C}$ be your favorite category. An *object with involutions* is a pair (A, s) such that A is an object of $\mathcal{C}$ and $s = (s_k)_{k \in \mathbb{Z}}$ is a family of involutions of A ($i.e.$, $s_k \circ s_k = id_A$ for all $k \in \mathbb{Z}$).

- A morphism between objects with involutions (A, s), (B, t) is a morphism $f : A \to B$ of $\mathcal{C}$ such that $t_k \circ f \circ s_k = f$ for all $k \in \mathbb{Z}$.

- Objects with involutions of $\mathcal{C}$ and their morphisms form a category $\mathrm{Inv}\mathcal{C}$. Moreover, if $\mathcal{C}$ is a model of $\mathbf{MELL}$, then so is $\mathrm{Inv}\mathcal{C}$.

- Define an endofunctor of $\mathrm{Inv}\mathcal{C}$ by $\S(A, s) = (A, (s_{k-1})_{k \in \mathbb{Z}})$, and acting as the identity on morphisms. If we define $!' = ! \circ \S$, we obtain a strict model of $\mathbf{ELL}$ (plus paragraph): $!'$ is a monoidal functor which is not a comonad but such that $!'A$ is a commutative comonoid.

Further reading

- Characterization of space classes: $\mathbf{PSPACE}$ (Gaboardi, Marion, Ronchi 2008), $\mathbf{L}$ (Schöpp 2007). The classes $\mathbf{L}$ and $\mathbf{coNL}$ have also being characterized using GoI5 (von Neumann algebras) by Girard (2010) and Aubert and Seiller (2013).

- Systems related to bounded linear logic (Dal Lago and Hofmann 2010, Dal Lago and Gabardi 2011).

- Sematic proofs of soundness, via intuitionistic realizability (Dal Lago and Hofmann 2008) or classical (Krivine's) realizability/forcing (Brunel 2013).

- Tons of other stuff, just ask me.