Principal types as lambda-nets

Pietro Di Gianantonio, Marina Lenisa
we present a (strict) connection between

- principal types for untyped $\lambda$-terms
- cut-free $\lambda$-nets (proof nets associated to untyped $\lambda$-term)

show that the connections can be used to obtain

- a TAS for principle types
- alternative proofs and explanations of some results relating types to computation
A \( \lambda \)-term \( M \) has principal type \( \tau \) if

- the set of instantiations of \( \tau \) coincides with
- the set of types \( \sigma \) assignable to \( M \)

\[ \vdash M : \sigma \] (in a suitable TAS)

Principle types synthesise by the typing algorithm of ML, Haskell

Here we consider intersection type (\( \rightarrow, \wedge \))

- in most approaches, principle types defined on \( \lambda \)-terms in normal form.
- the synthesis of principle intersection type for arbitrary \( \lambda \)-terms is complex [Ronchi88] [Kfoury,Wells02]
Lambda-nets

Proof-nets for $\lambda$-terms

- a $\lambda$-term $M$ represents a proof $\pi$
- $\pi$ can be represented also by proof-net $P$
- just some proof-nets in special form are obtained in this way
- normalisation of $M$ (roughly) correspond to cut-elimination in $P$
Correspondence illustrate by an example

Take the Church number 2 $\lambda f. \lambda x. f(fx)$

the corresponding proof nets is:
The principle type is obtained by replacing each:

- $\otimes$ and $\bigotimes$ by $\to$
- dereliction and promotion by $!$
- contraction by intersection
- axiom by two instances of the same variable
- box by symbols to representing box boundaries
The result is:

That is: \(! (\Box j \! \alpha \to \beta) \land \Box j \! (\Box i \! \alpha \to \beta) \to (\Box j \Box i \! \alpha \to \gamma)\)

normally written as: \(((\beta \to \gamma) \land (\alpha \to \beta)) \to (\alpha \to \gamma)\)
To design an algorithm to synthesise principle types

- principle type are primarily defined on $\lambda$ terms in normal forms
- main problem: if $M : \tau$ and $N : \sigma$, what is the principle type of $MN$?

A first possible solution:

- from $\tau$ and $\sigma$, derive the lambda-nets $\pi_1$ and $\pi_2$,
- perform cut elimination on the composition of $\pi_1, \pi_2$,
- from the normalize lambda-net derive the principle type of $MN$

A second solution:

- mimic cut-elimination on the principle types
An alternative view for normalisation

If

- $\vdash M : \tau_1 \rightarrow \tau_2$
- $\vdash N : \sigma$

The principle type of $MN$ is obtained by:

- defining the MGU $U$ between $\tau_1$ and $\sigma$
  - making the two types equal by
    - instantiating variable (like in standard MGU)
    - duplicating boxes
  - return the application of $U$ to $\tau_2$

Normalisation as an MGU algorithm
Further consequences

Relate types with computations

- $M : \tau$ (standard type) iff
- $M$ has a principle type iff
- cut-elimination on the $\lambda$-net for $M$ converges
- $M$ is normalizable
Therefore:

- if $M$ is typable, then it is normalisable
- subject reduction holds
  - reduction preserve principle types
to find a type for a term is as complex as normalising it
  one build a type that is an expansion of the normal form

- types contain information about the number of reduction steps in normalisation
  normalisation is related to cut-elimination and the number of steps cut-elimination depends on complexity of the formula
- inhabitability of types is decidable, and related to correctness of proof structures
  a type as only a finite set of possible candidates as principle type, check for any of them if it is the representation of a term
Further consequences

Synthesise a type system for call-by-value $\lambda$-calculus

Consider:

- the encoding of the call-by-value $\lambda$-calculus in LL, through $\lambda$-nets $O \sim!(O \rightarrow O)$
- the corresponding principle types, through the above translation
- the set of types instantiation of principle types
Thanks for your attention