Curry-Howard Correspondence for Classical Logic

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Stéphane Graham-Lengrand

CNRS, Laboratoire d'Informatique de l'X

`Stephane.Lengrand@Polytechnique.edu`

Lecture IV
Classical Realisability

Overview

def. by induction on t	def. by induction on A	
t *is of type* A	t *realises* A	t is a proof of A
$t : A$	$t \Vdash A$	
Typing	Realisability	

Proof-Search — Curry-Howard corr.

Computational interpretation of logic

Like typing, realising is a relation between terms and formulae

Example

Consider a closed term t:

$\vdash t : A \to B$ if $t = \lambda x.t'$ with $x : A \vdash t' : B$

or $t = t_1\ t_2$ with $\begin{cases} t_1 : C \to A \to B \\ t_2 : C \\ \text{for some } C \end{cases}$

have to consider terms with free variables

$t \Vdash A \to B$ if for all t' such that $t' \Vdash A$, we have $t\ t' \Vdash B$

may consider closed terms only

Why is this interesting?

Typing has been sold to you for safety:

"Well-typed programs cannot go wrong" (Milner)

Hence the expression "**type-safety**"

We could argue that in fact, we do not care about typing, but realisability:

When implementing a function from integers to integers,

we do not care whether our code t satisfies $t : \texttt{int} \to \texttt{int}$

(in other words, whether $t = \lambda x.t'$ with... or $t = t_1\ t_2$ with...)

But what we really care about is whether,

when applying t to an integer, we compute a integer.

In other words, whether $t \Vdash \texttt{int} \to \texttt{int}$

So, why did we ever do typing?

Well, because realisability is undecidable

(given t and A, determining whether $t \Vdash A$)

whereas typing is (usually) decidable (with exceptions like Curry-style System F, etc)

(given t and A, determining whether $t : A$)

BUT typing implies realising:

$$\text{if } t : A \text{ then } t \Vdash A$$

It is the **Adequacy Lemma**:

typing is about syntax, realisability is about semantics

(writing $t \Vdash A$ for $t \in [\![A]\!]$, using notation of previous lectures)

A slogan:

Typing is a decidable approximation of realisability

Origins

Introduced by Kleene to formalise the Brouwer–Heyting–Kolmogorov interpretation of intuitionistic logic

$t \Vdash A_1 \wedge A_2$	if $t = \langle t_1, t_2 \rangle$ with $t_1 \Vdash A_1$ and $t_2 \Vdash A_2$
$t \Vdash A_1 \vee A_2$	if $t = \mathrm{inj}_i(t')$ with $t' \Vdash A_i$ for $i = 0$ or $i = 1$
$t \Vdash A_1 \rightarrow A_2$	if t is a computable function such that, whenever $u \Vdash A_1$, $t(u) \Vdash A_2$
$t \Vdash \exists x A(x)$	if $t = \langle a, t' \rangle$ with a an element of the "model" and $t' \Vdash A(a)$
$t \Vdash \forall x A(x)$	if t is a computable function such that, for all elements a of the "model", $t(a) \Vdash A(a)$

Parameterised by a way to interpret atomic formulae

t ranges over mathematical objects such as pairs, computable functions, etc

can be implemented as a number

(ok for pairs, injections, & computable functions can be assigned their Gödel numbers)

can be implemented as an untyped λ-term (untyped λ-calculus being Turing-complete)

there comes Curry-Howard correspondence

A few remarks

But typed λ-calculus is not Turing-complete:

if we only use typed λ-terms as realisers, we are missing some computable functions,

and hence some potential realisers!

Also, think of how to realise $\forall x^S A(x)$ and how to prove it:

In order to realise $\forall x^S A(x)$, we can, taking an inhabitant n of S as input,

give different realisers of $A(n)$ depending on n (in any computable way)

In order to prove $\forall x^S A(x)$, we need to produce a single proof, of $A(x)$

(i.e. a generic way of proving $A(n)$, not depending on n)

And what about classical logic

Origins are really about constructivism:

a realiser of $\exists x A(x)$ can only be a pair whose first component is a witness

a realiser of $A_1 \vee A_2$ can only be one of the 2 injections

Doing something similar in classical logic seems difficult

But, since Griffin's connection between control and classical proofs,

realisability has received renewed attention, mostly by Krivine et al.

Disclaimer:

Classical realisability only works for confluent restrictions of classical calculi

(e.g. CBV, CBN, polarity-based reduction)

Principles of classical realisability

- take an orthogonality relation $\perp$ between "proofs" and "counter-proofs" (i.e. between things that could be realisers), closed under anti-reduction
- define an interpretation of formulae using orthogonal constructions

$[A_1 \vee A_2]_\sigma$	$:=$	$\{\mathsf{inj}_i(t) \mid t \in [\![A_i]\!]_\sigma\}$	
$[\exists x A]_\sigma$	$:=$	$\{\langle a, t\rangle \mid t \in [\![A]\!]_{\sigma, x\mapsto a}\}$	
$[\![N]\!]_\sigma$	$:=$	$(\left[N^\perp\right]_\sigma)^\perp$	if N is $A_1 \wedge A_2$ or $\forall x A$
$[\![P]\!]_\sigma$	$:=$	$([P]_\sigma)^{\perp\perp}$	if P is $A_1 \vee A_2$ or $\exists x A$

By taking $t \Vdash A$ to mean $t \in [\![A]\!]$, **Adequacy** now works in classical logic too:

If $\vdash_c t : A$ (t classical proof of A), then **for any** $\perp$ (closed under anti-reduction) $t \Vdash A$

Remarks

- Syntax for t deliberately left abstract, but can use Curien-Herbelin-Wadler's calculus

  (see exercise sheet)

- You can take realisers to not be terms themselves,

  but a semantic interpretation of terms (in a specific model)

- By picking such interpretations & the orthogonality relation,

  Adequacy can give you properties of typed terms, e.g. **Strong Normalisation**

  (Again: for those confluent restrictions of classical calculi such as CBV/CBN, etc

  Otherwise, more advanced technique required: symmetric reducibility candidates)

- Some properties lost (compared to intuitionistic realisability):

  Because we have taken orthogonals,

  From $t \Vdash A_1 \vee A_2$ we do not necessarily have t of the form $\mathsf{inj}_i(t')$ with $t' \Vdash A_i$

  From $t \Vdash \exists x A(x)$ we do not necessarily have t of the form $\langle a, t' \rangle$ with a witness and ...

  Witness extraction fails in classical realisability (as expected)...

  ...unless $A(x)$ is of a particular form! (see exercise sheet)

Conclusion

Realisability is a semantical notion

- that is entailed by typing
- that can be adapted to classical logic,

  despite having been introduced for very constructivist motivations
- that relates to polarisation and focusing (see Dale's lectures)
- that allows to build models from other models to prove relative consistency theorems:

  To prove "Theory A is consistent if Theory B is consistent",

  it suffices to transform a given model of B into a model of A.

  Set theorists do this everyday with the notion of forcing: $p \Vdash A$ "p forces A"

  Krivine showed that realisability generalises forcing.

  With realisability, set theory axioms can be explained with computational notions

  (control, clock, global state and memory management, etc.)

Questions?