Logical Relations

Olek Gierczak

(1) What are they?
How are they used?

(2) How did they enter PL?

(3) How did we extend
their use beyond "toy"
languages?
- an answer: step indexing

What are Logical Relations? How are they used?

- A technique for proving Language properties:

  1) define a relation on terms by induction over types

     "family of Relations indexed by types"

  2) prove relevant terms are in relation

  3) prove terms in the relation have desired property

- Provide stronger induction hypotheses!

Ex. Normalization for STLC

$$e ::= x \mid \lambda x{:}\tau.\, e \mid e\ e \mid c$$

$$v ::= \lambda x{:}\tau.\, e \mid c$$

$$\tau ::= \tau \to \tau \mid \beta$$

$$\Gamma ::= \cdot \mid \Gamma, x{:}\tau$$

$$\vdots$$

$$(\lambda x{:}\tau.\, e)\ v \longrightarrow e[v/x]$$

$$\vdots$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.\, e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}$$

$$\vdots$$

Normalization:

All well-typed terms run to a value.

Or, if $\vdash e : \tau$, then

$\exists v.\ e \rightarrow^* v$

Let's try a proof by induction over $\vdash e : \tau$.

Lambda Case: $\boxed{e = \lambda x : \tau_1 . e'}$

We know $\dfrac{x : \tau_1 \vdash e' : \tau_2}{\vdash e : \tau_1 \to \tau_2}$

e is a value, done!

Application Case: $\boxed{e = e_1\ e_2}$

We know: $\dfrac{\vdash e_1 : \tau_1 \to \tau_2 \qquad \vdash e_2 : \tau_1}{\vdash e_1\ e_2 : \tau_2}$

We want to show:

$$\exists v.\ e_1 e_2 \longrightarrow^* v$$

By IH, we know

IH$_1$: $\exists v_1.\ e_1 \to^* v_1$

IH$_2$: $\exists v_2.\ e_2 \to^* v_2$

So, $e_1\ e_2 \to^* v_1\ v_2$

Since $\vdash v_1 : \tau_1 \to \tau_2$,

$v_1 = \lambda x{:}\tau_1.\ e'$

So, $e_1 e_2 \to^* (\lambda x{:}\tau_1.\ e')\ v_2$

$\to e'[v_2/x]$??

Does this terminate?

We don't know!

Need Stronger Induction Hypothesis:

"well typed lambdas terminate when applied to values"

Logical Relations to the Rescue

Define a unary relation containing normalizing terms:

1) define a relation on terms by induction over types
2) prove relevant terms are in relation
3) prove terms in the relation have desired property

$\mathcal{E}[\![\tau]\!]$ = all well typed terms at type τ that run to a value

More formally:

$$\mathcal{E}[\![\tau]\!] = \{ e \mid \vdash e : \tau \ \wedge \ e \to^* v \ \wedge \ v \in \mathcal{V}[\![\tau]\!] \}$$

"e behaves like a normalizing term at type τ if:

- it is well typed
- it runs to a value
- the value behaves like a normalizing value at type τ"

$$\mathcal{V}[\![B]\!] \simeq \{ c \}$$

$$\mathcal{V}[\![\tau_1 \to \tau_2]\!] =$$

$$\{ \lambda x : \tau . e \mid \forall v \in \mathcal{V}[\![\tau_1]\!].$$

$$e[v/x] \in \mathcal{E}[\![\tau_2]\!] \}$$

"a lambda expression behaves like a normalizing value if applying it to normalizing values produces normalizing terms."

Yay! Induction!

Does the relation have an infinite regress?

Is it Well Founded?

Will we always reach a base case?

$$\mathcal{E}[\![\tau]\!] \simeq \{ e \mid \vdash e : \tau \;\wedge\; e \to^* v \;\wedge\; v \in \mathcal{V}[\![\tau]\!] \}$$

$$\mathcal{V}[\![B]\!] \simeq \{ c \}$$

$$\mathcal{V}[\![\tau_1 \to \tau_2]\!] \simeq \{ \lambda x{:}\tau.\, e \mid \forall e' \in \mathcal{V}[\![\tau_1]\!].\; e[e'/x] \in \mathcal{E}[\![\tau_2]\!] \}$$

$\tau_1 < \tau_1 \to \tau_2$ "strictly smaller"

$\tau_2 < \tau_1 \to \tau_2$

We're Good!

Normalization

- We've finished 1 by defining $\mathcal{E}$.
- 3 is easy, because it's baked in

1) define a relation on terms by induction over types
2) prove relevant terms are in relation
3) prove terms in the relation have desired property

- What about 2?

"Fundamental Property"

$$\Gamma \vdash e : \tau \Longrightarrow [\![\Gamma \vdash e : \tau]\!]$$

think: $e \in \mathcal{E}[\![\tau]\!] \triangleq [\![\vdash e : \tau]\!]$

- Lots of work to show, but trust me, it's true!

- Normalization is just one of many uses of Logical Relations in PL
  - Type Soundnes
  - Parametricity
  - Program Equivalence

.
.
.

Tait '67

Girard '72

How did LRs enter PL?

How did LRs enter PL?

- Came from proof theory and logic
  - Tait '67 - Normalization for STLC
  - Girard '72 - Normalization for System F
- different notation, different goals
  - calls it "Convertible"
  - interpretation of types in intuitionistic arithmetic - Tait
- not very relevant to my talk

Tait '67

Girard '72

Plotkin '77

How did LRs enter PL?

How did LRs enter PL?

Plotkin '77

- Plotkin introduced and studied PCF:
  - STLC + Recursion
    + Boolean operations
    + Integer Arithmetic
- Examined connections between domain semantics and operational semantics
- Used LR to show: "soundness"

domain equivalence $\Rightarrow$ operational equivalence

PCF:

$$e ::= x \mid \lambda x{:}\tau.\, e \mid e\ e \mid c \mid Y_\tau\ e$$

$$v ::= \lambda x{:}\tau.\, e \mid c$$

$$\tau ::= \tau \to \tau \mid B$$

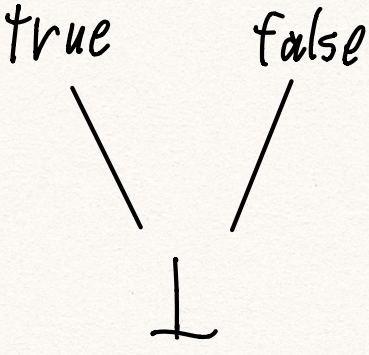
$$\Gamma ::= \cdot \mid \Gamma, x{:}\tau$$

$$Y_\tau\ e \to e\ (Y_\tau\ e)$$

$$\frac{}{\Gamma \vdash Y_\tau : (\tau \to \tau) \to \tau}$$

Domain Semantics for PCF

- Represent types as "domains"

eg. $[\![Bool]\!] =$ 

- $\bot$ "means" non termination

- Plotkin wanted to show

domain equivalence $\Rightarrow$ operational equivalence

which we get from

equal to constant $\Leftrightarrow$ runs to constant

- more formally, $\forall e, c,$

$$[\![e]\!] = [\![c]\!] \iff e \to^* c$$

The hard part is the $\Rightarrow$

How to prove? Logical Relation!

$$\mathcal{E}[\![B]\!] = \{e \mid \vdash e : B \ \wedge \ [\![e]\!] = c \Rightarrow e \to^* c\}$$

$$\mathcal{E}[\![\tau_1 \to \tau_2]\!] = \{e \mid \vdash e : \tau_1 \to \tau_2 \ \wedge \ \forall e' \in \mathcal{E}[\![\tau_1]\!], e\ e' \in \mathcal{E}[\![\tau_2]\!]\}$$

(Note, Call by Name)

Hard part of fundamental property, or

$$\vdash e : \tau \Rightarrow e \in \mathcal{E}[\![\tau]\!]$$

is showing

$$Y_\tau \in \mathcal{E}[\![(\tau \to \tau) \to \tau]\!]$$

ie want to show

$$\forall e \in \mathcal{E}[\![\tau \to \tau]\!],\ Y_\tau\ e \in \mathcal{E}[\![\tau]\!]$$

Problem:

$$Y_\tau\ e \to e\ (Y_\tau\ e)$$

Circularity!

Solution:

Stratify!

Consider all unrollings

$$\Omega_B \triangleq Y_B\,(\lambda x{:}C.\,x)$$

$$\Omega_{\tau_1 \to \tau_2} \triangleq \lambda y{:}\tau_1.\,\Omega_{\tau_2}$$

$$Y^0_\tau \triangleq \Omega_{(\tau \to \tau) \to \tau}$$

$$Y^{n+1}_\tau \triangleq \lambda f{:}(\tau \to \tau).\, f\,(Y^n_\tau\, f)$$

Plotkin shows:

$$[\![Y_\tau]\!] = \bigsqcup_n [\![Y_\tau^n]\!]$$

"the denotation of Y_τ is the combination of the denotations of all unrollings"

And uses this to complete the proof, $Y_\tau\ e \in \mathcal{E}[\![(\tau \to \tau) \to \tau]\!]$

Recap:

- Soundness of domain semantics
- method: $[\![e]\!] = [\![c]\!] \iff e \to^* c$

Take away:

We can resolve circularity by stratifying

Tait '67

Girard '72

Plotkin '77

Pitts/Stark '93

Pitts '98

Birkedal/Harper '99

How did we extend the use of LRs beyond "toy" languages?

What Happened After Plotkin?

- Attempts at more realistic language features / types:
  - Pitts-Stark '93
    - references
  - Pitts '98
    - quantified types
  - Birkedal/Harper '99
    - quantified + recursive
- All were either incomplete, or too complicated

Tait '67

Girard '72

Plotkin '77

Pitts/Stark '93

Pitts '98

Birkedal/Harper '99

Appel/McAllester '01

How did we extend the use of LRs beyond "toy" languages?

Towards "Real" Languages

- Appel / McAllester '01
- Examining STLC + recursive Type
- Provide Simple LR for Type Soundness
- Use step indexing, a form of stratification close to operational semantics

Recursive Types

We often work with lists:

> An Int List is one of:
> - Unit
> - Int x (Int List)

How do we type this?

Int List = Unit + (Int x (Int List))

Not a well defined Type!

Need Recursive Types

We write $\boxed{\mu\alpha.\tau}$ to mean

"α is a recursive type variable in τ"

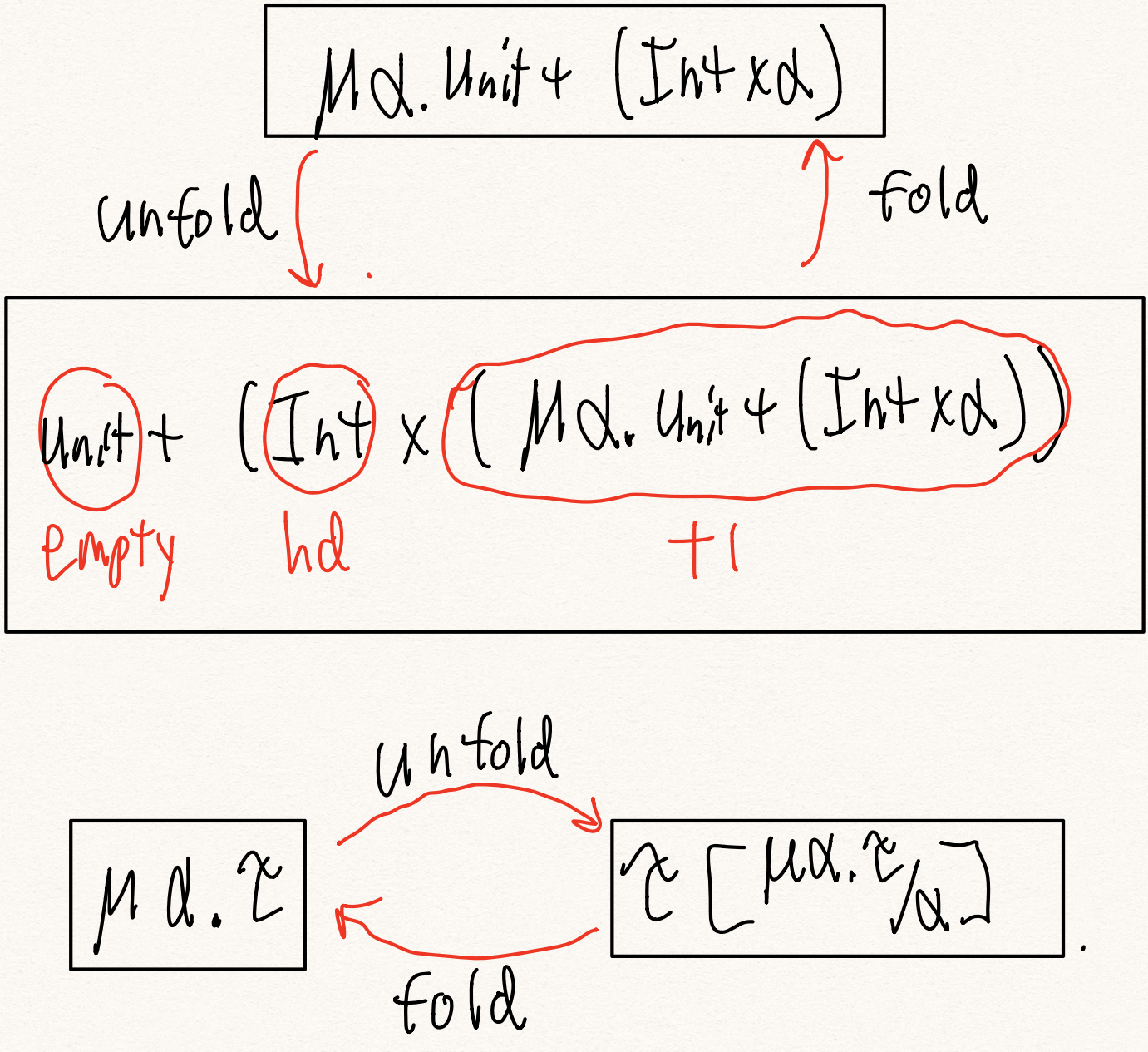
So our definition becomes:

$$\boxed{\text{Int List} = \mu\alpha.\text{Unit} + (\text{Int} \times \alpha)}$$

How do we construct expressions with type $\mu\alpha.\tau$?

We introduce "fold e" and "unfold e"

View change operations:



STLC with Recursive Types

$$e ::= x \mid \lambda x{:}\tau.\, e \mid e\ e \mid c \mid \text{fold}\ e \mid \text{unfold}\ e$$

$$v ::= \lambda x{:}\tau.\, e \mid c \mid \text{fold}\ v$$

$$\tau ::= \tau \to \tau \mid C \mid \mu\alpha.\, \tau$$

$$\Gamma ::= \cdot \mid \Gamma, x{:}\tau$$

$$\vdots$$

$$\text{unfold}\ (\text{fold}\ v) \longrightarrow v$$

$$\vdots$$

$$\frac{\Gamma \vdash e : \tau\,[\mu\alpha.\tau/\alpha]}{\Gamma \vdash \text{fold}\ e : \mu\alpha.\tau} \qquad \frac{\Gamma \vdash e : \mu\alpha.\tau}{\Gamma \vdash \text{unfold}\ e : \tau\,[\mu\alpha.\tau/\alpha]}$$

$$\vdots$$

Let's try a Type Soundness Logical Relation

$\mathcal{E}$, $\mathcal{V}[\![B]\!]$, $\mathcal{V}[\![\tau_1 \to \tau_2]\!]$ are the same.

What about $\mu\alpha.\tau$?

$$\mathcal{V}[\![\mu\alpha.\tau]\!] = \{ \text{fold } v \mid v \in \mathcal{V}[\![\tau[\mu\alpha.\tau/\alpha]]\!] \}$$

$\tau[\mu\alpha.\tau/\alpha] > \mu\alpha.\tau$

Problem: Circularity!

Appel/McAllester – Step Indexing

– Solution from Appel/McAllester '01:
Stratify the entire relation
by number of steps.

$e \in \mathcal{E}_k[\![\tau]\!]$ means

"e behaves like type
τ for k steps"

$v \in \mathcal{V}_k[\![\tau]\!]$ means

"when we use v, it
behaves like type τ for
k steps"

So how do we define $\mathcal{V}_k[\![\mu\alpha.\tau]\!]$?

"fold v behaves like type $\mu\alpha.\tau$ for k steps if unfold (fold v) behaves like type $\tau[\mu\alpha.\tau/\alpha]$ for k steps"

formally:

$$\mathcal{V}_k[\![\mu\alpha.\tau]\!] = \{\text{fold } v \mid \forall j \leq k, \text{unfold (fold } v) \in \mathcal{V}_j[\![\tau[\mu\alpha.\tau/\alpha]]\!]\}$$

Recap:

- Appel/McAllester prove Type Soundness for STLC with recursive types.

Take away:

- Circularity can be resolved by step indexing, or stratifying by number of steps

Tait '67

Girard '72

Plotkin '77

Pitts/Stark '93

Pitts '98

Birkedal/Harper '99

Appel/McAllester '01

Ahmed '04

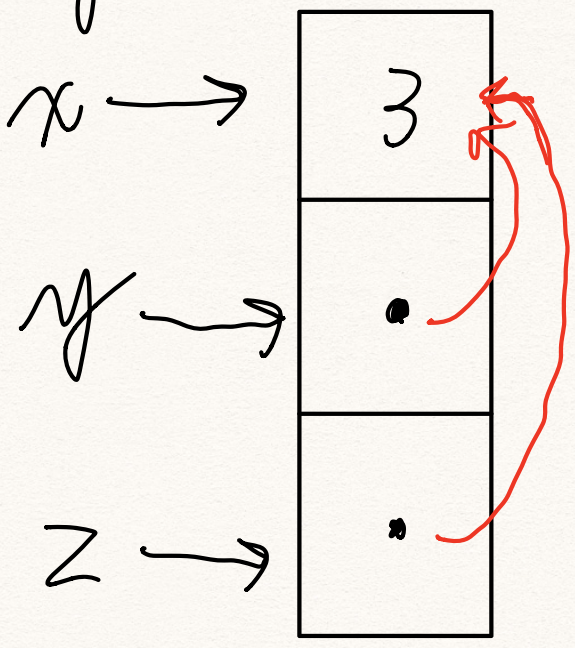
How did we extend the use of LRs beyond "toy" languages?

Logical Relations for "Real" Langs

- Ahmed '04 presents accessible techniques for designing and using LRs
  - (notation for this talk came from this thesis)
- develops Logical Relations for languages with mutable references
  - Solves Circularity with a step indexed model

How should we type reference cells?

- Should we allow type updates along with value updates?



x := new 3

x : ref Int

y := new x

y : ref (ref Int)

z := new x

z : ref (ref Int)

Now what if we update x to be a bool? **x := true**

- We'd need to do an alias analysis and update types for y, z

- Instead, we consider mutable references with **type invariance:**

  "The type of every allocated reference remains unchanged for the duration of the program"

- Seen in Java and ML

STLC with Mutable References

$$e ::= x \mid \lambda x{:}\tau.\, e \mid e\ e \mid c \mid l \mid \mathrm{new}(e) \mid !e \mid e := e$$

$$v ::= \lambda x{:}\tau.\, e \mid c \mid l$$

$$\tau ::= \tau \to \tau \mid \beta \mid \mathrm{ref}\ \tau$$

$$\Gamma ::= \cdot \mid \Gamma, x{:}\tau$$

$$S ::= l \overset{\text{fin}}{\to} v$$

$\overset{\text{fin}}{\to}$ as finite maps

$$\vdots$$

$$\frac{l \notin \mathrm{dom}(S)}{(S, \mathrm{new}\ v) \to (S[l \to v], l)}$$

$$\frac{l \in \mathrm{dom}(S)}{(S, !l) \to (S, S(l))}$$

$$\frac{\ell \in dom(S)}{(S, \ell := v) \to (S[\ell \to v], unit)}$$

How do we know what type $S(\ell)$ is?

answer: we track it.

$$\Psi ::= \ell \xrightarrow{fin} V[\![\tau]\!]$$

we model types as sets

What are the elements of the LR?

$$(S, \Psi, \nu)$$

"l has type $\text{ref}\ \tau$ if Σ maps l to τ and if S maps l to a value that behaves like a τ"

$$\mathcal{V}[\![\text{ref}\ \tau]\!] =$$

$$\{(S, \Sigma, l) \mid \Sigma(l) = \tau \wedge (S, \Sigma, S(l)) \in \mathcal{V}[\![\tau]\!]\}$$

before:

$\mathcal{V}[\![\tau]\!] \simeq$ values

how:

$\mathcal{V}[\![\tau]\!] \simeq$ Store $\times$ Store typing

$\times$ values

Store typing $\simeq$ locations $\times \mathcal{V}[\![\tau]\!]$

Problem: Circularity!

Solution due to Ahmed:

Step index the store typing:

$V_k[\![\tau]\!] \simeq$ Store $\times$ Store typing$_k$

$\times$ values

Store typing$_k \simeq$ locations $\times V_{k-1}[\![\tau]\!]$

$k-1 < k$

no circularity!

$$\mathcal{V}_k[\![\text{ref } \tau]\!] =$$

$$\{ (s, \Psi_{k-1}, l) \mid$$

$$\Psi_{k-1}(l) = \mathcal{V}_{k-1}[\![\tau]\!]$$

$$\wedge \; (s, \Psi_{k-1}, s(l)) \in \mathcal{V}_{k-1}[\![\tau]\!] \}$$

Recap:

- What was achieved?
  - Type Soundness LR for STLC with mutable refs!

- Plenty of details skipped

  eg:

  - need to show type hierarchy is well founded

  - quantified types make this more complex

- See Ahmed '04 if interested

Summary

- Logical Relations are a proof technique for achieving a stronger induction hypothesis
- Useful for many different properties
- Started as only applicable to "toy" languages
- Step indexing brought LRs for real languages