Graded Types Parametricity:
Principles and Application to Abstract Interpretation

Flavien BREUVART

LIPN, Paris XIII

Chocola: 15 November 2016
Project CoGITARe:
Combining Graded and Intersection Types for the Analyses of Resources

ANR JCJC with funding
Begin in March 1 PhD and 1y posdoc

Treated Issue: Abstract Interpretation of Functional Programs
An old objective with no satisfactory solution

(Naive ?) Solution: Performing the Analysis at Type Level
Tried several times since the 90’s with mixed results

Modern approach: Graded, Intersection and Refinement Types
We will revisit old issues with a modern point of view

Today: Brief overview and focus in a specific issue and associated research directions
Overview

1. Abstract Interpretation
   - Abstract Interpretation for Dummies
   - In Functional Programming
   - Polymorphic Abstractions

2. Graded types
   - Idea and Bibliography
   - Issues
   - Resource-polymorphism

3. Parameterisation
   - Definitions
   - Nice properties

4. Oidification
   - Definitions
   - Model
Contents

1 Abstract Interpretation
   - Abstract Interpretation for Dummies
   - In Functional Programming
   - Polymorphic Abstractions

2 Graded types
   - Idea and Bibliography
   - Issues
   - Resource-polymorphism

3 Parameterisation
   - Definitions
   - Nice properties

4 Oidification
   - Definitions
   - Model
Abstract Interpretation for Dummies

**Objective**: bound the set of reachable states

Both lower and upper bounds (Safety, liveness, quantitative)
**Idea 1**: Represent the program as an automata with memory (state)

\[
y := 0 ; \\
\text{while } (x > 100) \{ \\
\quad x := x - 100 ; \\
\} \\
\text{while } (x > y) \{ \\
\quad y := y + 2x ; \\
\quad x := x - 1 ; \\
\} \\
\]

\[
x := x - 100 \\
x > 100 \\
x \leq 100 \\
x := x - 1 \\
x > y \\
x \leq y \\
y := y + 2 \\
y > y \\
\]
**Idea 1**: Represent the program as an automata with memory (state)

**New goal**: Propagate possible states to the end

```plaintext
y := 0;
while (x > 100) {
    x := x - 100;
}
while (x > y) {
    y := y + 2x;
    x := x - 1;
}
```

```
x, y ∈ ℕ

y := 0

? _______ x := x - 100

? _______ x ≤ 100

? _______ x := x - 1

? _______ y := y + 2

? _______ x > y

? _______ x ≤ y

? _______ x > 100

? _______ x := x - 1
```
Abstract Interpretation for Dummies

Idea 1: Represent the program as an automata with memory (state)

New goal: Propagate possible states to the end

\[
y := 0; \\
\text{while } (x > 100) \{ \\
\quad x := x - 100; \\
\} \\
\text{while } (x > y) \{ \\
\quad y := y + 2x; \\
\quad x := x - 1 \\
\} \\
\]

\[
x, y \in \mathbb{N} \\
y := 0 \\
y = 0, x \in \mathbb{N} \\
x := x - 100 \\
x \leq 100 \\
x := x - 1 \\
x > 100 \\
x > y \\
x \leq y \\
y := y + 2 \\
\]
Idea 1: Represent the program as an automata with memory (state)

New goal: Propagate possible states to the end
Abstract Interpretation for Dummies

**Idea 1**: Represent the program as an automata with memory (state)

**New goal**: Propagate possible states to the end

\[
\begin{align*}
&\text{\(y := 0\)}; \\
&\text{\textbf{while} (\(x > 100\)) \{} \\
&\quad \text{\(x := x - 100\)}; \\
&\text{\textbf{}} \}
\\
&\text{\textbf{while} (\(x > y\)) \{} \\
&\quad \text{\(y := y + 2x\)}; \\
&\quad \text{\(x := x - 1\)}; \\
&\text{\textbf{}} \}
\end{align*}
\]
Abstract Interpretation for Dummies

**Idea 1**: Represent the program as an automata with memory (state)

**New goal**: Propagate possible states to the end

**Issue**: Loops are fixpoints

\[
\begin{align*}
x, y & \in \mathbb{N} \\
y & := 0 \\
\text{while } (x > 100) \{ \\
    x & := x - 100 \\
\} \\
\text{while } (x > y) \{ \\
    y & := y + 2x \\
    x & := x - 1 \\
\}\end{align*}
\]
Abstract Interpretation for Dummies

Idea 2: The best set of states is the least fixpoint

\[ y := 0 ; \]
while \((x > 100)\) {
\[ x := x - 100 ; \]
} 
while \((x > y)\) {
\[ y := y + 2x ; \]
\[ x := x - 1 ; \]
}
Abstract Interpretation for Dummies

**Idea 2**: The best set of states is the least fixpoint

**New goal**: Find the least fixpoint

```plaintext
y := 0 ;
while (x > 100) {
    x := x - 100 ;
}
while (x > y) {
    y := y + 2x ;
    x := x - 1 ;
}
```

$x, y \in \mathbb{N}$

$y := 0$

$y = 0, x \in \mathbb{N}$

$y = 2, 0 < x \leq 100$

$x := x - 100$  $x \leq 100$

$x > 100$  $x > y$

$y = 0, x \leq 100$

$y := y + 2$

$x = y = 0$

$y = 0, 0 < x \leq 100$
Abstract Interpretation for Dummies

Idea 2: The best set of states is the least fixpoint

New goal: Find the least fixpoint

Issue: Too long ...

\[
x, y \in \mathbb{N}
\]

\[
y := 0
\]

\[
y = 0, x \in \mathbb{N}
\]

\[
x := x - 100
\]

\[
y = 0, x > 100
\]

\[
y = 2, 0 < x \leq 100
\]

\[
y := y + 2x
\]

\[
x := x - 1
\]

\[
x := x - 1
\]

\[
x \leq y
\]

\[
x > y
\]

\[
x = y = 0
\]

\[
y = 0, 0 < x \leq 100
\]
Idea 3: Abstract the set of states while preserving fixpoints

Abstraction:

\[
P(\mathbb{N} \times \mathbb{N})
\]

\[
P\{e, o\} \times P\{e, o\}
\]
Abstract Interpretation for Dummies

Idea 3: Abstract the set of states while preserving fixpoints

New goal: Find the least abstract fixpoint

Abstraction:

\[ P(\mathbb{N} \times \mathbb{N}) \]
\[ P\{e, o\} \times P\{e, o\} \]

\[ x, y \in \{e, o\} \]
\[ y := 0 \]
\[ y = e, x \in \{e, o\} \]
\[ x := x - 100 \]
\[ x > 100 \]
\[ x \leq 100 \]
\[ x := x - 1 \]
\[ x > y \]
\[ x \leq y \]
\[ y := y + 2 \]
\[ y = e, x \in \{e, o\} \]
\[ y = e, x \in \{e, o\} \]
\[ y = e, x \in \{e, o\} \]
\[ y = e, x \in \{e, o\} \]
Abstract Interpretation for Dummies

Idea 3: Abstract the set of states while preserving fixpoints

New goal: Find the least abstract fixpoint

Result: You get a concrete fixpoint

\[ x, y \in \mathbb{N} \]

\[ y := 0 \]

\[ y \text{ even}, x \in \mathbb{N} \]

\[ x := x - 100 \]

\[ x \leq 100 \]

\[ x > 100 \]

\[ x \leq y \]

\[ x > y \]

\[ y := y + 2 \]

\[ y \text{ even}, x \in \mathbb{N} \]

\[ y \text{ even}, x \in \mathbb{N} \]

\[ y \text{ even}, x \in \mathbb{N} \]

\[ y \text{ even}, x \in \mathbb{N} \]

\[ y \text{ even}, x \in \mathbb{N} \]
From Theory to Practice

Theoretical Foundations

- Based on domain theory
  - We work with dcpos in order to get a notion of fixpoint

- Galois connection/adjunction
  - The abstraction is performed by a Galois connection (an adjunction)

Practical Foundations

- Compile into a state machine
  - Refine locations and variables into more uniform ones
    - inlining, static single assignment (SSA)...

- Choose an abstract domain
  - Choose a more precise/simple-specific abstract domain
    - segments, polygons, arithmetic...

- Methods to find fixpoints
  - Use fast/powerful methods to search for fixpoints
    - widening, narrowing, finite domains...
AI for Functional Programming: Works in Theory but Not in Practice

Domain and categorical theories for FP have been understood and refined for 40 years!

Dozens of full abstraction results or powerful types systems...

In the 90’s, figures like Cousot and Abramsky thought that it was time to adapt AI to FP, but they ultimately failed.

Methods to find fixpoints do not work on functional spaces

Finite spaces
tower blow up on the (monomorphic) type order

Widening and narrowing are not inherited by functional spaces.

Other issues

Complex flow of execution

Abstracting ADT
Type Order of Functional Programs is Practically Bounded by 3

False in the absolute
any CPS, closure, etc, will easily lift the order of your programmes to 5 or 6...

True up-to polymorphism
Polymorphic function of order 2 are instantiated as monomorphic functions of order 5.

What is an abstract domain for a polymorphic function?

[Abramsky1992]
Model for polymorphism and polymorphic treatment of totality

Types Systems
A type system with polymorphism is an abstraction of a kind.
Polymorphic abstractions can only abstract resource usage and preservation

A term \( t : \forall A, (A \rightarrow A) \rightarrow A \rightarrow A \) is somehow characterised by its number of use of its first argument.

Graded types systems seem to capture them

Most* polymorphic abstractions in the literature can be embedded into a kind of graded type system

* all those I know

Example

\((A^1 \rightarrow A)^4 \rightarrow A^1 \rightarrow A\) has a unique inhabitant in \( \text{nf} : \lambda fx. f(f(f(f(fx)))) \)
Contents

1 Abstract Interpretation
   • Abstract Interpretation for Dummies
   • In Functional Programming
   • Polymorphic Abstractions

2 Graded types
   • Idea and Bibliography
   • Issues
   • Resource-polymorphism

3 Parameterisation
   • Definitions
   • Nice properties

4 Oidification
   • Definitions
   • Model
Type systems for verification fall under\(^1\) one of those:

- encode a whole external proof (ex: coq)
- use intersection types (ex: model checking of HO schemes)
- attach quantitative information to ground-types\(^2\) (ex: refinement types)
- attach to any arrow a information on resources (ex: Hoar types)

1: at least 2: and ADTs

Kind of information carried by these annotations

- postcondition / raised effect / resource modification
- precondition / required resource
- usage of the arguments
Graded Structures

Syntactical intuition

let $F$ a functor with nice properties (exponential, monad)
Suppose that $FA = \exists i : I, F_i A$, what are the conditions on $I$ ?
Natural transformation $\rightarrow$ operators $\rightarrow$ diagrams $\rightarrow$ equations

For a monad $M = \exists i : I, i \cdot A$

$(I,1,\ast)$ is a monoid with $\eta_{i,A} : A \rightarrow i \cdot A$ $\mu_{i,j,A} : i \cdot (j \cdot A) \rightarrow (i \ast j) \cdot A$

Categorically formalised [Fuji, Katsumata, Mellies]

Preservation of lax algebraic structures by adjoin functors
Linear Logic with Graded Exponential

Logic of a graded exponential (me_\ll)

\[
\frac{\Gamma \vdash B}{\Gamma, A^0 \vdash B} \quad \text{Weak} \quad \frac{\Gamma, A^1 \vdash B}{\Gamma, A^1 \vdash B} \quad \text{Der} \quad \frac{\Gamma, A^i, A^j \vdash B}{\Gamma, A^{i+j} \vdash B} \quad \text{Contr}
\]

\[
\frac{\Gamma, A^i \vdash B}{\Gamma, A^j \vdash B} \quad i \leq j \quad \text{Der} \quad \frac{A_1^{i_1}, \ldots, A_n^{i_n} \vdash B}{A_1^{i_1*+j}, \ldots, A_n^{i_n*+j} \vdash B^j} \quad \text{Prom}
\]

Semiring Galois connection

If \( S \perp R \) then any proof (i.e. any term) in me_R is a proof in me_S
Issues To Be Dealt With in CoGITARe

No ADTs

Too many systems

- graded monads and exponentials
- any other graded structures (arrows ?)
- a lot of morally graded system do not enter those frameworks (BLL, DFuzz, indexed LL...)

A lack of expressive power

We can compose of non-matching gradation:

\[
\frac{A_1^{i_1}, \ldots, A_n^{j_n} \vdash B^j \rightarrow C}{\text{App}}\quad \frac{A_1^{k_1}, \ldots, A_n^{k_n} \vdash B}{A_1^{i_1+j*k_1}, \ldots, A_n^{i_n+j*k_n} \vdash C}
\]

but the \( A_i \) have to be the same (no identification of deep gradations)

\( \leadsto \) lack of expressivity: \( \lambda f.f(\lambda xy.x); f(\lambda xy.y) \) untypable in me_\( \mathbb{N} \)ll.
Resource Polymorphism for More Expressivity

$$\lambda f. x. f x : \forall i. (A^i \rightarrow B) \rightarrow (A^i \rightarrow B)$$

Resource polymorphism is not type polymorphism

- trivial monomorphic resource \( \not\Rightarrow \) trivial polymorphic one
  \( \forall \alpha, \beta . \alpha A \rightarrow \beta B \rightarrow \beta \alpha \beta C \)
- substitution and binders may be "algebraised"

Two Different Kinds of Dependency

- **Algebraisation of the syntactical polymorphism**
  Polymorphic labels are resources and substitution is an algebraic operation.

- **Oidification of the logic**
  Resources have source and target and the bind the resources inside the term they are applied to
Abstract Interpretation

1. Abstract Interpretation for Dummies
2. In Functional Programming
3. Polymorphic Abstractions

Graded types

2. Idea and Bibliography
3. Issues
4. Resource-polymorphism

Parameterisation

3. Definitions
4. Nice properties

Oidification

4. Definitions
5. Model
Schedules Types for Weak Head-Reduction

We use Weak-head reduction
Trad-off between algebraic and coalgebraic behaviour observations

Definition: Schedules
An ordered monoid \((\mathcal{S}, \epsilon, \cdot, \leq)\) containing labels \(\alpha, \beta, \gamma \cdots \in \mathcal{S}\) with

- a substitution \(S[\alpha \mapsto T] \in \mathcal{S}\) th:
  \[\alpha[\alpha \mapsto T] = T \quad \beta[\alpha \mapsto T] = \beta\]
- behaving well...

Definition : Schedules Types
\((\text{Types})\)

\[\mathcal{T} \quad A, B, C \quad := \quad \forall \alpha. B \quad | \quad S \cdot A \rightarrow T \cdot B \quad | \quad \cdots\]

Reynolds-style local-variable block and rational schedules

\[\text{new} : \delta(\alpha \text{int} \rightarrow \beta(\forall \gamma. \gamma \text{int} \rightarrow \gamma \text{com}) \rightarrow \beta(\alpha + \beta)^* \cdot \text{com}) \rightarrow \delta \cdot \text{com}\]
Extending Graded Monads and Exponentials

Graded monads using polymorphic-free schedules

If $\alpha = \varepsilon$ and $S[\alpha \to T] = S$ always, then we are recovering the graded monads for weak-head reduction.

Scheduled types are polymorphic graded monads in the first place...

Graded exponentials using vectors schedules

Let $R$ a semiring.

If $S = \bigoplus \alpha_i R$, is set of $R$-valued monomials over the labels, so that

$S \ast T := S + T$ is the sum of monomials,

$S[\alpha \to T]$ is the substitution of $\alpha$ by $T$ in the monomial $S$.

Then we are recovering a graded exponential for weak-head reduction.

4th church numeral for $\bigoplus \alpha_i \text{Nat}$

$(A^1 \to A)^4 \to A^1 \to A \equiv \alpha(\forall \beta. \beta A \to \beta A) \to \gamma A \to (4\alpha + \gamma)A$

Few differences due to graded exponential not being well defined in WHR
In additions to usual AI tools (finite domains, widening...), we can restrict the scopes of the foralls:

### Local-polymorphism

Only arrows of the following form for \( \alpha \not\in \text{dom}(A) \) such that...

\[
\forall \alpha. (\alpha \cdot A \rightarrow T \cdot B) 
\]

\( \rightsquigarrow \) deterministic application rule

\[
\frac{\Gamma \vdash S \quad \forall \alpha. \alpha A \rightarrow T B}{\Gamma \vdash S \cdot T[\alpha \rightarrow U] B[\alpha \rightarrow U]} \quad \Gamma \vdash U A
\]

### Let-polymorphism

Let-sequents of the following form with monomorphic \( B_j \) and \( T \) and...

\[
(\alpha_i \cdot A_i) \vdash S \beta_1 B_1 \rightarrow \cdots \beta_k B_k \rightarrow T \downarrow
\]

\( \rightsquigarrow \) ML let-polymorphism inference?
Higher-Order Resource Polymorphism?

Future work: need for 2nd order

We still do not have that
\[ \lambda f.x.f : \forall i.(A^i \to B) \to (A^i \to B) \]

Indeed, the exponent is already a 1st order parametric information, we need something like:
\[ \lambda f.x.f : \forall h.(\forall \alpha.A \to h(\alpha).B) \to (\forall \alpha.A \to h(\alpha).B) \]

Would be also useful to understand strong head reduction or unification algorithms.

Question 1: is such a 2nd order variable a function? 
Question 2: Where do we stop?
Absence of Model

For the moment we failed at providing either categorical or denotational models...

Some leads using game semantics, but only for strong head reduction
Contents

1 Abstract Interpretation
   • Abstract Interpretation for Dummies
   • In Functional Programming
   • Polymorphic Abstractions

2 Graded types
   • Idea and Bibliography
   • Issues
   • Resource-polymorphism

3 Parameterisation
   • Definitions
   • Nice properties

4 Oidification
   • Definitions
   • Model
Oidification of the semiring

**Oidification**: Generalising an algebraic structure by associating to elements a source and a target
⇝ algebras becomes categories
⇝ the original structures are the one-point categories

\[
\begin{array}{c}
\text{Monoids} \rightarrow \text{Categories} \\
\text{Groups} \rightarrow \text{Groupoids}
\end{array}
\]

**Idea : Grading the exponentials by morphisms**
We get resource environments and exponentials are binders.

**Similar examples in the literature**

\[
\begin{array}{c}
\text{LL(I)}: \\
\sigma | \Gamma \vdash B \\
f : \sigma \rightarrow \rho \\
\rho | \Gamma^f \vdash B^f \\
\text{Prom}
\end{array}
\]

\[
\begin{array}{c}
\text{BLL}: \\
\rho, x | \Gamma \vdash B \\
f : \mathbb{N}^\rho \rightarrow \mathbb{N} \\
\rho | \Gamma^{x<f} \vdash B^{x<f} \\
\text{Prom}
\end{array}
\]

LL(I) results from an attempt at understanding logical relations [Bucc,Ehrh.00]
BLL is a system characterising polynomial algorithms [Gir,Sce,Sco.92]
Substitution

Dereliction

In such a system, the dereliction is performing an action:

\[
\frac{\Gamma, f_*(A) \vdash_\sigma B}{\Gamma, A^f \vdash_\sigma B} \quad \text{Der}
\]

Such action is only available on specific \( f \) (playing the previous role of \( 1 \)) which are basically projections.

From an operational point of view, they are substitutions

\[
\text{ex. in BLL:} \quad \frac{\Gamma, A[x:=0] \vdash_\sigma B}{\rho | \Gamma, A^{x<1} \vdash_\sigma B} \quad \text{Der}
\]

Categorically we need “substitutions” \( s : \rho \rightarrow \sigma \)

they correspond to specific resources and that commutes with them:

\[
A[s]^f \simeq A^{s.f} \quad \text{and} \quad A^f[s] \simeq A^{f.s}
\]
**An oidified semiring inside linear categories**

**L**: A linear category

\[ m_1, m_\rho, \rho : \text{monoidality of } ! \]

**The environments \( E \); monoids**

The \( \otimes \)-monoids \((\rho, \mu_\rho, \eta_\rho)\) over \( L \).

**The resource category \( C \); \( f : !\rho \to \tau \)**

The Kleisli category of monoid morphisms for the monad \((\rho, \mu, \eta) \mapsto (!\rho, (m_\rho, \rho; \mu), (m_\bot; \eta))\).

Spelling out, the morphisms the \( f : !\rho \to \tau \) commuting:

\[
\begin{array}{ccc}
!\mathbb{1} & \xrightarrow{!\eta_\rho} & !\rho \\
\downarrow m_\bot & & \downarrow f \\
\mathbb{1} & \xrightarrow{\eta_\tau} & \tau \\
\end{array}
\quad
\begin{array}{ccc}
!(\rho \otimes \rho) & \xrightarrow{!\mu_\rho} & !\rho \\
\downarrow m_\rho, \rho & & \downarrow f \\
!\rho \otimes !\rho & \xrightarrow{f \otimes f} & \tau \otimes \tau \\
\downarrow & & \downarrow \mu_\tau \\
\tau & \xrightarrow{\tau} & \tau \\
\end{array}
\]

**The rewriting/substitution category \( R \); \( i : \rho \to \tau \)**

The category of \( \otimes \)-monoid morphisms.
Slicing linear categories into an hyperdoctrine

- Formula $A$ of domain $\rho$ are coslices, i.e., morphisms $A : |A| \rightarrow \rho$
- Programs (or derivations) $x : A \vdash_\rho t : B$ of domain $\rho$ are morphisms $t : |A| \rightarrow |B|$ such that:

$$
\begin{array}{ccc}
|A| & \xrightarrow{A} & \rho \\
\downarrow t & & \downarrow B \\
|B| & \xrightarrow{\rho} & \rho
\end{array}
$$

- Exponentials and substitutions are the non-linear and linear coextensions of contexts:

$A^f := !|A| !\rho \xrightarrow{f} \tau$

$A[s] := |A| \xrightarrow{A} \rho \xrightarrow{s} \tau$

- Tensor product is defined by:

$A \otimes B := |A| \otimes |B| \xrightarrow{A \otimes B} \rho \otimes \rho \xrightarrow{\mu_\rho} \rho$

- Cartesian coproduct and product are defined by:

$A \oplus B := |A| \oplus |B| \xrightarrow{\{A, B\}} \rho$

$A \& B := |A| \& |B| \xrightarrow{A \& B} \rho \& \tau =: \rho + \tau$

However, the monoidal closure has to be required...
Exponential formulae

\[
\begin{align*}
&d_A := d_{|A|} \\
&\text{Nat}(d) \\
&|A| \\

&p'_A, f, g := p'|_A \\
&\text{Nat}(p') \\
&!!|A| \\

&m_f := m_f \\
&m_{|A|, |B|} \\
&\text{Nat}(m) \\
&!(|A| \otimes |B|) \\

&\mu \rho \\

\end{align*}
\]
I hope to have convinces you that:

- abstract interpretation for functional programming is an open and interesting field of study, even for theorists,
- graded types are the future! (or just that they may be used for real thing in years to come...)
- resources’ dependency/polymorphism is really different from terms’, but still quite fun.

Project CoGITARe

begin in March for 42 months

1 funded PhD (+internship) funding for 1y posdoc

Contact: breuvart@lipn.fr