Parametricity, Quotient Types, and Theorem Transfer

Brian Huffman

Galois Tech Seminar

March 5, 2013

Outline

I Parametricity, free theorems

II Quotient types, subtypes (type abstraction)

III Theorem transfer, Isabelle/HOL automation

Parametricity

Parametricity

Parametrically polymorphic functions

- may be instantated at different types
- all instances behave uniformly
- limited in what they can do with their arguments

How to make this precise?

Idea: Types as Relations

Mapping $[\![-]\!]$ takes type expressions to binary relations

Idea: Types as Relations

Mapping $[\![-]\!]$ takes type expressions to binary relations

Ground types are identity relations

- $[\![\texttt{Int}]\!] = \mathit{Id}_{\texttt{Int}}$
- $[\![\texttt{Bool}]\!] = \mathit{Id}_{\texttt{Bool}}$

Idea: Types as Relations

Mapping $[\![-]\!]$ takes type expressions to binary relations

Ground types are identity relations

- $[\![\texttt{Int}]\!] = \mathit{Id}_{\texttt{Int}}$
- $[\![\texttt{Bool}]\!] = \mathit{Id}_{\texttt{Bool}}$

Functions are related if they take related input to related output

- $[\![\tau_1 \rightarrow \tau_2]\!] = [\![\tau_1]\!] \Rrightarrow [\![\tau_2]\!]$
- $(A \Rrightarrow B)\ f\ g \Longleftrightarrow (\forall x\, y.\ A\, x\, y \implies B\,(f\ x)\,(g\ y))$

Idea: Types as Relations

Mapping $[\![-]\!]$ takes type expressions to binary relations

Ground types are identity relations

- $[\![\texttt{Int}]\!] = Id_{\texttt{Int}}$
- $[\![\texttt{Bool}]\!] = Id_{\texttt{Bool}}$

Functions are related if they take related input to related output

- $[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \Mapsto [\![\tau_2]\!]$
- $(A \Mapsto B)\ f\ g \Longleftrightarrow (\forall x\, y.\ A\, x\, y \implies B\,(f\ x)\,(g\ y))$

Type variables map to arbitrary relations

- $[\![\texttt{a}]\!] = A$
- $[\![\texttt{b}]\!] = B$

The Parametricity Theorem

Theorem. If term f has type τ, then $[\![\tau]\!]\ f\ f$

The Parametricity Theorem

Theorem. If term f has type τ, then $[\![\tau]\!]\ f\ f$

Example:

- $\texttt{foo} :: \texttt{a} \to \texttt{b} \to \texttt{a}$
- $[\![\texttt{a} \to \texttt{b} \to \texttt{a}]\!]\ \texttt{foo}\ \texttt{foo}$
- $(A \Mapsto B \Mapsto A)\ \texttt{foo}\ \texttt{foo}$ (for arbitrary A, B)
- $A\,x\,x' \wedge B\,y\,y' \implies A\,(\texttt{foo}\,x\,y)\,(\texttt{foo}\,x'\,y')$
- Implies e.g. that $\texttt{foo}\,x\,y = x$

Proof of the Parametricity Theorem

Lambda calculus typing rules:

$$\frac{\Gamma \vdash f : \tau_1 \to \tau_2 \qquad \Gamma \vdash x : \tau_1}{\Gamma \vdash f\, x : \tau_2}\ \text{App}$$

$$\frac{\Gamma, x : \tau_1 \vdash u : \tau_2}{\Gamma \vdash \lambda x.\, u : \tau_1 \to \tau_2}\ \text{Abs} \qquad\qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{Var}$$

Proof of the Parametricity Theorem

Lambda calculus typing rules:

$$\frac{\Gamma \vdash f : \tau_1 \to \tau_2 \qquad \Gamma \vdash x : \tau_1}{\Gamma \vdash f\ x : \tau_2}\ \text{App}$$

$$\frac{\Gamma, x : \tau_1 \vdash u : \tau_2}{\Gamma \vdash \lambda x.\ u : \tau_1 \to \tau_2}\ \text{Abs} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{Var}$$

Inference rules for relations:

$$\frac{\Gamma \vdash (R_1 \Mapsto R_2)\ f\ g \qquad \Gamma \vdash R_1\ x\ y}{\Gamma \vdash R_2\ (f\ x)\ (g\ y)}\ \text{App}$$

$$\frac{\Gamma, R_1\ x\ y \vdash R_2\ u\ v}{\Gamma \vdash (R_1 \Mapsto R_2)\ (\lambda x.\ u)\ (\lambda y.\ v)}\ \text{Abs} \qquad \frac{R\ x\ y \in \Gamma}{\Gamma \vdash R\ x\ y}\ \text{Var}$$

Parametricity with Datatypes

Data structures are related if

- they have the same shape
- elements are related pointwise

Parametricity with Datatypes

Data structures are related if

- they have the same shape
- elements are related pointwise

Pairs

- $(A \otimes B)\,(x, y)\,(x', y') \iff A\,x\,x' \wedge B\,y\,y'$

Lists

- $A^*\,[\,]\,[\,]$
- $A^*\,(x : xs)\,(x' : xs') \iff A\,x\,x' \wedge A^*\,xs\,xs'$

Constructors satisfy parametricity theorem

- $(A \Mapsto B \Mapsto A \otimes B)\,(,)\,(,)$
- $(A \Mapsto A^* \Mapsto A^*)\,(:)\,(:)$

Theorems for Free! (Wadler)

Recipe for generating free theorems:

1. Start with parametricity theorem for the given type
2. Instantiate relations with graphs of functions
3. Simplify

Theorems for Free! (Wadler)

Recipe for generating free theorems:

1. Start with parametricity theorem for the given type
2. Instantiate relations with graphs of functions
3. Simplify

Example:

- $\texttt{reverse} :: [\mathrm{a}] \to [\mathrm{a}]$
- $(A^* \Rrightarrow A^*)\, \texttt{reverse}\, \texttt{reverse}$
- Let $A = \mathit{graph}(f)$
- Then $A^* = \mathit{graph}(\texttt{map}\ f)$
- $A^*\ xs\ ys \implies A^*\ (\texttt{reverse}\ xs)\ (\texttt{reverse}\ ys)$
- $\texttt{map}\ f\ xs = ys \implies \texttt{map}\ f\ (\texttt{reverse}\ xs) = \texttt{reverse}\ ys$
- $\texttt{map}\ f\ (\texttt{reverse}\ xs) = \texttt{reverse}\ (\texttt{map}\ f\ xs)$

Not-Quite-Parametric Functions

Some functions are polymorphic, but not completely parametric

Not-Quite-Parametric Functions

Some functions are polymorphic, but not completely parametric

Example:

- $(=) :: \mathtt{a} \to \mathtt{a} \to \mathtt{Bool}$
- Its type suggests $(A \Mapsto A \Mapsto Id_{\mathtt{Bool}})\,(=)\,(=)$
- I.e. $A\,x\,x' \wedge A\,y\,y' \implies (x = y \iff x' = y')$

Not-Quite-Parametric Functions

Some functions are polymorphic, but not completely parametric

Example:

- $(=) :: \mathtt{a} \to \mathtt{a} \to \mathtt{Bool}$
- Its type suggests $(A \Mapsto A \Mapsto Id_{\mathtt{Bool}})\,(=)\,(=)$
- I.e. $A\,x\,x' \wedge A\,y\,y' \implies (x = y \iff x' = y')$

Not true for all A, but for some A

- Valid iff A is single-valued in both directions (bi-unique)
- $\mathit{bi\text{-}unique}(A) \implies (A \Mapsto A \Mapsto Id_{\mathtt{Bool}})\,(=)\,(=)$
- Extra assumption works like `Eq` constraint

Not-Quite-Parametric Functions

Some functions are polymorphic, but not completely parametric

Example 2:

- $(\forall) :: (\mathtt{a} \to \mathtt{Bool}) \to \mathtt{Bool}$
- Its type suggests $((A \Mapsto \mathit{Id}_{\mathtt{Bool}}) \Mapsto \mathit{Id}_{\mathtt{Bool}})\ (\forall)\ (\forall)$
- I.e. $(\forall x\, y.\ A\ x\ y \implies p\ x \Leftrightarrow q\ y) \implies (\forall x.\ p\ x) \Leftrightarrow (\forall y.\ q\ y)$

Not true for all A, but for some A

- Valid iff A is surjective in both directions (bi-total)
- $\mathit{bi\text{-}total}(A) \implies ((A \Mapsto \mathit{Id}_{\mathtt{Bool}}) \Mapsto \mathit{Id}_{\mathtt{Bool}})\ (\forall)\ (\forall)$
- Extra assumption works like a class constraint

Parametricity in Higher Order Logic

Theorems for ~~free~~ cheap!

Non-parametric polymorphic functions exist $(=, \forall)$

- Can't infer theorems from types alone
- Must prove parametricity theorem for each constant
- Easy syntax-directed proof (App/Abs/Var rules)
- Some constants need *bi-unique* or *bi-total* constraints

Parametricity in Higher Order Logic

Isabelle/HOL maintains a database of parametricity theorems

- Wadler-style free theorems are one application

How else can we use parametricity?

Quotient Types

Quotients and subtypes are everywhere

- integers
- rationals
- reals
- n-bit words
- multisets
- finite sets
- finite maps
- vectors $\mathbb{R}^n$
- balanced trees
- ...

Quotients and subtypes are **type abstractions**

Hidden details of type construction and representation

Properties encoded in type signatures

- functions maintain datatype invariant
- functions respect equivalence relation

Equality on abstract type $\longleftrightarrow$ other relation on raw type

Formalizing a new abstract type

1. Define representation ("raw") type
2. Define raw operations
3. Prove theorems about raw operations
4. Construct abstract type
5. **Lift operations** from raw to abstract
6. **Transfer theorems** from raw to abstract

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\texttt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\texttt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\mathtt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\mathtt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$
3. Raw theorem: $a \leq_{\mathtt{raw}} b \implies a +_{\mathtt{raw}} c \leq_{\mathtt{raw}} b +_{\mathtt{raw}} c$

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\texttt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\texttt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$
3. Raw theorem: $a \leq_{\texttt{raw}} b \implies a +_{\texttt{raw}} c \leq_{\texttt{raw}} b +_{\texttt{raw}} c$
4. Construct quotient $\mathbb{Z} = (\mathbb{N} \times \mathbb{N}) / \approx$
   (requires that $\approx$ is an equivalence relation)

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\texttt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\texttt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$
3. Raw theorem: $a \leq_{\texttt{raw}} b \implies a +_{\texttt{raw}} c \leq_{\texttt{raw}} b +_{\texttt{raw}} c$
4. Construct quotient $\mathbb{Z} = (\mathbb{N} \times \mathbb{N})/ \approx$
   (requires that $\approx$ is an equivalence relation)
5. Lift $+_{\texttt{raw}}$ to $+_{\texttt{int}} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
   Lift $\leq_{\texttt{raw}}$ to $\leq_{\texttt{int}}: \mathbb{Z} \to \mathbb{Z} \to \texttt{Bool}$
   (requires that $+_{\texttt{raw}}$ and $\leq_{\texttt{raw}}$ respect $\approx$)

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\texttt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\texttt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$
3. Raw theorem: $a \leq_{\texttt{raw}} b \implies a +_{\texttt{raw}} c \leq_{\texttt{raw}} b +_{\texttt{raw}} c$
4. Construct quotient $\mathbb{Z} = (\mathbb{N} \times \mathbb{N}) / \approx$
   (requires that $\approx$ is an equivalence relation)
5. Lift $+_{\texttt{raw}}$ to $+_{\texttt{int}} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
   Lift $\leq_{\texttt{raw}}$ to $\leq_{\texttt{int}}: \mathbb{Z} \to \mathbb{Z} \to \texttt{Bool}$
   (requires that $+_{\texttt{raw}}$ and $\leq_{\texttt{raw}}$ respect $\approx$)
6. Transfer theorem to $a \leq_{\texttt{int}} b \implies a +_{\texttt{int}} c \leq_{\texttt{int}} b +_{\texttt{int}} c$

Example: Quotient construction of $\mathbb{Z}$

1. Raw type: $\mathbb{N} \times \mathbb{N}$
2. Raw operation: $(x, y) +_{\mathtt{raw}} (u, v) = (x + u, y + v)$
   Raw operation: $(x, y) \leq_{\mathtt{raw}} (u, v) = (x + v \leq u + y)$
   Equivalence relation: $(x, y) \approx (u, v) \Leftrightarrow x + v = u + y$
3. Raw theorem: $a \leq_{\mathtt{raw}} b \implies a +_{\mathtt{raw}} c \leq_{\mathtt{raw}} b +_{\mathtt{raw}} c$
4. Construct quotient $\mathbb{Z} = (\mathbb{N} \times \mathbb{N}) / \approx$
   (requires that $\approx$ is an equivalence relation)
5. Lift $+_{\mathtt{raw}}$ to $+_{\mathtt{int}} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
   Lift $\leq_{\mathtt{raw}}$ to $\leq_{\mathtt{int}}: \mathbb{Z} \to \mathbb{Z} \to \mathtt{Bool}$
   (requires that $+_{\mathtt{raw}}$ and $\leq_{\mathtt{raw}}$ respect $\approx$)
6. Transfer theorem to $a \leq_{\mathtt{int}} b \implies a +_{\mathtt{int}} c \leq_{\mathtt{int}} b +_{\mathtt{int}} c$

Steps 4–6 are all automated in Isabelle/HOL

Theorem Transfer

Automating Theorem Transfer

Goal:

- Prove equivalence between corresponding propositions
  e.g. $(\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\texttt{raw}} x) \Leftrightarrow (\forall y : \mathbb{Z}.\ y \leq_{\texttt{int}} y)$

Automating Theorem Transfer

Goal:

- Prove equivalence between corresponding propositions
  e.g. $(\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\mathtt{raw}} x) \Leftrightarrow (\forall y : \mathbb{Z}.\ y \leq_{\mathtt{int}}\ y)$

Idea:

- Think in terms of binary relations:
  $Id_{\mathtt{Bool}}\ (\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\mathtt{raw}} x)\ (\forall y : \mathbb{Z}.\ y \leq_{\mathtt{int}}\ y)$

Automating Theorem Transfer

Goal:

- Prove equivalence between corresponding propositions
  e.g. $(\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\texttt{raw}} x) \Leftrightarrow (\forall y : \mathbb{Z}.\ y \leq_{\texttt{int}} y)$

Idea:

- Think in terms of binary relations:
  $\mathit{Id}_{\texttt{Bool}}\ (\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\texttt{raw}} x)\ (\forall y : \mathbb{Z}.\ y \leq_{\texttt{int}} y)$
- Use syntax-directed App/Abs/Var rules,
  just like deriving parametricity theorems

Automating Theorem Transfer

Goal:

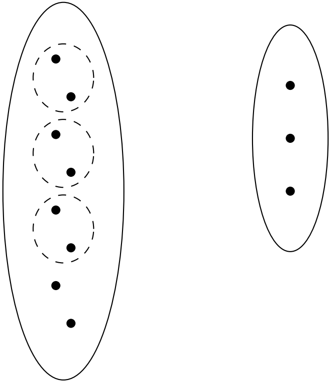
- Prove equivalence between corresponding propositions
  e.g. $(\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\texttt{raw}} x) \Leftrightarrow (\forall y : \mathbb{Z}.\ y \leq_{\texttt{int}} y)$

Idea:

- Think in terms of binary relations:
  $\mathit{Id}_{\texttt{Bool}}\ (\forall x : \mathbb{N} \times \mathbb{N}.\ x \leq_{\texttt{raw}} x)\ (\forall y : \mathbb{Z}.\ y \leq_{\texttt{int}} y)$
- Use syntax-directed App/Abs/Var rules,
  just like deriving parametricity theorems
- Along with parametricity theorems, use **transfer rules**

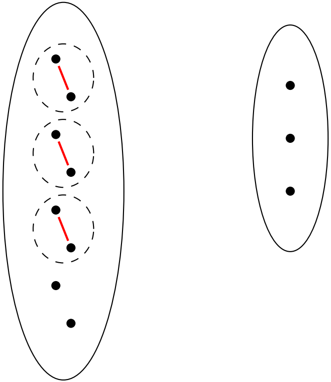
The Four Parts of a Quotient

`Quotient R Abs Rep T`



The Four Parts of a Quotient

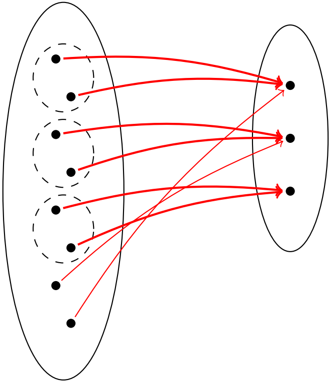
`Quotient R Abs Rep T`



Equivalence relation

The Four Parts of a Quotient

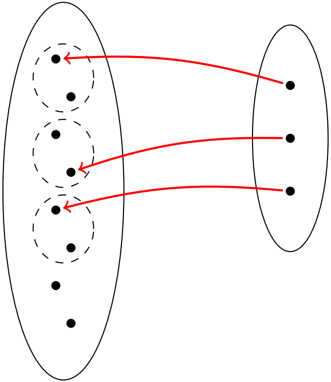
`Quotient R Abs Rep T`



Abstraction function

The Four Parts of a Quotient

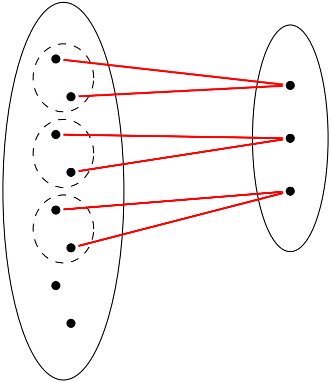
`Quotient R Abs Rep T`



Representation function

The Four Parts of a Quotient

`Quotient R Abs Rep T`



Transfer relation

Transfer Rules

Parametricity theorems relate instances of the **same** function:

- $(A \Rrightarrow A^* \Rrightarrow A^*)\ (:)\ (:)$
- $(A^* \Rrightarrow A^*)\ \texttt{reverse}\ \texttt{reverse}$
- $(\mathit{Id}_{\texttt{Bool}} \Rrightarrow \mathit{Id}_{\texttt{Bool}} \Rrightarrow \mathit{Id}_{\texttt{Bool}})\ (\implies)\ (\implies)$
- $\mathit{bi\text{-}total}(A) \implies ((A \Rrightarrow \mathit{Id}_{\texttt{Bool}}) \Rrightarrow \mathit{Id}_{\texttt{Bool}})\ (\forall)\ (\forall)$

Transfer rules relate **different** functions, using **transfer relations**:

- $(\mathrm{T}_{\texttt{int}} \Rrightarrow \mathrm{T}_{\texttt{int}} \Rrightarrow \mathrm{T}_{\texttt{int}})\ (+_{\texttt{raw}})\ (+_{\texttt{int}})$
- $(\mathrm{T}_{\texttt{int}} \Rrightarrow \mathrm{T}_{\texttt{int}} \Rrightarrow \mathit{Id}_{\texttt{Bool}})\ (\leq_{\texttt{raw}})\ (\leq_{\texttt{int}})$
- $(\mathrm{T}_{\texttt{int}} \Rrightarrow \mathrm{T}_{\texttt{int}} \Rrightarrow \mathit{Id}_{\texttt{Bool}})\ (\approx)\ (=)$

Using Transfer Rules

Syntax-directed derivation of $(\forall x.\, x \le_{\mathtt{raw}} x) \Leftrightarrow (\forall y.\, y \le_{\mathtt{int}} y)$:

$$\dfrac{\dfrac{\dfrac{\overline{(\mathrm{T}_{\mathtt{int}} \Mapsto \mathrm{T}_{\mathtt{int}} \Mapsto \mathit{Id}_{\mathtt{Bool}})\,(\le_{\mathtt{raw}})\,(\le_{\mathtt{int}})} \quad \overline{\mathrm{T}_{\mathtt{int}}\; x\; y}}{(\mathrm{T}_{\mathtt{int}} \Mapsto \mathit{Id}_{\mathtt{Bool}})\,(x \le_{\mathtt{raw}})\,(y \le_{\mathtt{int}})} \qquad \overline{\mathrm{T}_{\mathtt{int}}\; x\; y}}{\mathit{Id}_{\mathtt{Bool}}\,(x \le_{\mathtt{raw}} x)\,(y \le_{\mathtt{int}} y)}}{(\mathrm{T}_{\mathtt{int}} \Mapsto \mathit{Id}_{\mathtt{Bool}})\,(\lambda x.\, x \le_{\mathtt{raw}} x)\,(\lambda y.\, y \le_{\mathtt{int}} y)}$$

$$\dfrac{\dfrac{\mathit{bi\text{-}total}(\mathrm{T}_{\mathtt{int}})}{((\mathrm{T}_{\mathtt{int}} \Mapsto \mathit{Id}_{\mathtt{Bool}}) \Mapsto \mathit{Id}_{\mathtt{Bool}})\,(\forall)\,(\forall)} \qquad \vdots}{\mathit{Id}_{\mathtt{Bool}}\,(\forall x.\, x \le_{\mathtt{raw}} x)\,(\forall y.\, y \le_{\mathtt{int}} y)}$$

Implementation in Isabelle/HOL

Quotient package

- `quotient_type` command
- Constructs quotient type from an equivalence relation

Lifting package

- `lift_definition` command
- Defines abstract function from raw function

Transfer package

- `transfer` proof method
- Replaces abstract goal with equivalent raw goal

Demo

Conclusions

Types-as-relations

- a versatile idea
- with practical applications

Automation

- Lifting and Transfer packages
- Used throughout Isabelle standard libraries
- Saves much manual effort

Paper

- "Lifting and Transfer: A Modular Design for Quotients in Isabelle/HOL"
- with Ondřej Kunčar, at Isabelle Workshop 2012