Refinement Calculus in Type Theory

Peter Hancock
Nottingham
hancock@spamcop.net

EffTT, Tallinn, 13-14 Dec 2007
Refinement Calculus

Type Theory

Tentativity
A calculus of monotone predicate transformers

\[ \Phi : \mathcal{P}(S') \rightarrow \mathcal{P}(S) \]

Semantics of a command:

*If for any postcondition \( U : \mathcal{P}(S') \) we know which preconditions \( \Phi(U) : \mathcal{P}(S) \) will guarantee termination in a final state satisfying the postcondition, then we know the meaning of the command.*
A calculus of monotone predicate transformers

\[ \Phi : \mathcal{P}(S') \rightarrow \mathcal{P}(S) \]

Semantics of a command:

*If for any postcondition \( U : \mathcal{P}(S') \) we know which preconditions \( \Phi(U) : \mathcal{P}(S) \) will guarantee termination in a final state satisfying the postcondition, then we know the meaning of the command.*

A wide-spectrum (\( \text{Spec} \sqsubseteq \cdots \sqsubseteq \text{Imp} \)) algebra of 2-party contracts, between an angel and a demon.
About 30-40 years old

- Floyd ’67, Hoare ’69. Flowchart/code annotations. (Partial correctness.)
About 30-40 years old

- Floyd ’67, Hoare ’69. Flowchart/code annotations. (Partial correctness.)
- Dijkstra ’75. Guarded commands, weakest (liberal) precondition. [De Bakker, & De Roever ’72, & Meertens ’75]
About 30-40 years old

- Floyd ’67, Hoare ’69. Flowchart/code annotations. (Partial correctness.)
- Dijkstra ’75. Guarded commands, weakest (liberal) precondition. [De Bakker, & De Roever ’72, & Meertens ’75]
About 30-40 years old

- Floyd ’67, Hoare ’69. Flowchart/code annotations. (Partial correctness.)
- Dijkstra ’75. Guarded commands, weakest (liberal) precondition. [De Bakker, & De Roever ’72, & Meertens ’75]
About 30-40 years old

- Floyd '67, Hoare '69. Flowchart/code annotations. (Partial correctness.)
- Dijkstra '75. Guarded commands, weakest (liberal) precondition. [De Bakker, & De Roever '72, & Meertens '75]
- '98. Back and von Wright’s Big Book.
About 30-40 years old

- Floyd ’67, Hoare ’69. Flowchart/code annotations. (Partial correctness.)
- Dijkstra ’75. Guarded commands, weakest (liberal) precondition. [De Bakker, & De Roever ’72, & Meertens ’75]
- ’98. Back and von Wright’s Big Book.
- . . . : parallel, reactive, object orientation . . . data refinement/simulation . . .
A trinity of categories

Gardiner, Martin, de Moore (1994)

- Sets and functions $f : A \to B$. 

Refinement Calculus
A trinity of categories

Gardiner, Martin, de Moore (1994)

- Sets and functions $f : A \rightarrow B$.
- Sets and relations $f : A \rightarrow \mathcal{P}(B)$. (Order enriched.)
A trinity of categories

Gardiner, Martin, de Moore (1994)

▶ Sets and functions $f : A \rightarrow B$.
▶ Sets and relations $f : A \rightarrow \mathcal{P}(B)$. (Order enriched.)
▶ Sets and (monotone) predicate transformers $f : A \rightarrow \mathcal{P}^2(B)$.

\[
\begin{align*}
    A & \rightarrow \mathcal{P}^2(B) \\
    \cong & \quad \mathcal{P}(B) \rightarrow \mathcal{P}(A) \\
    \cong & \quad \mathcal{P}(A \times \mathcal{P}(B))
\end{align*}
\]

Refinement order $=$ pointwise inclusion.
A trinity of categories

Gardiner, Martin, de Moore (1994)

- Sets and functions \( f : A \to B \).
- Sets and relations \( f : A \to \mathcal{P}(B) \). (Order enriched.)
- Sets and (monotone) predicate transformers \( f : A \to \mathcal{P}^2(B) \).

\[
A \to \mathcal{P}^2(B) \\
\simeq \mathcal{P}(B) \to \mathcal{P}(A) \\
\simeq \mathcal{P}(A \times \mathcal{P}(B))
\]

Refinement order = pointwise inclusion.

- ↓. Skew-span. ‘Weak pullovers’. (Lax weak pullbacks.)
A trinity of categories

Gardiner, Martin, de Moore (1994)

- Sets and functions \( f : A \to B \).
- Sets and relations \( f : A \to \mathcal{P}(B) \). (Order enriched.)
- Sets and (monotone) predicate transformers \( f : A \to \mathcal{P}^2(B) \).

\[
\begin{align*}
A & \to \mathcal{P}^2(B) \\
\cong & \quad \mathcal{P}(B) \to \mathcal{P}(A) \\
\cong & \quad \mathcal{P}(A \times \mathcal{P}(B))
\end{align*}
\]

Refinement order = pointwise inclusion.

- ↓. Skew-span. ‘Weak pullovers’. (Lax weak pullbacks.)
- ↑. Maps.
An algebra of contracts

- Lattice structure: $\sqcup, \sqcap_i, \sqcap;\text{ abort, magic.}$
An algebra of contracts

- Lattice structure: \(\sqsubseteq, \sqcup_i, \sqcap_i, \text{abort}, \text{magic}\).
- Modal structure: \(\langle R \rangle, [R]\), where \(R \subseteq S \times S'\)

\[
\langle R \rangle, [R] : \mathcal{P}(S') \rightarrow \mathcal{P}(S)
\]
\[
\langle R \rangle U \overset{\Delta}{=} \{ s \mid \exists s'. s R s' \land U(s') \}
\]
\[
[R]U \overset{\Delta}{=} \{ s \mid \forall s'. s R s' \rightarrow U(s') \}
\]

Both: \(\langle f \rangle = [f] = \) assignment. \(\langle U \rangle\) assertion, \([U]\) assumption.
An algebra of contracts

- Lattice structure: $\sqsubseteq, \sqcup_i, \sqcap_i, \text{abort, magic}$.  
- Modal structure: $\langle R \rangle, [R]$, where $R \subseteq S \times S'$

\[
\begin{align*}
\langle R \rangle, [R] : & \mathcal{P}(S') \to \mathcal{P}(S) \\
\langle R \rangle U & \triangleq \{ s \mid \exists s'. s R s' \land U(s') \} \\
[R] U & \triangleq \{ s \mid \forall s'. s R s' \to U(s') \}
\end{align*}
\]

Both: $\langle f \rangle = [f] =$ assignment. $\langle U \rangle$ assertion, $[U]$ assumption.

- Sequential structure: skip, $\emptyset$.  

Peter Hancock Nottingham hancock@spamcop.net: Refinement Calculus in Type Theory
An algebra of contracts

- Lattice structure: $\subseteq$, $\sqcup_i$, $\sqcap_i$, \texttt{abort}, \texttt{magic}.
- Modal structure: $\langle R \rangle$, $[R]$, where $R \subseteq S \times S'$
  $$\langle R \rangle, [R] : \mathcal{P}(S') \rightarrow \mathcal{P}(S)$$
  $$\langle R \rangle U \triangleq \{ s \mid \exists s'. s R s' \land U(s') \}$$
  $$[R] U \triangleq \{ s \mid \forall s'. s R s' \rightarrow U(s') \}$$

Both: $\langle f \rangle = [f] = \text{assignment}$. $\langle U \rangle$ assertion, $[U]$ assumption.

- Sequential structure: \texttt{skip}, $\&$.
- $\Phi^*(U) = (\mu V) U \cup \Phi(V)$, $\Phi^\infty(U) = (\nu V) U \cap \Phi(V)$,
- Monoidal closed structure: $\otimes$, $\rightarrow$, $!$. (If there's time.)
An algebra of contracts

- Lattice structure: $\sqsubseteq, \sqcup_i, \sqcap_i, \text{abort}, \text{magic}$.
- Modal structure: $\langle R \rangle, [R]$, where $R \subseteq S \times S'$

\[
\langle R \rangle, [R] : \mathcal{P}(S') \rightarrow \mathcal{P}(S)
\]

\[
\langle R \rangle U \triangleq \{ s \mid \exists s'. s R s' \land U(s') \}
\]

\[
[R]U \triangleq \{ s \mid \forall s'. s R s' \rightarrow U(s') \}
\]

Both: $\langle f \rangle = [f] = \text{assignment.} \quad \langle U \rangle \text{ assertion, } [U] \text{ assumption.}$

- Sequential structure: skip, $\triangledown$.
- $\Phi^*(U) = (\mu V) U \cup \Phi(V), \quad \Phi^\infty(U) = (\nu V) U \cap \Phi(V)$.
- Monoidal closed structure: $\otimes, \rightarrow, !$. (If there’s time.)
- Many ‘healthiness’ conditions: conjunctive, strict, continuous, ...

Peter Hancock Nottingham hancock@spamcop.net: Refinement Calculus in Type Theory
Two notions of subset (at least)

- Set-valued function on $X$. (Contravariant.)
  \[ \mathcal{P} : \text{Type} \to \text{Type} \]
  \[ \mathcal{P}(X) \overset{\Delta}{=} X \to \text{Set} \]
Two notions of subset (at least)

- Set-valued function on $X$. (Contravariant.)
  \[ P : \text{Type} \to \text{Type} \]
  \[ P(X) \triangleq X \to \text{Set} \]

- Set-indexed function into $X$. (Covariant.)
  \[ F : \text{Type} \to \text{Type} \]
  \[ F(X) \triangleq (\Sigma I : \text{Set}) \, I \to X \]
Two notions of subset (at least)

- Set-valued function on $X$. (Contravariant.)
  \[ \mathcal{P} : \text{Type} \to \text{Type} \]
  \[ \mathcal{P}(X) \triangleq X \to \text{Set} \]

- Set-indexed function into $X$. (Covariant.)
  \[ \mathcal{F} : \text{Type} \to \text{Type} \]
  \[ \mathcal{F}(X) \triangleq (\Sigma I : \text{Set}) I \to X \]

- Predicative: if $A$ is a set, neither $\mathcal{P}(A)$ nor $\mathcal{F}(A)$ is a set. (Quite important: we are going to iterate $\mathcal{F}$.)
Two notions of subset (at least)

- Set-valued function on $X$. (Contravariant.)
  \[ \mathcal{P} : \text{Type} \to \text{Type} \]
  \[ \mathcal{P}(X) \triangleq X \to \text{Set} \]

- Set-indexed function into $X$. (Covariant.)
  \[ \mathcal{F} : \text{Type} \to \text{Type} \]
  \[ \mathcal{F}(X) \triangleq (\Sigma I : \text{Set}) I \to X \]

- Predicative: if $A$ is a set, neither $\mathcal{P}(A)$ nor $\mathcal{F}(A)$ is a set.
  (Quite important: we are going to iterate $\mathcal{F}$.)

‘Set’ = data type; given exhaustively by constructors (the operands of which need not be data . . . ).
Two notions of subset (at least)

- Set-valued function on $X$. (Contravariant.)
  \[ \mathcal{P} : \text{Type} \to \text{Type} \]
  \[ \mathcal{P}(X) \triangleq X \to \text{Set} \]

- Set-indexed function into $X$. (Covariant.)
  \[ \mathcal{F} : \text{Type} \to \text{Type} \]
  \[ \mathcal{F}(X) \triangleq (\Sigma I : \text{Set}) I \to X \]

- Predicative: if $A$ is a set, neither $\mathcal{P}(A)$ nor $\mathcal{F}(A)$ is a set.
  (Quite important: we are going to iterate $\mathcal{F}$.)

‘Set’ = data type; given exhaustively by constructors (the operands of which need not be data . . .).
Representations of predicate transformers

- $\Phi : S \rightarrow \mathcal{F}^2(S')$. (Petersson and Synek, 1989.)

\[
\Phi = (\lambda s) \langle C(s), (\lambda c) \langle R(s, c), (\lambda r) n(s, c, r) \rangle \rangle
\]

$C : S \rightarrow \text{Set}$ 

$R : (\Pi s : S) C(s) \rightarrow \text{Set}$ 

$n : (\Pi s : S, c : C(s)) R(s, c) \rightarrow S'$ 

Next state

Commands

Responses
Representations of predicate transformers

\[ \Phi : S \to F^2(S') \]  
(Petersson and Synek, 1989.)

\[ \Phi = (\lambda s) \langle C(s), (\lambda c) \langle R(s, c), (\lambda r) n(s, c, r) \rangle \rangle \]

\[ C : S \to \text{Set} \]
\[ R : (\Pi s : S) C(s) \to \text{Set} \]
\[ n : (\Pi s : S, c : C(s)) R(s, c) \to S' \]

Next state

Extension: \( \| \Phi \| : \mathcal{P}(S') \to \mathcal{P}(S) \)

\[ U : \mathcal{P}(S') \leftrightarrow \{ s : S \mid (\Sigma c : C(s)) (\Pi r : R(s, c)) U(n(s, c, r)) \} \]

Eg, abort: \( C(s) = \emptyset \);

magic: \( R(s, c) = \emptyset \).
Free monad

\[ \Phi : S \rightarrow \mathcal{F}^2(S) \]
Free monad

\[ \Phi : S \rightarrow \mathcal{F}^2(S) \]

\[ \|\Phi^*\|X = (\mu Y) X \cup \Phi(X) \]
Free monad

\[ \Phi : S \to \mathcal{F}^2(S) \]

**Programs** \( C^*(s) \)

\[
C^*(s) = 1 + (\Sigma c : C(s)) (\Pi r : R(s, c)) C^*(n(s, c, r)) \\
= 1 + \|\Phi\|(C^*, s)
\]

\[ \|\Phi^*\|X = (\mu Y) X \cup \Phi(X) \]
Free monad

$$\Phi : S \rightarrow \mathcal{F}^2(S)$$

$\|\Phi^*\| X = (\mu Y) X \cup \Phi(X)$

- Programs $C^*(s)$

$$C^*(s) = 1 + (\Sigma c : C(s))(\Pi r : R(s, c)) C^*(n(s, c, r))$$

$$= 1 + \|\Phi\|(C^*, s)$$

- Traces $R^*(s, p)$ where $s : S, p : C^*(s)$:

$$R^*(s, \text{Exit}) = 1$$

$$R^*(s, \text{Call}(c, f)) = (\Sigma r : R(s, c)) R^*(n(s, c, r), f(r))$$
Free monad

\[ \Phi : S \rightarrow \mathcal{F}^2(S) \]

\[ \| \Phi^* \| X = (\mu Y) X \cup \Phi(X) \]

- Programs \( C^*(s) \)

\[
C^*(s) = 1 + (\Sigma c : C(s)) (\Pi r : R(s, c)) C^*(n(s, c, r))
= 1 + \| \Phi \| (C^*, s)
\]

- Traces \( R^*(s, p) \) where \( s : S, p : C^*(s) \):

\[
R^*(s, \text{Exit}) = 1
R^*(s, \text{Call}(c, f)) = (\Sigma r : R(s, c)) R^*(n(s, c, r), f(r))
\]

- Exit state \( n^*(s, p, t) \) where \( s : S, p : C^*(s), t : R^*(s, p) \):

\[
n^*(s, \text{Exit}) = 1
n^*(s, \text{Call}(c, f), \langle r, t \rangle) = n^*(n(s, c, r), f(r), t)
\]
Morphisms

- Objects: *interaction structures*

\[
\langle S : \text{Set}, \Phi : S \rightarrow \mathcal{F}^2(S) \rangle
\]
Morphisms

- **Objects:** *interaction structures*

\[ \langle S : \text{Set}, \Phi : S \to \mathcal{F}^2(S) \rangle \]

- **Morphisms:** *simulations*

\[ \langle S : \text{Set}, \Phi : S \to \mathcal{F}^2(S) \rangle \downarrow \]

\[ \langle S' : \text{Set}, \Psi : S' \to \mathcal{F}^2(S') \rangle \]

given by \( R \subseteq S' \times S \) such that

\[ \langle R \rangle \cdot \| \Phi \| \sqsubseteq \| \Psi \| \cdot \langle R \rangle \]
What this means

\[ R(s, s') \rightarrow \sum f : C(s) \rightarrow C'(s') \]
\[ g : (\prod c : C(s)) R'(s', f(c)) \rightarrow R(s, c) \]
\[ \prod c : C(s) \]
\[ r' : R'(s', f(c)) \]
\[ R(n(s, c, g(c, r')), n'(s', f(c), r')) \]
What this means

\[ R(s, s') \rightarrow \sum f : C(s) \rightarrow C'(s') \]
\[ g : (\prod c : C(s)) R'(s', f(c)) \rightarrow R(s, c) \]
\[ \prod c : C(s) \]
\[ r' : R'(s', f(c)) \]
\[ R(n(s, c, g(c, r')), n'(s', f(c), r')) \]

Er, hang on, that's a coalgebra for ... an interaction structure! (That's where some linear logic comes in...).
What this means

\[ R(s, s') \rightarrow \sum f : C(s) \rightarrow C'(s') \]
\[ g : (\prod c : C(s)) R'(s', f(c)) \rightarrow R(s, c) \]
\[ \prod c : C(s) \]
\[ r' : R'(s', f(c)) \]
\[ R(n(s, c, g(c, r')), n'(s', f(c), r')) \]

Er, hang on, that's a coalgebra for ... an interaction structure!
(That's where some linear logic comes in...).
Composition is relational composition. (This isn't great....)
What is the logical form of a specification?

What a silly question! It depends...
What is the logical form of a specification?

What a silly question! It depends...

- A *client* (transaction) program:
  \[
  \begin{align*}
  \mathcal{U} \subseteq \Phi^*(\mathcal{V}) \\
  \text{Precondition} & \quad \text{Postcondition}
  \end{align*}
  \]
What is the logical form of a specification?

What a silly question! It depends…

▶ A *client* (transaction) program:

\[
\begin{align*}
\text{Precondition} & \subseteq \Phi^*(\text{Postcondition}) \\
\end{align*}
\]

▶ A *server* program:

\[
\begin{align*}
\text{Initialisation} & \trianglerighteq (\Phi^\sim)^\infty(\text{Safety property}) \\
\end{align*}
\]

\(\Phi^\sim\) is *inversion* of \(\Phi\):

\[
\begin{align*}
C^\sim(s) & = (\prod c : C(s)) R(s, c) \\
R^\sim(s, \_ ) & = C(s) \\
n^\sim(s, f, c) & = n(s, c, f(c))
\end{align*}
\]
Thinking more practically

- An Instruction set: $\langle I, O \rangle : \mathcal{F}($Set$)$
Thinking more practically

- An *Instruction set*: $\langle I, O \rangle : \mathcal{F}(\text{Set})$
- Specifications (man pages) in the ‘real world’ seem to have the form:

$$S \rightarrow (\Pi i : I) \mathcal{F}(\mathcal{P}(O(i) \times S))$$

The data here is

$$P : \mathcal{P}(S \times I)$$
$$N : (\Pi \langle s, i \rangle : S \times I) P\langle s, i \rangle \rightarrow \mathcal{P}(O(i) \times S)$$

A species of non-deterministic Mealey machine?
A relative of Lynch’s IO-automata?
Some opinions (on ‘the IO problem’)

- We need old ideas. For example predicate transformers. Simulations. Streams. This ought to be reassuring.
- Stop obsessing about monads!
- Stop obsessing about programs (terms). Start obsessing about specifications!
- Forget about computers. A program is a guide to action. Action directed towards an end.
Some opinions (on ‘the IO problem’)

Some opinions (on ‘the IO problem’) 


- We need old ideas. For example predicate transformers. Simulations. Streams. This ought to be reassuring.
Some opinions (on ‘the IO problem’)

- We need old ideas. For example predicate transformers. Simulations. Streams. This ought to be reassuring.
- Stop obsessing about monads!
Some opinions (on ‘the IO problem’)  

- We need old ideas. For example predicate transformers. Simulations. Streams. This ought to be reassuring.  
- Stop obsessing about monads!  
- Stop obsessing about programs (terms). Start obsessing about specifications!
Some opinions (on ‘the IO problem’) 

- We need old ideas. For example predicate transformers. Simulations. Streams. This ought to be reassuring.
- Stop obsessing about monads!
- Stop obsessing about programs (terms). Start obsessing about specifications!
- Forget about computers. A program is a guide to action. Action directed towards an end.
A few references I

Ralph-Johan Back and Joakim von Wright.
Refinement calculus, A systematic introduction.

Ralph-Johan Back and Joakim Von Wright.
Encoding, decoding and data refinement.

Paul H. B. Gardiner, Clare E. Martin, and Oege de Moor.
An algebraic construction of predicate transformers.

Peter Hancock and Pierre Hyvernat.
Programming interfaces and basic topology.

Peter Hancock and Anton Setzer.
Interactive programs in dependent type theory.

Peter Hancock and Anton Setzer.
Guarded induction and weakly final coalgebras in dependent type theory.
A few references II

Pierre Hyvernat.
Predicate transformers and linear logic: yet another denotational model.

Markus Michelbrink.

Carroll Morgan.
Programming from Specifications.

Markus Michelbrink and Anton Setzer.

Kent Petersson and Dan Synek.
A set constructor for inductive sets in Martin-Löf’s type theory.
Examples

Papertape IO

\[ S = \text{Char}^\omega \times \text{Char}^* \]
\[ C \langle i, o \rangle = \{\text{Get}\}|\{\text{Put } c \mid c : \text{Char}\} \]
\[ R \langle i, o \rangle \text{Get} = (\Sigma c : \text{Char}) c = \text{hd } i \]
\[ n \langle i, o \rangle \text{Get} \langle c, \_ \rangle = \langle \text{tl } i, o \rangle \]
\[ R \langle i, o \rangle (\text{Put } c) = \{\text{Ack}\} \]
\[ n \langle i, o \rangle (\text{Put } c) \text{ Ack} = \langle i, o ++ \langle c \rangle \rangle \]
Examples

Nim

\[
\begin{align*}
S &= Z \\
C l &= (l = 0) + (l > 0) + (l > 1) \\
R l (\text{in}_{0-}) &= \{\} \\
R l (\text{in}_{1-}) &= C(l - 1) \\
R l (\text{in}_{2-}) &= C(l - 2) \\
n l (\text{in}_{1-}) (\text{in}_{0-}) &= -1 \\
n l (\text{in}_{1-}) (\text{in}_{1-}) &= l - 2 \\
n l (\text{in}_{1-}) (\text{in}_{2-}) &= l - 3 \\
n l (\text{in}_{2-}) (\text{in}_{0-}) &= -1 \\
n l (\text{in}_{2-}) (\text{in}_{1-}) &= l - 3 \\
n l (\text{in}_{2-}) (\text{in}_{2-}) &= l - 4
\end{align*}
\]
Examples

Nim

\[ S \quad = \quad Z \]
\[ C l \quad = \quad (l = 0) + (l > 0) + (l > 1) \]
\[ R l (\text{in}_{0-}) \quad = \quad \{\} \]
\[ R l (\text{in}_{1-}) \quad = \quad C(l - 1) \]
\[ R l (\text{in}_{2-}) \quad = \quad C(l - 2) \]
\[ n l (\text{in}_{1-}) (\text{in}_{0-}) \quad = \quad -1 \]
\[ n l (\text{in}_{1-}) (\text{in}_{1-}) \quad = \quad l - 2 \]
\[ n l (\text{in}_{1-}) (\text{in}_{2-}) \quad = \quad l - 3 \]
\[ n l (\text{in}_{2-}) (\text{in}_{0-}) \quad = \quad -1 \]
\[ n l (\text{in}_{2-}) (\text{in}_{1-}) \quad = \quad l - 3 \]
\[ n l (\text{in}_{2-}) (\text{in}_{2-}) \quad = \quad l - 4 \]

\[(\mu X) [l > 0][M][l > 0][M][X] \]

where \( M \triangleq (l := l' | l - 2 \leq l' \leq l - 1) \)
Some programming constructs

\[
\begin{align*}
\|\text{skip}\| U &= U \\
\|P ; Q\| &= \|P\| \cdot \|Q\| \\
\|\text{abort}\| U &= \text{false} \\
\|\text{magic}\| U &= \text{true} \\
\|v := e\| U &= U[v \leftarrow e] \\
\|\langle P \rangle\| U &= P \land U \\
\|\lbrack P \rbrack\| U &= P \rightarrow U \\
\|\langle R \rangle\| U &= \{s \mid R(s) \not\subseteq U\} \\
\|\brack R\| U &= \{s \mid R(s) \subseteq U\} \\
\|\uplus_i P_i\| U &= \lor_i \|P_i\| U \\
\|\cap_i P_i\| U &= \land_i \|P_i\| U
\end{align*}
\]