Formalising Generalised Substitutions

Jeremy Dawson

Logic and Computation Program, NICTA [1]

Automated Reasoning Group,
Australian National University, Canberra, ACT 0200, Australia
http://users.rsise.anu.edu.au/~jeremy/

September 3, 2007

[1]National ICT Australia is funded by the Australian Government's Dept of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Centre of Excellence program.

Outline

Outline

Introduction

Dunne developed a theory of the generalised substitutions used in the B method.

He then proved properties of these generalised substitutions.

Theory based on total correctness: contrast to abstract command language, based on general correctness

Both underlying "models of computation" based on a monad

Each of these is a compound monad

Outline

The General Correctness Operational Model

Want to distinguish computations which (on a given initial state)

- fail to terminate
- terminate in final state s
- non-deterministically, either of the above

Neither *wlp* / partial correctness
nor *wp* / total correctness does this.

General correctness refinement (Dunne):

$$A \sqsubseteq B \equiv wp(A, Q) \Rightarrow wp(B, Q) \wedge wlp(A, Q) \Rightarrow wlp(B, Q)$$

The General Correctness Operational Model

Type of Computations

A computation (on given state) produces a set of outcomes

An outcome is either

- `NonTerm`, indicating non-termination, or
- `Term` s, indicating termination in the state s.

In Isabelle: `datatype` σ `TorN = NonTerm | Term` σ

For a non-deterministic computation (from given initial state), result is a set of outcomes.

type $outcome = \texttt{TorN}\ state$

type of computations is $state \rightarrow set\ \texttt{TorN}\ state$

The Total Correctness Operational Model

Only interested in total correctness (weakest preconditions).

Any computation which may fail to terminate fails every post-condition.

Such computation is refinement-equivalent to a computation which does fail to terminate.

Type of results is either

- `NonTerm`, indicating possible non-termination, or
- `Term` S, indicating termination in a state $s \in S$.

type of result *tcres* ("total correctness result") = `TorN` *set state*

type of computations is *state* $\rightarrow$ `TorN` *set state*

weakest precondition function (hence refinement):

$$[C]\ Q\ s = \exists S.\ (\forall x \in S.\ Q\ x) \wedge C\ s = \texttt{Term}\ S$$

Confirming the Models

In each case, to confirm model is appropriate,

- we show two computations refinement-equivalent iff they are the same function (of type used in model)
- $A \sqsubseteq_{tc} B \ \wedge B \ \sqsubseteq_{tc} A \ \Rightarrow \ A = B$
- equivalently, $[A] = [B] \Rightarrow A = B$
- we define operations operationally, and prove these definitions correspond to Dunne's definitions (which use weakest preconditions)

(Caveat: we ignore "frames").

Outline

Monads

Long known in category theory

Define unit and extension functions, satisfying rules

$$\begin{aligned} \mathit{unit} &: \alpha \to M\alpha \\ \mathit{ext} &: (\alpha \to M\beta) \to (M\alpha \to M\beta) \end{aligned}$$

$$\begin{aligned} \mathit{ext}\ f \circ \mathit{unit} &= f \\ \mathit{ext}\ \mathit{unit} &= \mathit{id} \\ \mathit{ext}\ (\mathit{ext}\ g \circ f) &= \mathit{ext}\ g \circ \mathit{ext}\ f \end{aligned}$$

or functions *unit*, *map* and *join* (7 axioms for these)

Can represent the structure of a computation (Moggi)

Monads — the Kleisli category

ext B models the action of B on result of previous computation

Define $B \odot A = ext\ B \circ A$: sequencing computations B and A.

unit models the computation which does nothing (*skip*).

$$\begin{aligned} f \odot unit &= f \\ unit \odot f &= f \\ h \odot (g \odot f) &= (h \odot g) \odot f \end{aligned}$$

These properties are as expected for sequencing and *skip*

they give another category — the *Kleisli* category

Monads — Examples

The non-termination monad: a computation either terminates in a new state, or fails to terminate.

$$
\begin{array}{rclrcl}
 & & & unit\_nt\ s & = & \mathtt{Term}\ s \\
map\_nt\ f\ \mathtt{NonTerm} & = & \mathtt{NonTerm} & map\_nt\ f\ (\mathtt{Term}\ s) & = & \mathtt{Term}\ (f\ s) \\
ext\_nt\ f\ \mathtt{NonTerm} & = & \mathtt{NonTerm} & ext\_nt\ f\ (\mathtt{Term}\ s) & = & f\ s
\end{array}
$$

Monads — Examples

The non-termination monad: a computation either terminates in a new state, or fails to terminate.

$$unit\_nt\ s = \mathtt{Term}\ s$$

$$map\_nt\ f\ \mathtt{NonTerm} = \mathtt{NonTerm} \qquad map\_nt\ f\ (\mathtt{Term}\ s) = \mathtt{Term}\ (f\ \ s)$$

$$ext\_nt\ f\ \mathtt{NonTerm} = \mathtt{NonTerm} \qquad ext\_nt\ f\ (\mathtt{Term}\ s) = f\ \ s$$

The set monad: models non-deterministic (but necessarily terminating) computations.

$$unit\_s\ s = \{s\} \qquad join\_s\ \mathcal{A} = \bigcup \mathcal{A}$$

$$map\_s\ f\ \ S = \{f\ \ s \,|\, s \in S\} \qquad ext\_s\ f\ \ S = \bigcup_{s \in S} f\ \ s$$

Compound Monads

Let M and N, each with unit and extension functions, be monads.

Then is $MN\alpha$ a monad? Need $unit_{MN} : \alpha \to MN\alpha$ and ext_{MN}

ext_{MN} "extends" a function f from domain α to $MN\alpha$.

$pext$, "partial extension", does part of this

$$\begin{aligned} ext_{MN} &: (\alpha \to MN\beta) \to (MN\alpha \to MN\beta) \\ pext &: (\alpha \to MN\beta) \to (N\alpha \to MN\beta) \end{aligned}$$

Compound Monads

Let M and N, each with unit and extension functions, be monads.

Then is $MN\alpha$ a monad? Need $unit_{MN} : \alpha \to MN\alpha$ and ext_{MN}

ext_{MN} "extends" a function f from domain α to $MN\alpha$.

pext, "partial extension", does part of this

$$\begin{aligned} ext_{MN} &: (\alpha \to MN\beta) \to (MN\alpha \to MN\beta) \\ pext &: (\alpha \to MN\beta) \to (N\alpha \to MN\beta) \end{aligned}$$

Definitions for a compound monad using *pext*

$$\begin{aligned} ext_{MN}\ g &= ext_M\ (pext\ g) \\ unit_{MN} &= unit_M \circ unit_N \end{aligned}$$

Also require that *pext* satisfies three rules

The General Correctness Compound Monad

Want *set* $\mathtt{TorN}\ \alpha$ is a monad;
in fact, for any monad M, $M\ \mathtt{TorN}\ \alpha$ is a monad

$$pext : (\alpha \to M\ \mathtt{TorN}\ \beta) \to (\mathtt{TorN}\ \alpha \to M\ \mathtt{TorN}\ \beta)$$

$$\begin{aligned} pext\ f\ (\mathtt{Term}\ a) &= f\ a \\ pext\ f\ \mathtt{NonTerm} &= unit_M\ \mathtt{NonTerm} \end{aligned}$$

Proof of *pext* rules easy.

The Total Correctness Compound Monad

Recall $tcres = \mathtt{TorN}\ set\ state$.

$$pext\_tc : (state \to tcres) \to set\ state \to tcres$$

defined using

$$prod\_tc : set\ tcres \to tcres$$

$$\begin{aligned} prod\_tc\ S &= \mathtt{NonTerm} \qquad \text{if } \mathtt{NonTerm} \in S \\ prod\_tc\ \{\mathtt{Term}\ s \mid s \in S\} &= \mathtt{Term}\ (\bigcup S) \end{aligned}$$

Relating the General and Total Correctness monads

$$\begin{aligned} \mathit{swap\_tc} : \mathit{set}\ \mathtt{TorN}\ \sigma &\rightarrow \mathtt{TorN}\ \mathit{set}\ \sigma \\ \mathit{swap\_tc}\ S &= \mathtt{NonTerm} \qquad \text{if } \mathtt{NonTerm} \in S \\ \mathit{swap\_tc}\ \{\mathtt{Term}\ s \mid s \in S\} &= \mathtt{Term}\ S \end{aligned}$$

swap_tc is also a monad morphism
from the general correctness monad to the total correctness monad.

$$\begin{aligned} \mathit{unit\_tc}\ a &= \mathit{swap\_tc}\ (\mathit{unit\_gc}\ a) \\ \mathit{ext\_tc}\ (\mathit{swap\_tc} \circ f)\ (\mathit{swap\_tc}\ x) &= \mathit{swap\_tc}\ (\mathit{ext\_gc}\ f\ \ x) \end{aligned}$$

Since it is *surjective*, could use monad axioms for general correctness monad to prove axioms for total correctness monad.

For some other results also, easiest to prove for general correctness and use *swap_tc* to "transfer" them to total correctness

Outline

Frames and Variables

Each substitution has a *frame*, loosely, the variables which "might" be affected. But $\mathit{frame}(x := x) = \{x\}$.

Behaviour of command (alone) doesn't determine frame.

Mostly we ignore frames.

For many generalised substitution operations (except assignment) consider abstract machine state, not variables

skip, sequencing, magic, abort

skip and sequencing given by *unit* and $\odot$ of monad

define *magic* and *abort* operationally;

```
magic_tc_def = "magic_tc s == Term {}"
abort_tc_def = "abort_tc s == NonTerm"
```

skip, sequencing, magic, abort

skip and sequencing given by *unit* and $\odot$ of monad

define *magic* and *abort* operationally;

```
magic_tc_def = "magic_tc s == Term {}"
abort_tc_def = "abort_tc s == NonTerm"
```

prove, as consequences, Dunne's definitions

```
magic_alt = "magic_tc = guard_tc (%s. False) unit_tc"
abort_alt = "abort_tc = precon_tc (%s. False) unit_tc"
```

preconditioned command, guarded command

Again, we define them operationally;

```
"precon_tc P C s = if P s then C s else NonTerm"
"guard_tc P C s = if P s then C s else Term {}"
```

preconditioned command, guarded command

Again, we define them operationally;

```
"precon_tc P C s = if P s then C s else NonTerm"
"guard_tc P C s = if P s then C s else Term {}"
```

and prove, as consequences, Dunne's definitions

```
"wp_tc (precon_tc P C) Q s = (P s & wp_tc C Q s)"
"wp_tc (guard_tc P C) Q s = (P s --> wp_tc C Q s)"
```

Definition of Commands — Choice

Defining by weakest precondition:
conjunction of individual weakest preconditions.

General Correctness: $\mathit{choice\_gc}\, \mathcal{C}\ s = \bigcup_{C \in \mathcal{C}} C\ s$.

Total Correctness:
$\mathit{choice\_tc}\, \mathcal{C}$ can fail to terminate if any $C \in \mathcal{C}$ can fail to terminate.

$\mathit{choice\_tc}\, \mathcal{C}\ s = \mathit{pext\_tc}\ (\lambda C.\ C\ s)\ \mathcal{C}$, expands to

- if $\{C\ s \mid C \in \mathcal{C}\}$ contains $\texttt{NonTerm}$ then $\mathit{choice\_tc}\, \mathcal{C}\ s = \texttt{NonTerm}$
- if $\{C\ s \mid C \in \mathcal{C}\} = \{\texttt{Term}\ S_C \mid C \in \mathcal{C}\}$, then $\mathit{choice\_tc}\, \mathcal{C}\ s = \texttt{Term}\ (\bigcup_{C \in \mathcal{C}} S_C)$

Use of Coinductive Definitions

$A^0 = skip$, $A^{n+1} = A \; ; A^n$

$A^\wedge$ is like choice of any A^n, also `NonTerm` if infinite repetition possible (an "infinite chain")

In [6] we defined an infinite chain by a function : $\mathbb{N} \to state$ (satisfying conditions)

Better to use Isabelle's coinductive definition facility, related proofs much easier

Use of Coinductive Definitions

infchs_tc C: the set of states from which it is possible to execute *C* infinitely many times sequentially

```
coinductive "infchs_tc C"
   intros "C s = Term S ==> ns : S ==>
                ns : infchs_tc C ==> s : infchs_tc C"
```

Use of Coinductive Definitions

infchs_tc C: the set of states from which it is possible to execute *C* infinitely many times sequentially

```
coinductive "infchs_tc C"
   intros "C s = Term S ==> ns : S ==>
                 ns : infchs_tc C ==> s : infchs_tc C"
```

reach_NT C: the set of states from which `NonTerm` is reachable

```
inductive "reach_NT C"
   intros "C s = NonTerm ==> s : reach_NT C"
    "C s = Term S ==> ns : S ==>
                  ns : reach_NT C ==> s : reach_NT C"
```

Use of Coinductive Definitions

icnt_tc C: the states from which $C\ s$ gives `NonTerm`

```
coinductive "icnt_tc C"
   intros "C s = NonTerm ==> s : icnt_tc C"
    "C s = Term S ==> ns : S ==>
                     ns : icnt_tc C ==> s : icnt_tc C"
```

(same intro rules as for `reach_NT` which was an inductive definition)

We then prove that $icnt\_tc\ C = reach\_NT\ C \cup infchs\_tc\ C$

Definitions of $A^\wedge$

Dunne defined $A^\wedge$ as a least-fixed point

We use the operational definition for $A^\wedge$ and then prove that $A^\wedge$ is least fixpoint of $\lambda X.\ (A; X)\ \Box\ skip$ (where $\Box$ means choice)

```
rephat_def = "rephat C state ==
          if state : icnt_tc C then NonTerm
                                 else Term (treach C state)"
fprep_tc_def = "fprep_tc A X ==
                        X = choice_tc seq_tc A X, unit_tc"
rephat_isfp = "fprep_tc A (rephat A)"
rephat_is_lfp = "fprep_tc A Y ==> ref_tc (rephat A) Y"
```

Other results

Least upper bound of set of generalised substitutions

Least upper bound of set of conjunctive predicate transformers

Generalised substitutions correspond to predicate transformers satisfying a non-empty conjunctivity condition

Discussion of results involving frames — how to express them

Assorted further results from Dunne [8], see paper, also Appendix