Reasoning with Powerdomains in Isabelle/HOLCF

Brian Huffman

Department of Computer Science
Portland State University

TPHOLs 2008, August 18–21

What are Powerdomains?

A powerdomain is

- a monad
- with a nondeterministic choice operator

Powerdomains adapt the notion of powersets to work with domain theory.

What are Powerdomains Good For?

Powerdomains are good for reasoning about

- Nondeterministic algorithms
  - Write algorithms monadically in a powerdomain
  - Works with arbitrary recursion
- Parallel computation
  - Resumption monad transformer models interleaving
  - Powerdomain models nondeterministic scheduler

Monads with Nondeterministic Choice

Haskell type class for monads with nondeterministic choice operator

```
class (Monad m) => MultiMonad m where
  (+|+) :: m a -> m a -> m a
```

Haskell lists can model nondeterministic computation

```
instance MultiMonad [] where
  xs +|+ ys = xs ++ ys
```

Using List Monad for Nondeterminism

Good for modeling in Haskell:

- Executable

Bad for denotational semantics:

- Not abstract enough
- Problems with partial/infinite values

Example Algorithms

- `f1` and `f2` should denote the same value

```
f1, f2 :: (MultiMonad m) => Int -> m Int
f1 n = return (n+1) +|+ return (n-1)
f2 n = return (n-1) +|+ return (n+1)
```

- `g1` and `g2` should denote different values

```
g1, g2 :: (MultiMonad m) => Int -> m Int
g1 n = return n +|+ g1 (n+1) +|+ g1 (n-1)
g2 n = return n +|+ g2 (n+1)
```

Example Algorithms

- f1 and f2 should denote the same value

```
f1, f2 :: (MultiMonad m) => Int -> m Int
f1 n = return (n+1) +|+ return (n-1)
f2 n = return (n-1) +|+ return (n+1)
```

- g1 and g2 should denote different values

```
g1, g2 :: (MultiMonad m) => Int -> m Int
g1 n = return n +|+ g1 (n+1) +|+ g1 (n-1)
g2 n = return n +|+ g2 (n+1)
```

Powerdomain Axioms

1. Return and bind satisfy monad laws

```
(return x) >>= f == f x
a >>= return == a
(a >>= f) >>= g == a >>= (\x -> f x >>= g)
```

2. Bind distributes over choice operator

```
(a +|+ b) >>= f == (a >>= f) +|+ (b >>= f)
```

3. Choice operator is associative, commutative, idempotent

```
(a +|+ b) +|+ c == a +|+ (b +|+ c)
a +|+ b == b +|+ a
a +|+ a == a
```

Notes on Different Kinds of Powerdomains

- Upper
  - "Demonic" nondeterminism
  - `undefined +|+ m = undefined`
  - "Possible non-termination is just as bad as never terminating"
  - Good for total correctness properties
- Lower
  - "Angelic" nondeterminism
  - `undefined +|+ m = m`
  - "I don't care about execution paths that don't terminate"
  - Good for partial correctness properties
- Convex
  - Distinguishes more values than upper or lower
  - "free domain-algebra" w.r.t. powerdomain axioms

Notes on Different Kinds of Powerdomains

- Upper
  - "Demonic" nondeterminism
  - `undefined +|+ m = undefined`
  - "Possible non-termination is just as bad as never terminating"
  - Good for total correctness properties
- Lower
  - "Angelic" nondeterminism
  - `undefined +|+ m = m`
  - "I don't care about execution paths that don't terminate"
  - Good for partial correctness properties
- Convex
  - Distinguishes more values than upper or lower
  - "free domain-algebra" w.r.t. powerdomain axioms

Notes on Different Kinds of Powerdomains

- Upper
  - "Demonic" nondeterminism
  - `undefined +|+ m = undefined`
  - "Possible non-termination is just as bad as never terminating"
  - Good for total correctness properties
- Lower
  - "Angelic" nondeterminism
  - `undefined +|+ m = m`
  - "I don't care about execution paths that don't terminate"
  - Good for partial correctness properties
- Convex
  - Distinguishes more values than upper or lower
  - "free domain-algebra" w.r.t. powerdomain axioms

Formalization in Isabelle/HOLCF

HOLCF logic in Isabelle2008 now includes powerdomain library

- Type constructors
  - `upper_pd`, `lower_pd`, `convex_pd`
- Operations on each type
  - `unit`, `bind`, `plus`, `map`, `join`
  - coercions from `convex_pd`
- Proof automation
  - ACI normalization
  - Simplifying inequalities

Summary

- Powerdomains are well-suited for reasoning about nondeterminism in functional programs
- You can do proofs about powerdomains right now in Isabelle/HOLCF

- Future work
  - Proofs about parallel code, using monad transformers