Type Correct Changes

A Safe Approach to Version Control Implementation

Jason Dagit
dagit@codersbase.com

Main Darcs Challenges

- Performance
- Conflict handling ← Summer of Code 2007
- Darcs source is fragile

Darcs Source is Fragile

- We would like to improve:
  - Transparency
  - Robustness
  - Approachability

Transparency * Robustness * Approachability

Darcs Source is Fragile

- Transparency means:
  - Is the source easy to understand?
  - Can you tell how darcs will behave?

Transparency * Robustness * Approachability

Darcs Source is Fragile

- Robustness means:
  - Reduced risk of regression
  - Can we refactor with confidence?

Transparency * Robustness * Approachability

Darcs Source is Fragile

- Approachability means:
  - Reduced learning curve for new devs
  - Source is self-documenting

Transparency * Robustness * Approachability

Darcs

- Based on data model known as Patch Theory
  - Still a novel approach to VCS
  - Manages significant complexity
- Provides relatively simple UI
  - Cherry picking
  - Automatic dependency calculation
- Inspired several other VCS

Outline

- Theory: Patches
- Tools: GADTs
- Solution: Type encoding
- Evaluation: Darcs source improvements

Patch Theory

- David Roundy developed Patch Theory
- See http://darcs.net/manual/node9.html
- Patches are similar to diffs, but also have an implicit dependency on context

Patch

- Invertible transformation of files and directories
- Depends on more than repository state

Patch Sequences

- Repository stores one sequence of patches
- Patch sequence defines a transformation of repository state

Sequences * Commutation * Context * Merge * Context Equality

Commute

- Given two patches, **A** and **B**:
  AB ↔ **B_1A_1**

- Partial relation

- Self-inverting:
  If **AB** ↔ **B_1A_1** then, **B_1A_1** ↔ **AB**

Sequences * Commutation * Context * Merge * Context Equality

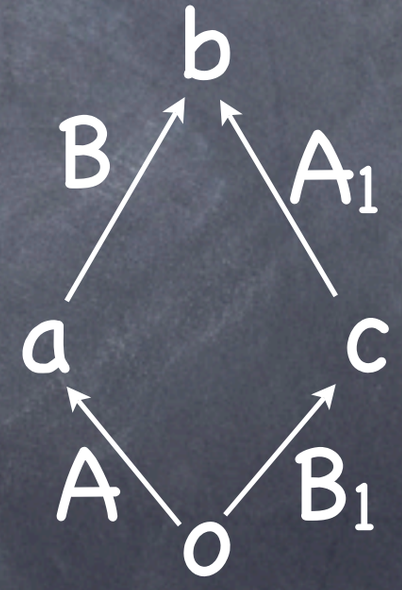
Patch Context

- Sequence of patches or any permutation of the sequence obtained by commutation.
- Notation:

  ${}^{o}\mathbf{A}^{a}$, ${}^{a}\underline{\underline{\mathbf{A}}}^{o}$, ${}^{o}\mathbf{A}^{a}\mathbf{B}^{b} = {}^{o}\mathbf{A}\mathbf{B}^{b}$, ${}^{o}\mathbf{A}\mathbf{B}^{b} \leftrightarrow {}^{o}\mathbf{B}_{1}\mathbf{A}_{1}{}^{b}$

Merge

- Given $^{o}\mathbf{A}^{a}$, $^{a}\mathbf{B}^{b}$, $^{c}\mathbf{A}_{1}{}^{b}$, $^{o}\mathbf{B}_{1}{}^{c}$ then,
  $^{a}\underline{\underline{\mathbf{A}}}\mathbf{B}_{1}{}^{c} \leftrightarrow {}^{a}\mathbf{B}\underline{\underline{\mathbf{A}}}_{1}{}^{c}$ iff $^{o}\mathbf{B}_{1}\mathbf{A}_{1}{}^{b} \leftrightarrow {}^{o}\mathbf{A}\mathbf{B}^{b}$.
- Symmetric
- Given $^{o}\mathbf{A}^{a}$ and $^{o}\mathbf{B}_{1}{}^{c}$:
  $^{o}\mathbf{A}\underline{\underline{\mathbf{A}}}\mathbf{B}_{1}{}^{c} \leftrightarrow {}^{o}\mathbf{A}\mathbf{B}\underline{\underline{\mathbf{A}}}_{1}{}^{c}$, discard $\underline{\underline{\mathbf{A}}}_{1}$
  $^{o}\mathbf{B}_{1}\underline{\underline{\mathbf{B}}}_{1}\mathbf{A}^{a} \leftrightarrow {}^{o}\mathbf{B}_{1}\mathbf{A}_{1}\underline{\underline{\mathbf{B}}}^{a}$, discard $\underline{\underline{\mathbf{B}}}$



Sequences * Commutation * Context * Merge * Context Equality

Context Equality

- Given $^{x}\mathbf{A}_1{}^{y}$ and $^{u}\mathbf{A}_2{}^{v}$, corresponding to same change*, if x = u or y = v then, $\mathbf{A}_1 = \mathbf{A}_2$.
- Note: not true for arbitrary patches

* $\mathbf{A}_1$ and $\mathbf{A}_2$ are related by commutation

Sequences * Commutation * Context * Merge * Context Equality

Outline

- Theory: Patches
- Tools: GADTs
- Solution: Type encoding
- Evaluation: Darcs source improvements

Generalized Algebraic Data Types

- Provide a uniform way to use:
  - Existential types
  - Phantom types
  - Witness types

Sealed

```
data Sealed a where
   Sealed :: a x -> Sealed a
```

- x is hidden
- Unsealing gives fresh distinct type, or eigenvariable, instead of x

Sealed * EqCheck * Forward List

EqCheck

- Type equality witness

```
data EqCheck a b where
    IsEq :: EqCheck a a
    NotEq :: EqCheck a b
```

- IsEq -- Proof that a = b
- NotEq -- No new information about a and b

Sealed * EqCheck * Forward List

Ordered Lists

```
data FL a x y where
  NilFL :: FL a x x
  (:>:) :: a x y -> FL a y z -> FL a x z
```

```
ord :: Char -> Int
chr :: Int -> Char

ord :>: chr :>: NilFL :: FL (->) Char Char
```

Sealed * EqCheck * Forward List

Outline

- Theory: Patches
- Tools: GADTs
- Solution: Type encoding
- Evaluation: Darcs source improvements

Type Encoding

- Add context to patches:

```
data Patch x y where
   Identity :: Patch x x
   FP :: FileName -> FilePatchType x y -> Patch x y
   ...
```

Context * Sequences * Commutation * Merge * Context Equality

Type Encoding

- Forward lists of patches:

```
p :>: q :>: NilFL :: FL Patch x z
```

  - Enforces ordering statically
  - Filter requires EqCheck
  - Not all list operations work on forward lists
    - No way to sort forward lists

Context * Sequences * Commutation * Merge * Context Equality

Type Encoding

- Original commute:

```
commute :: (Patch, Patch) -> Maybe (Patch, Patch)
```

- New commute:

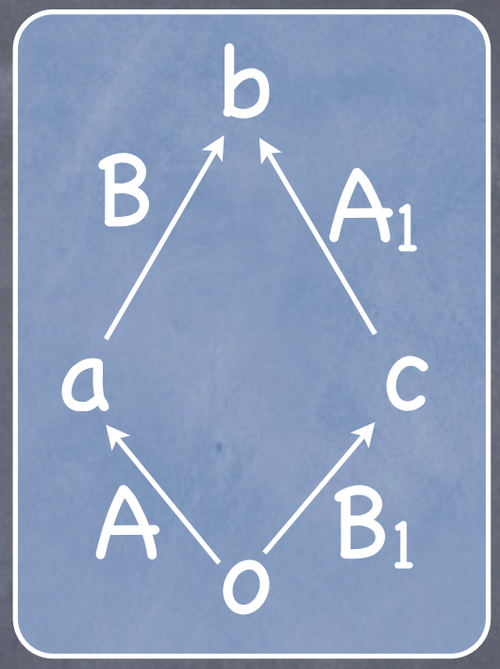
```
data (a1 :> a2) x y where
  (:>) :: a1 x z -> a2 z y -> (a1 :> a2) x y

commute :: (Patch :> Patch) x y -> Maybe ((Patch :> Patch) x y)
```

Context * Sequences * **Commutation** * Merge * Context Equality

Type Encoding

Recall:



- Original merge

```
merge :: (Patch, Patch) -> Maybe (Patch, Patch)
```

- New merge

```
data (a1 :/\: a2) where
  (:/\:) :: a1 x y -> a2 z y -> (a1 :/\: a2) x z

data (a1 :\/: a2) where
  (:\/:) :: a1 x y -> a2 x z -> (a1 :\/: a2) y z

merge :: (Patch :\/: Patch) a c
      -> Maybe ((Prim :/\: Prim) a c)
```

Context * Sequences * Commutation * **Merge** * Context Equality

Type Encoding

Recall:

```
data EqCheck a b where
    IsEq :: EqCheck a a
    NotEq :: EqCheck a b
```

- Only rarely required:

```
(=\/=) :: Patch a b -> Patch a c -> EqCheck b c
```

```
(=/\=) :: Patch a c -> Patch b c -> EqCheck a b
```

- Lifting run-time check requires `unsafeCoerce`
  - Hard to avoid due to existential types

Context * Sequences * Commutation * Merge * Context Equality

Outline

- Theory: Patches
- Tools: GADTs
- Solution: Type encoding
- Evaluation: Darcs source improvements

Darcs Defects Found

- Example defects found by our type encoding:
  - Interactive changes
  - Removed `rempatch` and `commute_by`
  - Refactor of `get_common_and_uncommon`
  - Buggy handling of pending state

Tricky Spots

- Error messages
- Inferred type is less polymorphic than expected
- My brain just exploded
- Wobbly types / Rigid type context

Tricky Spots

- Caution needed with (=\/=) and (=/\=) to avoid unsound definitions
- EqCheck provides proof that a = b

```
data Patch a b = P

unsafeCoerce :: forall a b. a -> b
unsafeCoerce x = case a =\/= b of
                   IsEq -> x
                   NotEq -> error “impossible”
  where (a, b) = (P, P) :: (P () a, P () b)
```

Improved Source Code

- How have the following changed?
  - Transparency
  - Robustness
  - Approachability

Transparency * Robustness * Approachability

Improved Source Code

- Is the source easy to understand?
  - Context-aware type signatures
- Can you tell how darcs will behave?
  - Machine checkable documentation

Transparency * Robustness * Approachability

Improved Source Code

- Reduced risk of regression:
  - Type signatures act as a contract
  - Statically verified
- We can refactor with confidence!

Transparency * Robustness * Approachability

Improved Source Code

- Reduced learning curve for new devs
  - Moved emphasis of understanding from Patch Theory to Haskell's type system
- Source is self-documenting
  - More type information means less guess work

Transparency * Robustness * Approachability

Future Work

- Make Repository IO Monad context-aware
  - Similar to Monad Regions
  - Track context transformations
  - Restrict IO actions

Conclusion

- Static types let the compiler do the hard work
- Refactoring is safer
- Machine checkable documentation is good
- Static analysis means no new run-time overhead

Thank You!
Questions?

Jason Dagit
dagit@codersbase.com