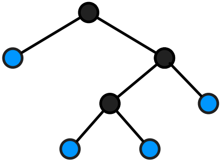
Isomorphisms for context-free types

Wouter Swierstra

April 7, 2006

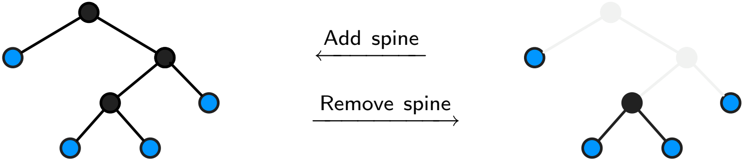
The University of Nottingham

Into the rabbit hole . . .



Into the rabbit hole . . .

Into the rabbit hole . . .



What is an isomorphism?

An isomorphism between two types σ and τ consists of functions psi :: $\sigma \to \tau$ and isp :: $\tau \to \sigma$ such that:

- $\mathsf{psi} \circ \mathsf{isp} = \mathsf{id}_\tau$
- $\mathsf{isp} \circ \mathsf{psi} = \mathsf{id}_\sigma$
- No peeking!

When are two types different?

- What should we do if we can't find an isomorphism between two types?
- We can show two data types are distinct by counting the number of inhabitants.
- Are the following familiar types isomorphic?

data List a = Nil | Cons a (List a)

data Tree a = Leaf | Node (Tree a) (Tree a)

What is a type?

Context-free types over an index set I are built from:

$0, \sigma + \tau$	coproducts
$1, \sigma \times \tau$	products
$i \in I$	parameters
$X, Y, \ldots$	recursive variables
$\mu X.\sigma$	least fixed point

For instance:

- Lists: $\mu X.1 + A \times X$
- Binary trees: $\mu X.1 + X \times A \times X$

Types and grammars

- These context-free types resemble context-free grammars.
- There are two important differences:
  1. Products commute $\sigma \times \tau \simeq \tau \times \sigma$
  2. Coproducts are not idempotent $\sigma + \sigma \not\simeq \sigma$
- Can we use parsing technology to distinguish different types?

Parser combinators

- Goal: Write a parser of type that recognizes when a given string is in a language or not:

$$I^* \to 2$$

- Intermediate: We write combinators of the following type:

$$I^* \to \mathcal{P}_{\text{fin}}(I^*)$$

- We can run an intermediate parser by checking if the entire input has been consumed.

Monadic parser combinators

- Lists and finite powersets have both certain structure.
- They form **monoids**.

$$0 :: a$$
$$\oplus :: a \rightarrow a \rightarrow a$$

- They form **monads**.

$$\mathrm{return} :: a \rightarrow m\, a$$
$$\gg\!= :: m\, a \rightarrow (a \rightarrow m\, b) \rightarrow m\, b$$

- We can define parser combinators using only these properties.

Rethinking the underlying monad

- How can we adapt monadic parser combinators to distinguish different types?
- It suffices to only change the underlying structure!
- Instead of powersets and lists we use multisets:
  - Order of input doesn't matter.
  - The number of parses is important.

Monadic parsers revisited

- Goal: Write a 'parser' that counts the number of inhabitants of a given type:

$$\mathcal{M}(I) \rightarrow \mathbf{N}$$

- Intermediate: We write combinators of the following type:

$$\mathcal{M}(I) \rightarrow \mathcal{M}(\mathcal{M}(I))$$

- We should show that multisets have the required structure...
- The actual parsers do not change!

Powerseries

- The multiset parsers give us a new interpretation of our types.
- We consider a type σ over a singleton index set I as:

$$\sum_{n \in \mathbf{N}} a_n \times X^n$$

where a_n is the result of running the σ parser on n.
- **Lemma** Two types are isomorphic iff their powerseries are equal.

Powerseries

- The multiset parsers give us a new interpretation of our types.
- We consider a type σ over a singleton index set I as:

$$\sum_{n \in \mathbf{N}} a_n \times X^n$$

where a_n is the result of running the σ parser on n.
- **Lemma** Two types are isomorphic iff their powerseries are equal.

The essence of a type is a powerseries.

Conclusions

- Formalizing these intuitions requires quite some work.
- We have a semi-algorithm for deciding whether or not two types are isomorphic.
- Is the problem decidable?
- Is there a subset of types for which isomorphism is decidable?