NOTES ON CLOSURES

ANDREA FALCONI

Abstract. These notes show how some well known relation closures may be seen as induction systems. At the same time, we provide Haskell code to compute an arbitrary inductive closure as well as the reflexive, symmetric, and transitive closure of a binary relation.

Induction Systems

An induction system is in a sense a generalization of the natural numbers.

$\mathfrak{S} \;:=\; (U,\ X,\ \Sigma)$	*induction system*
$U \neq \emptyset,\ X \subset U$	U and X are sets
$\Sigma = \{\sigma : U^{k_\sigma} \longrightarrow U,\ k_\sigma \in \mathbb{N}\}$	set of finitary functions on U

With an induction system we associate a sequence of sets and a new one built out of their union.

$$\begin{aligned} &X_0 := X \\ &X_{n+1} := X_n \ \cup\ \{\sigma(x_1, \dots, x_{k_\sigma}) \mid x_i \in X_n,\ \sigma \in \Sigma\} \\ &\overline{X} := \cup X_n \end{aligned}$$

The above process is referred to as the *inductive definition* of $\overline{X}$, which is called the *inductive closure* of X under Σ. The following holds:

$$X_0 \subset \overline{X} \tag{1}$$

$$\forall\sigma : \quad \sigma \in \Sigma \ \wedge\ x_1, \dots, x_{k_\sigma} \in \overline{X} \ \Rightarrow\ \sigma(x_1, \dots, x_{k_\sigma}) \in \overline{X} \tag{2}$$

i.e. $\overline{X}$ is $\mathfrak{S}$-*closed*; moreover it is the *smallest* $\mathfrak{S}$-*closed* set:

$$(1) \wedge (2) \text{ hold for } \Gamma \ \Rightarrow\ \overline{X} \subset \Gamma \tag{3}$$

Date: September, 2008.

Key words and phrases. Induction systems, closures, Haskell.

Computing an Inductive Closure

We now look at a Haskell program to compute the inductive closure $\overline{X}$ of a set X given the finitary functions Σ. The main idea is that of using a function to produce the sequence of sets X_n, which we represent with a Haskell *infinite* list; also, out of convenience, we decide to use lists again to represent sets and tuples. Thus, each σ will have the Haskell type [u]→u where u is the type of the elements of U.

```
induct :: Eq u ⇒ [u] → [([u]→ u, Int)] → [[u]]
induct xs sigmas = ys : induct ys sigmas
    where
    ys = next xs sigmas
```

The next function in the above definition builds X_{n+1} given X_n and the list of finitary functions in Σ; each σ is paired up to its arity, hence the type of sigmas becomes [([u]→ u, Int)]. For example

```
take 3 $ induct [0] [(λ[x]→x+1,1)]                    ⇒ [[0,1],[0,1,2],[0,1,2,3]]
take 3 $ induct [0] [(λ[x]→x+1,1),(λ[x]→x−1,1)]   ⇒ [[0,1,−1],[0,1,−1,2,−2],
                                                          [0,1,−1,2,−2,3,−3]]
```

Because the arity k_σ of each σ is given as input, we can easily construct all the tuples in $X_n^{k_\sigma}$ which are to be fed into σ (in the form of a list) to generate the elements of X_{n+1}. This leads to the following definition of next:

```
next :: (Eq u) ⇒ [u] → [([u]→ u, Int)] → [u]
next xs sigmas = nub (xs ++ ys)
    where
    ys = concat [img s k xs | (s,k)←sigmas]
    img s k = map s ∘ lists k
```

where the lists function generates all lists of length k from the input list.

```
lists :: Int → [u] → [[u]]
lists n xs
    | n ≤ 0     = [[]]
    | otherwise = concat [map (x:) rs | x←xs]
    where
    rs = lists (n−1) xs
```

Examples:

```
lists 1 [1,2]  ⇒  [[1],[2]]
lists 2 [1,2]  ⇒  [[1,1],[1,2],[2,1],[2,2]]

next [0] [(λ[x]→x+1,1)]                    ⇒  [0,1]
next [0] [(λ[x]→x+1,1),(λ[x]→x−1,1)]   ⇒  [0,1,−1]
```

By construction, $X_n \subset X_{n+h}$ $\forall h > 0$; a useful remark is that if

$$\overline{X} \text{ finite} \;\Rightarrow\; \exists m \forall h:\; X_m = X_{m+h} \tag{4}$$

Proof. Let $\lambda(x) = min\{n \mid x \in X_n\}$, for any given $x \in \overline{X}$ and let $m = max\ \lambda(\overline{X})$. Then, out of necessity, we have $X_m = X_{m+1} = \cdots = \overline{X}$. (For if that were not the case, $\exists q \exists x : m < q \;\wedge\; x \in X_q \setminus X_m$, against $\lambda(x) \leq m$.) □

The following function exploits this fact to compute the inductive closure of a list xs (representing the initial set X_0) under the given sigmas, assuming the closure is finite. (This function will never terminate if the closure is infinite and so it should only be used when the closure is already known to be finite.)

```
finClosure :: Eq u ⇒ [u] → [([u]→ u, Int)] → [u]
finClosure xs sigmas = pickWhenSameSize 0 $ induct xs sigmas
   where
   pickWhenSameSize m (x:xs)
      | m == n      = x
      | otherwise = pickWhenSameSize n xs
      where
      n = length x
```

Closures of Binary Relations

Let A be a set and $R \subset A \times A = A^2$ a binary relation on A. We would like to determine the smallest possible extension to R which is reflexive, i.e. the *reflexive closure* rR of R. More precisely, we ask: what is the smallest relation rR on A such that

$$R \subset rR \tag{5}$$

$$\forall a:\; a \in A \;\Rightarrow\; (a,a) \in rR \tag{6}$$

$$(5) \wedge (6) \text{ hold for } \Gamma \;\Rightarrow\; rR \subset \Gamma \tag{7}$$

Now define $\forall a$ the constant function $\rho_a : \{()\} = A^0 \to A^2$ as $\rho_a() = (a,a)$ and let $\Sigma := \{\rho_a \mid a \in A\}$. Trivially, (6) is equivalent to

$$\forall \rho_a:\; \rho_a \in \Sigma \;\Rightarrow\; \rho_a() \in rR$$

and thus if we consider the induction system $\mathfrak{S} := (A^2,\ R,\ \Sigma)$, we have that rR is the inductive closure of R under Σ — i.e. $rR = \overline{R}$. Obviously if A is finite, so is rR.

We can use a similar procedure to see that the *symmetric closure* sR of R is the inductive closure of R under $\Sigma := \{\sigma\}$, where $\sigma : A^2 \to A^2$ is defined by

$\sigma\ (x,y) = (y,x)$. Finally, for the *transitive closure* tR, take $\Sigma := \{\tau\}$, where $\tau : A^2 \times A^2 \to A^2$ is defined by

$$(u,v)\tau(x,y) = \begin{cases} (u,y) & \text{if } v = x, \\ (u,v) & \text{otherwise.} \end{cases}$$

Again, if A is finite, so are sR and tR.

Computing Some Closures

We can now use the above to obtain concise, straightforward Haskell code to compute the the reflexive, symmetric, and transitive closures of a relation R on a *finite* set A.

```
rxClosure xs rs = finClosure rs sigmas
    where
    sigmas = map (λx → (λ_→(x,x),0)) xs

symClosure rs = finClosure rs sigmas
    where
    sigmas = [(λ[(x,y)] → (y,x), 1)]

trClosure rs =  finClosure rs [(t, 2)]
    where
    t [(x,y),(u,v)]
        | y == u     = (x,v)
        | otherwise = (x,y)
```

Examples:

```
rxClosure [1,2,3] [(1,2)]        ⇒  [(1,2),(1,1),(2,2),(3,3)]

symClosure [(1,2),(1,3)]         ⇒  [(1,2),(1,3),(2,1),(3,1)]

trClosure [(1,2),(2,3),(3,4)]    ⇒  [(1,2),(2,3),(3,4),(1,3),(2,4),(1,4)]
```