The Weird World of Bi-directional Programming

Benjamin C. Pierce

University of Pennsylvania

Abstract. Programs generally work in just one direction, from input to answer. But sometimes we need a program to work in two directions: after calculating an answer, we want to *update* the answer and then somehow calculate backwards to find a correspondingly updated input. Of course, in general, a given update to the answer may not correspond to a unique update on the input, or to any at all; we need an “update translation policy” that tells us which updates can be translated and how to choose among translations when there are many possibilities. The question of how to determine such a policy has been called the *view update problem* in the database literature.

Many approaches to this problem have been devised over the years; most have taken existing database query languages (such as SQL) as their starting points and then proposed ways of describing or inferring update policies. More recently, several groups have begun working to design entirely new languages in which programs are inherently bi-directional – i.e., in which every program can be read from left to right as a map from inputs to answers *and* from right to left as (roughly) a map from updated answers to updated inputs. Moreover, bi-directionality in these languages is treated compositionally: each primitive works in both directions, and the two directions of compound programs can be derived from the two directions of their subcomponents.

This talk charts some interesting regions of the world of bidirectional programming and bi-directional language design, using as a touchstone our experiences at the University of Pennsylvania in the context of the Harmony project, where bi-directional languages – one for transforming trees and another for relational data – play a crucial role in the architecture of a universal data synchronizer.