Two Strategies to Data-Refine an Equivalence to a Forest

Philipp Heuberger*
Turku Centre for Computer Science (TUCS) and Åbo Akademi
DataCity, Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland
e-mail: pheuberg@abo.fi

Abstract. Well-known strategies give incentive to the algorithmic refinement of programs and we ask in this paper whether general patterns also exist for data refinement. In order to answer this question, we study the equivalence relation problem and identify the motivations to replace the equivalence relation by a data structure suitable for efficient computation.

1 Introduction

The equivalence relation problem consists of the specification and implementation of a manager that maintains an equivalence relation over a given set. The manager allows for the dynamic addition of related (equivalent) elements together with an equivalence test of two elements.

Abrial distinguishes between three orthogonal aspects of refinement [1], i.e. the decreasing of non-determinism, the weakening of the precondition and the changing of the variable space. The third aspect is illustrated by our treatment of the equivalence relation problem, where a relation in an initial specification is replaced by a function. The motivation for using two different mathematical objects to present the problem is that a relation is well suited for the abstract description and the pointer structure is close to the concrete representation in the implementation.

Algorithmic refinement has singled out reoccuring patterns for the introduction of a loop construct. Dijkstra [6] motivates formally each development step with great care and Gries [8] and Cohen [4] formulate strategies like "deleting a conjunct" or "replacing constants by fresh variables". In the examples treated by them variables are not changed and consequently no strategy for doing so is formulated.

There are many ways to specify an equivalence relation manager. We present a variety of them in the B specification method and propose general strategies that guide the changing of the equivalence to the data structure known as "Union-Find" or "Fischer/Galler-Trees".

* This work has been financed by the Swiss-TMR-Fellowship of the Federal Office for Education and Science and is supported by The Swiss National Science Foundation.
The equivalence relation problem is well known and described in many algorithmic textbooks, e.g. [6, 3, 5]. It has been formally treated thoroughly before in VDM [12], based on the mathematical object of a set partition. Fraer has carried out a development in B on a larger example that includes the treatment of the equivalence relation problem [7]. His major strategy is "design for provability". Our approach is different: we look for heuristical guidance to find a suitable data refinement in order to calculate the refined program by proof and in this paper we present two strategies to data-refine an equivalence to a forest. The goal is to identify general strategies for data refinement. In an initial attempt documented in [10], we have refined a specification corresponding to the VDM treatment on which we have partly reported in [11]. We have used the B-Toolkit [2, 13] in the initial development and have generated the machines in this document with AtelierB.

2 The Equivalence Relation Problem

An equivalence relation (or equivalence for short) $\theta$ on a set $A$ is a subset of $A \times A$ such that the following three properties hold for $a, b, c \in A$, where $a \theta b$ stands for $(a, b) \in \theta$:

- Reflexivity: $a \theta a$
- Symmetry: $a \theta b \Rightarrow b \theta a$
- Transitivity: $a \theta b \land b \theta c \Rightarrow a \theta c$

For a better understanding of the algebraic definition, let us consider the representation by graphs. In general, every relation corresponds to a graph. The domain set $A$ of a relation corresponds to the set of nodes in a graph and the relation $\theta$ corresponds to the set of edges in the graph. Equivalences belong to a particular type of graph, i.e. graphs which consist of fully connected subcomponents only. The equivalence relation problem basically is the construction of the smallest equivalence containing an arbitrary user-given relation. In mathematics, the construction is called the reflexive, symmetric and transitive closure or equivalence closure for short. In figure 1, the relation represented as a graph to the right is the equivalence closure of the one to the left. Observe that the single-edged cycles at each node are due to the reflexivity and the bi-directed edges to the symmetry. Moreover, symmetry and transitivity together demand an edge between $b$ and $c$.

The equivalence closure gives a very static and functional view of the problem, i.e. there is one input, i.e. the user-given relation, and one output, i.e. the equivalence. For an equivalence manager though, the problem is more involved. The relation is not passed as a whole, but every pair of connected elements is submitted separately. The user may update the relation and inquire the equivalence dynamically many times in arbitrary order. Translated into the functional view, the transitive closure is applied to a dynamically changing input.

An equivalence manager basically has two operations: an update operation and a query operation. For simplicity, pairs of related elements are only added by