High-Level Optimization of Abstract Data Types

Anthony Hunt[1][0009−0005−1085−3343] and Emil Sekerinski[2][0000−0001−9788−5842]

Department of Computing and Software, McMaster University, Hamilton, ON L8S4L8, Canada {hunta12,emil}@mcmaster.ca

Sets, sequences, and relations are essential to describing system behaviours in formal specification and modelling languages like Event-B, Alloy, and Dafny. Conversely, high-level programming languages (like Python and Haskell) support only a subset of these types and operators, preferring the use of implementation-aware types like arrays and classes. In ongoing work, we use specification language semantics to enhance programming language performance. We propose a language capable of high-level expressions for abstract data types with efficient, guaranteed bounds for runtime and memory consumption.

Consider a service system that tracks all attendees and workshops within a conference. Each visitor may attend at most one workshop per day, and only one workshop may be held in a particular room. Then, if a workshop organizer needs to order meals for each attendee in a *room*, an appropriate model is:

$$location : Workshop \rightarrowtail Room \tag{1}$$

$$attends : Visitor \nrightarrow Workshop \tag{2}$$

$$meals \coloneqq card((location^{-1} \circ attends^{-1})[\{room\}]) \tag{3}$$

We envisage that programmers write such expressions and efficient code is generated. While naive compilation constructs intermediate relations for $location^{-1}$ and $attends^{-1}$, more efficient code computes the result directly, as in Fig. 1 (a). Semi-automatic data type refinement could then produce code like Fig. 1 (b).

In contrast to a conventional language's implementation-aware operations, the semantics of specification expressions closely follow set theory; theorems become provable, high-level term rewriting passes. Our prototype compiler successfully rewrites queries similar to (3) into the efficient loop structures of Fig. 1.

$meals := 0$
for p, r **in** $location$ **do**
  if $r = room$ **then**
    for v, p' **in** $attends$ **do**
      if $p = p'$ **then**
        $meals := meals + 1$

(a) Running time is $O(|location||attends|)$.

$meals := 0$
for v, p **in** $attends$ **do**
  if $location[p] = room$ **then**
    $meals := meals + 1$

(b) If *location* is a hashmap, the running time may be reduced to $O(|attends|)$.

Fig. 1: Two implementations of query (3).

[1] Presenter, recent graduate, and current Master's student at McMaster University.

[2] Full professor at McMaster University in the Department of Computing and Software. He received his Ph.D. from the University of Karlsruhe, Germany, and was at Åbo Akademi, Finland, before joining McMaster. Link to homepage.