Science of Computer Programming 138 (2017) 26–48

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Projecting programs on specifications: Definition and implications

Jules Desharnais [a], Nafi Diallo [b], Wided Ghardallou [c], Ali Mili [b,*]

[a] Département d'informatique et de génie logiciel, Université Laval, Québec, Canada
[b] New Jersey Institute of Technology, Newark, NJ, United States
[c] Faculté des Sciences de Tunis, El Manar, Tunisia

ARTICLE INFO

Article history:
Received 8 January 2016
Received in revised form 17 November 2016
Accepted 19 November 2016
Available online 2 February 2017

Keywords:
Absolute correctness
Relative correctness
Refinement
Projection
Lattices

ABSTRACT

Given a specification R, it is common for a candidate program P to be doing more than R requires; this is not necessarily bad, and is often unavoidable, due to programming language constraints or to otherwise sensible design decisions. In this paper, we introduce a relational operator that captures, for a given specification R and candidate program P, the functionality delivered by P that is relevant to R. This operator, which we call the *projection* of P over R (for reasons we explain), has a number of interesting properties, which we explore in this paper.

© 2017 Elsevier B.V. All rights reserved.

1. Projecting programs on specifications

Given a specification R and a program P that is written to satisfy R, we refer to P as a *candidate* program for specification R, and we refer to R as the *target* specification of program P, regardless of whether P does or does not satisfy specification R. Given a specification R and a candidate program P, the program P could well be falling short of some of the requirements of R, while at the same time exceeding (i.e. doing more than needed) on some other requirements. There is no shortage of reasons why a program may fall short of the requirements mandated by the specification, but there are also ample reasons why a program may do more than required: these include cases where excess functionality is a byproduct of normal design decisions, cases where it stems from programming language constructs, and more generally the need to bridge the gap between a potentially non-deterministic specification and a deterministic program. Consider for example the following relational specification on space S defined by integer variables x and y:

$$R = \{(\langle x, y\rangle, \langle x', y'\rangle) | x' = x + y\},$$

and consider the following program on the same space:

```
while (y!=0) {x:=x+1; y:=y-1}.
```

* Corresponding author.
E-mail addresses: Jules.Desharnais@ift.ulaval.ca (J. Desharnais), ncd8@njit.edu (N. Diallo), wided.ghardallou@gmail.com (W. Ghardallou), mili@cis.njit.edu (A. Mili).

http://dx.doi.org/10.1016/j.scico.2016.11.006
0167-6423/© 2017 Elsevier B.V. All rights reserved.

For non-negative values of y this program computes the sum of x and y in x while placing 0 in y; for negative values of y, it fails to terminate. Hence its function can be written as:

$$P = \{(\langle x, y\rangle, \langle x', y'\rangle) | y \geq 0 \wedge x' = x + y \wedge y' = 0\}.$$

This program does not do everything that specification R requires, since it fails to compute the sum of x and y into x for negative values of y; on the other hand, it puts 0 in y even though the specification does not ask for it (but this is a side effect of the algorithm we have chosen to satisfy the specification). Hence looking at specification R and program P, we would like to think that the functionality of P that is relevant to (mandated by) R is captured by the following relation:

$$\pi = \{(\langle x, y\rangle, \langle x', y'\rangle) | y \geq 0 \wedge x' = x + y\}.$$

Whatever else P does (e.g. it sets y to 0) is not relevant to specification R; on the other hand, whatever else the specification mandates (computing the sum of x and y into x for negative values of y), program P is not delivering. In other words, relation π fails to capture the condition $y' = 0$ because that is not mandated by the specification, and it fails to capture the condition the case $y < 0$ because that is not delivered by the candidate program P.

In this paper, we introduce a relational operator that captures, for a given specification R and program P, the functionality delivered by P that is relevant to (i.e. mandated by) R; we refer to this operator as the *projection of P over R*, for reasons that we elucidate in our subsequent discussions. We define this operator in section 4, and we discuss its properties and implications in section 5. The concept of projection of a program on a specification comes about as a byproduct of the definition of relative correctness, i.e. the property of a program to be more-correct than another with respect to a specification. Relative correctness, due to [3], is discussed in section 3. In section 2 we introduce the mathematical background that we use throughout this paper. In section 6 we briefly discuss related work; because no other authors we know of have written about projections, the closest work to ours pertains to relative correctness. Finally, in section 7 we summarize and assess our work, and we discuss possible future directions of research.

2. Mathematical background

2.1. Relational notations

In this section, we introduce some elements of relational mathematics that we use in the remainder of the paper to carry out our discussions. We assume the reader familiar with relational algebra, and we generally adhere to the definitions of [2,19]. Dealing with programs, we represent sets using a programming-like notation, by introducing variable names and associated data types (sets of values). For example, if we represent set S by the variable declarations

$$x : X;\ \ y : Y;\ \ z : Z,$$

then S is the Cartesian product $X \times Y \times Z$. Elements of S are denoted in lower case s, and are triplets of elements of X, Y, and Z. Given an element s of S, we represent its X-component by $x(s)$, its Y-component by $y(s)$, and its Z-component by $z(s)$. When no risk of ambiguity exists, we may write x to represent $x(s)$, and x' to represent $x(s')$.

A relation on S is a subset of the Cartesian product $S \times S$; given a pair (s, s') in R, we say that s' is an *image* of s by R. Special relations on S include the *universal* relation $L = S \times S$, the *identity* relation $I = \{(s, s') | s' = s\}$, and the *empty* relation $\phi = \{\}$. Operations on relations (say, R and R') include the set-theoretic operations of *union* ($R \cup R'$), *intersection* ($R \cap R'$), *difference* ($R \setminus R'$) and *complement* ($\overline{R}$). They also include the *relational product*, denoted by $R \circ R'$, or (RR', for short) and defined by:

$$RR' = \{(s, s') | \exists s'' : (s, s'') \in R \wedge (s'', s') \in R'\}.$$

The *power* of relation R is denoted by R^n, for a natural number n, and defined by $R^0 = I$, and for $n > 0$, $R^n = R \circ R^{n-1}$. The *reflexive transitive closure* of relation R is denoted by R^* and defined by $R^* = \{(s, s') | \exists n \geq 0 : (s, s') \in R^n\}$. The *converse* of relation R is the relation denoted by $\widehat{R}$ and defined by

$$\widehat{R} = \{(s, s') | (s', s) \in R\}.$$

Finally, the *domain* of a relation R is defined as the set $dom(R) = \{s | \exists s' : (s, s') \in R\}$, and the *range* of relation R is defined as the domain of $\widehat{R}$. Operator precedence is adopted as follows: unary operators apply first, followed by product, then intersection, then union.

A relation R is said to be *reflexive* if and only if $I \subseteq R$, *symmetric* if and only if $R = \widehat{R}$, *antisymmetric* if and only if $R \cap \widehat{R} \subseteq I$, *asymmetric* if and only if $R \cap \widehat{R} = \phi$, and *transitive* if and only if $RR \subseteq R$. A relation is said to be a *partial ordering* if and only if it is reflexive, antisymmetric, and transitive. Also, a relation R is said to be *total* if and only if $I \subseteq R\widehat{R}$, and a relation R is said to be *deterministic* (or: a *function*) if and only if $\widehat{R}R \subseteq I$. In this paper we use a property to the effect that two functions f and f' are identical if and only if $f \subseteq f'$ and $f'L \subseteq fL$. A relation R is said to be a *vector* if and only if $RL = R$; a vector on space S is a relation of the form $R = A \times S$, for some subset A of S; we use vectors to represent subsets of S; in particular, we use the product RL as a relational representation of the domain of R.

Download English Version:

https://daneshyari.com/en/article/4951808

Download Persian Version:

https://daneshyari.com/article/4951808

Daneshyari.com