A Array Representation

We consider arrays as first-order objects that can be parameters of functions, predicates, and can be quantified over. In first-order logic, this means that we have sorts for arrays, and one sort for each dimension. In the following, we denote by int the integer sort, and $array_k$ the k-dimensional array sort, where $k \geq 1$.

To denote the value of an array at some indices, for each $k \geq 1$, we introduce a special function named $d\mathrm{k}array$ of the arity:

$$d\mathrm{k}array : \ array_k \times int^k \to int,$$

as we consider only integer valued arrays. Thus, $d1array(a, i)$ denotes the value of a one-dimensional array a at index i, i.e. $a[i]$ under a conventional notation, and $d2array(b, i, j)$ stands for $b[i][j]$ for two-dimensional array b. We can also introduce a function to denote the size of an array.

Recall that we generate a set of axioms for a program P under a language $\vec{X}$. The generated set of axioms captures the changes of P on $\vec{X}$, so $\vec{X}$ needs to include all functions and predicates that can be changed by P. Therefore, if a program makes changes to, say a two-dimensional array, then $\vec{X}$ must include $d2array$.

When we translate a program to first-order axioms, we need to convert expressions in the program to terms in first-order logic. This is straightforward, given how we have decided to represent arrays. For example, if E is $a(1, 2)+b(1)$, where a is a two-dimensional array and b a one-dimensional array, then $\hat{E}$, the first-order term that corresponds to E, is $d2array(a, 1, 2) + d1array(b, 1)$.

We are now ready to describe how we generate axioms for assignments, First, for integer variable assignments:

Definition A.1 *If P is* `V = E`*, and $V \in \vec{X}$, then $\Pi_P^{\vec{X}}$ is the set of the following axioms:*

$$\forall \vec{x}.X1(\vec{x}) = X(\vec{x}), \text{ for each } X \in \vec{X} \text{ that is different from } V,$$
$$V1 = \hat{E}$$

where for each $X \in \vec{X}$, we introduce a new symbol $X1$ with the same arity standing for the value of X after the assignment, and $\hat{E}$ is the translation of the expression `E` *into its corresponding term in logic as described above.*

For example, if P_1 is

```
I = a(1,2)+b(1)
```

and $\vec{X}$ is $\{I, a, b, d1array, d2array\}$ (a and b are for the two array variables in

the assignment, respectively), then $\Pi^{\vec{X}}_{P_1}$ is the set of following axioms:

$$
\begin{aligned}
&I1 = d2array(a, 1, 2) + d1array(b, 1),\\
&a1 = a,\\
&b1 = b,\\
&\forall x, i.d1array1(x, i) = d1array(x, i),\\
&\forall x, i, j.d2array1(x, i, j) = d2array(x, i, j).
\end{aligned}
$$

Again, we remark that we assume all array accesses are legal. Otherwise, we would need axioms like the following to catch array errors:

$$
\begin{aligned}
&\neg \textit{in-bound}(1, b) \rightarrow arrayError,\\
&\neg \textit{in-bound}((1, 2), a) \rightarrow arrayError,
\end{aligned}
$$

where $\textit{in-bound}(\vec{i}, array)$ means that the index $\vec{i}$ is within the bound of $array$, and can be defined using array sizes.

Definition A.2 *If P is* `V(e1,e2,...,ek) = E`*, then $\Pi^{\vec{X}}_{P}$ is the set of the following axioms:*

$$
\begin{aligned}
&\forall \vec{x}.X1(\vec{x}) = X(\vec{x}),\ \textit{for each } X \in \vec{X} \textit{ which is different from } dkarray,\\
&dkarray1(x, i_1, ..., i_k) =\\
&\qquad ite(x = V \wedge i_i = \hat{e_1} \wedge \cdots \wedge i_k = \hat{e_k}, \hat{E}, dkarray(x, i_1, ..., i_k)),
\end{aligned}
$$

where $ite(c, e, e')$ is the conditional expression: if c then e else e'.

For example, if P_2 is `b(1)=a(1,2)+b(1)`, and $\vec{X}$ is $\{I, a, b, d1array, d2array\}$, then $\Pi^{\vec{X}}_{P_2}$ is the set of following axioms:

$$
\begin{aligned}
&I1 = I,\\
&a1 = a,\\
&b1 = b,\\
&\forall x, i.d1array1(x, i) =\\
&\qquad ite(x = b \wedge i = 1, d2array(a, 1, 2) + d1array(b, 1), d1array(x, i)),\\
&\forall x, i, j.d2array1(x, i, j) = d2array(x, i, j).
\end{aligned}
$$

Notice that $b1 = b$ means that while the value of b at index 1 has changed, the array itself as an *object* has not changed. If we have array assignments like `a=b` for array variables `a` and `b`, they will generate axioms like $a1 = b$.

We now give two simple examples of how the inductive cases work described in the tables[1] provided as supplementary material mentioned previously. See [1] for more details.

Consider P_3 which is the sequence of first P_1 then P_2:

[1] https://goo.gl/2ZBGUr

```
I = a(1,2)+b(1);
b(1)=a(1,2)+b(1)
```

The axiom set $\Pi^{\vec{X}}_{P_3}$ is generated from $\Pi^{\vec{X}}_{P_1}$ and $\Pi^{\vec{X}}_{P_2}$ by introducing some new symbols to connect the output of P_1 with the input of P_2:

$$
\begin{aligned}
&I2 = d2array(a,1,2) + d1array(b,1),\\
&a2 = a,\\
&b2 = b,\\
&\forall x,i.d1array2(x,i) = d1array(x,i),\\
&\forall x,i,j.d2array2(x,i,j) = d2array(x,i,j),\\
&I1 = I2,\\
&a1 = a2,\\
&b1 = b2,\\
&\forall x,i.d1array1(x,i) =\\
&\qquad ite(x = b2 \wedge i = 1, d2array2(a2,1,2) + d1array2(b2,1), d1array2(x,i)),\\
&\forall x,i,j.d2array1(x,i,j) = d2array2(x,i,j),
\end{aligned}
$$

where $I2, a2, b2, d1array2, d2array2$ are new symbols to connect P_1's output with P_2's input. If we do not care about the intermediate values, these temporary symbols can often be eliminated. For this program, eliminating them yields the following set of axioms:

$$
\begin{aligned}
&I1 = d2array(a,1,2) + d1array(b,1),\\
&a1 = a,\\
&b1 = b,\\
&\forall x,i.d1array1(x,i) =\\
&\qquad ite(x = b \wedge i = 1, d2array(a,1,2) + d1array(b,1), d1array(x,i)),\\
&\forall x,i,j.d2array1(x,i,j) = d2array(x,i,j).
\end{aligned}
$$

The most important feature of the approach in [1] is in the translation of loops to a set of first-order axioms. The main idea is to introduce an explicit counter for loop iterations and an explicit natural number constant to denote the number of iterations the loop executes before exiting. It is best to illustrate by a simple example. Consider the following program P_4:

```
while I < M {
  I = I+1;
}
```

Let $\vec{X} = \{I, M\}$. To compute $\Pi^{\vec{X}}_{P_4}$, we need to generate first the axioms for the body of the loop, which in this case is straightforward:

$$
\begin{aligned}
&I1 = I + 1,\\
&M1 = M
\end{aligned}
$$

Once the axioms for the body of the loop are computed, they are turned into inductive definitions by adding a new counter argument to all functions and predicates that may be changed by the program. For our simple example, we get

$$\forall n.I(n+1) = I(n) + 1, \tag{1}$$
$$\forall n.M(n+1) = M(n), \tag{2}$$

where the quantification is over all natural numbers. We then add the initial case, and introduce a new natural number constant N to denote the terminating index:

$$\begin{aligned} &I(0) = I \wedge M(0) = M, \\ &I1 = I(N) \wedge M1 = M(N), \\ &\neg(I(N) < M(N)), \\ &\forall n.n < N \rightarrow I(n) < M(n). \end{aligned}$$

One advantage of making counters explicit and quantifiable is that we can then either compute closed-form solutions to recurrences like (1) or reason about them using mathematical induction. This is unlike proof strategies like k-induction where the counters are hard-wired into the variables. Again, for more details about this approach, see [1] which has discussions about related work as well as proofs of the correctness under operational semantics.

B
Appendix: Details of Instantiation Strategy

Whenever an array element assignment occurs inside a loop, our system will generate an axiom like the following:

$$\begin{aligned} &\forall x_1, x_2 ... x_{k+1}, n.d\mathrm{k}array_i(x_1, x_2 ... x_{k+1}, n+1) = \\ &\quad ite(x_1 = A \wedge x_2 = E_2 \wedge ... \wedge x_{k+1} = E_{h+1}, E, \\ &\qquad d\mathrm{k}array_i(x_1, x_2 ... x_{k+1}, n)) \end{aligned} \tag{3}$$

where

- A is a k-dimensional array,
- $d\mathrm{k}array_i$ is a temporary function introduced by the translator,
- x_1 is an array name variable introduced by the translator universally quantified over arrays of k dimensions,
- x_2,....,x_{k+1} are natural number variables representing array indices universally quantified over natural numbers,
- n is the loop counter variable universally quantified over natural numbers,

- $E, E_2, ..., E_{k+1}$ are expressions.

For an axiom like (3), our system performs two types of instantiations:

- **Instantiating arrays**: this substitutes each occurrence of variable x_1 in the axiom (3) by the array constant A, and generates the following axiom:

$$\forall x_1, x_2 ... x_{k+1}.dk array_i(A, x_2 ... x_{k+1}, n+1) = ite(x_2 = E_2 \wedge ... \wedge x_{k+1} = E_{h+1}, E, dk array_i(A, x_2 ... x_{k+1}, n)) \quad (4)$$

- **Instantiating array indices**: This substitutes each occurrence of the variables x_i, $2 \leq i \leq k$ in the axiom (4) by E_i, and generates the following axiom:

$$\forall n.dk array_i(A, E_2 ... E_{k+1}, n+1) = E \quad (5)$$

References

[1] F. Lin, "A formalization of programs in first-order logic with a discrete linear order," *Artificial Intelligence*, vol. 235, pp. 1 – 25, 2016.