Proving your Algorithms

Louis-Noël Pouchet
pouchet@cse.ohio-state.edu

Dept. of Computer Science and Engineering, the Ohio State University

September 2010

888.11
Motivation

You need to prove your algorithms are correct:

- Power of the solution: Conjecture vs. Lemma!

- Building a proof may help find (and fix) mistakes
Categories of Proofs

Disclaimer: this is not exhaustive!

- **Correct / Complete / Terminate**
  - Simple in several situation, but may be very complex too...
  - Based on pre/post condition of the algorithms

- Logic programming / Lambda calculus equivalence
  - Rigourous
  - Can be software-assisted (Coq)
Simple Correctness Proof

Two main conditions:

- The algorithm is complete/correct: the post-condition is respected on all possible inputs satisfying the pre-condition
  - Precondition: a predicate $I$ on the input data
  - Postcondition: a predicate $O$ on the output data
  - Correctness: proving $I \Rightarrow O$

- The algorithm terminates
  - For all possible input, the algorithm exits
Proving the bubblesort algorithm

Algorithm

```
Algorithm bubblesort
Input:
Integer[]: A
Output:
Integer[]: sorted by increasing order

for i ← 1 to A.size - 1 do
    for j ← i + 1 to A.size do
        if A[i] > A[j] then
            tmp ← A[i]
            A[j] = tmp
        end if
    end for
end for
return A
```

Example: bubblesort:
Loop Invariants

One possible scheme: prove an invariant is true for all iterations

1. **Initialization**: the invariant(s) is true prior to the first iteration of the loop

2. **Maintenance**: if the invariant is true for iteration $n$, it is true for iteration $n + 1$

3. **Termination**: when the loop terminates, the invariant is true on the entire input

For bubblesort, the invariant is "At iteration $i$, the sub-array $A[1..i]$ is sorted and any element in $A[i+1..A.size]$ is greater or equal to any element in $A[1..i]$"
Initialization

For $i = 0$, the invariant is respected: the sub-array $A[1..0]$ is sorted, trivially (it contains no element).
Maintenance

Given the sub-array A[1..n − 1] sorted. Iteration \( n \) inserts at position \( n \) the smallest of the remaining unsorted elements of A[n..A.size], as computed by the \( j \) loop. A[1..n − 1] contains only elements smaller than A[n..A.size], and A[n] is smaller than any element in A[n + 1..A.size], then A[1..n] is sorted and the invariant is preserved.
Termination

Proving 101

- Proving the algorithm terminates (i.e., exits) is required at least for recursive algorithms.
- For simple loop-based algorithms, the termination is often trivial (show the loop bounds cannot increase infinitely).

- Finding invariants implies to carefully write the input/output of the algorithm.
- The proof can be tedious, "simpler" proofs are acceptable.
Another completeness / correctness / termination proof

Scheme:
- All cases are covered: completeness
  - Show all possible inputs are processed by the algorithm, may be trivial

- For a given (arbitrary) case, it is correctly processed: correctness
  - May need to cover individually all branches/cases of the algorithm
  - For each case, show the processing generates the expected output

- in all cases, the algorithm exits: termination
Another Example:

**Example**

**Algorithm**

**BuildSearchSpace**: Compute $\mathcal{T}$

*Input:*
$pdg$: polyhedral dependence graph

*Output:*
$\mathcal{T}$: the bounded space of candidate multidimensional schedules

\[
d \leftarrow 1
\]

**while** $pdg \neq \emptyset$ **do**

\[
d \leftarrow createUniversePolytope
\]

**for each** dependence $\mathcal{D}_{R,S} \in pdg$ **do**

\[
W_{\mathcal{D}_{R,S}} \leftarrow buildWeakLegalSchedules(\mathcal{D}_{R,S})
\]

\[
\mathcal{T}_d \leftarrow \mathcal{T}_d \cap W_{\mathcal{D}_{R,S}}
\]

**end for**

**for each** dependence $\mathcal{D}_{R,S} \in pdg$ **do**

\[
S_{\mathcal{D}_{R,S}} \leftarrow buildStrongLegalSchedules(\mathcal{D}_{R,S})
\]

\[
\text{if } \mathcal{T}_d \cap S_{\mathcal{D}_{R,S}} \neq \emptyset \text{ then}
\]

\[
\mathcal{T}_d \leftarrow \mathcal{T}_d \cap S_{\mathcal{D}_{R,S}}
\]

\[
pdg \leftarrow pdg - \mathcal{D}_{R,S}
\]

**end if**

**end for**

**end do**
Proof (kind-a)

▶ Correctness: For each level $d$, $T_d$ is the contains only schedules such that for all unsatisfied dependences, $\Theta_S - \Theta_R \geq 0$. Hence the semantics is preserved for all schedules. Since only satisfied dependence are removed from the set, the lexicopositivity of dependence satisfaction is respected.

▶ Completeness: trivial, no assumption is made on $pdg$ and a dependence can always be at least weakly satisfied if the input program accepts at least one schedule

▶ Termination: At least one dependence can be solved per time dimension, and the dependence graph of a program is finite.
Exercise

Given the algorithm for the following problem:
Input:
- The starting address of a matrix of integer $A$ of size $n \times n$
- The starting address of a matrix of integer $B$ of size $n \times n$
- A function $matrix(16x16) : getBlock(address : X, int : i, int : j)$ which returns a sub-matrix (a block) of the matrix starting at address $X$, of size $16 \times 16$ whose first element is at position $i,j$

Output:
- An integer $c$, the sum of the diagonal elements of the product of $A$ and $B$

Exercise: Prove it computes $tr(A.B)$