CS 3EA3 - Lecture Notes
Algorithm Derivation

Marina Mansour

Wednesday, February 15, 2017

Today, we present several problems and derive their algorithms, the objective being to illustrate the use of invariants in loop construction in the simplest possible context.

1 Summing the Elements of an Array

This problem is quickly introduced in Section 13.3.1 of the course text, but here we will provide the derivation in full.

Given $0 \leq N$, establish

$$R : total = (\oplus i \mid 0 \leq i < N \bullet A[i])$$

Note: if $\oplus$ means:

- **add** $(+)$ then **add** the elements of array A.
- **product** $(*)$ then **multiply** the elements of array A.
- **and** $(\wedge)$then **all** elements of array A are true.
- **or** $(\vee)$ then there **exists** at least one element of array A that is true.

Let's take the invariant:

$$P : total = (\oplus i \mid 0 \leq i < n \bullet A[i]) \wedge \ 0 \leq n \leq N$$

This invariant holds the notion of "so far". We do not examine all the elements of the array (up to N), but we examine a smaller segment of it (up to n).
We need to keep the property of the invariant (P) *true* from the beginning to the end. Thus, we need to make it *true* initially.

Initially truthify P by:

$$P[t, n := e, 0] \equiv true \qquad\qquad \text{where e is the unit of } \oplus$$

Note: This is a proof obligation.

So, our program so far:

$$
\begin{array}{ll}
\{ \quad 0 \leq N \quad \} & \text{Precondition} \\
total,\ n := e,\ 0 & \\
\{ \quad invariant\ P \quad \} & \\
& \\
& \\
\{ \quad R \quad \} & \text{Post-condition}
\end{array}
$$

To reach the post-condition R we would need to reach:

$$P \wedge n = N$$

Note: This is another proof obligation.

So our program so far:

$$
\begin{array}{ll}
\{ \quad 0 \leq N \quad \} & \text{Precondition} \\
total,\ n := e,\ 0 & \\
\{ \quad invariant\ P \quad \} & \\
& \\
& \\
\{ \quad P \wedge n = N \quad \} & \\
\{ \quad R \quad \} & \text{Post-condition}
\end{array}
$$

To get to $n = N$, we should first have a loop that when terminated, $n = N$ becomes *true*. This loop will be:

$$
\begin{array}{l}
do\ n \neq N \\
\quad \rightarrow ?
\end{array}
$$

So our program so far:

$$
\begin{array}{ll}
\{ \quad 0 \leq N \quad \} & \text{Precondition} \\
total,\ n := e,\ 0 & \\
\{ \quad invariant\ P \quad \} & \\
\mathbf{do}\ n \neq N & \\
\quad \rightarrow ? & \\
\mathbf{od} & \\
\{ \quad P \wedge n = N \quad \} & \\
\{ \quad R \quad \} & \text{Post-condition}
\end{array}
$$

Now, we need to find a bound function (bf) in order to make progress to terminate the loop. We know that

$$\begin{array}{ll} & P \wedge n \neq N \implies bf > 0 \\ = & \quad \{ \text{ where } 0 \leq n < N \wedge n \neq N \quad \} \\ & n < N \implies bf \\ = & \quad \{ \text{ arithmetic } \} \\ & N - n > 0 \implies bf \end{array}$$

Now, we have our bound function:

$$bf : N - n$$

Note: This is another proof obligation.
So our program so far:

$$\begin{array}{ll} \{ \quad 0 \leq N \quad \} & \text{Precondition} \\ \mathit{total},\ n := e,\ 0 & \\ \{ \quad \mathit{invariant}\ P\ ; bf : N - n \quad \} & \\ \mathbf{do}\ n \neq N & \\ \quad \rightarrow ? & \\ \mathbf{od} & \\ \{ \quad P \wedge n = N \quad \} & \\ \{ \quad R \quad \} & \text{Post-condition} \end{array}$$

Now, we need to to make progress towards termination by decreasing the bound function (bf). To decrease $N - n$ we will will increment n:

$$\{ \ bf = s \ \} \ n := n + 1 \ \{ \ bf < s \ \}$$

Note: This is another proof obligation.
This gave us information on what to do with n, but what happens to *total*? Let's call it the arbitrary term X for now.
So our program so far:

$$\begin{array}{ll} \{ \quad 0 \leq N \quad \} & \text{Precondition} \\ \mathit{total},\ n := e,\ 0 & \\ \{ \quad \mathit{invariant}\ P \quad ; \ bf : N - n \quad \} & \\ \mathbf{do}\ n \neq N & \\ \quad \rightarrow \ \mathit{total},\ n := X,\ n + 1 & \\ \mathbf{od} & \\ \{ \quad P \wedge n = N \quad \} & \\ \{ \quad R \quad \} & \text{Post-condition} \end{array}$$

Rather than guessing, let's solve for X.

$$
\begin{array}{ll}
 & P[t,\ n := X,\ n+1] \\
= & \quad \{ \text{ Definition of P and textual Substitution } \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \wedge\ 0 \leq n+1 \leq N \\
= & \quad \{ \text{ Order of } \mathbb{N} \ \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \wedge\ n+1 \leq N \\
= & \quad \{ \text{ Successors strictly above and } <\text{-arithmetic } \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \wedge\ n < N \\
= & \quad \{ \text{ Definition of strict order } \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \wedge\ n \leq N \wedge\ n \neq N \\
= & \quad \{ \text{ From invariant } P \ \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \wedge\ n \neq N \\
= & \quad \{ \text{ From loop guard } \ \} \\
 & X = (\oplus i \mid 0 \leq i < n+1 \bullet A[i]) \\
= & \quad \{ \text{ Split off term theorem and using invariant } P \ \ \} \\
 & X = \mathit{total} \oplus A[n]
\end{array}
$$

Finally, our program:

$$
\begin{array}{ll}
\{ \quad 0 \leq N \quad \} & \text{Precondition} \\
\mathit{total},\ n := e,\ 0 & \\
\{ \quad \mathit{invariant}\ P \ \ ;\ \mathit{bf} : N - n \quad \} & \\
\mathbf{do}\ n \neq N & \\
\qquad \rightarrow\ \mathit{total},\ n := \mathit{total} \oplus A[n],\ n+1 & \\
\mathbf{od} & \\
\{ \quad P \wedge n = N \quad \} & \\
\{ \quad R \quad \} & \text{Post-condition}
\end{array}
$$

And the C code will be:

```
total = e;          //e is the unit of oplus
for(int n = 0 ; n < N ; n ++)
    total = oplus( total , f(n) );
```

To derive algorithms like this one, recall the lecture on January 9th, 2017 on Games. There we said we need to do the following 7 steps:

1. Formalise '*Givens*' and '*Requireds*' of the problem.
2. Obtain an invariant P and initialise the variables to make it true.
3. Bridge from invariant to post-condition: solve for B in
$$P \wedge \neg B \implies R$$
4. If $\neg B$ holds then were done, otherwise we construct a loop to obtain it.
5. Solve for a "bound function" bf in $P \wedge B \implies bf > 0$
6. Make progress towards termination: solve for S in
$$\{\ bf = c\ \}\ S\ \{\ bf < c\ \}$$
where c can be thought of as any "candidate number of iterations remaining".
7. Ensure that such a program S maintains our invariant!

And the general schema looks like:

$$\begin{array}{ll}
\{ \quad G \quad \} & \text{Precondition} \\
\text{initialisation} & \\
\{ \quad \text{invariant } P \ \ ; \text{bound } bf \quad \} & \\
\textbf{do } B \ \rightarrow \{\ bf = c\ \}\ S\ \{\ bf < c\ \} \textbf{ od} & \\
\{ \quad P \wedge \neg B \implies R \quad \} & \\
\{ \quad R \quad \} & \text{Post-condition}
\end{array}$$

2 Summing the Elements of an Array 2.0

Given $0 \leq len$, establish s the sum of array $r[0 \ldots len - 1]$. So our program will look like:

$$
\begin{array}{l}
s,\ n := 0,\ 0 \\
\textbf{do}\ n \neq len \\
\quad \rightarrow\ s,\ n := s + r[n],\ n + 1 \\
\textbf{od}
\end{array}
$$

3 All True

Given $x \geq 1$, establish 'a' as true *iff* all elements of the array $f[0 \ldots x]$ are true.
The C code will be:

```
a = true;        //true is the unit of &&
for(int i = 0 ; i < x ; i ++)
    a = f[i] && a;
```

4 Factorial

Given $n \geq 0$, establish $fact = n! = (* \, i \mid 0 \leq i < n \bullet i + 1)$
The C code will be:

```
fact = 1;        //1 is the unit of *
for(int i = 0 ; i < b ; i ++)
    fact = fact * (i + 1);
```

5 Tricky but Works!

Given $n \geq 0$, establish $1/n!$, assuming you cannot divide at the end when you have $n!$. The C code will be:

```
fact = 1;        //1 is the unit of /
for(int i = 0 ; i < n ; i ++)
    fact = fact / (i + 1);
```

The reason why this is tricky is because in the general schema, oplus ($\oplus$) has to be associative and division ($/$) is **not** associative.
Thus this program cannot be justified or proven using the general schema defined earlier.