CSECS 2011, pp. 000 - 000

The 7 Annual International Conference on
Computer Science and Education in Computer Science,

July 06-10 2011, Sofia, Bulgaria

APPLICATIONS OF CUMULATIVE SUBGOAL FULFILLMENT TO LINEAR PROGRAMMING

ERIC BRAUDE

Abstract: We show how the CSF program design approach can be used to synthesize the classical linear programming algorithm as well as Seidel's external algorithm.

Keywords: programming, simplex, correctness, synthesis, coding

ACM Classification Keywords: D.2.3 Coding Tools and Techniques

1. Introduction

Cumulative subgoal fulfillment (CSF), introduced in [Braude, 2007], is an approach to creating procedures that are also, in the classical sense, provably correct. In this paper, we apply CSF to the linear programming problem, showing how it produces both Dantzig's classical algorithm [Dantzig, 1998], as well as Seidel's Seidel, 1990]. The latter was actually generated without a priori knowledge of Seidel's work. The results are obtained by selecting different cumulative subgoals. Some general strategies and techniques for selecting subgoals were described in [Braude, 2007]; others are to appear elsewhere. A common one is to replace a constant with a variable; i.e., if a postcondition involves accomplishing an objective for N things, a candidate subgoal is to accomplish the objective for i of them.

2. Cumulative Subgoal Fulfillment

CSF is based on principles of physical construction. In such construction, each stage can be thought of as fulfilling a subgoal: one which remains valid while we construct additional parts. Informally, CSF consists of a sequence of code blocks, each of which fulfills a subgoal and (this is the key part) leaves invariant all subgoals already fulfilled.

Formally, consider a procedure P for which *pre*, *inv*, and *post* are the conjunctions of the preconditions, invariants, and postconditions respectively. We define an *algorithm plan* for P as a sequence $s_1, s_2, ..., s_n$ of predicates for which $pre \wedge s_1 \wedge s_2 \wedge ... \wedge s_n \wedge inv \Rightarrow post \wedge inv$. A CSF implementation of P consists of an algorithm plan $s_1, s_2, ..., s_n$ and a sequence $c_1, c_2, ..., c_n$ of code blocks satisfying the following Hoare triples.

(1) $inv \wedge pre \{c_1\}\ inv \wedge s_1$, and

(2) $inv \wedge s_1 \wedge s_2 \wedge ... \wedge s_{i-1} \{ c_i \}\ inv \wedge s_1 \wedge s_2 \wedge ... \wedge s_i$ for $i = 2, 3, ..., n$.

(A Hoare triple $A\{c\}R$ denotes "If A is true, and code c is executed, then R is true.")

A predicate p for which $p \wedge post$ = true will be called a *cumulative subgoal* for P. Thus, algorithm plans consist of cumulative subgoals. In most cases, the code blocks $c_1, c_2, c_3, ...$ can be constructed via a *perturb/restore* process as follows.

// Fulfill s_1: (i.e., the following is c_1)

<use *pre* to fulfill s_1 by perturbing variable(s) >

<restore *inv*>

// Fulfill s_2:

<use *inv* and s_1 to fulfill s_2>

<restore *inv*>

<restore s_1>

// Fulfill s_3:

<use *inv*, s_1, and s_2 to fulfill s_3>

<restore *inv*>

<restore s_1>

<restore s_2>

...

A special case of this is where we fulfill a subgoal s_i by using a loop as follows, in which "perturbing variable(s)" is often the incrementing of an index.

// Fulfill s_i ...

while(!s_i){ // prove termination ...

< perturb variable(s) using *inv*, s_1, s_2, ... , and s_{i-1} to fulfill s_i more closely>

<restore *inv*>

<restore s_1>

<restore s_2>

...

<restore s_{i-1}>}

We use the convention that *returnX* is the variable whose value is returned by the procedure. We use the '*x* and *x*' notation to denote the respective value of *x* before and after the relevant set of operations.

3. Linear Programming

For the purpose of demonstrating the application of CSF to linear programming, it is sufficient to describe the process for two dimensions. The linear programming problem can then be expressed as follows, which includes explicit limits on the variables.

double linProgMax(*Function anObjectiveFn*, *Equation*[] *aConstraint*)

// Pre1: *anObjectiveFn* is linear

// Definition: *M* == *Double*.*MAX_VALUE*

// Pre2: The elements of *aConstraint* are linear

// AND *aConstraint*[0,3] = {(*x*<=*M*), (*x*>=-*M*), (*y*<=*M*), (*y*>=-*M*)}

The Title of the Section

// Pre3: The members of *aConstraint* are distinct

// Post: *returnV* is the maximum of *anObjectiveFn*() subject to every member of *aConstraint*.

Most, if not all, approaches depend on the observation that maxima occur at the intersection of constraints.

4. The Classical Algorithm via CSF

We describe Danzig's classical linear programming solution here [Dantzig, 1998] in terms of CSF. For now, we take for granted the second half of precondition 2 above. We will also assume, without compromising the point of this paper, that the constraints form an inner polygon (illustrated in Figure 1), one of whose vertices is the locus of the maximum.

Figure 1: Linear Programming Domain in 2 Dimensions

The cumulative property of CSF subgoals allows the reader to verify that if all subgoals were true, the postconditions will have been fulfilled.

The first subgoal selected here exploits the fact that the maximum occurs at an intersection of two (or more) constraints. This can be done by labeling such a point, which is a candidate for the return value.

> SG1 (Max occurs at a vertex): returnV is vertex of the interior polygon
>
> AND returnV = aConstraint[s] $\cap$ aConstraint[t]

We next encapsulate Danzig's idea of traveling around the polygon.

> For $x \neq y$, define $v(x,y)$ as *aConstraint*[x] $\cap$ *aConstraint*[y].
>
> SG2 (Improvement direction identified):
>
> anObjectiveFn(returnV) $\geq$ anObjectiveFn(v(u,s))
>
> AND v(u,s) is on the interior polygon

The final subgoal describes a successful conclusion to this (finite) process.

> SG3 (No further improvement possible) :
>
> *anObjectiveFn(returnV)* $\geq$ *anObjectiveFn(v(t,w))*
>
> *AND v(t,w) is on the interior polygon*

SG1 can be fulfilled by selecting any vertex of the inner polygon. SG2 can be fulfilled by comparing the value at this vertex with those at its neighbors on the inner polygon. SG3 can be fulfilled by repeatedly moving in the direction of the unvisited neighbor with higher or equal value. Each of these fulfillments can be readily accomplished in a way that preserves its predecessors.

5. Seidel's Algorithm via CSF

CSF is used in this case in a more straightforward manner than in the classical case. We use once again the observation that a maximum occurs at an intersection but there is no need to identify a inner polygon. We invoke all of precondition 3. The author applied this alternative CSF reasoning process independently and afterwards learned of its essential equivalence with Seidel's algorithm [Seidel, 1990]. He describes the algorithm for arbitrary dimensions, and shows its improved efficiency.

By fulfilling the constraints one at a time as (cumulative!) subgoals, the following algorithm plan results.

> SG1: Subject to *the constraints in s, anObjectiveFn*() attains its maximum at *returnV*
>
> SG_0: *s* contains *aConstraint*[0]
>
> …
>
> SG_i: *s* contains *aConstraint*[j]
>
> ...
>
> SG_{last}: *i = aConstraint.length*()-1

SG1 – SG_i can be fulfilled together by selecting *i*=3 and returning a corner point or the intersection with the $2M\times 2M$ square, as shown in Figure 2.

Figure 2: Fulfilling SG1-SG_4

SG$_{last}$ can be effected by a loop which increments *i* and restores each prior subgoal. The key observation in the restorations is that if ‘*returnV* fails to satisfy *aConstraint*[*i*+1], then *returnV*’ lies at the intersection of *aConstraint*[*i*+1] and one of *aConstraint*[0], *aConstraint*[1], ..., *aConstraint*[*i*]. This is illustrated by Figure 3.

Figure 3: Introducing Next Constraint (i+1)

This amounts to comparing values of *anObjectiveFn*() at various points on *aConstraint*[*i*+1], which is effectively liner programming in dimension 1. In general, solving the problem in dimension *d* can be performed by invoking liner programming recursively for dimension *d*-1.

6. Conclusion

The CSF approach was used to generate both the classical and the Seidel approach to the linear programming problem. These can be thought of as a perimeter and an external approach respectively. For future work it would be interesting to see if CSF can yield an internal approach such as [Karmarkar, 1984], and whether it can be used improve linear programming implementations.

Bibliography

The Title of the Section

Braude, Eric J. *Cumulative Subgoal Fulfillment in Software Development*, Proceedings of the 11th IASTED International Conference on Software Engineering and Applications, 480-485 (2007)

Dantzig, George *Linear Programming and Extensions*, Princeton University Press, (1998).

Karmarkar, Narendra *A New Polynomial Time Algorithm for Linear Programming*, Combinatorica, v.4, no.4 (1984), pp 373–395.

Seidel, Raimund, *Linear Programming and Convex Hulls Made Easy*, SCG '90 Proceedings of the Sixth Annual Symposium on Computational Geometry, p211-215, ACM Press (1990).

Authors' Information

ERIC BRAUDE, Ph.D., Associate Professor of Computer Science, Boston University Metropolitan College, 808 Commonwealth Ave 205, Boston, MA 02215 USA; ebraude@bu.edu

Major Fields of Scientific Research: Software Engineering, Artificial Intelligence