Inference of Linear Upper-Bounds on the Expected Cost by Solving Cost Relations

Alicia Merayo\textsuperscript{1} and Samir Genaim\textsuperscript{2}

\textsuperscript{1} Universidad Complutense de Madrid, Spain
amerayo@ucm.es
\textsuperscript{2} Universidad Complutense de Madrid, Spain
sgenaim@ucm.es

1 Introduction

Resource usage analysis (a.k.a. cost analysis) aims at statically determining the number of resources required to safely execute a given program. A resource can be any quantitative aspect of the program, such as memory consumption, execution steps, etc. Over the past decade several cost analysis frameworks, for different programming languages, have been developed [6, 8, 12, 14, 3, 9]. They can infer precise upper-bounds on the worst-case cost.

There are problems that involve probabilistic choices and for which worst-case cost is not the adequate. For example, consider a program that transmits packets of data over the network. Due to network failures, the transmission of a packet might fail and it has to be transmitted again. Assuming that the probability of success (resp. failure) is $\frac{3}{4}$ (resp. $\frac{1}{4}$), our goal is to estimate the number of attempts required in order to successfully transmit $n$ packets. This problem can be modeled using the following loop:

\begin{verbatim}
while ( n>0 ) {
    n=n-1; \oplus \frac{3}{4} \text{skip;}
    \text{tick(1);}
}
\end{verbatim}

where $\oplus$ is a probabilistic choice operator and $\text{tick(1)}$ is a cost annotation. Note that the left-hand (resp. right-hand) side of $\oplus$ represents a successful (resp. failed) transmission.

If we consider the probabilistic choice as a non-deterministic choice, and analyze this program using a worst-case cost analyzer, we would not obtain an upper-bound since the loop is considered non-terminating. The adequate notion of cost for this setting, i.e., in the presence of such probabilistic operations, is the expected cost. Let $S$ be an infinite sequence of independent choices for the probabilistic operation (i.e., left or right), $\text{cost}(n, S)$ be the cost of the execution when taking the choices as indicated in the sequence $S$, and $\text{Pr}(S)$ is the probability of taking such choices (i.e., multiplying the probabilities of all choices), then the expected cost is defined as $\sum_S \text{Pr}(S) \ast \text{cost}(n, S)$. Note that the sum is over all possible sequences $S$, and that $\sum_S \text{Pr}(S) = 1$. In this case the expected cost can be reduced to solving the following recurrence relation for $n > 0$ (for $n \leq 0$ we let $C(n) = 0$):

$$C(n) = 1 + \frac{3}{4}C(n-1) + \frac{1}{4}C(n)$$

The function $\max(0, \frac{3}{4}n)$ is a possible solution. Note that if we turn $= \to \geq$ in the above equation, we get a definition for an upper-bound on the expected cost.

Cost analysis of probabilistic programs and term-rewriting has been recently considered in several works [13, 7, 5, 11]. For example, the work of [13], which is based on an extension of amortized cost analysis, is developed for a simple imperative language and able to infer polynomial bounds on the expected cost of challenging problems.
2 Inference of Linear Upper-Bounds on the Expected Cost by Solving Cost Relations

while ( x<\ n ) {
    x=x+1;

    if (x
    x=x-1;

    tick(1);
}
into a closed-form upper bound. Before solving the cost relations we typically simplify them using unfolding as done in [2]. This simplification step usually has an impact on both the performance and the precision of the solving process.

**Generating cost relations.** Each regular node \( n \in N_r \) with outgoing transitions \( n \xrightarrow{\varphi_i} n_i \), \( 1 \leq i \leq k \), contributes \( k \) cost relations of the form

\[
C_n(\bar{x}) = \text{cost}(n) + C_{n_i}(\bar{x}^{'}) \mid \varphi_i
\]

and each probabilistic node \( n \in N_p \) with outgoing edges \( n \xrightarrow{p_i, \varphi_i} n_i \), for \( 1 \leq i \leq k \), contributes one cost relation

\[
C_n(\bar{x}) = \text{cost}(n) + \sum_{i=1}^{k} p_i \cdot C_{n_i}(\bar{x}^{'}) \mid \bigcup_{i=1}^{k} \varphi_i
\]

where \( \bigcup_{i} \varphi_i \) is the union of all \( \varphi_i \), after renaming the primed variables \( \bar{x}^{'}, i \) of each \( \varphi_i \) to fresh variables \( \bar{x}_i \). Note that we view a conjunction of linear constraints as a set, this is why we use a union. For the CFG of Figure 1 we generate the following cost relations:

\[
\begin{align*}
C_A(x, n) &= C_F(x', n') & \{ x \geq n, x' = x, n' = n \} \\
C_B(x, n) &= C_B(x', n') & \{ x < n, x' = x, n' = n \} \\
C_B(x, n) &= C_C(x', n') & \{ x = x + 1, x'_2 = x - 1, n'_1 = n, n'_2 = n \} \\
C_D(x, n) &= C_D(x', n') & \{ x' = x, n' = n \} \\
C_E(x, n) &= 1 + C_A(x', n') & \{ x' = x, n' = n \} \\
C_F(x, n) &= 0
\end{align*}
\]

The meaning of a cost relation is as expected, it recursively defines the cost of node \( n \) in terms of its successors in the context of some constraints (those of the corresponding transitions). Note that, due to non-determinism, a query \( C(\bar{v}) \) might evaluate to several possible values, and thus, an upper-bound is defined to be an upper-bound on the set of all such values. The correctness follows from the weakest-precondition reasoning of [11].

**Solving cost relations.** Solving cost relations into an upper-bound, on expected cost of the CFG, means seeking functions \( f_n : \mathbb{Z}^m \rightarrow \mathbb{R}^+ \), for each node \( n \), such that for each cost relation \( (C_i(\bar{x}) = c + \sum_{j} p_j C_j(\bar{x}_j) \mid \varphi) \) the following formula is satisfied (here \( \bar{z} \) is the set of all variables that appear in the cost relation):

\[
\forall \bar{z}. \varphi \models f_i(\bar{x}) \geq c + \sum_{j} p_j \cdot f_j(\bar{x}_j),
\]

This formula states that \( f_i \) is enough for paying the local cost \( c \) of \( C_i \) plus the cost of its successors (taking into account the probabilities \( p_j \)). The set of all resulting formulas is then automatically solved as described in [4], with some modifications to take the multiplicands that correspond to probabilities into account. This procedure is based on the use of Farkas’ lemma and SMT solving. Briefly, we first define a template function \( f_n(\bar{x}) = \max(0, a_0 + \sum a_i x_i) \) for each node \( n \) (where \( a_i \) are the template parameters) and substitute them in the formulas, then Farkas’ lemma is used to transform each formula as the one above into a set of (possibly non-linear) constraints over the template parameters \( a_i \), and finally by solving these constraints using an SMT solver we obtain concrete values for the template parameters, i.e., we instantiate each template \( f_n \). Note that the non-linearity in the generated constraints
Inference of Linear Upper-Bounds on the Expected Cost by Solving Cost Relations

4

while (x+3 <= n) {
    if ( y<n ) {
        skip; @x y=y+1;
    } else {
        skip; @x x=x+1; @x x=x+2; @x x=x+3;
        tick(1);
    }
}

\[ \varphi_1 = \{ x+3 \leq n \} \cup \varphi_5 \]
\[ \varphi_2 = \{ x+3 > n \} \cup \varphi_5 \]
\[ \varphi_3 = \{ y < m \} \cup \varphi_5 \]
\[ \text{cost}(F) = \text{cost}(M) = 1 \]
\[ \varphi_4 = \{ y' \geq m \} \cup \varphi_6 \]
\[ \varphi_6 = \{ x' = x, n' = n, y' = y, m' = m \} \]
\[ \varphi_7 = \{ x' = x + 1, n' = n, y' = y, m' = m \} \]
\[ \varphi_8 = \{ x' = x + 2, n' = n, y' = y, m' = m \} \]
\[ \varphi_9 = \{ x' = x + 3, n' = n, y' = y, m' = m \} \]

Figure 2 Example 2

comes from the fact that our templates are not linear, they include expressions like \( \max(0, f) \).
In order to apply Farkas’ lemma we need to eliminate the max first, which we do by splitting the corresponding formula into two cases for \( f \leq 0 \) and \( f \geq 0 \). This introduce \( f \leq 0 \) and \( f \geq 0 \) to the left-hand of the formula, and since \( f \) has template variables we get non-linear constraints when applying Farkas’ lemma.

In order to infer an upper-bound for the cost relations that we generated above, we first simplify them using unfolding into the following:

\[
C_A(x, n) = 0
\]
\[
C_A(x, n) = 1 + \frac{3}{2} \cdot C_A(x, n') \quad \{ x < n, x' = x+1, x'' = x-1, n' = n \}
\]

and then solving as described above, using the template \( f_A(x, n) = \max(0, a_0 + a_1 n + a_2 x) \), results in the upper-bound \( f_A(x, n) = \max(0, 3n - 3x) \).

Let us now consider the program depicted in Figure 2, taken from [13], and its corresponding CFG that is depicted in the same figure. Generating the cost relations and simplifying them using unfolding we obtain the following cost relations:

\[
C_A(x, y, m, n) = 0 \quad | \{ x + 3 > n \}
\]
\[
C_A(x, y, m, n) = 1 + \sum_{i=0}^{1} \frac{1}{2} \cdot C_F(x, y, n, m) \quad | \{ x + 3 \leq n, y \geq m, y'_0 = y, y'_1 = y + 1 \}
\]
\[
C_A(x, y, m, n) = 1 + \sum_{i=0}^{1} \frac{1}{2} \cdot C_F(x', y, n, m) \quad | \{ x + 3 \leq n, y \geq m, x'_0 = x, x'_1 = x + 1, x'_2 = x + 2, x'_3 = x + 3 \}
\]

The first one corresponds to the loop exit, and the second (resp. third) one to executing the then (resp. else) branch of the if-condition. Now in order to solve these cost relations we start by choosing the template \( f_A(x, y, m, n) = \max(0, a_0 + a_1 x + a_2 y + a_3 n + a_4 m) \), however, the solver fails to find a solution using this template. In such case we refine our template to include one more component, namely we use \( f_A(x, y, m, n) = \max(0, a_0 + a_1 x + a_2 y + a_3 n + a_4 m + a_5 m) + \max(b_0 + b_1 x + b_2 y + b_3 n + b_4 m) \). Now the solver succeeds to find the upper bound \( f_A(x, y, m, n) = \max(0, 2m - 2y) + \max(0, \frac{7}{2} n - \frac{3}{2} x) \). Note that, this example required a more complicated template since the loop executes in two independent phases. In the first one we increase \( y \) until \( y \geq m \) holds, and in the second we increase \( x \) until \( x + 3 > n \) holds.
3 Concluding remarks

We described a preliminary work on inferring upper-bounds on the expected cost for CFGs via cost relations, with the goal of integrating this process in the SACO tool. It can also be used to prove almost sure termination. Our approach generates a set of cost relations from a given CFG and solves them using a technique similar to the one described in [4]. It is currently limited to linear upper-bounds and can handle small examples, e.g., most of the examples with a linear cost that are described in [13].

Out tools is still not as powerful as [13]. We plan to carry on this research direction, mainly to improve the implementation to handle larger programs and handle cases where complex invariants are needed which we do not support yet. We plan to extend the ABS language [10] with probabilistic constructs and integrate our implementation in the SACO.

References