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

Alicia Merayo[1] **and Samir Genaim**[2]

1 **Universidad Complutense de Madrid, Spain**
amerayo@ucm.es
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:

```
while ( n>0 ) {
  n=n-1; ⊕3/4 skip;
  tick(1);
}
```

where $\oplus$ is a probabilistic choice operator and `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), $\texttt{cost}(n, S)$ be the cost of the execution when taking the choices as indicated in the sequence S, and $\texttt{Pr}(S)$ is the probability of taking such choices (i.e., multiplying the probabilities of all choices), then the expected cost is defined as $\sum_S \texttt{Pr}(S) * \texttt{cost}(n, S)$. Note that the sum is over all possible sequences S, and that $\sum_S \texttt{Pr}(S) = 1$. In this case the expected cost can be reduced to solving the following recurrence relation for $n > 0$ (for $n \leq 0$ we let $C(n) = 0$):

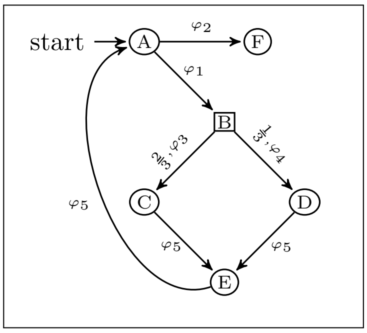
$$C(n) = 1 + \frac{3}{4}C(n-1) + \frac{1}{4}C(n)$$

The function $\max(0, \frac{4}{3}n)$ 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.

licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
while ( x<n ) {
  x=x+1;⊕2/3 x=x-1;
  tick(1);
}
```



$$
\begin{aligned}
\varphi_1 &= \{x \geq n, x' = x, n' = n\} \\
\varphi_2 &= \{x < n, x' = x, n' = n\} \\
\varphi_3 &= \{x' = x + 1, n' = n\} \\
\varphi_4 &= \{x' = x - 1, n' = n\} \\
\varphi_5 &= \{x' = x, n' = n\} \\
&\quad \texttt{cost}(E) = 1
\end{aligned}
$$

Figure 1 Example 1

The goal of our work is to support the inference of expected cost in our cost analyzer SACO [1], which infers upper-bounds on the worst-case cost of ABS programs [10] — an abstract behaviour modeling language based on concurrent objects. The underlying techniques of SACO for inferring cost are based on first generating a set of cost relations (a general form of recurrence relations) that describes the cost of the program and then solving them into closed-form upper-bounds using a dedicated solver [2]. Thus, we are interested in developing a tool for inferring the expected cost for such cost relations, since it would make the integration with SACO straightforward. In this extended abstract, we report on preliminary results in this direction, in particular on a preliminary implementation that can infer linear upper-bounds on the expected cost of simple control-flow graphs by solving corresponding cost relations.

2 An informal account to our approach

A program in our setting consists of a control-flow graph (CFG) and a tuple of integer variables $\bar{x} = \langle x_1, \cdots, x_m \rangle$. A CFG consists of a set of nodes N representing locations, and a set of edges E representing transitions. Each node n is annotated with a non-negative cost, denoted by $\texttt{cost}(n)$, that corresponds to the resources consumed by each visit to n. Note that we could also annotate transitions with cost, this requires no change in our approach. Each edge $n_1 \xrightarrow{\varphi} n_2$ is annotated with a conjunction of linear constraints φ over variable $\bar{x}$ and primed variables $\bar{x}'$, it defines the transition relation $\{(\bar{v}, \bar{v}') \in \mathbb{Z}^{2m} \mid \bar{x} = \bar{v} \wedge \bar{x}' = \bar{v}' \models \varphi\}$ between nodes n_1 and n_2. For every node we assume that its outgoing transitions, if any, cover the whole state space, i.e., it is always possible to advance. The set of nodes N consists of regular nodes N_r (we draw them as circles) and probabilistic nodes N_p (we draw them as squares). A probabilistic node represents a probabilistic choice, and thus its *outgoing edges are annotated with probabilities that sum to* 1 (in addition to the transition constraints). Moreover, we assume that such edges include only deterministic updates, i.e., the corresponding constraints include only equalities of the form $x' = a_0 + \Sigma_i a_i x_i$ (we can refine this to any non-deterministic update as far as it is always applicable, i.e., covers the whole state space). The idea is that when the execution is at a probabilistic node, the choice of the next node depends on the probabilities associated to the corresponding transitions, and the next state is calculated using the corresponding deterministic update. On the other hand, when the execution is at a regular node the next node is chosen in a non-deterministic way from all applicable transitions. A simple program (taken from [13]) and a corresponding CFG are depicted in Figure 1.

The inference of a linear upper-bound on the expected cost of a given CFG is done in two steps: (1) generation of cost relations from the CFG; and (2) solving the cost relations 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

$$C_n(\bar{x}) = \texttt{cost}(n) + C_{n_i}(\bar{x}') \quad | \quad \varphi_i$$

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

$$C_n(\bar{x}) = \texttt{cost}(n) + \sum_{i=1}^{k} p_i \cdot C_{n_i}(\bar{x}'_i) \quad | \quad \biguplus_{i=1}^{k} \varphi_i$$

where $\uplus\varphi_i$ is the union of all φ_i, after renaming the primed variables $\bar{x}'$ of each φ_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:

$$\begin{array}{lll}
C_A(x,n) = & C_F(x',n') & \{x \geq n, x' = x, n' = n\} \\
C_A(x,n) = & C_B(x',n') & \{x < n, x' = x, n' = n\} \\
C_B(x,n) = & \frac{2}{3} \cdot C_C(x'_1,n'_1) + \frac{1}{3} \cdot C_D(x'_2,n'_2) & \{x'_1 = x+1, x'_2 = x-1, n'_1 = n, n'_2 = n\} \\
C_C(x,n) = & C_E(x',n') & \{x' = x, n' = n\} \\
C_D(x,n) = & C_E(x',n') & \{x' = x, n' = n\} \\
C_E(x,n) = & 1 + C_A(x',n') & \{x' = x, n' = n\} \\
C_F(x,n) = & 0 &
\end{array}$$

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 $f_n : \mathbb{Z}^m \mapsto \mathbb{R}^+$, for each node n, such that for each cost relation $\langle C_i(\bar{x}) = c + \sum_j p_j C_j(\bar{x}_j) \mid \varphi \rangle$ 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

```
while (x+3 <= n) {
  if ( y<m ) {
    skip; ⊕1/2 y=y+1;
  else
    skip; ⊕1/4 x=x+1; ⊕1/4 x= x+2;⊕1/4 x=x+3;
  tick(1);
}
```

$$
\begin{aligned}
\varphi_1 &= \{x+3 \leq n\} \cup \varphi_5 \\
\varphi_2 &= \{x+3 > n\} \cup \varphi_5 \\
\varphi_3 &= \{y < m\} \cup \varphi_5 \qquad \texttt{cost}(F) = \texttt{cost}(M) = 1 \\
\varphi_4 &= \{y' \geq m\} \cup \varphi_5 \\
\varphi_5 &= \{x' = x, n' = n, y' = y, m' = m\} \\
\varphi_6 &= \{x' = x, n' = n, y' = y+1, m' = m\} \\
\varphi_7 &= \{x' = x+1, n' = n, y' = y, m' = m\} \\
\varphi_8 &= \{x' = x+2, n' = n, y' = y, m' = m\} \\
\varphi_9 &= \{x' = x+3, n' = n, y' = y, m' = m\}
\end{aligned}
$$

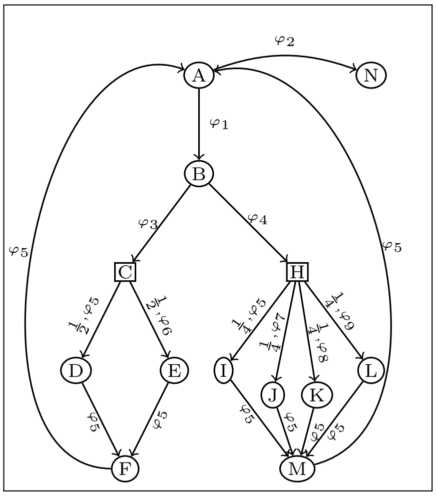


Figure 2 Example 2

comes from the fact that our templates are not linear, they include expressions like $\max(0, f)$. In order to apply Farka's 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:

$$
\begin{array}{lll}
C_A(x,n) = & 0 & \{x \geq n\} \\
C_A(x,n) = & 1 + \frac{2}{3} \cdot C_A(x_1', n') + \frac{1}{3} \cdot C_A(x_2', n') & \{x < n, x_1' = x+1, x_2' = x-1, n' = n\}
\end{array}
$$

and then solving them 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:

$$
\begin{array}{lll}
C_A(x,y,m,n) = & 0 & \mid \{x+3 > n\} \\
C_A(x,y,m,n) = & 1 + \sum_{i=0}^{1} \frac{1}{2} \cdot C_F(x, y_i', n, m) & \mid \{x+3 \leq n, y \geq m, y_0' = y, y_1' = y+1\} \\
C_A(x,y,m,n) = & 1 + \sum_{i=0}^{3} \frac{1}{4} \cdot C_F(x_i', y, n, m) & \mid \{x+3 \leq n, y \geq m, x_0' = x, \\
& & \quad x_1' = x+1, x_2' = x+2, x_3' = x+3\}
\end{array}
$$

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 $f_A(x,y,m,n) = \max(0, a_0 + a_1 x + a_2 y + a_3 n + a_4 m) + \max(b_0 + b_1 x + b_2 y + b_3 n + b_4 m)$. Now the solver succeeds to find the upper bound $f_A(x,y,m,n) = \max(0, 2m - 2y) + \max(0, \frac{2}{3}n - \frac{2}{3}x)$. 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

1. E. Albert, P. Arenas, A. Flores-Montoya, S. Genaim, M. Gómez-Zamalloa, E. Martin-Martin, G. Puebla, and G. Román-Díez. SACO: Static Analyzer for Concurrent Objects. In *Proc. of TACAS'14*, volume 8413 of *LNCS*, pages 562–567. Springer, 2014.
2. E. Albert, P. Arenas, S. Genaim, and G. Puebla. Closed-Form Upper Bounds in Static Cost Analysis. *Journal of Automated Reasoning*, 46(2):161–203, 2011.
3. E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Cost Analysis of Object-Oriented Bytecode Programs. *Theoretical Computer Science (Special Issue on Quantitative Aspects of Programming Languages)*, 413(1):142–159, 2012.
4. D. E. Alonso-Blas and S. Genaim. On the Limits of the Classical Approach to Cost Analysis. In *Proc. of SAS 2012*, volume 7460 of *LNCS*, pages 405–421. Springer, 2012.
5. M. Avanzini, U. Dal Lago, and A. Yamada. On probabilistic term rewriting. In *In Proc. of FLOP'18*, volume 10818 of *LNCS*, pages 132–148. Springer, 2018.
6. M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Analyzing runtime and size complexity of integer programs. *ACM Trans. Program. Lang. Syst.*, 38(4):13:1–13:50, 2016.
7. K. Chatterjee, H. Fu, and A. Murhekar. Automated recurrence analysis for almost-linear expected-runtime bounds. In *In Proc. of CAV'17*, volume 10426 of *LNCS*, pages 118–139. Springer, 2017.
8. S. Gulwani, K. K. Mehra, and T. M. Chilimbi. Speed: Precise and Efficient Static Estimation of Program Computational Complexity. In *Proc. of POPL'09*, pages 127–139. ACM, 2009.
9. J. Hoffmann, K. Aehlig, and M. Hofmannn. Multivariate Amortized Resource Analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14:1–14:62, 2012.
10. E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A Core Language for Abstract Behavioral Specification. In *Proc. of FMCO'10 (Revised Papers)*, volume 6957 of *LNCS*, pages 142–164. Springer, 2012.
11. B. L. Kaminski, J.-P. Katoen, C. Matheja, and F. Olmedo. Weakest precondition reasoning for expected run-times of probabilistic programs. In *In Proc. of ESOP'16*, volume 9632 of *LNCS*, pages 364–389. Springer, 2016.
12. Z. Kincaid, J. Breck, A.F. Boroujeni, and T. W. Reps. Compositional recurrence analysis revisited. In *In Proc. of PLDI'17*, pages 248–262. ACM, 2017.
13. V. C. Ngo, Q. Carbonneaux, and J. Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. In *In Proc. of PLDI'18*, 2018. To appear.
14. M. Sinn, F. Zuleger, and H. Veith. Complexity and resource bound analysis of imperative programs using difference constraints. *J. Autom. Reasoning*, 59(1):3–45, 2017.