1 VC GENERATION FOR PROGRAMS WITH FUNCTION CALLS

Thus far, we have focused on formulating the VC for basic blocks of heap manipulating programs and have shown a decidable logic of lists and list-measures that solves the delta-logic VC by building a decision procedure for the corresponding contextual logic. However, most useful programs that manipulate lists also contain function calls, and a lot of such programs are also best expressed as recursive functions. As we have stated earlier, we recommend the continued use of frame reasoning to perform an inference of the post-state across a function call. However, it is not obvious how to generate VCs for such a situation because our hypothesis of the context being singular would fail immediately upon the prescence of a function call. In this section, we show how to generate VCs for a general setup that has both basic blocks and function calls. To illustrate the key idea of our technique better, we shall restrict ourselves to the simple case of the delta-logic of lists and list-measures introduced earlier. A formulation for the general delta-logics is certainly possible along the same lines.

Let us first consider the case of a program that contains a basic block, followed by a function call, followed by another basic block. We would like to generate the VC for this. Observe that we already know how to generate the VC for the first basic block: we encode the precondition and the program transformation for that basic block in our decidable framework. Let us call this formula VC_1. We then mandate that the semantics of the function call is that the heaplet of the function call (which is the underlying heaplet of its precondition) is havoced entirely and merely satisfies the postcondition ensured by the function. We also use frame reasoning to ensure that any formula whose underlying heaplet does not intersect with the heaplet of the function call, if holds before the call holds after the call as well. This can also be done systematically by checking the truth of all the predicates on all variables in the formula and inferring them in the post-state of the function call if frame reasoning allows it. Here, we denote the precondition and postcondtion of the function call by FC_{pre} and FC_{post} respectively.

However, this brings forth a problem: what are the recursive functions that can be used describe the post-state? We certainly cannot use the parameterized functions describing the pre-state for the context has changed (possibly) entirely. This requires us to have *new* parameterized recursive definitions, where we have replaced any pointer field p_i with p_i' in the definitions themselves. In the case of the decidable logic of lists and list-measures, this is done by having two sets of parameterized recursive functions (each of which appear with two sets of parameters denoting the pre- and post-state of their respective basic blocks), and modelling the abstraction of the context for the second basic block using a *different* uninterpreted function T', which plays the role of T described in Section **??** for the second basic block. The idea is that the postcondition and the frame reasoning will connect the two different recursive definitions, thereby connecting T and T' indirectly, and this will accurately capture the effect of the changing context. Therefore, the generalised VC would be of the form:

$$\begin{gathered} VC_1(\mathcal{R}^{P1}, \mathcal{R}^{P2}, n, \overline{x}) \wedge \Big(FC_{pre}(\mathcal{R}^{P2}, n, \overline{x}) \Rightarrow FC_{post}(\mathcal{R}'^{P3}, n', \overline{x}')\Big) \wedge \\ \textit{frame-reasoning}(\text{heaplet}(FC_{pre})) \wedge VC_2(\mathcal{R}'^{P1}, \mathcal{R}'^{P4}, n', \overline{x}') \end{gathered}$$

where $\mathcal{R}^{P_i}$ refers to the recursive functions in the state of the program before the function call, i.e, with the first context and $\mathcal{R}'^{P_j}$ refers to the recursive functions that are defined on the second context: the one after the function call. n and n' refer to the respective pointers, and $\overline{x}$ and $\overline{x}'$ refer to the total set of program variables in the pre- and post-states of the function call (these need have no correspondence, of course, but the function contract can refer to both). Finally, *frame-reasoning* is a placeholder term referring to a formula that systematically infers valuations of recursive functions on the post-state of the function call from the corresponding valutions on the pre-state, depending on the heaplet of the function call.

It is easy to see that this technique can be extended to any number of basic blocks interrupted by function calls. This gives us a generalised VC generation technique that helps us verify more interesting programs, examples of which are discussed in Section **??**.