FAQ's based on the paper 'Verification of parameterized concurrent programs by modular reasoning about data and control'

August 21, 2013

This FAQ is the result of long exchange of mails with Zachary (Zak) who is one of the author of this paper. Questions/doubts presented in this FAQ came up while reading the above mentioned paper by the author(chinmay@cse.iitd.ac.in). The example mentioned in the last question of this FAQ is by Suvam Mukherjee(suvam@csa.iisc.ernet.in) and Deepak D'Souza(deepakd@csa.iisc.ernet.in). Zak's patience in explaining things was exemplary and I wish to learn this virtue from him. This FAQ is organized in form of a series of questions and the corresponding explanations from Zak. I would suggest interested readers to first go through the paper carefully before looking at this FAQ. You should download the corrected and the updated version of this paper from Zak's web page http://www.cs.toronto.edu/~zkincaid/ instead of downloading it from the ACM's proceedings.

Observation. In Sequential Data Flow Graph construction a data flow edge $u \rightarrow^x v$ is added for the variable x irrespective of whether it is local or global but in the subsequent iterations of *Coarsen* algorithm only data flow edges of the form $u \rightarrow^x v$ are added (as a result of interference analysis) where x is a global variable. This is because only global variables can pass values from one thread to the another.

Q. What is the intuition to have an abstract annotation $\iota^{\#}(u)$ at location u, such that $\iota(u) \rightarrow \iota^{\#}(u)$ and all free variables of $\iota^{\#}(u)$ are global variables?
Ans. Chinmay: I think the intuition is that threads only synchronize via global variables and $\iota^{\#}(u)$ captures the valuations of global variables that reach at location u. This valuation along with the enabledness condition at some location v determine if the location (u, v) can be reached together. This notion is captured in rules *coreachable* and *mayReach* used in the interference analysis.
Zak: The really important thing about $\iota^{\#}(u)$ is that it is a finite-domain abstraction of i(u), which allows us to do our interference analysis efficiently using binary decision diagrams. You're right that having global variables makes it (conceptually) easier to do the enabled-ness checks because all the threads know how to evaluate global variables. I would say the intuition is that $\iota^{\#}(u)$ is intended to capture synchronization information, and the way that threads synchronize is to use global variables (since they can't see each other's locals).

Q. Does the function *enabled*(u) change if new data flow edges are added to the graph?
Ans. Function *enabled*(u) depends on the choice of observable predicates (C), which is determined by the predicates used in assume and lock/unlock constructs, but not on the annotation $\iota(v)$. Therefore enabling conditions do not change over the iterations of the coarsening loop.

Q. For the running example of the paper, I think $enabled(u1) = lock2 = 1$, $enabled(u2) = lock1 = 1/\ lock2 = 0$ and $enabled(v4) = counter > 0$ hold. Am I right here? Also, for non-assume and non-lock statements what will be enabled function? Will it be true for such statements?
Ans. It depends on how we choose C. But if C is $lock1 = 1, lock2 = 1, counter > 0, counter >= 0$ then we would have

- enabled(u1): lock2 = 1
- enabled(u2),enabled(v1),enabled(v3): lock1=1
- enabled(u3): counter<=0
- enabled(v4): counter>0

and yes, non-assume, non-lock statements will have the enabling condition "true".

Q. If Assume determines enabled at a location and enabled give a observable formula F#(GV) with only GV as free variables what will be the enabled function for those assume constructs which have both global variable as well as local variables in it. And what about those assume constructs which have only local variables in it, for example u11 and u14 in the running example.
Ans. We can compute the enabled function using predicate abstraction. Suppose that C is the set of observable conditions, and suppose we want to compute enabled(v) for some vertex v labeled with "assume(phi)". We do this by taking the conjunction of all c in C such that phi implies C (checking the implication with an SMT solver). For example, if C is $\{x >= 0, y >= 0\}$, then

- enabled(assume($x > 0$)) = $x >= 0$
- enabled(assume($z >= 0$)) = true
- enabled(assume($z >= 0 \wedge y >= z$)) = $y >= 0$

As a result of this, assume statements that refer only to local variable typically have an enabling condition of "true". The only case where this is not the case is something like $enabled(assume(local > local))$ for which it is false.

Q. It is clear that the assume constructs only referring to local variables will have enabling condition as true but what about assume of the form $(global > local)$? As said in your reply to the previous question, Its enabled condition is conjunction of all c in C(Observable Predicates) such that $global > local \rightarrow \wedge_i c_i$. If we do not use the annotation ι to get the abstraction about the local variables at this point(which means assertion on local vars) how can SMT solver be used

to check for the satisfiability of above formula? or is it the case that local vars are made universally quantified in the formula ($\mathit{global} > \mathit{local}$) and then SMT solver is used?
Ans. Yes, that's right - the local variables are (implicitly) existentially quantified. As you suggested, a more precise way to do this would be to use the annotation ι to add information about the local variables (which would still be existentially quantified, but we'd have more information about them). The procedure is sound without using the annotation, however.

Q. There is a statement used in 2-3 places where observable conditions and observable formula are defined. 'that is as strong as any other observable formula with that property'; I am not sure I understand its meaning and the consequence fully. For example in the definition of $\mathit{enabled}(v)$ and $\iota^{\#}(v)$ in section 4, paragraph 3 and 4.
Ans. In the terminology of abstract interpretation, this just means that we want the best abstraction of something in the domain of observable formulae. It can be computed as I suggested above (taking the conjunction of all c in C such that phi implies C (checking the implication with an SMT solver)). For example, suppose that C is $\{x >= 0, x <= 0\}$, and we want to compute an observable formula that is implied by "$x = 0$". There are four choices for this:

- true
- $x >= 0$
- $x <= 0$
- $x >= 0 \wedge x <= 0$

We use the phrase "as strong as any other observable formula with that property" to ensure that we pick the last one.

Q. mod set of an assume construct contains every variable and as is mentioned it is to make sure that data flows to condition before flowing further to the enclosing block (in if else/while case). I believe that the transition function for assume is going to be identity relation because it 'actually' never modifies any variable. am I Correct? Further, I assume that in subsequent iterations (after finishing sequentialDFG analysis) it is mainly enabled function which controls if any statement after an assume construct is going to be reachable (coreachable/mayreach) or not. For example, the fact that enabled(v4)=$\mathit{counter} > 0$ and $\iota^{\#}(u10)$=$\mathit{counter} = 0$ is the reason that $\mathit{mayReach}(u10, x, v4, v5)$ does not hold and therefore no data flow edge is added from u10 to v5.

But the same inference (why no edge exist from u10 to v5) can also be obtained by considering that $\mathit{assume}(\mathit{counter} > 0)$ has counter in its mod set and therefore the path from u10 to v5 is not added. This is the argument used in section 5 (before theorem 5.2) to argue why no edge is between u10 to v5. But I think that the argument that $\iota(u10) \wedge \mathit{enabled}(v4)$ do not have a satisfying assignment ($\mathit{counter} = 0 \wedge \mathit{counter} > 0$) seems to be more 'correct' reason of not having an edge between u10 and v5.
Ans. The transition relation for an assume is a *partial* identity relation (it doesn't change the values of any variables, but it might block/have no successors on some states). The paragraph just before section 3 gives a formal definition.

For your u10/v5 example, it is not enough for counter to be in the mod set of "assume($counter > 0$)" to prove that absence of data flow. Consider the trace:
u1 u2 u8 u9 v1 v4 u10 v5

This trace witnesses the data flow $u10 \rightarrow^{counter} v5$, but it's i-infeasible because $\iota^{\#}(u9) \wedge enabled(v4)$ is unsatisfiable.

It's also not enough for enabled(v4) to be incompatible with $\iota^{\#}(u10)$. Consider the trace
u1 u2 u9 u9 u10 u14 u15 v1 v4 v5

This trace is i-feasible (since $\iota^{\#}(u15)$ is just "true"), but it doesn't witness the data flow $u10 \rightarrow^{counter} v5$ because counter is in the mod set of v4.

So: both the way that we treat "mod" for assume and the way we handle synchronization are required in order to prove the assertion at v6.

Q. Consider the following example program running two threads,

$$\begin{tikzcd}
\{true\}[a]x := 0 \arrow[d] & \{true\}[b]x := 1 \arrow[d] \\
\{x = 0\}[c]assume(x >= 1) & \{x = 1\}[d]skip
\end{tikzcd}$$

with inductive assertions ι attached to them (after sequential analysis). Is the trace (0,a)(1,b)(0,c) an ι-feasible trace with respect to this ι or not? If it is an ι-feasible trace then does it witness the edge $b \rightarrow^{x} c$?

Ans. It is not ι-feasible. However, the trace (0,a)(1,b) is ι-feasible and witnesses the data flow $b \rightarrow^{x} c$. Further, $(endloc((0,a)(1,b),0)$ is c.

To be a little more formal, paths are sequence of actions, and each action consists of a thread ID and a control flow edge. So the path (0,a)(1,b) should really be written as (0,(a,c))(1,(b,d)), which makes it a little more clear that the end location of thread 0 should be c.