Review of Symbolic Verification of Translation Model Transformations

Levi Lúcio Bentley James Oakes Hans Vangheluwe

October 28, 2014

Overall evaluation

In a rather impressive attempt to redress the technical shortcomings identified in the second version, the authors have rewritten large parts of their paper (though, of course, the overall message has remained the same). Most if not all of the gaps in the formalisation have now been filled, but (sorry to say) in many cases not in a very satisfactory way: apart from instances of sloppiness and awkward formulation — which are in principle straightforward to repair — there are also unclarities where it is not possible to guess the intended meaning, and worse, points where the paper is actually erroneous.

The main problem with the formalisation, however, is that (partly because of the awkward formulation) it is now so incredibly baroque that it has actually become impossible to understand. A prime example of this is the construction of the path condition in Section 4.3, which is layered through a staggering total of 8 distinct transition-like relations, the majority of which raise some, or in some cases a lot of, questions of mathematical correctness.

In my opinion, this complexity is, or should be, unnecessary. The fundamental programme of the paper is clear and indeed well-known: you have a concrete operational semantics and an over-approximating but finite abstract semantics. The baroqueness of the latter in the case at hand is a sign of a poorly designed abstract semantics or a poorly chosen way to explain it, or possibly a combination of both.

I will give concrete suggestions for improvement below, but an exhaustive exposition would take more time than I have available or feel I should spend on a paper review. Indeed, I am approaching the point where the number of pages of review I have written for this paper exceeds that of many papers I have authored. To be explicit, this review is the result of two full days spent on a dedicated effort trying to understand the first half of the paper.

The above observations unfortunately mean that the paper needs yet another revision that is far too encompassing to be called minor. Since it is now in its third iteration, I am with great reluctance forced to conclude that the team of authors is trying to achieve something for which they do not have the right background. I therefore recommend to reject the paper altogether.

General comments

Here I will address some discussion points and shortcomings of the paper that are not directly of a technical nature. The order of discussion is that of the paper, not of importance of the issues addressed.

Note that I have stopped trying to understand the formalisation somewhere in Section 4; the absence of comments from thereon is not evidence of the absence of potential problems.

1. On Page 2, left, mid-page you state that your methods "differ from previous work"in that you require no intermediate representation. It is certainly not preordained that this difference is an advantage: such an intermediate representation typically gives access to general-purpose tooling, e.g., for model checking or theorem proving. You would have to argue that you are better off reimplementing that functionality in your own setting. Is there any evidence for that? An example of such evidence (in a somewhat different context) is for instance provided by Pennemann in his PhD thesis, where by experiments he shows that his dedicated, graph-based resolution outperforms that of a general-purpose theorem prover.

2. The paper is inconsistent (maybe due to the iteration of submitted versions) in what it says about its own structure. Page 2, right, 1st paragraph states that the language and its samentcs are presented in Section 2; Section 2 (1st paragraph) says that the semantics is presented in Appendix B; most of it is actually in Section 3. Appendix B indeed repeats essentially all of Section 3, a duplication which certainly serves no purpose and should be remedied. In fact, I would much prefer *all* of Appendix B to be moved to the main text, as the only part that is now only in the appendix, the definition of a transformation, is absolutely central to the paper.

   As it currently stands, Section 3.4 has a similar problem: first it states that the semantics is not in the main text, then it refers to Section 3.2 for the semantics (which the reader has just gone through), then in Def. 17 jumps to a notation that is only introduced in Appendix B.

3. Your formalisation is made more complex than necessary because you do not put the power of typed graphs to work. If your source and target metamodels have disjoint types (which you can assume without loss of generality) you can take their union, augmented with *trace* edge types from all target node types to all source node types, and take that to be the meta-model for an input-output model, and also for a rule. Suddenly you don't have to carry arround the distinction between *Match* and *Apply* (for rules) or *input* and *output* (for input-output models) around any more: they are just the projections onto the types of the source, resp. target metamodel.

4. Section 4.2.2 on "backward links" is rather confusing, as it speaks about traceability links being added to rules. You never add traceability links to rules anywhere in the paper; in fact, your formal notion of a rule has no room for them. Instead, rules only have backward links. (The clarity of the the situation is not improved by the fact that those are labelled *trace*.)

   I think the situation would be much more clear if you were to use the common notion of left hand side and right hand side of a rule. The left hand side of a

rule rl is precisely your $||rl||$, the right hand side is the entire rl augmented by your traceability links, as in 4.2.2. Suddenly we are back in a well-known graph transformation formalism, and your rules are in fact so simple (no deletion, no negative application conditions, injective matching) that I believe it makes no difference whether you use algebraic graph transformation or some more constructively defined variant. The only standard notion is that of transitive closure in the left hand side; but there are plenty of GT tools that offer this extension.

In fact I see no reason why your rule application would then not precisely coincide with the the same notion in, say, SPO graph rewriting; and I hope you agree that this would help you no end in explaining what you are doing. (If, on the other hand, there is after all some difference then this, too, would be interesting to know, as that difference is not at all apparent right now.)

Viewed like this, moreover, I think your notions of input-output models and rules are very close to triple graph transformation as used by Schrr at all. I think the main (if not only) difference lies in the fact that your "glue graph" is actually not a graph; instead, you use traceability edges directly from the target model to the source model. If you were to turn those edges into nodes with source and target edges, even that difference would disappear.

5. In Section 4, rather than starting first with the construction of path conditions (Section 4,4), it would be more natural to discuss their meaning in terms of transformation executions (Section 5), since that should be the justification for your constructions.

Formalisation

In this section I will collect the technical issues I have encountered in the formalisation. These are of varying degrees of severity; again, I am using the order of the paper rather than importance.

Section 3

6. Graph union (Def. 2) is only well-defined if $s \cup s'$, $t \cup t$ and $\tau \cup \tau'$ yield functions, which in turn is only the case if s and s' etc. coincide on common elements. This is neither stated as a requirement here nor checked anywhere in the use of the union. (Since I sure you implicitly had this restriction in mind, this may be counted as sloppiness; you can probably assume this to hold w.l.o.g. wherever you need it.)

7. For multigraphs (which may contain parallel edges with the same label) it is necessary to include edge images in the homomorphism (Def. 3). If you do not do that, then graphs with parallel edges are isomorphic to graphs with single edges (between the same nodes), so you might as well restrict to simple graphs.

   Concretely: Right now the def. requires the existence of *some* unfixed e' in the target graph, but in fact f should be extended to $E \to E'$ so that $e' = f(e)$ in this situation.

8. Def. 6: Since the codomain of τ is apparently $VT \times \{abstract, concrete\} \cup ET \times \{containment, reference\}$, it is certainly never going to be a bijection. However,

I think you actually only want injectiveness; what purpose is served by requiring it to be surjective?

As an aside, I think this is quite an awkward way to encode the notions of abstractness and containment: I think it would be much more elegant to define abstractness as a predicate over *VT* and containment as a predicate over *ET*. To go even one step further, it would in fact be more natural (and more standard) to equate the notions of type graph and metamodel altogether, as they certainly serve the same conceptual purpose.

9. Def. 7: is an expanded metamodel a metamodel? Then you need to allow non-injective τ, at least on edges, as the requirement $\tau(e) = \tau(e')$ in the third bullet will certainly cause non-inejectiveness.

10. Def. 9 and many thereafter incrementally define new concepts on top of graphs. Most of these definitions are rather hard to read, for several reasons:

    - It is common to use an unprimed version for the original object and a primed version for the constructed object, whereas you turn this convention around
    - Sloppiness: in more than 50% of all cases the $\subseteq$ should be $\supseteq$ or vice versa; in this particular instance, you really mean $\tau' \supseteq \tau$ and $E = E' \cup E_c^*$.
    - It is really more readable to define the construction as a construction, so:

    $$\begin{aligned} E &= E' \cup E_c^* \\ s &= s' \cup \{(e_1 \cdots e_n, s(e_1)) \mid e_1 \cdots e_n \in E_c^*\} \\ t &= t' \cup \{(e_1 \cdots e_n, t(e_n)) \mid e_1 \cdots e_n \in E_c^*\} \\ \tau &= \tau' \cup \{(e, \mathit{indirect}) \mid e \in E_c^*\} \end{aligned}$$

    (Note that this also takes care of s and t, which you have forgotten here and in all other similar definitions.) The set E_c^* should itself also be more carefully defined, as

    $$E_c^* = \{e_1 \cdots e_n \in E'^* \mid \forall 1 \leq i < n.t(e_i) = s(e_{i+1})\}$$

11. Def. 10: Here you certainly should *not* require τ to be injective. The requirement on containment relations is mathematical nonsense: if $e' \in E'$ has target b then $b \in V'$ and so $f(b)$ does not exist, and most certainly is not the source of any edge in E. In fact, I really have no idea what you want to say here; the explanation below the definition does not clarify matters at all.

12. Def. 14: According to your own criterion, $||rl||$ is *not* a rule as it does not create either a node or an edge.

13. The second half of Def. 16 raises a lot of questions. First of all, neither $\sqsubseteq$ nor $\cong$ are defined over rules; however, I can imagine what it would look like. Then, what role does *rl* play; do you just mean $||rl_1|| \sqsubseteq ||rl_2||$ given a suitable isomorphic copy? What if $||rl_1|| \cong ||rl_2||$ (as a special case); is your condition then not unsatisfiable as rl_1 must appear after rl_2 as well as vice versa?

    More importantly though, why do you care? Your semantics makes sure that rules are never considered for application twice in succession, so dependencies are always resolved unambiguously.

Section 4

14. Def. 18 suddenly introduces the notion of "rule copies" (*Rulecop*) which are really not explained properly. My guess is that they are isomorphic representatives of rules. On page 13 under "notation" this component is called *Rule* and referred to as a relation, and in Def. 28 it is denoted as *Rules* and required to be "the same as the set of transformation rules used to build [the transformation execution]" (and hence *not* isomorphic copies?). Needless to say, these inconsistencies do not help understandability.

15. Def. 19 can only be understood once it is clear that your plan is to make isomorphic rule copies in which the node and edge identities already coincide with those in your path condition. (Hence my "isomorphic representative" guess in the previous remark.) This is very much nonstandard in the works of graph transformation, and poorly motivated. What are the advantages of this approach over relying on partial morphisms from a fixed rule to the path condition? As it is, you have to explain that you only consider isomorphic copies that differ in relevant parts, where "relevant" is determined by the question whether a node/edge identity coincides with one on the path condition. (You actually do not explain that at all: strictly following your definitions one would have to consider an infinite number of isomorphic rule copies, assuming that the number of available identities is infinite (as it must be).)

16. The right hand side of the conclusion of Def. 20 has $AC \cup \bigcup_{pc' \in AC} pc'$, of which I cannot make sense. AC is a set, but the pc' are not, so what operation is $\bigcup_{pc' \in AC} pc'$? Do you mean $\bigsqcup_{pc' \in AC}$ instead? But then the result is not a set, so how can you take its union with AC? The same problem reoccurs in other definitions.

17. Def. 24: it seems to me you are making a very serious mistake here. $\blacktriangleright$ tests for the existence of a morphism, but in Def. 19 you are relying on coincident node/edge identities. (See my remark 15.) Maybe you mean that the morphism from *rl* to *pc* should drive the choice of node/edge identity in the isomorphic copy of *rl* to be combined with *pc*, but this is really a wild guess on my part.

    In fact, though the intuitions behind the constructions in Section 4.4 still make sense to me, the formal constructions do not. At this point the problems with the mathematics have accumulated so much that I have given up trying to understand or guess what was meant. As stated above, I feel that the authors have failed to identify the right abstractions to make this construction believeable. Possibly things could be cleared up by following the suggestions in my remarks 3 and 4.

Appendix B

18. Def. B.12. First of all, I wonder why you define *match* as a function rather than just defining what a match is; namely, an element of the set you are constructing here. Secondly (and more importantly), why do you not use the notion of homomorphism instead of explicitly talking about ismorphic subgraphs? Your way, you lose the connection between *glue* and $||rl||$: you know they are isomorphic, but what if $||rl||$ has symmetries — then there is more than one match, which in your definition cannot be distinguished.

You pay the price for this loss of information in Def. B.13, which now is broken: a_Δ depends on m_{glue} (which was called *glue* in Def. 12, why the change in notation?) in a way depending on the mapping from m_{glue} to *rl* which you have discarded.

Def. B.13 is broken in another way as well: it does not properly define E' but merely imposes some requirements that are also fulfilled by choosing $E' = E$. Instead I know from the discussion about traceability links in Section 4 (which logically comes *after* this appendix!) that you really want to add such links for all newly created nodes of the output graph to all pre-existing nodes of the input graph; but that is not made clear anywhere at this point.

19. Def. B.14: It seems to me rather roundabout to sequentially apply all rules, accumulate the results and then take their union. Why do you not take the union simultaneously with the parallel application of all rules (of a given layer), as you are essentially doing for all applications of a single rule in Def. B.13? This would get rid of the artificial sequential ordering, and the whole issue of confluence would not even have to be raised as there is no non-determinism any more.

20. In the constructions defined in Appendix B, you ignore the required absence of containment cycles. Might such cycles not appear in the union of the graphs, even if the individual graphs are cycle-free?

Textual comments

p2 right, l–8 "must" → "will always"

p3 right, l–10 "means" → "mean"

p3 right, l–2 "Figure 1b" → "Figure 3b"

p4, Fig. 3 I would suggest adding another node f5 superviesd by f4, to illustrate the notion of transitive closure. Another remark: I think the black containment diamonds do not belong in an instance model, as they are a metamodel notion.

p4 left, l–11 "subset" → "subgraph"

p4 left, l–4 "the expressiveness of the transformations our algorithm can examine" → "the expressiveness of the rule language"

p5, Fig. 4a The two "agent" edges in the postcondition whould be marked "female" resp. "male"

p5 left, l2 "This would be" → "For instance, take"

p5 right, l2 of "Indirect Match Links" "a nonempty path" (I think). The label of the containment associations does not matter?

p6 right, l–2 of Def. 1 Font of e should be e

p7 left, Def. 3 and all subsequent definitions g has an unexplained component st, is this the same as the pair (s,t) in Def. 1?

p7 right, Def. 6 and many other places A word in LaTeXmath mode should always be in `\mathit` to avoid ugly spacing as in $reference$.

p8 left, l1 "relation" → "type"

p8 left, l6 "the formal treatment"

p8, l–1 above Def. 9 "as enforced by EMF"?? Do you mean "in analogy to EMF"?

p8 left, l–1 The difference between these stars is too small to be noticeable.

p8 right, l5/4 above Def. 11 "the cardinality of the target class"?? Are you talking about edge multiplicities here (which you have chosen not to include)?

p8 right, Def. 11 Why is *Input* a model and *Output* a metamodel instance? Note that $\setminus$ binds stronger than $\cup$ in $E \setminus E' \cup E''$. "metch-apply patterns" → "input-output patterns"

p9 left, l–4/–3 of Def. 13 e in $\tau(e) = trace$ is still bound by $\nexists e$, so "and" should be $\wedge$

p9 right, Def. 15 The first bullet is a special case of the second, so can be removed.

p9 right, l2 above Def. 16 "important such that" → "important to ensure that"

p10 right, Def. 17 There is a copy/paste error w.r.t. Def. B.16 (which only serves to strengthen my point, made above, about avoiding this kind of duplication). In this case, B.16 is the correct version.

p11 right, l1/2 of 4.2.1 "allow for dependencies to be specified on" → "specify"

p11 right, l–2 "s solid line" → "solid lines"

p13 left, l–13/–10 "Recall ...": we cannot recall this as it has not yet been presented; the abstraction relation is defined only in Section 5. See my comment 5 above.

p13 right, Def. 19 "joint" → "overlapping"? Why are the type graphs actually not identical, as *pc* and *rl* are defined over the same source and target metamodels? Last line: "rl" → "$\{rl\}$"

p4 left, l5 above Sect 4.1.1 "below figures" → "figures below"

p20 left, l9 of Def. 28 "built" → "build"

p45 left, l7 above Def. B.13 "Its role is to extend"

p46 left, l9 "built" → "build"

p46 left, Def. B.15 I believe tr, T and R in this definition stand for the same thing.