PE

INDIAN INSTITUTE OF TECHNOLOGY DELHI

FORM PG 13

Ph.D. No. 11753

Name of the Candidate	Abhishek Rose
Title of the thesis	*MODELING DYNAMIC ALLOCATIONS AND DEALLOCATIONS OF LOCAL MEMORY FOR TRANSLATION VALIDATION*

1. **Please note that this form should be accompanied by** a report commenting on the strengths and weaknesses as well as specific suggestions for improvement of the thesis **as per the guidelines given in the covering letter. This recommendation form is to mainly facilitate the Dean (Academics) in deciding appropriate action. Check (V) for your recommendation.**

Option I.	The thesis is recommended for **award in its present form.**	
Option II.	The thesis be accepted for the award after **minor revision**. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	V
Option III.	The thesis be accepted for the award after **minor revision.** I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option IV.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	
Option V.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option VI.	**Re-writing of thesis after further research** is recommended. The re-written thesis should be sent to me for re-evaluation.	
Option VII.	Thesis may be **rejected** outright.	

2. At IIT Delhi, theses are considered for the award of "**Distinction in Doctoral Research**". Only upto 10% of the thesis may be given this award. The basis for this award is the excellence in doctoral research reflected by the quality of research produced as documented in the Ph.D. thesis as well as in the public domain.

   Would you recommend this thesis for consideration of the award? **Yes**/No

   Additional Comments, in this regard, if any: **See next page**

Name of the Examiner [redacted] **Signature of the Examiner**

Date: November 5, 2024 [redacted]

Additional comments on PhD Thesis of Abhishek Rose,
By [redacted], November 2024

This thesis is a very capable *tour-de-force*. It addresses a particular aspect of compiler correctness proofs: the use of the stack frame to implement local variables and allocation using the alloca() primitive. Previous formally verified compilers, such as CompCert (Leroy et al., 2006-2024) and CakeML (Myreen et al., 2014-2020) address the issue of local variables but cannot handle local allocation using alloca(). Compilers that do implement alloca() (such as gcc and LLVM) are not formally verified. For compilation that includes alloca(), this thesis shows how to do formal *validation* (proving that any particular compilation is correct), which is perhaps not as desirable as verification (proving that all compilations will be correct) but still very respectable.

The techniques used are sophisticated, state-of-the-art, and clearly explained in very great detail. The implementation is carefully measured with a benchmark suite that gives insight into the practicality of the work.

Unfortunately, the results show that this technique is not yet practical: it does not scale to medium-size programs, it takes many minutes or hours on small programs, and it does not handle the full range of optimizations that gcc or LLVM perform. That does not detract from the quality of the PhD thesis research work: results like this are how science advances.

I have several minor comments about the presentation. It is not necessary to address all of them in the final version of the thesis; I leave this up to the judgment of the author.

Minor comments:
page 2, "correct by construction": This is not what correct-by-construction means. C-by-C means that the programming language works in such a way that the compiled program *must* be correct w.r.t. the specification, and the specification *is* , in effect, the source program. C-by-C works only in specialized application domains, such as parsing or network configuration.

What CakeML and CompCert do is not C-by-C. It is "formally verified". That is, the Gallina program that implements CompCert is not C-by-C, it would be easy to write a different, incorrect, Gallina program. But that program is accompanied by a formal proof.

page 4, "GCC alloc() operator": alloca() is not special to gcc, it was present in 1970s-era Unix C compilers long before gcc was envisioned.

Page 7: You (correctly) state the "backward" correctness: For each execution of A, there exists an execution of C. But in the next few paragraphs, your informal proof implicitly goes "forward". This can be OK if either (1) the program is deterministic, or (2) your argument is "loosely formal". I am not sure that (1) holds, because we don't know what scanf() does. Clearly (2) does hold, you've said so, but perhaps you should point out this issue.

Page 32: "non-terminating": This terminology is ambiguous, because "nonterminating" usually means "infinite-looping" which is not what you mean here.

Page 51: sentence beginning "An informal argument in favor": I don't understand this entire sentence, and especially this comma; it seems like it's a run-on sentence; should the last comma be a period?
And then, I just don't understand the argument. Is it Adot or A that has more executions? The phrase "additional executions over A" is unclear.

Page 53, paragraph beginning "Recall that": This is very significant. Gcc does an amazing job of tail-call optimization, and LLVM does almost as well (though not quite). Given the trade-off between having alloca() or having tail-call optimization, there are many applications in which tail calls are more important than alloca(). Therefore, this is a significant limitation of this work, and it should be acknowledged in the introductory chapters.

Page 58, "A local variable may either be stack-allocated or virtual-allocated, but not both": Many compilers do

"live range splitting". That is, the optimizer will break up the live range of a local variable x into two local variables x1 and x2 with an assignment x2=x1 at the split point. This allows x1 to be spilled while x2 is kept in registers, which would be a good thing if, for example, x2 is in a loop while x1 is in rarely executed error-exit code.

For an extreme form of this, see:
Andrew W. Appel and Lal George. 2001. Optimal spilling for CISC machines with few registers. In Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation (PLDI '01). Association for Computing Machinery, New York, NY, USA, 243–253. https://doi.org/10.1145/378795.378854

It can also be argued that SSA itself is a kind of live-range splitting, because each source variable is split by phi-functions into several LLVM variables.

So, does this section mean that live-range splitting is completely forbidden?

Page 85, "Refinement failure due to elimination of alloca():" In contrast to my very significant concerns (noted elsewhere) about the inability to accommodate tail-call optimizations or live-range splitting, it seems to me that the restriction described here is entirely reasonable and harmless.

Page 114, "safety-relaxed semantics": It would be a good idea here to add another sentence explaining, informally, what "safety-relaxed" means. That is, what is being relaxed, and why? This would help motivate the formal statements (a), (b), (c).

Page 174, "performance anomaly": Please quantify how much this cost in performance, as a percentage of run time of the whole program. I imagine it would be very little. Is it really significant, or not?

IE

INDIAN INSTITUTE OF TECHNOLOGY DELHI

FORM PG 13

Ph.D. No. 11753

Name of the Candidate	**ABHISHEK ROSE**
Title of the thesis	**Modeling Dynamic Allocations and Deallocations of Local Memory for Translation Validation**

1. **Please note that this form should be accompanied by** a report commenting on the strengths and weaknesses as well as specific suggestions for improvement of the thesis **as per the guidelines given in the covering letter. This recommendation form is to mainly facilitate the Dean (Academics) in deciding appropriate action. Check (√) for your recommendation.**

Option I.	The thesis is recommended for **award in its present form.**	√
Option II.	The thesis be accepted for the award after **minor revision**. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	
Option III.	The thesis be accepted for the award after **minor revision**. I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option IV.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	
Option V.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option VI.	**Re-writing of thesis after further research** is recommended. The re-written thesis should be sent to me for re-evaluation.	
Option VII.	Thesis may be **rejected** outright.	

2. At IIT Delhi, theses are considered for the award of "**Distinction in Doctoral Research**". Only upto 10% of the thesis may be given this award. The basis for this award is the excellence in doctoral research reflected by the quality of research produced as documented in the Ph.D. thesis as well as in the public domain.

   Would you recommend this thesis for consideration of the award? Yes

   Additional Comments, in this regard, if any: The research presented in the thesis is very high quality systems research, that solves a difficult problem. I have worked in the area of translation validation, and am of the opinion that this thesis can open the doors to more powerful and versatile translation validators than what are available today.

[redacted]

Name of the Examiner [redacted] **Signature of the Examiner**

Date: Feb 14, 2025

Comments on Ph.D. Thesis of Abhishek Rose

The dissertation titled "Modeling Dynamic Allocations and Deallocations of Local Memory for Translation Validation" by Mr. Abhishek Rose is a record of very high quality systems research on a problem of significant academic and applied interest. Translation validation has long been recognized as a practical alternative to verified compilation. Yet, there are very few translation validators out there that can be used on real-life code. It's not that earlier researchers haven't tried to build practical translation validators. The truth is that building a translation validator that can be applied to code from practical applications is exceedingly difficult. In addition to the inherent computational hardness of the problem (translation validation is undecidable in general), a lot of the difficulty stems from the need to support various source language features commonly used by programmers, and in a manner that is faithful to the language reference semantics. Allocation and deallocation of local memory is one such feature that is ubiquitous in code from diverse applications. This is also one of the most challenging feature to accommodate in a translator validator, given the difference in memory abstraction at the source and assembly levels. Mr. Rose's dissertation is a bold step in trying to fill in this gap. It has been a pleasure reading the dissertation, and I wholeheartedly support acceptance of the dissertation for the award of a Ph.D. degree.

Having said the above, there are a few comments I had about the thesis. I am listing them below. Perhaps Mr. Rose can address these comments when preparing the final version of the thesis.

- Fig 1.1 is referred to in page 3. But there is no figure labeled or numbered 1.1
- The compiler + end-to-end TV flow isn't exactly doing the same thing as a verified compiler. Why would a verified compiler ever give an output saying that the compiler did something erroneous? But a compiler + end-to-end TV can certainly produce an output saying the compiler did something erroneous.
- Typo on pg 16: "first-logic queries" to be replaced by "first-order logic queries"
- The correctness of the technique proposed in the thesis depends crucially on insertion of alloc and dealloc annotations in the assembly, and also on insertion of dealloc instructions in the unoptimized IR. Given the computational difficulty of inserting these instructions at the right places, there are heuristics at play in figuring out where the dealloc is inserted in the optimized IR (especially for alloc instructions corresponding to variable or parameter allocation), and also in figuring out where the alloc and dealloc's are inserted in the assembly. I am left wondering what would happen if the dealloc's in the unoptimized IR and also in the annotated assembly are inserted incorrectly, but in a way that causes the SMT based

verification to sail through. Is this not a possibility? Couldn't the SMT based verification still go through with wrongly inserted dealloc's in both the unoptimized IR and the annotated assembly?

- On pg 29, it is mentioned that Σ_P^{hp} remains constant throughout P's execution. It isn't clear why this is necessarily true, especially if P has heap manipulating instructions (e.g. building and then freeing a dynamically allocated linked list). What purpose does the heap serve if it's not changing at all in a program that manipulates the heap?
- Chapter 2 is very impressive in terms of the details of modeling the execution semantics of both the unoptimized IR and the assembly instructions. The gradual development of more "permissive" refinement relations to capture increasingly faithfully the semantics of C programs with allocations, deallocations and re-orderings is very commendable. I went through the detailed modeling, and while the presentation does get tedious at times, I appreciate the rigour with which the treatment has been done. My only comment about this chapter is that the notation could perhaps have been simplified to enhance readability. The notation does come in the way of reading and understanding this chapter.
- I must especially commend the student for taking into account undefined and non-well-formedness conditions in an elegant way in the formulation.
- While the refinement relation with three dots (for lack of a better terminology) is clearly what we'd like to have, it is understandable that this requires a heavier cost to be paid in terms of annotation insertion and final checking of Hoare triples. The authors use this justification to proceed with the refinement relation with two dots (again, for lack of better terminology) as a more pragmatic tradeoff.
  While I agree with the above rationale in general, I would have liked to see an experimental determination of the overhead of using the refinement relation with three dots vis-a-vis that of using the relation with two dots. I presume the authors have actually done such experiments before arriving at the decision to proceed with the relation with two dots. However, I didn't see this experimental evidence in the thesis.
- Sec 3.4: Callers' Virtual Smallest Semantics is simply described by a set of "... is replaced with ...". It would have been good to motivate here why it is beneficial to use caller's virtual smallest semantics in the first place. We are only told much later (Sec 3.5) that this helps in SMT encoding.
  A similar comment applies to Sec 3.5: We suddenly start seeing Safety-Relaxed Semantics as a set of bulleted points "... is replaced with ..." It would certainly have helped to motivate these alternate semantics by giving an example of how it helps with SMT encoding.
- In Sec 4.1, a set of conditions are listed for Dynamo to be complete. Of these, conditions (a) through (d) are concretely checkable or enforceable. However, I don't think this is the case for conditions (e) and (f), which would be extremely difficult to enforce or even check.

Specifically, condition (e) states that the desired annotation to the assembly program is identifiable either through search heuristics or through user-supplied and/or compiler hints. Since nothing is stated about the class of search heuristics or hints to be used, and given that the "desired" annotation is one that allows us to prove the desired refinement relation, this is equivalent to stating that Dynamo is complete on the class of translations for which it can prove the refinement relation (a tautological statement). A similar comment applies for condition (f) that requires the invariant inference procedure to identify just the right invariants at the right nodes of the network.

- In the Dynamo algorithm, is there a specific reason that cut-points are considered in reverse post-order?
- How is the parameter μ chosen when invoking correlatedPathsInCOpts()? Is it a number chosen once and for all, or does this change (dynamically determined) for each translation validation run? Is μ inferred automatically using some heuristics, or is it something for which we must depend on the user?
- Pg 105: first enumeration item, line 2: "Then" to be replaced by "then"
- Algorithm asmAnnotOpts (Algorithm 4) lies at the heart of Dynamo. It appears from the presentation that this is currently implemented as a set of very effective heuristics. Is this true, or is there some notion of completeness of the algorithm (asmAnnotOpts), as presented?
- Pg 132: The text under invariant inference actually doesn't talk anything about how the invariants are being inferred. In general, inferring the right invariants is undecidable, so how are invariants being inferred in Dynamo? I think a paragraph on this should be added.
- Several key steps in Dyanmo involve trying out alternatives from a set of candidates, and as I see, the set of candidates can become large very soon. Is it possible to understand how large C and assembly program are likely to be amenable to the analysis by Dynamo, within reasonable time (say few tens/hundreds of minutes)? The experiments provide some estimate of the lines of C/assembly for the benchmarks that were successfully validated. It would have been wonderful to see (in a plot, perhaps) how the translation validation time increases with size of the code. Of course, the time depends not only on the size of the code, but on what the code really is doing. So I am aware that designing such an experiment is not easy. Yet, this would be something on my wish-list for a thesis of the high quality that this one is.
- Chapter 6 mentions that Dynamo is 400K SLOC in C/C++. Was all of this code written as part of this dissertation, or did this dissertation build on some kind of existing code base for Dynamo?
- Were the benchmarks reported in Table 6.1 hand-crafted, or taken from a benchmark source? If it is the former, I am curious how it was decided

that this set of benchmarks is sufficient – clearly this set doesn't cover all features of the language.

- Sec 6.2.3: What was the rationale for introducing the `O1-` option? How were the optimizations chosen to be disabled selected in the first place?
- Sec 6.2.4 is much appreciated. Understanding the causes of failures is a big part of understanding the strengths and limitations of Dynamo.
  I am a bit curious though why only bzip2 was chosen from SPEC2000. What would it take to run Dynamo on other benchmarks from SPEC200?
- An overall suggestion: The presentation is extremely notation heavy.at times (superscript of subscript of superscript of ...), and this does hurt the readability of the thesis. It is hard at times to keep track of all that is being represented by such complex notation. It would have been good to simplify the notation (I am sure this could have been done) to enhance readability.

S

INDIAN INSTITUTE OF TECHNOLOGY DELHI

FORM PG 13

Ph.D. No. 11753

Name of the Candidate	Abhishek Rose
Title of the thesis	MODELING DYNAMIC ALLOCATIONS AND DEALLOCATIONS OF LOCAL MEMORY FOR TRANSLATION VALIDATION

1. **Please note that this form should be accompanied by** a report commenting on the strengths and weaknesses as well as specific suggestions for improvement of the thesis **as per the guidelines given in the covering letter. This recommendation form is to mainly facilitate the Dean (Academics) in deciding appropriate action. Check (√) for your recommendation.**

Option I.	The thesis is recommended for **award in its present form.**	**YES** ✓
Option II.	The thesis be accepted for the award after **minor revision**. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	
Option III.	The thesis be accepted for the award after **minor revision**. I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option IV.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do not** wish to examine the response further, and would like the Examination Board to evaluate the response and ensure that the corrections are incorporated in the revised thesis.	
Option V.	The thesis be accepted for the award after **major revision** requiring rewriting a portion/chapter of the thesis incorporating some additional work. I **do** wish to examine the response further, before recommending for award of degree. The revised thesis and candidate's responses to comments may be sent back to me for re-examination.	
Option VI.	**Re-writing of thesis after further research** is recommended. The re-written thesis should be sent to me for re-evaluation.	
Option VII.	Thesis may be **rejected** outright.	

2. At IIT Delhi, theses are considered for the award of **"Distinction in Doctoral Research"**. Only upto 10% of the thesis may be given this award. The basis for this award is the excellence in doctoral research reflected by the quality of research produced as documented in the Ph.D. thesis as well as in the public domain.

   Would you recommend this thesis for consideration of the award? **YES**

   (Yes) No

   Additional Comments, in this regard, if any:

SBS

Name of the Examiner Sorav Bansal **Signature of the Examiner**

Date: 9th November 2024

Comments on thesis draft by Abhishek Rose

November 9, 2024

Abhishek's thesis tackles the problem of translation validation from C to assembly code in the presence of dynamic memory allocations, across aggressive compiler transformations including loop and vectorizing transformations. This is the first verification attempt across a translation that supports a wide variety of memory allocation and deallocation strategies that may be implemented within a compiler that translates the C code to assembly code. Translation validation is a fundamental and classic problem in computer science and formal methods, and Abhishek's thesis makes a fundamental contribution in this space. This is among the best PhD theses I have guided, and I have no hesitation in recommending the thesis for the award of a PhD, and also for the "Distinction in Doctoral Research" award.

On potential to carry out independent research

Abhishek has demonstrated several qualities such as curiosity, perseverance, hard-work, keen-ness to learn independently, writing skills, presentation skills, etc., during his interactions with me over the duration of his thesis. I feel convinced that Abhishek is ready to carry out independent research in future.

On the standard of presentation of the thesis

Good.

OFFICE OF DEAN ACADEMICS

Ph.D. No. 11753
February 17, 2025

Subject: **Ph.D. thesis submitted by Abhishek Rose (2017CSZ8584)**

Dear Prof. Bansal,

Reports from all the examiners on the aforesaid Ph.D. thesis have now been received. Both examiners have asked for minor revisions in the report and desired correction. An annotated thesis received from one of the examiners is being emailed for your perusal and further necessary action. A detailed response should be submitted by the candidate addressing the examiners' queries at the earliest possible. The same be forwarded by the supervisor(s) after ensuring that the examiners' concerns are addressed.

With regards,

(Dean, Academics)

Prof. Sorav Bansal, CSE

Dear reviewer,

I would like to sincerely thank you for taking the time to review my dissertation and for providing such thoughtful and constructive feedback. Your insights will undoubtedly help me improve the overall quality of my work. I appreciate the care and attention you have given to my work, and I am grateful for the opportunity to address your comments and suggestions.

Below, I have provided a point-by-point response to your concerns, along with the revisions I plan to make to address them. Please let me know if any additional clarifications or changes are needed.

Thank you once again for your guidance and support.

Regards,
Abhishek Rose

Point-by-point response to reviewer #1

> page 2, "correct by construction": This is not what correct-by-construction means. C-by-C means that the programming language works in such a way that the compiled program *must* be correct w.r.t. the specification, and the specification *is* , in effect, the source program. C-by-C works only in specialized application domains, such as parsing or network configuration.
>
> What CakeML and CompCert do is not C-by-C. It is "formally verified". That is, the Gallina program that implements CompCert is not C-by-C, it would be easy to write a different, incorrect, Gallina program. But that program is accompanied by a formal proof.

Thank you for pointing this out. Indeed, a correct-by-construction compiler would be a more specific type of "formally verified" compiler. I will fix this in the final revision.

> page 4, "GCC alloc() operator": alloca() is not special to gcc, it was present in 1970s-era Unix C compilers long before gcc was envisioned.

Noted. Will omit GCC in the final revision.

> Page 7: You (correctly) state the "backward" correctness: For each execution of A, there exists an execution of C. But in the next few paragraphs, your informal proof implicitly goes "forward". This can be OK if either (1) the program is deterministic, or (2) your argument is "loosely formal". I am not sure that (1) holds, because we don't know what scanf() does. Clearly (2) does hold, you've said so, but perhaps you should point out this issue.

Thank you for pointing this out. I will be try to be clearer in the text.

> Page 32: "non-terminating": This terminology is ambiguous, because "nonterminating" usually means "infinite-looping" which is not what you mean here.

I will add clarification and a reference to definition of "terminating" node.

> Page 51: sentence beginning "An informal argument in favor": I don't understand this entire sentence, and especially this comma; it seems like it's a run-on sentence; should the last comma be a period? And then, I just don't understand the argument. Is it Adot or A that has more executions? The phrase "additional executions over A" is unclear.

Adot has more executions. I will revise this sentence for the final revision.

> Page 53, paragraph beginning "Recall that": This is very significant. Gec does an amazing job of tail-call optimization, and LLVM does almost as well (though not quite). Given the trade-off between having alloca() or having tail-call optimization, there are many applications in which tail calls are more important than alloca(). Therefore, this is a significant limitation of this work, and it should be acknowledged in the introductory chapters.

I will add a line mentioning this limitation in the list of contributions in section 1.3.

> Page 58, "A local variable may either be stack-allocated or virtual-allocated, but not both": Many compilers do "live range splitting". That is, the optimizer will break up the live range of a local variable x into two local variables x1 and x2 with an assignment x2=x1 at the split point. This allows x1 to be spilled while x2 is kept in registers, which would be a good thing if, for example, x2 is in a loop while x1 is in rarely executed error-exit code.
>
> For an extreme form of this, see: Andrew W. Appel and Lal George. 2001. Optimal spilling for CISC machines with few registers, In Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation (PLDI '01). Association for Computing Machinery, New York, NY, USA, 243-253. https://doi.org/10.1145/378795.378854.
>
> It can also be argued that SSA itself is a kind of live-range splitting, because each source variable is split by phi-functions into several LLVM variables.
>
> So, does this section mean that live-range splitting is completely forbidden?

I think some cases should be permitted, especially those where the compiler does *not* recycle the stack space.

In your example, if the stack space for x1 is not reclaimed by the compiler till the end of the lifetime of x, then we can still *pretend* that x2 is stack-allocated (in the space reserved for x1), even though it is never spilled into the stack.

An important fact to note here is that if a variable is *not* address-taken (i.e., its address is *never* observed), then we can always virtual-allocate it. Even if it is spilled to stack entirely or partially.

So, the case that does not work is where the stack space of an address-taken variable is reclaimed during its lifetime. I would imagine it to be a rare occurrence.

Thank you for asking this question. I will add a paragraph on this in the revised version.

> Page 85, "Refinement failure due to elimination of alloca():" In contrast to my very significant concerns (noted elsewhere) ahout the inability to accommodate tail-call optimizations or live-range splitting, it seems to me that the restriction described here is entirely reasonable and harmless.

Noted. I will explicitly mention these limitations in page 62 (or page 85 of the PDF file).

> Page 114, "safety-relaxed semantics": It would be a good idea here to add another sentence explaining, informally, what "safety-relaxed" means. That is, what is being relaxed, and why? This would help motivate the formal statements (a), (b), (c).

Noted.

> Page 174, "performance anomaly": Please quantify how much this cost in performance, as a percentage of run time of the whole program. I imagine it would be very little. Is it really significant, or not?

Yes, it would be very little.

Point-by-point response to reviewer #2

> Fig 1.1 is referred to in page 3. But there is no figure labeled or numbered 1.1.
>
> Typo on pg 16: "first-logic queries" to be replaced by "first-order logic queries"
>
> Pg 105: first enumeration item, line 2: "Then" to be replaced by "then"

Thank you for spotting these typos! I will fix them in the final version.

> The compiler + end-to-end TV flow isn't exactly doing the same thing as a verified compiler. Why would a verified compiler ever give an output saying that the compiler did something erroneous? But a

> compiler + end-to-end TV can certainly produce an output saying the compiler did something erroneous.

Indeed, a TV-based verified compiler may report internal errors (due to TV failure) more often than a verified compiler. But this can potentially be mitigated, perhaps at the cost of time budget, by, for example, iteratively disabling optimization passes (if possible) until TV succeeds.

On the other hand, while an ideal verified compiler would only produce error due to invalid input or insufficient resources, real-world verified compilers such as the CompCert compiler have phases that invoke translation validation like approaches (CompCert's register allocation pass is an example) and hence may fail with internal error (if the validation fails) even in the presence of valid input and sufficient resources.

> The correctness of the technique proposed in the thesis depends crucially on insertion of alloc and dealloc annotations in the assembly, and also on insertion of dealloc instructions in the unoptimized IR. Given the computational difficulty of inserting these instructions at the right places, there are heuristics at play in figuring out where the dealloc is inserted in the optimized IR (especially for alloc instructions corresponding to variable or parameter allocation), and also in figuring out where the alloc and dealloc's are inserted in the assembly. I am left wondering what would happen if the dealloc's in the unoptimized IR and also in the annotated assembly are inserted incorrectly, but in a way that causes the SMT based verification to sail through. Is this not a possibility? Couldn't the SMT based verification still go through with wrongly inserted dealloc's in both the unoptimized IR and the annotated assembly?

The dealloc insertion in the unoptimized IR is rather straightforward by using the LLVM `stacksave` and `stackrestore` intrinsics. As I have explained in section 2.1.1 (page 21), these intrinsics are emitted by the Clang frontend for code generation purposes and are derived from scope information in the source C code. Unlike the assembly annotation, inserting deallocs from this information does not rely on heuristics – the locations are computed by identifying the allocations reaching a postdominating `stackrestore` (which would indicate the end of scope).

As mentioned in section 6.1.1 (page 157), the translation from C to unoptimized IR (including insertion of dealloc) is a trusted component in the system and any error here may result in soundness issues (i.e., reporting successful TV when in fact refinement does not exist) and our SMT-based verification cannot help here.

In the case of assembly, for refinement to hold, the (de)alloc annotations must be well-formed (e.g., $alloc_s$ allocated region belongs to stack etc.) and must match the (de)alloc in unoptimized IR (such that the two are in lockstep). The latter is insured because we generate traces recording each (de)allocation event in the form of write (`wr`) instructions. An "incorrect" annotation would either fail this

trace requirement or the well-formedness of the annotation check (note that these are the (Equivalence) and (Safety) requirements in the product graph).

In other words, the "correctness" of assembly annotations is defined in terms of the unoptimized IR. A soundness error, in this case, would be due to incorrect dealloc insertion in the unoptimized IR.

> On pg 29, it is mentioned that heap remains constant throughout P's execution. It isn't clear why this is necessarily true, especially if P has heap manipulating instructions (e.g. building and then freeing a dynamically allocated linked list). What purpose does the heap serve if it's not changing at all in a program that manipulates the heap?

As mentioned in section 2.1.2 (pg 25), we do not interpret heap allocation procedure calls like `malloc()` and therefore P's heap cannot change during its execution. Heap, in our case, refers to an allocated region that is not a variable (global or local), parameter, or stack.

> While the refinement relation with three dots (for lack of a better terminology) is clearly what we'd like to have, it is understandable that this requires a heavier cost to be paid in terms of annotation insertion and final checking of Hoare triples. The authors use this justification to proceed with the refinement relation with two dots (again, for lack of better terminology) as a more pragmatic tradeoff.
>
> While I agree with the above rationale in general, I would have liked to see an experimental determination of the overhead of using the refinement relation with three dots vis-a-vis that of using the relation with two dots. I presume the authors have actually done such experiments before arriving at the decision to proceed with the relation with two dots. However, I didn't see this experimental evidence in the thesis.

The generation of non-silent trace events in refinement relation with two dots (borrowing the terminology) enables a significant pruning of the search space for annotation: an annotation only needs to be added to an A path if the correlated C path has an (de)alloc. Further, the arguments to the annotated instruction can be deduced based on the correlated (de)alloc (section 4.1.3, page 112). Lastly, the restrictions on annotation of $alloc_v$ enable an efficient SMT encoding which is around 4 times faster than the naive encoding (section 6.2.1, page 164)

These advantages are lost in the refinement relation with three dots where there are no counterparts to the `s2v` and `v2s` instructions in C.

Guided by this reasoning, the fact that the three dots definition would potentially require a different set of cross product requirements (and associated algorithm for its automatic construction), and the results from initial experimental evaluation where the refinement relation with two dots sufficed in almost all cases, we decided against implementation of the refinement relation with three dots.

> Sec 3.4: Callers' Virtual Smallest Semantics is simply described by a set of "... is replaced with ...". It would have been good to motivate here why it is beneficial to use caller's virtual smallest semantics in the first place. We are only told much later (Sec 3.5) that this helps in SMT encoding. A similar comment applies to Sec 3.5: We suddenly start seeing Safety-Relaxed Semantics as a set of bulleted points "... is replaced with ...". It would certainly have helped to motivate these alternate semantics by giving an example of how it helps with SMT encoding.

Thank for the suggestion. I will make sure to amend it in the final revision.

> In Sec 4.1, a set of conditions are listed for Dynamo to be complete. Of these, conditions (a) through (d) are concretely checkable or enforceable. However, I don't think this is the case for conditions (e) and (f), which would be extremely difficult to enforce or even check.
>
> Specifically, condition (e) states that the desired annotation to the assembly program is identifiable either through search heuristics or through user-supplied and/or compiler hints. Since nothing is stated about the class of search heuristics or hints to be used, and given that the "desired" annotation is one that allows us to prove the desired refinement relation, this is equivalent to stating that Dynamo is complete on the class of translations for which it can prove the refinement relation (a tautological statement). A similar comment applies for condition (f) that requires the invariant inference procedure to identify just the right invariants at the right nodes of the network.

Both the annotation identification algorithm and the invariant inference algorithm can be made more powerful through user-provided or externally generated untrusted hints. For example, our prototype implementation uses hints from an instrumented Clang/LLVM for annotations (section 6.1.4). Similar enrichment is possible for invariant inference where more powerful invariant inference techniques can be utilized for identifying richer invariants.

The Dynamo algorithm allows replacement of these two components (annotation identification and invariant inference) with potentially more powerful substitutes which in turn can enable it to prove refinement across more transformations (conversely, using less powerful substitutes may cause it to fail across certain transformations). This parametric property is the primary reason points (e) and (f) sound almost tautological.

I must also mention that (f) is identical to the completeness requirement in CoVaC [48] (page 12) and (e) is included as an extrapolation of (f) to the guessing required by our algorithm.

[48] Anna Zaks and Amir Pnueli. "CoVaC: Compiler Validation by Program Analysis of the Cross-Product". In: Proceedings of the 15th International

Symposium on Formal Methods. FM '08. Turku, Finland: Springer-Verlag, 2008, pp. 35–51. isbn: 978-3-540-68235-6. doi: 10.1007/978-3-540-68237-0_5.

> In the Dynamo algorithm, is there a specific reason that cut-points are considered in reverse post-order?

Quoting from second paragraph on page 102: The paths are enumerated in reverse postorder (i.e., the source and sink nodes are enumerated in reverse postorder) so that annotation (if any) over a path from q to q^t is performed before the address sets (potentially modified by annotation) are used in successor instructions of q^t. This property ensures consistency in the invariant network and enables incremental building of the product program X.

> How is the parameter μ chosen when invoking correlatedPathsInCOpts()? Is it a number chosen once and for all, or does this change (dynamically determined) for each translation validation run? Is μ inferred automatically using some heuristics, or is it something for which we must depend on the user?

μ is provided as a parameter to the algorithm by the user and can be different for each run.

> Algorithm asmAnnotOpts (Algorithm 4) lies at the heart of Dynamo. It appears from the presentation that this is currently implemented as a set of very effective heuristics. Is this true, or is there some notion of completeness of the algorithm (asmAnnotOpts), as presented?

In the blackbox setting, indeed, it is heuristic-driven and it would be hard to provide a completeness claim.

> Pg 132: The text under invariant inference actually doesn't talk anything about how the invariants are being inferred. In general, inferring the right invariants is undecidable, so how are invariants being inferred in Dynamo? I think a paragraph on this should be added.

Section 4.2 details the invariant inference scheme: The invariant inference procedure is a counterexample guided algorithm described in previous work [12].

[12] Shubhani Gupta, Abhishek Rose, and Sorav Bansal. 2020. Counterexample-guided correlation algorithm for translation validation. Proc. ACM Program. Lang. 4, OOPSLA, Article 221 (November 2020), 29 pages. https://doi.org/10.1145/3428289

> Several key steps in Dynamo involve trying out alternatives from a set of candidates, and as I see, the set of candidates can become large very soon. Is it possible to understand how large C and assembly program are likely to be amenable to the analysis by Dynamo, within reasonable time (say few tens/hundreds of minutes)? The experiments provide some estimate of the lines of C/assembly for the benchmarks that were successfully validated. It would have been wonderful to

> see (in a plot, perhaps) how the translation validation time increases with size of the code. Of course, the time depends not only on the size of the code, but on what the code really is doing. So I am aware that designing such an experiment is not easy. Yet, this would be something on my wish-list for a thesis of the high quality that this one is.

Indeed, the runtime does not seem to correlate highly with number of instructions. I would imagine a combination of a set of "features" of C and assembly programs such as number of paths to be correlated, number of allocated locals, number of procedure calls, number of loops etc. to be useful for making a "prediction" of the estimated runtime. On the other hand, the order in which the alternatives are tried also affect the runtime. For example, the number of backtrackings performed (a proxy to the number of alternatives tried) may explain why `generateMTFValues` (with ALOC 144) takes more time than `undoReversibleTransformation_fast` (with ALOC 221) – the former backtracks 16 times while the latter only 6.

I will try to include a scatter-plot of ALOC vs runtime in the final revision.

> Chapter 6 mentions that Dynamo is 400K SLOC in C/C++. Was all of this code written as part of this dissertation, or did this dissertation build on some kind of existing code base for Dynamo?

No, Dynamo builds upon an existing code base and was also used in two equally impressive dissertations:

1. "Black-box Equivalence Checking across Compiler Transformations" by Manjeet Dahiya (https://www.cse.iitd.ac.in/~dahiya/thesis.pdf)
2. "Counterexample-Guided Equivalence Checking" by Shubhani (https://sorav.compiler.ai/pubs/shubhani_thesis.pdf)

The code base comes from a decade-long project and had many contributors over the years.

> Were the benchmarks reported in Table 6.1 hand-crafted, or taken from a benchmark source? If it is the former, I am curious how it was decided that this set of benchmarks is sufficient — clearly this set doesn't cover all features of the language.

The benchmarks were hand-crafted: we tried to be as exhaustive as possible while still keeping the benchmarks small enough to be tractable. I would be curious to know which local allocation pattern we missed.

> Sec 6.2.3: What was the rationale for introducing the 01- option? How were the optimizations chosen to be disabled selected in the first place?

Some of these transformations cannot be handled by the path correlation algorithm of Dynamo ((a)), some require a richer invariant inference ((b) and (c)), and the remaining require inter-procedural reasoning ((d),(e), and (f)). The

common theme being: they make TV possible for a broader range of procedures while having reasonably low impact on the performance of the generated executable.

> Sec 6.2.4 is much appreciated. Understanding the causes of failures is a big part of understanding the strengths and limitations of Dynamo. I am a bit curious though why only bzip2 was chosen from SPEC2000. What would it take to run Dynamo on other benchmarks from SPEC2000?

bzip2 was selected because it is a well-known and useful program but is small enough to be tractable for TV. It should be possible to run Dynamo on other similarly sized benchmarks from SPEC2000. As indicated by our experiments, large procedures (>= ~140 assembly lines of code) are still very much out of reach.

> Chapter 2 is very impressive in terms of the details of modeling the execution semantics of both the unoptimized IR and the assembly instructions. The gradual development of more "permissive" refinement relations to capture increasingly faithfully the semantics of C programs with allocations, deallocations and re-orderings is very commendable. I went through the detailed modeling, and while the presentation does get tedious at times, I appreciate the rigour with which the treatment has been done. My only comment about this chapter is that the notation could perhaps have been simplified to enhance readability. The notation does come in the way of reading and understanding this chapter.
>
> I must especially commend the student for taking into account undefined and non-well-formedness conditions in an elegant way in the formulation.
>
> An overall suggestion: The presentation is extremely notation heavy, at times (superscript of subscript of superscript of ...), and this does hurt the readability of the thesis. It is hard at times to keep track of all that is being represented by such complex notation. It would have been good to simplify the notation (I am sure this could have been done) to enhance readability.

Thank you for the kind words of appreciation and the suggestions. I will try my best to incorporate them in the final revision.

Subject **Re: To Sup_Minor Revisions_Abhishek Rose_11753**
From Sorav Bansal <sbansal@cse.iitd.ac.in>
To Dean (Academics) <Dean.Academics@admin.iitd.ac.in>
Date 19-02-2025 12:36

- review_response 4.pdf(~137 KB)

Dear Prof. Kurur,

Thank you! Please find attached a response from Abhishek, duly reviewed by me.

Regards,

Sorav

On Mon, 17 Feb, 2025, 13:01 Dean (Academics), <Dean.Academics@admin.iitd.ac.in> wrote:

> Dear Prof. Bansal,
>
> Reports from all the examiners on the Ph.D. above thesis have now been received. Both examiners have asked for minor revisions in the report and desired corrections.
>
> An annotated thesis received from one of the examiners is attached for your perusal and further necessary action.
>
> A detailed response should be submitted by the candidate addressing the examiners' queries at the earliest possible. The same be forwarded by the supervisor(s) after ensuring that the examiners' concerns are addressed.
>
> With regards,
> ---
> Narayanan Kurur | deanacad@admin.iitd.ac.in
> Dean Academics | +91 11 2659 1708