Bounded Verification of Sparse Matrix Computations

Tristan, Alper, John

July 20, 2019 at 11:36am

Abstract

Show how to model and reason about BOTH structure and behavior, can model rich state and operations, sparse matrix representations and algorithms on them.

Use Alloy, a declarative modeling language with a SAT solver, automatic proofs within a scope.

Show how to perform tests for refinement and equivalence, the form of the "proof obligations" (need a better phrase).

Also introduce a new Alloy idiom for state changes, for bounded iteration and changing state. Other contributions?

1 Introduction

-sparse matrices:

Sparse matrix computations are central to many applications in scientific computing ... or say avoiding zeros by not assembling, etc., but must be dealt with. Also sparse matrices for big data, machine learning, right?

Mention common *representations*, used historically, libraries available, but also new representations being created to take advantage of new hardware characteristics (cite), and new architectures like GPUs (cite).

Mention common *operations* on sparse matrices, primarily sparse matrix vector multiplication for solving linear systems and eigenvalues, etc.

-our particular interest:

Our own interest arose when optimizing ADCIRC, a popular large-scale ocean model, trying to assure that changes preserved correctness. Long run times, testing is hard.

First use (by our group?) for subdomain modeling, can make incremental model changes and simulate at an incremental computational cost. Huge performance gains. Maybe show an image of a subdomain?

Using Alloy, a declarative language with automatic analysis, can simulate and check ... not a common application for state-based formal methods ... (predicate abstraction type things?)

Continuing work on internal algorithms, solvers (and the particular representations they require) ... e.g., ADCIRC uses itpack-v and ellpack format ... other solvers CSR

Assembly from meshes works directly on the data structures for performance, sometimes hard to ensure correct in all cases, corner cases not missed.

Alper, this is done for performance reasons, right? Simple setter/getter functions could be used to hide internal representation of a sparse matrix, but guess (?) it's difficult to create operations like that that don't compromise performance?

-scope and organization of paper:

...

2 Alloy and Bounded Verification (or "Approach")

Uses a SAT solver, automatic, no reals.

Small scope hypothesis

Figure: cube in 3 axes (Alloy) ... 3 axes and random dots (testing)

Unit testing and test driven development... aim to reveal bugs by testing individual components, incomplete and less effective at testing combinations of components

Libraries provide nice abstractions on structure and behavior, but often still require a lot of work to be done on the developer's side, and to maintain performance may not perform checks like one might assume. Great example of this here: `https://github.com/scipy/scipy/issues/8778` where we see scipy does not check the representation invariant of a csr matrix, specifically for performance reasons.

Another possibly useful example... rep invariant is maintained but the internal sorting is reversed... unintentional side effects may go unnoticed, some solvers require ordering to be maintained... `https://github.com/scipy/scipy/issues/9639`

Implicitness in spec ... Alloy "generates" what might be viewed as input/data for operations ... arbitrary sparse matrices

IMPORTANT: for numerical software, hard to perform "parts" of a large-scale simulation by simply extracting code ... have to create "valid" data for input, populate data structures ... here we say properties required of data and Alloy generates that for us ... so in some ways easier than testing a part of a large-scale program.

Alloy for structure and behavior

Not obvious for numerical software, probably.

Importance of proper specification to which we are designing? An example here of a discussion about the 'correct' behavior when storing explicit zeros in a representation: `https://github.com/scipy/scipy/issues/3343`

Structure:

- Class-like sigs for modeling rich state
- But weak support for Ints (and no reals)

On the latter, two approaches ... for some problems predicate abstaction is sufficient (cf. subdomain modeling), for others, may just need to rely on numbers being “distinct” (the latter is the case for sparse matrices):

From Jackson’s text: To figure out whether integers are necessary, ask yourself what properties are actually relied upon. For example, a communication protocol that numbers its messages may RELY ONLY ON THE NUMBERS BEING DISTINCT; or it may rely on them increasing; or perhaps even being totally ordered.

Behavior:

- Since class-like features are based on relations, we have a relational language, declarative, but it has navigation expressions (generalizing dot as relational join) that make data modeling natural, as well as first-order predicate calculus + transitive closure, a rich modeling language.

- Of course numerical programs are written in imperative languages like Fortran, C++.

**We can avoid overspecifying order of computations, and when we need an ordering can impose it ... usually simply. Alloy has an idiom, adding a “time” column to a relation. Later we describe a complementary approach that works well for matrices and bounded (needs to be bounded?) iteration.

3 Sparse Matrix Representations in Alloy

Basic representation of zero and non-zero values.

$$Value = \{Zero, Value_0, Value_1, \ldots, Value_n\}$$

Also, a diagram showing inheritance...Value is a signature, Zero is an extension of Value.

Signatures for CSR (just, or also ELL?).

Hoare checks of an ADT for correctness (or earlier?) ... sound ... commuting diagram (general, figure) ... math expression for the refinement check (here, or next section?)

Abstraction function and representation invariant, define what the terms mean and present them for CSR.

4 Sparse Matrix Operations in Alloy

The three methods we’ve used to model structured loops... one simple one and two more complex that introduce time-varying state:

(1) set comprehensions for simple cases that are easier to reason about – “en masse”,

(2) method used in ellcsr that makes use of the ’some’ quantifier to populate a sequence of loop values (kpos) – for semi-structured looping needing time-varying state...

(3) method of using a signature to represent internal loop state, for cases in which internal looping structure is more complicated (time-varying state again), arrays are incrementally modified, as in transpose...

Description of each method with a simple example, maybe the counting of even numbers models?

Now the following sections make use of each of these methods in order of growing complexity... MVM has no time-varying state, translation needs only the first time-varying state method, transpose makes use of all three methods.

Declarative languages need a way to express time-varying state.

Variety of idioms particular to the language, for lazy functional languages recursively defined infinite sequences [**?**].

5 Matrix-Vector Multiplication

SumProd, representation of expressions ... sums and products, zeros and non zeros, commutative ... rep that's easy to use and check ... a lightweight approach, not Seigel's completely general approach.

Also no time-varying state here ... again, simple ...

nature of the check ... a triangle, show in figure and in math and Alloy

6 Translation Between Formats

Ellpack to CSR

We say it in intro (?), but could elaborate on why of interest to us ... use of PARDISO and other solvers in ADCIRC that require CSR ... ADCIRC assembles into ELL ... can reimplement assembly or translate between ELL and CSR.

Other solvers we'd like to use... ITPACK and Pardiso use CSR, cuSPARSE uses BCSR, Armadillo uses CSC.

Have an algorithm from SPARSKIT, want to determine if it can be performed in-place.

Here we have time-varying state (kpos variable) ... introduce a new idiom: bounded iteration with time-varying state, has a conditional so you can't "pre-compute" values ... do this using an array + stuttering ... sort of like util/ordering in Alloy, but variable is an array indexed by "time" ticks.

7 Matrix Transpose

Algorithm in pseudo-code and definition in Alloy.

Sould be simple? (in the sense of an "en masse" description without state updates ... no need for "time varying" state).

Abstract transpose is "en masse", CSR is not quite as simple. Looking like all three methods described above will be used...

Show the nature of the refinement check in Alloy.

After we check this, can point out it's really CSC, define the abstraction function for CSC, and do another check with its abstraction function.

Boundary Conditions in ADCIRC? (probably not)

Documentation says symmetric (we checked it) ...

We thought the zeroing of rows was incorrectly modifying values that need to be saved before the zeroing of columns occurs... turns out that is not that case and the order is correct.

We also thought columns weren't supposed to be zeroed out, and they are ... can check, we did. (I know Alper already knows rows and columns are both zeroed out).

8 Dicussion ... or "Testing"?

What kinds of bugs can be found ... "mutation" testing?

I wonder to what extent our Alloy models can catch off-by-one or other indexing problems. Is it possible some get missed? I mean, for transpose, for instance, an error would have to show up in the corresponding abstract matrix (via alpha, the abstraction function) for our approach to "notice" it. Since Alloy predicates are "total" (they produce sort of "null" results, empty sets, when things go wrong), the effects may be "visible" in the abstract matrix, or maybe some can be missed ... maybe?

9 Related Work

a. Arnold - Isabelle/HOL ... "full functional correctness" (a side goal), looking at compilers and optimization, created a little functional language

from paper:

Before settling on the design presented in this paper, we set as our goal the full functional verification of imperative sparse code, in the style presented in Section 2.1. However, even the simple CSR format turned out to be rather overwhelming. We attempted to verify its correctness in multiple ways: (i) manually with Hoare-style logic, both with first-order predicates and inductive predicates; (ii) with ESC/Java [6]; (iii) with TVLA [13]; and (iv) using a SAT-based BOUNDED MODEL CHECKER. The results were unsatisfactory either because it took weeks to develop the necessary invariants (i, ii), the abstraction was too complex for us to manage (iii), or BECAUSE THE CHECKER SCALED POORLY (iv). Eventually, we concluded that we needed to verify sparse codes at a higher level of abstraction (and separately compile the verified code into efficient low-level code). Turning our attention to functional programs allowed us to replace explicit loops over arrays with maps and a few fixed reductions over lists, which in turn simplified the formulation and encapsulation of inductive invariants.

Rule reuse was significant, but still requires knowledge of lambda-calculus. Does not support modeling of assembly process.

b. Kotlyar -

A relational approach to the compilation of sparse matrix programs

[?]

From Stodghill's dissertation

A RELATIONAL APPROACH TO THE AUTOMATIC GENERATION OF SEQUENTIAL SPARSE MA-

TRIX CODES

The idea of using viewing sparse computations as relational queries from which joins could be discovered and scheduled using query optimization was first suggested by Kotlyar [81].

Libraries are one approach to alleviating the user s burden but often libraries do not give the user sufficient control over algorithmic and storage format decisions Also the natural abstraction layer between the user s code and the library code often serves as a barrier to important policy and optimization decisions It seems that instead of having the sparse implementation provided a priori what is needed is a tool that will generate portions of the sparse implementation from the dense specification . We call such a tool a sparse compiler.

c. Siegal - expression tables for checking equivalence (types of equivalence: Herbrand, IEEE, and real)

it's not sparse matrices, the symbolic handling relevant

d. Our work

No other sparse matrix work of ours, but any use of Alloy is relevant, would like to include the KeYmaera X work if we can say why.

Alper's KeYmaera X work

A little more about subdomain modeling?

other past work?

10 Conclusions

and future work:

a. newer, more complex formats, hybrid formats

"A hybrid format for better performance of sparse matrix-vector multiplication on a GPU" (cite) (I like this article because it shows that improvements to formats are still being made, and they're becoming more complex, I guess. Also, the perspective on ELL and CSR is good. By Gropp.

b. include mesh representations in Alloy to model assembly, to optimize

c. complexities of GPU problems