Leveraging Intermediate Forms for Analysis

Andrew Reiter, Ayal Spitz and Jared Carlson

Veracode Research
Intrepid Pursuits

Analysis via Formal Methods

- What are Formal Methods?
- Why should they be used?
- Who should be using them? FM & LLVM Community…
- Our Approach to integrating FM into LLVM…

What are Formal Methods?

- Formal Methods are a set of techniques used to construct and/or verify a mathematical model of a system, in this case a software system…
- Hoare Logic, 1969, formal set of logical rules to reason about computer programs.

The Basics Start in *101

- The idea of a particle is a model - in reality there's no such thing.
- Many of the engineering approximations we use in day to day constructions are derived from more generalized physics; Maxwell's equations -> Ohm's Law; F = m*a (is only true for constant mass, and is a simplification of the Hamiltonian).

Some Physics Examples

$$\oint_c E \bullet dl = \frac{d}{dt}\int_s B \bullet ds \rightarrow V = iR$$

Electromagnetic equations leads to circuit analysis

$$H = \sum_i \frac{p_i^2}{2m_i} + \sum_{i<j} V(r_i - r_j)$$

$$\frac{dp_i}{dt} = \frac{\partial H}{\partial r_i}; \frac{dx_i}{dt} = \frac{\partial H}{\partial p_i}; \rightarrow F = m \cdot a$$

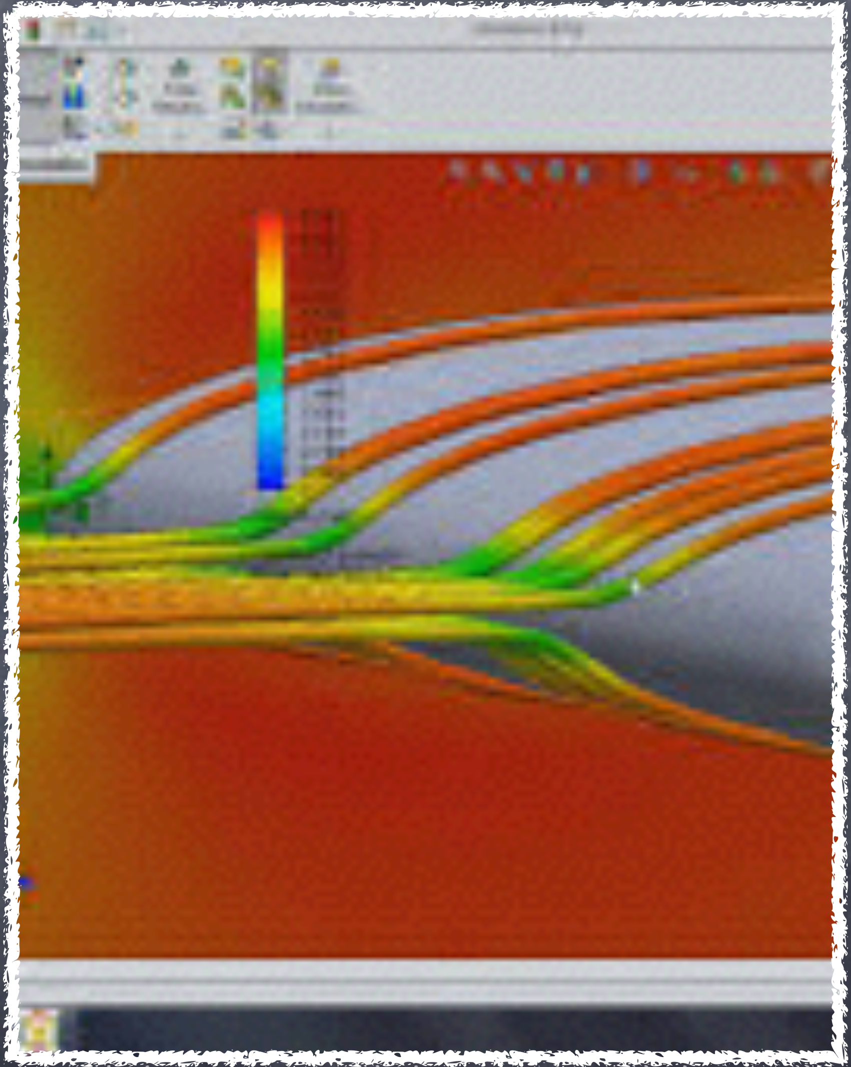
Hamiltonian leads to every day usage (and the common simplification) of Newton's Law.

Formal Methods in Engineering

- in ME, EE, AE, etc, we dont use the term "Formal Methods"; instead we have "Free Body Diagrams", instead of a verifier, we have CAD, FE Solvers, etc.
- These tools all "model" the underlying physics and engineering approximations.

- Pencil/Paper or whiteboard
- CAD
- Numerical Simulation
- 3-D print or small prototyping for scaled tests
- Build it!

- Pencil/Paper or whiteboard
- Alloy
- Coq
- tests - logic/unit/etc.
- Deploy!



Software ought to have more modeling uses. The tools were a bit lacking but these days are actually quite good - see Kathleen Fischer (USENIX 2015)

In Software…

- In software a model is usually thought of in terms of MVC (Model View Controller) paradigms, where the engineer is separating the components.
- This isn't a true "model" as the M is written in code and <u>actually</u> implemented.

Formal Methods

- Fundamentally Software can be approached via logic - the mathematical underpinnings…
- In physics we are usually concerned with the states of time, position, velocity…
- In software, we're concerned with state information, as defined by available types to the program.

- In other words, we're generally replacing differential equations (equations of motion, electromagnets, differential geometry, etc.) with logical operations such as joins, unions, disjunction, and so forth. Replacing calculations with Calculational Logic (see R. Blackhouse for text examples).
- A few quick examples:
    - Disjunction… Basically an OR statement; but DNF is fundamental in analysis for proving theorems (DNF, each variable only appears once in every clause).
    - floor(x) can be defined for all float x, the answer is an integer such that n <= x; this means n <= floor(x); (float)n <=x; implying floor(x) <= x; demanding that "floor" always rounds down.

- Assignment: Given a pre and post conditions for assignment it can be possible to calculate an appropriate assignment statement.
- Suppose s and n satisfy s=n^2 and we want to maintain this while incrementing n (n++); pre s=n^2; post s = (n+j)^2 and holds for all j (1, ...). Then s is incremented by 2*n + 1.
- While this is fairly trivial these calculations are much more reliable than an educated guess, which is often implemented. How often have you seen "Oh I just did..." and then later on...

Formal Methods Generally

- In Software, foundational uses of logic allow us to transfer code or pseudo code into a form for modeling…
- This could be "find" a model, a means a prototyping via logic
- Or this could be "checking" a model, involving the construction of appropriate tests, evaluation of code, etc.

FM Jargon 101

- Invariants - true condition for duration of the program.
- Intervals

- Abstract Domains, polyhedra..

- Fixed Point Iteration - convergence
- Linear constraints - defines boundaries/limits
- and so on....

FM and LLVM

- Samples of LLVM & FM: VeLLVM, IKOS, SMACK, along with others.. So folks have been and are continuing to work here!
- LLVM IR is not ideal for FM as a standalone but it's not meant to! :<) But it's a great starter..

Opportunities

- Moving beyond IR, LLVM is modular and has so many tools for development that FM ought to move into this space.
- Specific tools for FM space should be developed and become part of the community, especially as LLVM continues to grow in the embedded space.

ϕ Issues?

- ϕ nodes; these represent a partial disjunction over at least some variables. For non-relational domains (think: intervals) this is fine, but for relational abstractions (think: polyhedra) which want to describe properties over all program variables this is "very challenging"
- We haven't seen anyone "daring" enough to really take this on.

Instruction Sets?

- Certain instructions can also represent challenges, as complex instructions are ideally regularized (simplified).
- As an example, a complex pointer arithmetic operation (gep) is replaced by pointer shifting (pshift).

IR Control Flow

- Conditional branch instructions can pose a problem where invariants might cross over basic blocks (propagation) for the branches.
- Typically analysis would desire abstract domains that are independent as possible.

Who's Doing What in FM?

- Formal Methods are increasingly used everywhere (but this is still a minority).
- Critical systems are the most common uses. NASA, NIST, DARPA, other government uses for infrastructure and so on in rapidly developing interesting technology and use cases.
- Facebook uses Confer, attempting to bridge the gap of FM and modern development life cycle.
- In Industry this has been gaining favor for a while as well. MSFT invested heavily in FM and greatly reduced the "blue screen of death" via SMACK.

VeLLVM

- VeLLVM (UPenn), created some verifiable LLVM passes.
- Formalized semantics of IR, for example, the `undef` value and intentional underspecification.
- Extracted an interpreter from formal semantics.

SeaHorn

- Takes program and generates IR for verification.
- Inline code, seahorn_assert(...), assume(...).
- Only linear constraints, interval domains...

IKOS

- Developed @ NASA by the formal methods group. NASA is very concerned with reliability issues in software
- Inference Kernel, generic operations for analysis provided.
- Example has an LLVM front-end.

Other Intermediate Forms: CIL

- Attempts to stay close to C in a “clean” representation.
- High level representation, attempting to retain the higher level information that is often encapsulated in source.
- Simplified branching, etc, are core concepts.
- Obvious issue is if you’re doing something outside of C…

Intermediate Forms: BAP

- If you saw the DARPA Grand Cyber Challenge, BAP (Binary Analysis Platform) was an essential component.
- Carnegie Mellon's entry (Mayhem) used BAP for automated security analysis.
- Uses an IL (Intermediate Language) but is often lifted to SSA form (per LLVM) for analysis passes, etc - BAP has LLVM bindings.

Intermediate Forms: AR

- AR (Abstract Representation) NASA AR is our choice. Replaced ϕ nodes are replaced with assignments, pointer arithmetic is simplified, etc.
- CFG based representation of the program is essential for domain construction.

Example

- Ok, let's do a "simple" example of applying FM...

Lattice Boltzmann Method

- LBM is a gas dynamics method for solving hydrodynamic equations.
- Based on the Boltzmann distribution, it's a rare - time dependent - physical model.
- Because of our emphasis on realizing models we should mention that this very strong theoretical physics model has some poor assumptions (only binary collisions!), but it still very successful (generally carefully constructed).

LBM Code ="simple"

```
  // compute density and velocity from the f's
void computeMacros(double* f, double* rho, double* ux, double* uy) {
    double upperLine  = f[2] + f[5] + f[6];
    double mediumLine = f[0] + f[1] + f[3];
    double lowerLine  = f[4] + f[7] + f[8];
    *rho = upperLine + mediumLine + lowerLine;
    *ux  = (f[1] + f[5] + f[8] - (f[3] + f[6] + f[7]))/(*rho);
    *uy  = (upperLine - lowerLine)/(*rho);
}

  // compute local equilibrium from rho and u
double computeEquilibrium(int iPop, double rho,
                          double ux, double uy, double uSqr)
{
    double c_u = c[iPop][0]*ux + c[iPop][1]*uy;
    return rho * t[iPop] * (
               1. + 3.*c_u + 4.5*c_u*c_u - 1.5*uSqr
           );
}

  // bgk collision term
void bgk(double* fPop, void* selfData) {
    double omega = *((double*)selfData);
    double rho, ux, uy;
    computeMacros(fPop, &rho, &ux, &uy);
    double uSqr = ux*ux+uy*uy;
    int iPop;
    for(iPop=0; iPop<9; ++iPop) {
        fPop[iPop] *= (1-omega);
        fPop[iPop] += omega * computeEquilibrium (
                                      iPop, rho, ux, uy, uSqr );
    }
}
```

In practice, we verify via the Physics!

- Conservation of mass, momentum, energy are relatively simple checks to ensure the calculations are correct.

$$\int f \, d\vec{v} = \rho = \sum_i f_i$$ mass

$$\int_c \vec{u} \cdot f \cdot d\vec{v} = \rho\vec{u} = \sum_i u_i f_i$$ momentum

- Notice these give us good pre/post conditions as discussed earlier. They look slightly different but really aren't that complicated.
- Our pre ∑f and post ∑f are the check to verify correct assignments.
- In this we are taking graduate level physics, and determining that simple checks for code quality can be constructed using assignment conditions at critical sections.
- Alloy (Daniel Jackson @MIT, especially) makes this excellent point - that FM doesn't need to be heavy, light-weight FM used well is a great choice!

This Brought us to Alloy

- Alloy is a first order (primarily) logic tool to find a model.
- Simple codes such as this embody complicated ideas in the form of simple code that we'd like to check under certain conditions (special cases).
- Alloy is an MIT initiative for light-weight FM. Nice, clear language that allows for a variety of solvers to be used. Easy to use, distributed as a JAR

Disadvantages

- We want to interact with the actual code! This isn't possible with Alloy as-is...
- Model finding AND checking is the goal...

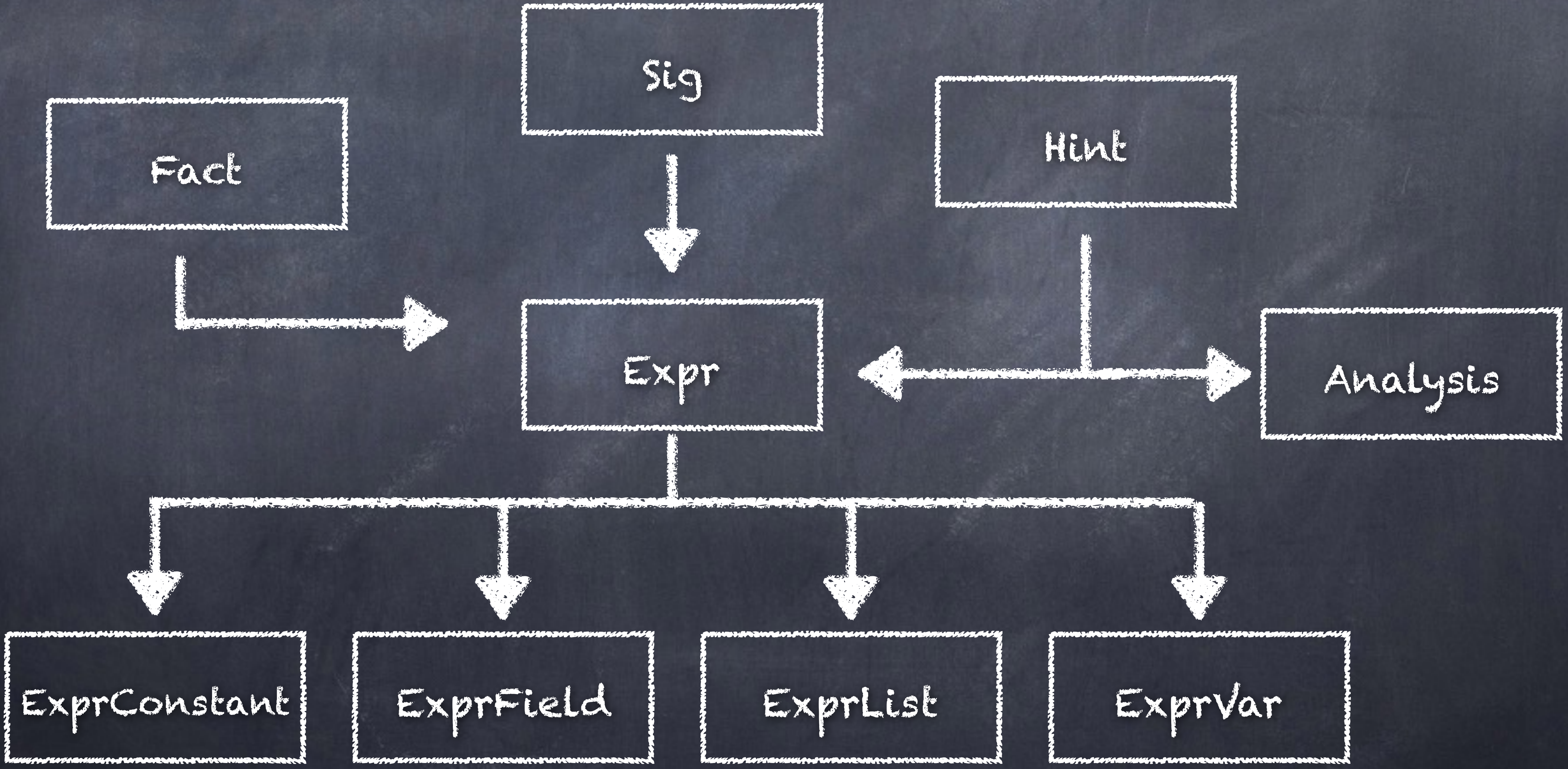
Code...

- So we started by importing a subset of the Alloy grammar. Instead of Java, using C+ + (mainly as we want to interface with C++ based tools).
- Created an interface to Z3 for a SAT solver (mirrored off of the very excellent project - "Souper").
- So this starts to find models...

But, Alloy Allows Exploration

- Our hacking does NOT reproduce one of the most essential features of Alloy - the ability to interactively find and explore models!
- Large caveat but… sometimes you can only do what you can do…

AST (snippet)


Sig
Fact
Hint
Expr
Analysis
ExprConstant
ExprField
ExprList
ExprVar

LBM Alloy Model

```
sig Lattice { many Node }

sig Node {

   rho: lone Float,

   p: lone Float

}

....
```

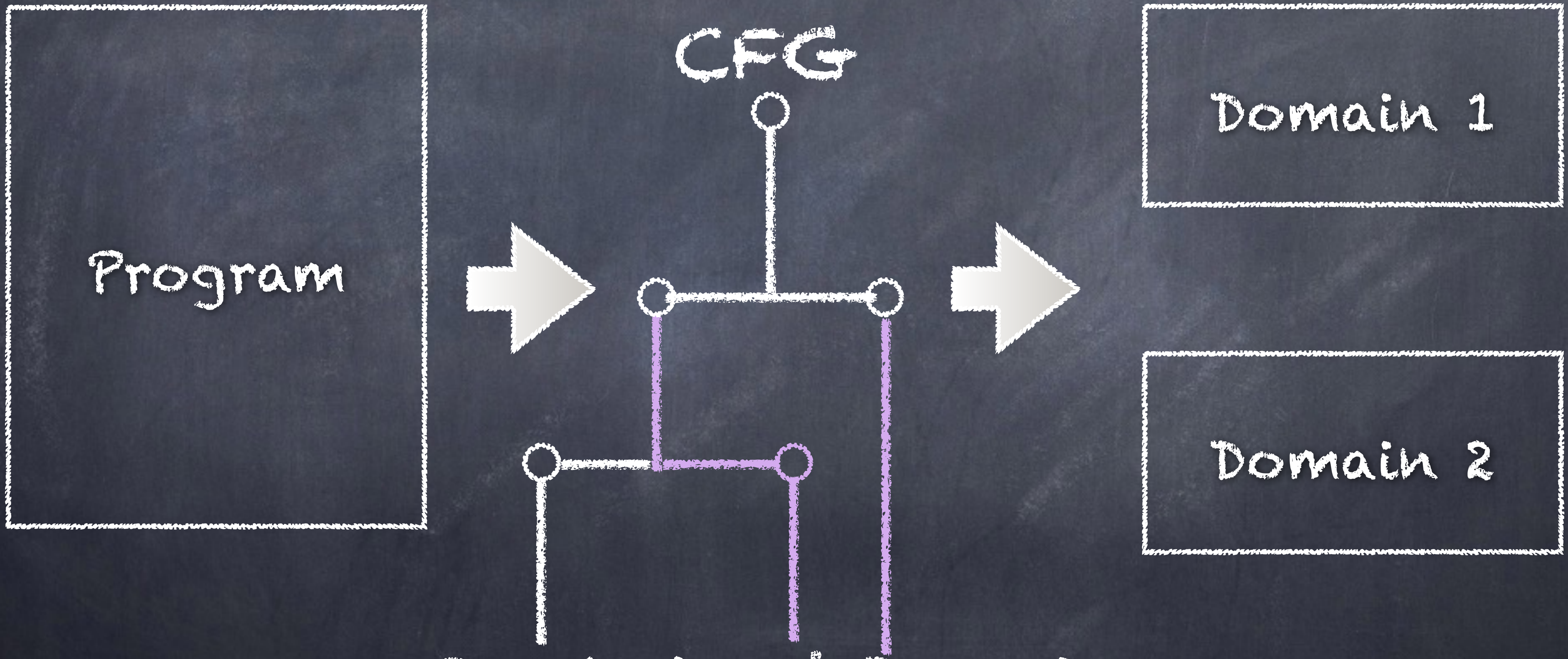
Annotations

```
  // compute density and velocity from the f's
__attribute__((annotate("calculation"))) void computeMacros(double* f, double* rho,
double* ux, double* uy) {
    double upperLine  = f[2] + f[5] + f[6];
    double mediumLine = f[0] + f[1] + f[3];
    double lowerLine  = f[4] + f[7] + f[8];
    *rho = upperLine + mediumLine + lowerLine;
    *ux  = (f[1] + f[5] + f[8] - (f[3] + f[6] + f[7]))/(*rho);
    *uy  = (upperLine - lowerLine)/(*rho);
}

  // compute local equilibrium from rho and u
double computeEquilibrium(int iPop, double rho,
                          double ux, double uy, double uSqr)
{
    double c_u = c[iPop][0]*ux + c[iPop][1]*uy;
    return rho * t[iPop] * (
               1. + 3.*c_u + 4.5*c_u*c_u - 1.5*uSqr
           );
}

  // bgk collision term
__attribute__((annotate("collision"))) void bgk(double* fPop, void* selfData) {
    double omega = *((double*)selfData);
    double rho, ux, uy;
    computeMacros(fPop, &rho, &ux, &uy);
    double uSqr = ux*ux+uy*uy;
    int iPop;
    for(iPop=0; iPop<9; ++iPop) {
        fPop[iPop] *= (1-omega);
        fPop[iPop] += omega * computeEquilibrium (
                                  iPop, rho, ux, uy, uSqr );
    }
}
```

Automated Pass Analysis

- Annotations are a nice “trick” to leverage to automatically link to a pass.
- Our first pass matches the IR Metadata to Alloy-spec.
- From there we generate AR using an additional pass for model queries.
- IR can be used for symbolic execution, other means to verify a specific state.

Slicing & Dicing



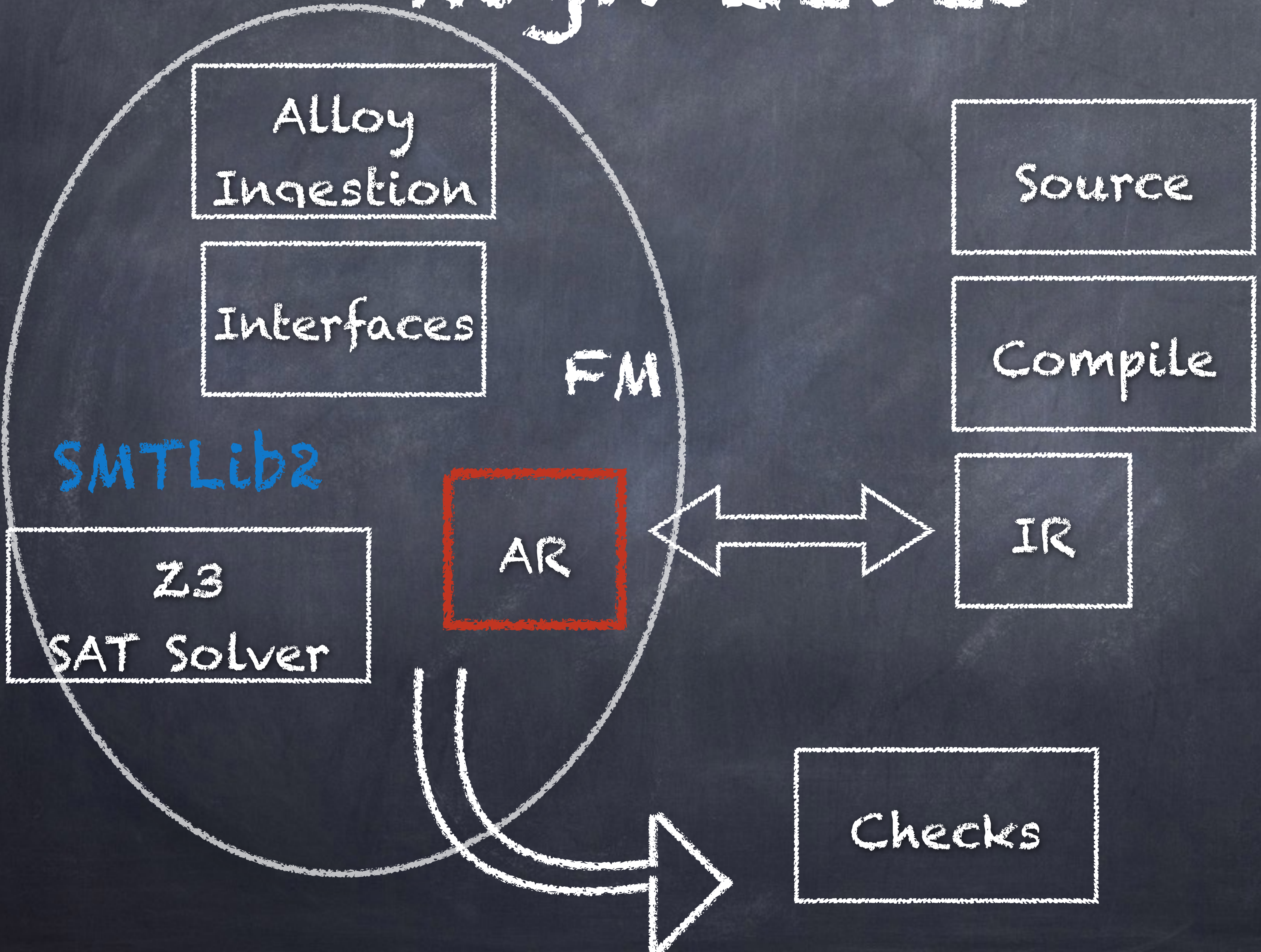
Break Apart Domains
based on CFG dependencies

Verification and Analysis

- LLVM Passes (perhaps a bit obvious, but maybe wrong?) leveraging IKOS*
- In LBM, the code (and CFG) is quite simple. Extraction from IR to AR is reasonably pleasant (so more to prove later).
- AR to IR (Transformation + Insertion) allows for various forms of checking.

* We have a colleague investigating more advanced toolsets (similar to IKOS - also from NASA; some of the same folks)

High Level


Alloy Ingestion
Interfaces
FM
SMTLib2
Z3 SAT Solver
AR
Source
Compile
IR
Checks

Diff's

- Converting to/from constraints AR is non-trivial. This example works nicely due to its simplicity.
- For non-trivial logical assumptions a great deal more of work is required.
- Once we can have the AR, because of its resemblance to IR inconsistencies are a bit easier.

Model Lang. Summary

- For now we've chosen to weakly (buggy) support Alloy functions and commands.
- Mainly this is due to resources but also because we want to be able to inject analysis directly to the AR processing, (think inline), and that's a more original contribution.

Disadvantages

- Our analysis run as passes and generally any optimizations that are part of the compilation chain are not verified as our approach tangentially.
- We're not analyzing everything, just critical code sections.
- If we fail, we currently don't have a good mechanism to inform for necessary changes - this is BIG by the way… (Similar to clang's ability to suggest a solution, we need a similar 'Diagnostics Engine' for FM we believe).

Results

- We can diff the found model vs. the resultant implementation.
- Also can inject code for sym. execution of the resultant implementation subject to assumptions.
- Moving beyond the simple, explaining results isn't all that easy... We need much friendlier errors!

Conclusion

- We used a modeling language (well, we hijacked one that we like) to find a model.
- Used this to enforce certain conditions, here, physics conservation laws for special cases (i.e. no energy source BC).
- Interact with the code, but leave the source minimally altered (ideally). Trying to separate the higher and lower logical ideas.
- Transformation at Intermediate forms allow exchange of the high/low level information for analysis. Ideally however the Representation for FM is VERY easily translated from LLVM IR.
- Wanted to tie this to the development toolchain…

Lot more work to be done here

- Growing field, an exciting field, especially for those mathematically inclined.
- Tools need to be integrated in our opinion.
- LLVM is a good place to continue work. But while we'd like to see LLVM incorporate more FM, FM should likely fit the LLVM "style" (usability especially).
- Lots of separate players - it was easy for us to talk to folks at DARPA, harder to find the right person at NASA for example. Different goals as well… (likely part of the problem).

What We're Up To...

- Think about maybe another layer for translations? SIL for Swift; maybe an IL to ease logic/FM translations to and from IR?
- Lots more work to evaluate backend tools...
- Error messages, translating modeling logic to code logic? Or maybe this is wrong and it should be in the code?
- Talk to FM folks and argue they NEED to work with compiler and tools groups!