30 Years of Abstract Interpretation

Thomas *Jaudon* Ball
Microsoft Research

[This is the text of a 20 minute talk I gave at the 30 Years of Abstract Interpretation Workshop in San Francisco, California, USA on January 9, 2008. The subsections below are the names of the slides in my talk.]

Declarations of 1776 and 1789

France and the United States have a rich history of shared ideals and principles. As evidence, I present to you that in 1776, the founding fathers of the United States of America signed the Declaration of Independence. With this document as a model, in 1789 the French Assembly published the Declaration of the Rights of Man and of the Citizen. Like the Declaration of Independence, the French Declaration of 1789 comprised a statement of principles rather than a constitution with legal effect. Just think about it: over two centuries ago, the founding fathers and the French Assembly appreciated the value of a good specification! Needless to say, the two countries diverged quite a bit when it came to implementation.

1977

Fast forward to January 1977 which I believe is the date we should be celebrating today. What happened? Well, according to Wikipedia, hell froze over – that is, snow fell in Miami, Florida (for the only time in its history); a peanut farmer from Georgia became the 39th President of the United States; Gary Gilmore was executed by firing squad in Utah after the reintroduction of the death penalty in this country.

Declaration of 1977

On a happier note, not yet commemorated by Wikipedia, January 1977 also witnessed the French declaration of Abstract Interpretation, presented by the Cousots at POPL 31 years ago and 380 miles south in Los Angeles, California. This declaration gave the world a new way to talk about program analysis. It presented a statement of principles, a specification, if you will, to guide the systematic design of program analysis. In my humble opinion, the principles of Abstract Interpretation presented in 1977 were three. The designer should:

- *compare* interpretations to one another, to establish the correctness and precision of interpretations;
- *compose* interpretations (using an algebraic approach);
- *celebrate* infinity!

We'll examine each of these three points in the rest of the talk.

Intellectual Influences

As we're starting off with a historical perspective, let's briefly mention influential figures from which Patrick undoubtedly drew inspiration:

- **Alan Turing**: the father of modern computer science described the limitations of computation via the Turing machine and the Halting Problem. Central to the Halting Problem is an infinite tape for computation and Turing machines that interpret input Turing machines. Abstract Interpretation addresses program analysis problems as hard as the Halting Problem, but without abandoning infinite precision in abstractions. This is, in my opinion, a most revolutionary idea. (This is a critical difference from the field of finite state model checking, for example.)

- **Évariste Galois**: a brilliant French mathematician laid the foundations for Galois Theory and Galois Connections, which form the basis for relating interpretations to one another. He died at the age of twenty, presumably in a duel over a woman. Very French.
- **Alfred Tarski**. One of the giants of modern mathematics and logic: among his many contributions was a formalization of logical truth, model theory, algebraic logic, and, of course, the existence of fixpoints.
- **Dana Scott**: Tarski student, Turing award winner and with Strachey, responsible for the lattice-theoretic, fixpoint semantics of programming languages that is the basis of modern interpretations of programs.

All in the Family (1977)

Now, nearly half the papers at POPL 1977 were about compiler construction: parsing, code generation, optimization, and dataflow analysis. Into this community, came the Cousots. Thanks to Kildall, the two communities shared a common lattice-theoretic concept of interpretation. But the motivations of the communities were quite different, which led them in different directions, theoretically and pragmatically. Even today, many in the dataflow analysis community think of program analysis as requiring finite-height lattices. Let's move on to consider key principles of Abstract interpretation: the comparison of interpretations, and composition of interpretations, and the celebration of infinity.

Compare

Here we see a very nice slide of Patrick's which I've reproduced by hand. It very concisely illustrates the idea of comparing interpretations. In the upper left, we have a set of integer points in the plane, which might represent the true behavior of a program. In the upper right, we overapproximate this set using the interval abstraction. Now this abstraction contains the original set, as well as many other points, which might represent false alarms. If there are too many false alarms, we might refine this abstraction using a conjunction of octagonal inequalities, which is strictly more powerful than intervals. In the lower right, we see an even more refined domain, the polyhedral domain, which is a conjunction of general linear inequalities. This concept of abstraction refinement is key to the design of program analyses, especially for verification and defect detection tools.

Diplomacy

Now, as I was pondering this key principle of Abstract Interpretation, that of comparing interpretations, I was struck by the similarity of this idea to another great French invention. Let me see: **comparing interpretations** leads us to **interpretation relations**, sounds sort of like **inter-nation relation**, **international relations**, and rhymes with **diplomacy**! Yes, we also have the French to thank for the practice and promulgation in the 1700s of diplomacy, defined by Webster as "skill in handling affairs without arousing hostility". Every French schoolchild must learn about Charles Maurice de Talleyrand considered one of the most skilled diplomats of all time and one of the authors of the Declaration of the Rights of Man. And here we have... well, Patrick does share many traits with Talleyrand: he's a great conversationalist, gourmand, and wine connoisseur! But alas, Patrick may not be a Talleyrand.

Patrick's Theme

Somehow I always feel like I am always at the losing end of a duel whenever I discuss program analysis with Patrick. It's like Patrick has a rifle and I am wielding a water pistol. These interactions often remind me of a song – would you like to know the name of the song? It was immortalized by a singer shown here. No, it's not **Carla Bruni**. It's Ethel Merman. The song is "Anything you can do, I can do better" from Irving Berlin's "Annie Get Your Gun". But this is not quite right. I think that the song would be better titled "Anything you can do, I can specify better". Abstract Interpretation really is a superior framework for designing our program analyses. Specify what the abstraction is, demonstrate the connections to other abstractions through comparison, decompose your abstraction into simpler ones, and use expressive (even infinite domains) abstract domains. Specify before you implement!

Compose

Let me now give an account of how Abstract Interpretation helped us to better understand the abstraction that the SLAM analysis engine computes. In May of 2000, fresh from visiting the Cousots, Andreas Podelski, ambassador of Abstract Interpretation, arrived at Microsoft Research. This was just after Sriram and I had completed the initial technical report that described the SLAM analysis process, which includes converting a C program into a Boolean program. Andreas asked us a very simple question: "what abstraction does SLAM compute?" Our answer was pretty poor: "we implemented an algorithm and proved it correct". Andreas was not very happy with the response.

Boolean (Predicate) Abstraction

Andreas said: "Look, we can formalize a Boolean abstraction very simply. A set of program states maps to a set of bit vectors of length n, each bit representing the evaluation of a predicate in a state. Likewise, a set of bit vectors maps to a set of program states. This mapping is a Galois connection. Now that we have this mapping relating sets of states and sets of bit-vectors, Abstract Interpretation gives us an ideal best transformer. Does the SLAM algorithm compute a transformer that is a best transformer?"

Apply Ideal

Let's look at an example of the best transformer in practice. Here we have a set of three predicates and a simple assignment statement. Now consider the set containing the bit vector <0,0,0>. What set of states does this represent? The set of states in which x is equal to 5 and y is not equal to 5. The strongest post-condition with respect to this set of states is the set of states in which y is not equal to 5 and x is equal to y. Abstracting this to the Boolean world yields two-bit vectors, one representing the set of states in which x is greater than 5 and y is not equal to 5, and one represents the set of states in which x is less than 5 and y is not equal to 5.

Apply SLAM

Now, let's look at the transformer computed by SLAM. SLAM generates a parallel assignment to update the three Boolean variables corresponding to the three predicates. Note that the assignment havocs the first two variables. Thus, given the input set containing the bit vector 0,0,0, the Boolean transformer will result in a set containing four bit vectors. So, SLAM's transformer is not the best transformer! But what is it doing?

Cartesian Abstraction

So, how can we formalize SLAM's abstraction? Andreas observed that the parallel assignment implies we are doing a Cartesian abstraction of the Boolean abstraction. Given a set of bit vectors, a Cartesian abstraction approximates this set by a single tri-vector, as shown here.

SLAM Abstraction

In the end, we showed that the abstraction that SLAM computes is the composition of two abstractions: a Cartesian abstraction of a Boolean abstraction.   That is, it is an overapproximation of the ideal Boolean abstraction. Conceptually, SLAM operates over tri-vectors. Its transformer is theoretically equivalent to converting a tri-vector to a set of bit vectors, converting the set of bit vectors to a set of program states, applying the strongest postcondition, mapping the states back to a set of bit-vectors and then taking the Cartesian abstraction of that set.

The high-level take away: Sriram and I made an implementation decision that forced us down a particular abstraction path. Using Abstract Interpretation, Andreas Podelski helped us uncover what the underlying abstraction was. Why is this important? Of course, we gained a better understanding by decomposing a monolithic abstraction into the functional composition of two simpler abstractions. More importantly, for defect detection tools such as SLAM, when you get false alarms you need to find the source of imprecision. If you don't even know what your abstraction is, this will be a difficult task.

Celebrate Infinity!

Now, as I mentioned before, Patrick is quite the gourmand. In fact, if you've been to dine with Patrick, you may have witnessed him walk into the kitchen to converse with the chef. Now, what is going on here?

I submit to you that Patrick's aim is to establish whether or not the chef has an infinite lattice from which he creates new recipes. Now, certainly we only have a finite number of ingredients in the kitchen from which to create recipes. But a great chef senses the infinite possibilities and is able to create a new dish that no one has ever tasted before.

In the same way, Abstract Interpretation tells us to celebrate infinite abstractions. Fundamentally, the design of an abstraction should be guided by the problem we are trying to solve. The abstraction should be expressive enough to capture the property of interest. Later, we will try to figure out how to compute with the abstraction.

Convex Polyhedra

A tour de force of abstract interpretation by Patrick and Nicolas Halbwachs in 1978 demonstrated the power of infinite abstract domains for the problem of analyzing buffer overflow.

First, a little personal history about the subject: In 1975, my cousin Ben Wegbreit published a paper about discovering linear equalities in programs. Building on Wegbreit's work, which required finite ascending chains, Karr presented a complete solution for discovering linear equalities.

Looking back thirty years, it really is amazing what we have in the 1978 paper: a clear exposition of why finite ascending chains are insufficient for precisely capturing linear inequalities; an exponential time algorithm for maintaining convex polyhedra. If 1977 delivered the theory and principles, 1978 really delivered the goods. Unfortunately, the machines of the time really were not up to the task. And, needless to say, buffer overflows had yet to lead to exploits costing the software industry billions of dollars. Zhe Yang from MSR says of the work:

> "When I started the ESPx project back in 2003, the benchmark for buffer overflow analysis was Nurit [Dor] et al's work, which was heavily based on the Cousot & Halbwachs 1978 paper on

> polyhedra. What I learned in Cousot & Halbwachs's seminal work became the reference point for the design of the abstraction in the espX work. I was amazed by how the geometric interpretation in the polyhedral work elegantly gives rise to the power of discovering invariances."

For those of you who aren't aware, espX is an analyzer regularly run over millions of lines of Microsoft source code to find buffer overflows.

Impact

In the end, the innovations are judged by their impact. How does Abstract Interpretation measure up? I think that after 30 year, it is very clear. The theoretic bedrock and design principles of Abstract Interpretation remain with us, having been elaborated and expanded upon by many here. This theory has led to many interesting abstractions and algorithms. Finally, these abstractions and algorithms have been realized in tools that have been put into use to effectively address software reliability issues.

At Microsoft, many of the tools we have created and deployed at scale owe much to Abstract Interpretation.

Finally, I think that based on the representation here, we can see that Abstract Interpretation truly is a global research endeavor that shines a bright light all around the earth. Thank you.