COMPUTER SCIENCE

200 Terabyte Proof Demonstrates the Potential of Brute-Force Math

MICHAEL BYRNE

Jul 30 2017, 3:00pm

$$\frac{xy \doteq x_1y_1 + z_1 \wedge y_1 \geq 0 \vdash xy \doteq x_1y_1 + z_1 \wedge y_1 \geq 0, y_1 \ jmod\ 2 \doteq 0}{xy \doteq x_1y_1 + z_1 \wedge y_1 \geq 0 \vdash xy \doteq 2x_1\lfloor y_1/2 \rfloor + z_1 + x_1 \wedge y_1 \geq 0, y_1 \ jmod\ 2 \doteq 0}$$

$$\frac{xy \doteq x_1y_1 + z_1 \wedge y_1 \geq 0 \vdash xy \doteq 2x_1\lfloor y_1/2 \rfloor + z_1 + x_1 \wedge \lfloor y_1/2 \rfloor \geq 0, y_1 \ jmod\ 2 \doteq 0}{1 + z_1 \wedge y_1 \geq 0 \vdash \{x_0 := 2x_1 || y_0 := y_1 \ jdiv\ 2 || z_0 := z_1 + x_1\}[](xy \doteq x_0y_0 + z_0 \wedge y_0 \geq$$

$$\frac{y_1 \geq 0 \vdash \{x_0 := x_1 || y_0 := y_1 \ jdiv\ 2 || z_0 := z_1 + x_1\}[\mathtt{x\_0 = x\_0 * 2;}](xy \doteq x_0y_0 + z_0 \wedge y}{y_1 \geq 0 \vdash \{x_0 := x_1 || y_0 := y_1 \ jdiv\ 2 || z_0 := z_1 + x_1\}[\mathtt{x\_0 = x\_0 * 2;}](xy \doteq x_0y_0 + z_0 \wedge y$$

$$\frac{\vdash \{x_0 := x_1 || y_0 := y_1 \ jdiv\ 2 || z_0 := z_1\}[\mathtt{z\_0 = z\_0 + x\_0; x\_0 = x\_0 * 2}](xy \doteq x_0y_0 +}{\{x_0 := x_1 || y_0 := y_1 || z_0 := z_1\}[\mathtt{y\_0 = y\_0/2; z\_0 = z\_0 + x\_0; x\_0 = x\_0 * 2;}](xy \doteq x$$

$$\frac{\ldots\} \ \mathtt{else} \ \{\ldots\}](xy \doteq x_0y_0 + z_0 \wedge y_0 \geq 0)}{= 0\}[\mathtt{while(y\_0 > 0)\{\ldots\}\ldots}](result \doteq xy)}$$

$$\frac{= y\}[\mathtt{int\ z\_0 = 0;\ldots}](result \doteq xy)}{\mathtt{x\_0, y\_0})](result \doteq xy)}$$

$$\frac{\mathtt{(x\_0, y\_0)}](result \doteq xy)}{\mathtt{o(x\_0, y\_0)}](result \doteq xy)}$$

Wikipedia

Automated verification finds renewed potential for making algorithms safe.

In computer science, the "brute force" solution is usually the suboptimal solution. It gets there eventually, but also inefficiently and without cleverness. The existence of a brute force solution to a problem usually implies the existence of a more elegant but perhaps less obvious solution.

You could take a basic algebra problem as an example: $2x + 100 = 500$. To solve this with brute force, we simply check every possible value of x until one works. We know, however, that it is far more efficient to rearrange the given equation using algebraic rules and in just two computations we get an answer.

Computer scientists are giving brute force another look, however. In a paper published in the current *Communications of the ACM*, Marijn Heule and Oliver Kullmann argue that we're entering a new era where brute force may have a key role to play after all, particularly when it comes to security- and safety-critical systems. This is thanks to a newish technology called Satisfiability (SAT) solving, which is a method of generating proofs in propositional logic.

Basically, we can put together optimized brute-force problem solving and accessible modern supercomputers and get a reasonable way of solving super-complex problems. Technology is making it so that we don't necessarily *need* cleverness in problem solving if we have access to a whole bunch of processor cores.

"Today, SAT solving on high-performance computing systems enables us to conquer problems of high complexity, driven by practice," Heule and Kullmann write. "This combination of enormous computational power with 'magical brute force' can now solve very hard combinatorial problems, as well as proving safety of systems such as railways."

In propositional logic, a proposition is just a statement that can be true or false. Crucially, such a statement has the property of being able to be *negated*. The statement "I am going to get a burrito," for example, in negated form is just "I am not going to get a burrito." So, the statement is a proposition. Propositional logic involves a whole lot proving that statements are true using logical operators such as AND and OR and NOT. As in my algebra example above, really complicated-seeming statements can often be rearranged into statements whose truth value is obvious.

Computer science involves a lot of propositions, or true/false statements. That's the essence of the whole thing, really. In formal propositional logic, we can imagine a bunch of individual atomic units of truth (that are either true or false combined using the above logical operators. For example, we might have the statement (*(true AND false) OR true) AND NOT (true OR false)* and be asked if that's ultimately true or not. It's not, but that's probably not obvious. These things get really hard to evaluate and if you've ever had to wrestle with formal logic you know that reaching a solution sometimes just involves guessing and checking, which is a brute-force method.

"Now the task is to live up to big complexities, and to embrace the new possibilities."

SAT is really more of a suite of approaches to brute-force problem solving. As it turns out, there are some actually clever ways of decomposing big problems—imagine the above statement but with millions of true and false clauses—by finding certain kinds of subproblems and by rooting out internal contradictions. Shortcuts, basically, but still brute

The end result of SAT solving is a proof, which is basically a tedious and exhaustive listing of steps needed to rearrange/decompose the initial statement into forms that are clear statements of truth or falsehood. As part of their research, Heule and Kullmann demonstrated an automated SAT solver that produced a 200 terabyte proof, which, as someone that's seen some shit when it comes to formal logic, makes me queasy.

As to why this matters for security and safety, it has to do with the notion of *correctness*. In computer science, this means that an algorithm meets a certain specification. It behaves as predicted and will eventually terminate. Correctness is demonstrated through the process of formal verification. Algorithms, or the statements that make up algorithms, can be reduced to propositional logic and so they can be proved correct. This is a pain in the ass, however, and as algorithms start to get complicated, formal verification gets more difficult. At the same time, algorithmic complexity makes formal verification more important. There's just more room for bugs to hide.

So, SAT methods offer a way into automated formal verification and the cheap supercomputers of 2017 offer a way into formal verification.

"Now the task is to live up to big complexities, and to embrace the new possibilities," Heule and Kullmann conclude. "Proofs must become objects for investigations, and understanding will be raised to the next level, how to find and handle them."