Why Software is Abstract, by PolR

Thursday, October 07 2010 @ 03:40 AM EDT

Contributed by: PJ

Why Software Is Abstract
by PolR

Following the ruling of the Supreme Court in *Bilski*, the USPTO asked, in substance, how to tell an abstract idea from an application of the idea. In this article I propose an answer to the question of what makes software abstract. It is a follow up to the previous article, Physical Aspects of Mathematics.

The logic is to look at why a mathematical calculation is abstract and then see if the same logic applies to software. It happens that it does. It is possible to show that software is abstract with references to the underlying mathematical aspects. This is not, however, the topic for this article. The argument is presented without any assumption as to whether or not software is mathematics. I work from the observation that a mathematical calculation solving a mathematical problem is abstract. Then I look at what makes it abstract. Then I observe that the exact same logic is applicable to all software whether or not the law sees it as an algorithm as defined by *Benson*. This is not surprising. Software is mathematics and this makes it abstract, but I don't use or rely on this fact in making the arguments in this article.

Abstraction Is Self-Containment

Let's start with experts, who can define for us what makes mathematics abstract, in some writings on the psychology of learning. See Mitchelmore, Michael and White, Paul — *Abstraction in Mathematics and Mathematical Learning* (where you can download the article as PDF)

> We claim that the essence of abstraction in mathematics is that mathematics is self-contained: An abstract mathematical object takes its meaning only from the system within which it is defined. Certainly abstraction in mathematics at all levels includes ignoring certain features and highlighting others, as Sierpinska emphasises. But it is crucial that the new objects be related to each other in a consistent system which can be operated on without reference to their previous

> meaning. Thus, self-containment is paramount.

These authors are university researchers specializing in the psychology of learning mathematics with a focus on the role of abstraction. They are authorities on the issue of what is an abstract idea when the psychology of learning mathematics is concerned.

Consider a calculation written down on paper. Let's say you have 128 boxes each containing 48 cans of peas. How many cans do you have? It is 128 × 48 = 6144 cans.

This is an arithmetical computation. What is the difference between this and computing 128 × 48 in the abstract without referring to boxes, cans, and peas? Does the computation change if it is cans of mushrooms instead? The difference is purely semantic. The computation is not changed. The written numbers do not record such information. The arithmetic process, the physical manipulation of the digits, is the same no matter what the written symbols represent in the real world.

This is why arithmetic is abstract. Mathematical calculations are self-contained in the sense that they are defined entirely within mathematics, independent of the real-world meaning of the numbers. This independence is the abstraction that makes mathematics an abstract idea.

This same idea of self-containment applies to computers for all type of computations, not only the ones restricted to computations about numbers. During the actual computation the real world meaning of the bits is not stored with the bits. Some mathematical meaning is hardwired in the computer. Boolean and numerical interpretations of bits are understood by the computer instruction sets. This is all the meaning computers can understand. When the computer carries out the computation any information about real-world meaning is unused and physically absent from the computer. Only the raw bits are present.

Abstractions may come in layers, each one being located at its own self-contained level. For example a particular program may be written in a programming language with an elaborate type structure that retains some information about real-world usage. For example, numbers may be tagged with the unit like inches and kilograms so we know what the number is used for. This source language layer of abstraction is self-contained. The information the program is able to understand is defined in terms of the capabilities of the programming language.

Then there is a layer closer to the hardware where the operations must be the machine implemented instructions. Unless the hardware has been designed especially for this purpose the elaborate types of the programming language are not recognized at this level. This degree of abstraction is also self-contained.

The activity of the computer is self-contained, then, in the sense given by Mitchelmore and White. The activity on the bits is entirely defined within the architecture of the computer with no reference to the real-world meaning. This means all computations made by the computer are abstract in the same manner that mathematical calculations are abstract.

To confirm that this is true, please examine the list of instructions from reference material like The Art of Assembly Language Programming. All the instructions a CPU is physically capable of are described in such reference material. None of the instructions ascribe or require any real-world meaning. The computer neither knows nor needs to know whether it adds oranges or apples. A requirement to understand the real-world interpretation of the bits will prevent computer execution because no computer has such capability[1].

An anonymous user replying to the previous article mentioned a related issue which is worth repeating here.

> This leads to my second point, that of severability and intent - which was triggered by your analysis of *Alappat*. The four circuits in that case _were_ severed, as you pointed out. As such, they were a general purpose computing device; general in the sense that they could be used for any purpose where the mathematical operation they embodied was useful. On the other hand, a mechanical analogue fire control computer is very limited -- it's pretty hard to think of another use, other than fire-control, for such a device. When someone tries to assert claims under various tests by tying a program to a GP computer, these parts are clearly severable.

Self-containment is not the ability to think of another use for the same abstraction or the same device. It is not being general purpose as opposed to being specific purpose in the sense of this comment. Imagine for example a complex mathematical model for climatology. It will be very hard to think of another use of these specialized mathematical formulas[2]. But they are still mathematical formulas and abstract in the self-containment sense. If we build a special purpose computer to carry this computation it would still be self-contained, even though this computer might have no other use than climatology.

The Irrelevance of Real-World Meaning

I understand that case law pays attention to the real-world meaning of symbols. I am not a lawyer, obviously, but from what I've read, patent law seems to care when the patent covers a real-world application, because this is one of the factors that help determine when the claimed subject matter is patentable. But failure to factor into the analysis the irrelevance of real-world meaning to the execution of an algorithm will bring inconsistencies. For an example, see *In re Alappat* where the meaning of the numbers was used to determine that the contested patent passed the useful, concrete and tangible result test.

As another example, this article on Patently O discussed *Ex Parte Gutta*, at a test to determine whether a machine that involves a mathematical algorithm is patentable subject matter:

> The BPAI's test for a claimed machine (or article of manufacture) involving a mathematical algorithm asks two questions. If the a claim fails either part of the two-prong inquiry, then the claim is unpatentable as not directed to patent eligible subject matter.
>
> 1. Is the claim limited to a tangible practical application, in which the mathematical algorithm is applied, that results in a real-world use (e.g., “not a mere field-of-use label having no significance”)?
>
> 2. Is the claim limited so as to not encompass substantially all practical applications of the mathematical algorithm either “in all fields” of use of the algorithm or even in “only one field?”

I don't know whether this test is still in use in the post-*Bilski* world. But still its analysis is instructive and relevant to this discussion.

Both prongs of the inquiry look at the purpose of the claim in terms of practical applications. In a scenario where the claimed machine does nothing but execute a computer program, this test depends exclusively on information that is both unused during the actual execution of the program and absent from the machine as it carries out the computation. There are scenarios, then, where the same machine can simultaneously be non-patentable subject matter and subject to several independent patents because this test is based on information that is irrelevant to its intended purpose. The outcome of the test depends on the purpose of the user and not on the technology he is patenting. This outcome may and will vary from user to user.

This observation applies to special purpose computers where the computation is carried out by application specific circuits. It also applies to software to the extent the courts believe that loading software in memory makes a new potentially patentable machine. Finally, although the BPAI test is applicable to machine patents, the observation would also apply to a similar test for process patents where the process is defined by the step-by-step execution of the computation on the bits.

If the law uses an inquiry that looks at the purpose of the claim in terms of practical applications to determine whether or not the claim is patentable or not because it's an abstract idea, where the claimed machine does nothing but execute a computer program, this inquiry would depend on information that is both unused during the actual execution of the program and absent from the machine as it carries out the computation. Depending on how that actual test is defined, there could be scenarios where the same machine can be simultaneously non-patentable subject matter and subject to several independent patents because the inquiry is based on information that is irrelevant to the actual technology. In this hypothetical scenario the outcome of the inquiry depends on the purpose of the user and not on the technology he is patenting. This outcome may and will vary from user to user.

This observation applies to special purpose computers where the computation is carried out by application-specific circuits. It also applies to software to the extent the courts believe that loading software in memory makes a new potentially patentable machine. Finally the observation would also apply to a similar test for process patents where the process is defined by the step-by-step execution of the computation on the bits.

Let me give an example. Suppose someone patents a method to draw the shape of a parabolic antenna for radio applications and drafts it as a machine patent on the computer running the program.

This is a practical application. But the program nowhere uses the information that it is the shape of an antenna that is being drawn[3]. The program is drawing a tridimensional geometric shape that will be displayed on screen or printed on paper. The fact that this shape is the shape of an antenna is totally irrelevant to the execution of the algorithm and has no effect on the structure of the computer that executes it. It is only when the output is produced that the human viewers will interpret the shape as an antenna.

Then along comes an engineer in the field of acoustics that has a use for a parabolic object. He may patent a method to draw his object and draft it as a machine patent. This likewise is a practical application. But both objects have the same shape and are computed by the same method. The very same computer can be used for both purposes. The code will be identical if the programmer so desires. In such a scenario, would a second patent on the same software be granted because a different application is contemplated? In such case when a user uses the software the patent that will be applicable will depend on the user's purpose and not on the software itself.

Then here comes a mathematician in the field of solid geometry, looking for a program to draw parabolic 3D objects. The very same code will work just fine but solid geometry is theoretical mathematical work and not a tangible practical application that leads to real-world use. For the purposes of this mathematician, the same software is not patentable subject matter.

In all three scenarios the "invention" is exactly the same. It is the same computer running the same software implementing the same algorithm and executing the same machine instructions. But this kind of inquiry gives different answers in each scenario. This indefiniteness is what we get if we ignore the abstraction inherent in software and give undue consideration to the real-world meaning of the bits.

Consider *Diamond v. Diehr*. The patent in that case involved an algorithm for computing the time required to cure rubber. But the patented process isn't content to just print the answer. It actually cures the rubber. It is the actual curing that separates this industrial process from a mathematical process because there is a functional relationship with the actual rubber. Such a relationship wouldn't occur if the process were just printing an answer.

In *Parker v. Flook*, the patent uses a mathematical algorithm to determine conditions that require human attention. An alarm is sounded where these conditions occur. An audible alarm is output.[4] Its purpose is to convey information. It *means* a condition occurred. It *answers* the question of whether or not something is worthy of attention. It doesn't *act* in the real world beyond communicating the information. The *Flook* patent covers a mathematical algorithm because the functional relationship with the required intervention is not being claimed.

In the antenna example, printing the shape of an antenna is output. The line between expressing and making a physical real-world use of the meaning is crossed when the antenna is built. Printing or displaying the shape of an antenna is a mathematical algorithm. Building an antenna is not.

Where Are the Boundaries of an Algorithm With the Real World?

So far we have gathered sufficient facts to answer two questions:

1. Why is software abstract?

2. In a process involving software, where are the boundaries of its abstract

parts?

We know the answer to the first question. It is the independence of the bits from their real-world meaning that makes software abstract. Let's answer the second, looking again at a mathematical example.

Consider a practical example. Here is a two-step process to compute the circumference of a circle with a precision of four decimal places:

1. Measure the diameter of the circle with a measuring instrument precise to four decimal places,

2. Multiply the diameter by 3.1416. The result is the circumference.

This is abstract because there is no correlation with the real world at all. The calculation is self-contained. A circle is a geometric figure and the arithmetic operations work on numbers.

What if we alter the process to make it a process to compute the circumference of a bottle? Would it make it more concrete?

Step b) is certainly not more concrete. A mathematical computation doesn't depend on the real-world meaning of the numbers. The fact that the diameter is the diameter of a bottle doesn't change anything about this step. The difference is in step a) because the measurement is now limited to measuring a bottle.

When a computation is done, the answer may have practical applications. For instance, the diameter may be used to compute the quantity of paint needed to paint a line around the bottle. This will involve some more abstract mathematical calculations, but eventually you put the exact quantity of paint in your bottle painting machine for the quantity of bottles you have. When you reach this point your activity is no longer independent of the real-world meaning. You are back into the concrete world.

From this example I infer the answers to the second question. In a process involving mathematics, its abstract mathematical part starts at the initial input, when the real world is described in mathematical terms by measurement or otherwise. It stops after the final output where the answer is provided, at the point when its meaning is acted upon.

In the actual process there is a flow of information. This flow starts at the data gathering step where information about the real world is collected. This information is represented with a mathematical object like a number or a collection of bits. This is the input. Then some mathematical operations occur. An answer in the form of another mathematical object is produced and communicated. This is the output. The information flow stops when the information is received by a party, human or device, who interprets the information to act upon its meaning. Everything that happens between the data gathering and the acting upon the meaning is abstract.

This logic is stated in terms of mathematical calculations. It applies also to abstract bits independently of whether or not the problem being solved is a mathematical problem in the sense of *Benson*. At the initial input when the real world is represented with bits, the abstract part of software begins. When the answer is acted upon after the final output the abstract software part has ended.

Would this not tell us where to draw the line between an abstract algorithm and an application of one? A court would know where are the boundaries of the algorithm and why it is abstract. The court should be able to work out how the law applies to a specific patent from this knowledge.

This analysis correlates well with the printed matter doctrine, or more exactly the notion of a functional relationship between the information and the substrate. Here is how it goes:

- When the real-world data is initially gathered it is input recorded on some media, be it a piece of paper, a USB key, a CD-ROM, a hard disk etc. Then it is printed matter with no functional relationship with the substrate.

- When the real-world data isn't initially gathered at the computer it must be brought to it somehow. This could be for example by communication over a network, an input connection or insertion of the media in a reader device. Again the real-world data is printed matter with no relationship with the substrate.

- When the data is processed by the computer the real-world meaning of the data has no functional relationship with the computer, because the computer never uses this meaning. This is why the computer program is abstract, and it is part of why the program is a mathematical algorithm.

- When the answer is produced, it is bits recorded in memory. Again this is printed matter with no functional relationship with the substrate.

- The answer may undergo some more steps. It may be communicated over a network or recorded on some storage device. If the answer is text or an image, it could be printed or displayed. If it is sound or video, it could be played back. In all scenarios, this is output. Again this is printed matter with no functional relationship with the substrate.

- Once the answer is interpreted and the real-world meaning is acted upon

> (as opposed to mere communication) then a functional relationship with the real world is found.

This analysis focuses exclusively on the real-world meaning of the data, ignoring its mathematical meaning. The functional relationship between computers and the mathematical meaning of the raw bits depends on the mathematically exact definition of algorithm and how this definition relates to actual computers.

In the entire chain of events there is no point where the real-world meaning has a functional relationship with the underlying computer processing until the end when a functional relationship with the meaning occurs[5]. Therefore, we should ask: do we have in the claim a legally significant element outside of this chain? If there is none, the claim is abstract.

This view is putting the emphasis on the claimed subject matter. Activity which is not explicitly claimed does not change a mathematical algorithm into a non-mathematical process. The mere presence of a real-world problem does not suffice to make software patentable. We also need to ask how this problem is solved. Is the real-world solution actually claimed? Or is it information processing that will eventually lead to the solution?

Consider the anti-lock brake system in a car. There is an embedded computer controlling the brakes. The software by itself is abstract. The apparatus formed by loading the software on the embedded computer is also abstract because it is entirely enclosed within the boundaries that delimit the abstract portion of the invention. But if we look at the entire system then the brakes are outside these boundaries. The brake system taken as a whole is not abstract.

This analysis implies that when the real-world problem requires nothing but the production of information from already known information the resulting patent claim is always abstract. Computers work in such manner that software solutions to these problems are always abstract. For example consider a database of scientific research on the chemistry of rubber with a search function to help an engineer locate a scientific article. We may say this is a real-world problem in the field of chemical engineering. The computer will never use the real-world meaning of the database content because it can't. It is all pure database algorithms that rely only on the raw bits. Like the formula to compute the circumference of a bottle, the solution to this database problem is abstract, because the processing of the bits is self-contained.

An Observation On Business Processes

This analysis also brings a partial answer to a question asked by the USPTO in their request for comments.

> The decision in Bilski suggested that it might be possible to "defin[e] a narrower category or class of patent applications that claim to instruct how business should be conducted," such that the category itself would be unpatentable as "an attempt to patent abstract ideas." *Bilski* slip op. at 12. Do any such "categories" exist? If so, how does the category itself represent an "attempt to patent abstract ideas?"

My answer would be this: Any process made entirely of information processing steps with no activity that acts upon the information is non-patentable because it is abstract. Communication of an answer is not enough to make a business process not abstract. A significant physical use of the real-world meaning of the information is required. This is true when all agents executing the process are human beings because then all steps are mental steps. If such a process is adapted to be implemented in whole or in part on computers or computer networks, then it remains non-patentable because no non-abstract element is introduced in the adaptation.

The Limitations of Self-Containment in Identifying Mathematical Algorithms

There is an implicit assumption that is pervasive in the whole article. It is that the meaning of the bits are external to the computer. What if the patent claim doesn't meet this assumption?

Suppose some computer engineer includes in the design of his CPU instructions to perform binary-to-BCD conversions using the Benson algorithm. Is this abstract? If the instructions are implemented in microcode this is software and I suppose the *Benson* Supreme Court precedent should be applicable. But what if the patent claims it as the making of a better CPU, calling it a method to set the bits to their right value in the registers? Can we say that the setting of the bits is a physical activity that acts on the bits instead of merely referring to their interpretation?

This question speaks directly to the issue that was the topic of the previous answer to the USPTO, Physical Aspects of Mathematics. Mathematical activity needs a physical representation in order to be carried out. Therefore the question as to whether a patent is on an abstract idea can be split into (at least) two separate questions.

1. Whether the patent is on abstract information about a real-world application or about the real thing.

2. Whether the patent is on a physical representation of mathematics.

This relates to the two directions of modeling that were discussed in *Physical*

Aspects of Mathematics.

1. Mathematics is used to provide a model of the physical reality. This is what happens, for example, when we use mathematics to describe the laws of physics.

2. A physical device of process is used to carry out an abstract mathematical operation. This is what happens, for example, when we use a calculator to carry out a calculation.

Self-containment is good at identifying the first type of issues. It is effective at telling whether the patent is on an abstract computation on abstract bits used to represent the external world as opposed to the real thing. This approach has the merit to work without having to get into the complexities of defining and identifying the notion of mathematical algorithm.

Self-containment by itself does not suffice to answer the second type of issues because they require recognizing when a mathematical calculation is done by physical means. This is not an inquiry of whether the actual processing is done at a level of abstraction different from the real-world application.

References Cases

Bilski v. Kappos The Supreme Court decision

Diamond v. Diehr, 450 U.S. 175, 182 (1981)

Gottschalk v. Benson, 409 U.S. 63, 71-72 (1972)

In re Alappat, U.S. Court of Appeals Federal Circuit, 33 F.3d 1526 July 29, 1994

In re Bilski the Court of Appeals for the Federal Circuit decision

In re Gulak, 703 F.2d 1381

Parker v. Flook 437 U.S. 584 (1978)

Footnotes

1 This bit level degree of abstraction may also be independent from actual hardware references. It is frequent that the computer is emulated in software. In such case the instructions are executed on foreign hardware that has no native ability to execute them. The universe of reference that defines the

software is not the actual hardware. It is its abstract specifications.

2 Perhaps someone with more imagination or a better understanding of climatology than I can find a way to reuse these mathematical equations. This is a feature of defining “abstract” as the inability to reuse. This is a definition that depends on the amount of knowledge and ability to imagine of human beings. It is always possible that some people will find creative reuse where others can't. This information may be present in descriptions of the algorithm like patent claims. But when writing the actual code this information does not translate into computer instructions. It has only documentation value. The instructions only depend on the abstract geometric properties of the antenna and not on the fact that it is an antenna.

3 This information may be present in descriptions of the algorithm like patent claims. But when writing the actual code this information does not translate into computer instructions. It has only documentation value. The instructions only depend on the abstract geometric properties of the antenna and not on the fact that it is an antenna.

4 In engineering the definition of symbols is recognizable physical states that can be used to represent information. A sound emitted vs no sound emitted is a pair of recognizable physical states. This is a pair of symbols able to convey a Yes/No type of information: whether or not something needs human attention.

5 This is applicable to analog information such as sound and video. When the inputs and outputs of a computer do nothing but a conversion between the analog and digital domains the boundaries of the abstract part of the process may extend across the analog portion until the points where the information is initially captured or eventually used. This is because even in analog format the information is self-contained, independent from the substrate carrier. I believe this too fits well with the printed matter doctrine because it is my understanding that this doctrine is applicable to analog information. For example the TV show that is aired is not patentable subject matter while equipment using radio-frequencies for television broadcasting is patentable.

557 comments

---

http://www.groklaw.net/article.php?story=20101007030644178