Constraint Satisfaction Problems

Prof. Deepak Khemani

Department of Computer Science

Indian Institute of Technology – Madras

Module - 1, Lecture - 04

Let's look at one more example of CSP and in fact this is an application and it kind of illustrates the power of using constraints. The area that I want to talk about is called model based diagnosis.

So the basic task that we are looking at is diagnosis essentially. And when we are talking about diagnosis here, we are talking about devices. We are not talking about human beings. So it's not as if what is wrong with the human being because this is based on creating a model of whatever you are diagnosing and so far, at least we have not made models of human beings though with progress in computation biology, who knows? I think maybe in ten years we may be able to create good models. But for the moment we will assume that we are working with devices. So things like physical devices that you're operating with.

And we want to look at an approach which is a fundamental approach. So there are two kinds of things that you can do for diagnosis. One is kind of experience based, memory based kind of a thing where you say that I accumulate experience and then I know that for example if my bicycle is not working then this must be the problem or if my speaker is making some crackling noise then this must be the problem and that kind of thing. So that is different. That is kind of memory and experience. What we are talking about here is by first principles, by fundamental principles that you create a model of the system and then use that model to reason with why it is not showing the expected behaviour.

This is also called consistency based diagnosis. And you might feel that this has something to do with logic. And in fact logic and constraints have a lot in common essentially. So when you are dealing with logic, for example, you might do something called inference and in constraints the equivalent thing is called propagation. So we saw some examples, for example, the cryptarithmetic puzzle or the Waltz algorithm which does Huffman Clowes labelling. They essentially do propagation that if you know something about one variable in the case of constraints then you can propagate that to a related variable essentially. And the

key common thing about both these processes is that these are local. You look at only a part of the data essentially. So you don't look at the entire data. Whether it's a knowledge based or whether it's a constraint satisfaction problem propagation or inference is local in nature. And that's what kind of makes them similar. And we also saw that, for example, making an inference can be seen as constraint satisfaction or constraint propagation essentially. So both the communities, logic community and constraint community would lay a claim to model based diagnosis or consistency based diagnosis.

So what is the basic idea behind consistency based diagnosis? It's that what assumptions or you might say what inferences make the observations. Here you are looking at the behaviour of a system and somehow the behaviour is not that is predicted essentially by the model. And now you are trying to reason and figure out as to what assumptions that you can make that make the observations consistent. And when I say assumptions, I basically mean about the term that we can use is broken components. So essentially we are trying to figure out if this component was broken then the behaviour that I'm seeing can be explained essentially. So that's the whole idea.

Now in model based diagnosis or in component based diagnosis, the behaviour that you are observing can be computed by looking at, or I should say device behaviour, by structure plus component behaviour. So the way that the model based community works is that you describe each component that goes into the device. Then you define the structure which is imposed when you connect components together. So there may be one component c1 here, another component c2 here, and another component c3 here.

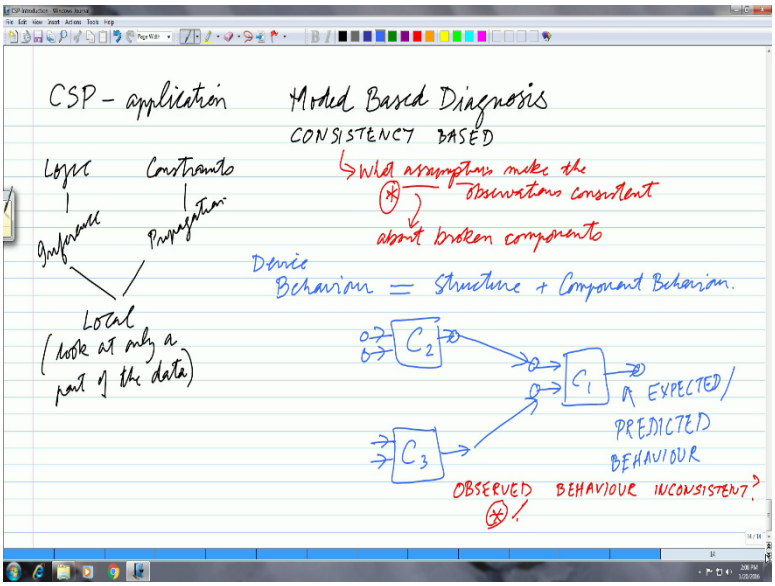
So you describe the behaviour of each component in terms of input and output. So what is the input it would take and what is the output it creates essentially. So you can talk about an adder, or a half adder, or an and gate, or a not gate or any such device. So let's say we are talking about electronic things. I have seen this applied to physical systems. So for example cars or braking systems and so on and some people I know who are in technical university in Munich have been working on building diagnosis systems for vehicles essentially. So you know you can ask the kind of questions that when I press the brake the car is swerving to the left. Then they have a system which will allow you to reason over this and try to explain why that is happening.

So you have components, in this example there are three components and then there is connectivity. So you give this output to this and maybe this output to this and so on and so

forth. So your structure plus component knowledge will give you the behaviour of the system essentially. So what you get here is predicted behaviour, or you might say expected behaviour. We will look at an example to make this idea a little bit clearer. So essentially once you create a model, and the model means structure plus components, then you can say that if I give this inputs to my device, then I should get this output. So you can compute what is the predicted behaviour essentially. Now if the behaviour that you see, if the observed behaviour is different, if the observed behaviour is inconsistent then you look for these assumptions that we spoke about that will explain or make it consistent.

So the simple approach that we are going to take here is that it's only the components which can go faulty. So components have terminals. So for example this component c2 has 2 input terminals and 1 output terminal. Likewise, c1 has 2 input terminals and 1 output terminal and connections are between terminals. So we will assume the connections don't go faulty. I mean obviously in real life circuits can go faulty because some connection is not working but in this approach we will try to just talk about components going faulty. So essentially we want to see how to model components so that we can reason about them.

(Refer Slide Time 9:42)



So we will model components using constraints because after all the output of the component is constrained by the input of the component. So if it's an adder, for example, then you know that the output should be the sum of the inputs essentially. So we can express this using constraints. But what I want to now highlight is how we bring this notion of consistency here.

So let's start with a simple thing which is an and gate. It takes two inputs and computes the output. So let's say the inputs are a and b and the output is o and let's call the gate g. Now first let's describe the consistent behaviour of the and gate which means that you have a kind of truth table with a, b and o. In this if you're thinking of this as domains, as constraints, then you can see that the domain of a = domain of b = domain of o. These are the three variables and the domain is simply 1 and 0.

And we'll also talk about the gate g and we'll say the domain of gate is equal to ok or Broken. You could have said ok or Not ok. Basically two values. Or you could have even said 1, 0.

So we have these four variables to model this component. Three of them are actually talking about the component and the fourth one is in some sense a meta variable which says that this device okay or not okay. So first we'll describe the normal behaviour and then we'll describe the faulty behaviour and together that will be a description of the component behaviour.

So what is a normal behaviour of this? If you see 1 and 1 as input, then output should be 1. If you give 1 and 0 as input, the output should be 0. If you give 0 and 1 as input, the output should be 0. If you give 0 and 0, the output should be 0. So that's a normal behaviour.

Anything which is not normal here would be faulty. So let's also describe the faulty behaviour. So some people say that this is a fault model in some sense. Now we are actually explicitly describing what is a faulty behaviour. Other people who have worked with model based systems say that if the behaviour is not normal it must be faulty. But in this simple example both the things mean the same thing.

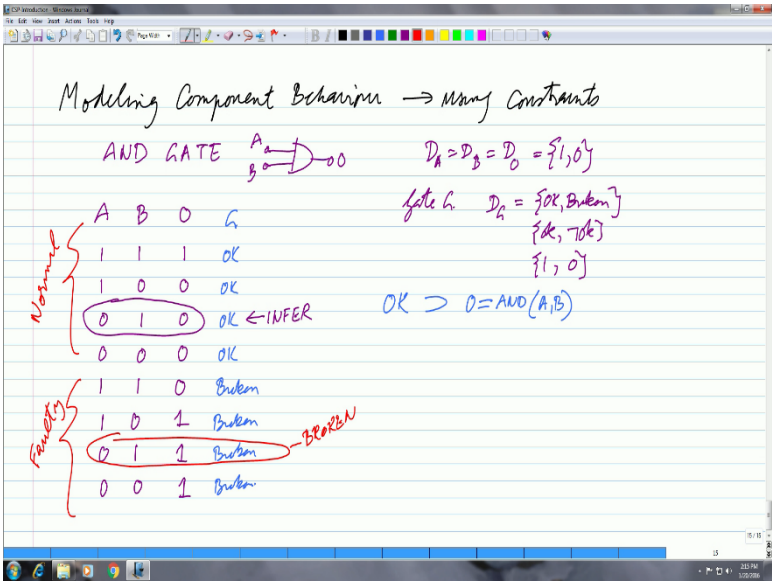
So what is a faulty behaviour? If I see 1 and 1 I see 0, its basically the opposite of what we have above. If I see 0 and 1 I see 1. If I see 0 and 0 I see 1. All these four constraints, so to speak, characterise faulty behaviour.

Now what we want to do is to build this system so that we add the health of the component as part of the behaviour. So we can do that by adding the variable g to the table. This is a constraint table. This is a now a relation between four variables a, b, o and g where g is the state of the adder.

So in some sense what we are saying here is that ok implies o is equal to and(a, b). So when we see the expected behaviour, then the device is ok otherwise it's not ok essentially. We set say 0 and 1 as inputs and 0 as output then we can infer output ok essentially. Constraint

propagation and inference is the same because the solution that will be consistent would be with 0 1 0, the fourth variable which is g can only get a value ok essentially. So in some sense you can infer ok as a value for the variable g. So g is a variable. It's simply says whether the device, the component is ok or not ok and our task is to basically figure out whether its ok or not ok and essentially this table or this set of constraints tell you which behaviours are ok and which behaviours are not ok. So you're not seeing the variable g. This is something that is part of your model. What you're only seeing is the variables a, b and o. You can see the two inputs and you can see one output and then based on that these constraints tell you whether the device is ok or not ok. Likewise, if we some other behaviour then its broken. We can infer that its broken essentially. So that's a basic idea that is used to talk about components.

(Refer Slide Time 16:30)



And then of course there is structure to components. So let me illustrate that with a very standard kind of example which has been used by everybody who talks about model based diagnosis.

So let's say you have a multiplier. So let's call it m1. We have 3 multipliers, 2 others are m2 and m3. So they must take 3 inputs. Two of them may take a common input. So let's say inputs are 2 and 3. So we are talking about adders and multipliers.

So essentially this has some tangled wiring I've drawn but essentially each multiplier is getting an input of 2 and an input of 3 which will be generated here. So you can expect that

the output generated by each of these three multipliers would be 6 with this particular input essentially.

And then you have 2 adders. a1 and a2. So adder one is taking 2 inputs. Adder two is also taking 2 inputs. And we have 2 outputs. So let's call them o1 and o2. We can give names to the terminals. And in fact it makes sense to give names to the terminals. So let me just call the product of the first multiplier p1. Another two are p2 and p 3 essentially. And there are the 6 inputs essentially.

Now each of this is modelled, I'm using slightly different notation here, m3 implies p3 and if the inputs are let's say i1, i2, i3 and i4 essentially. So m3 implies that p3 should be equal to i1 into i2.

Now of course this is a logical statement. Its either true or its false. But we can, as we saw in the previous example, we can express it as a constraint. So I'm not going to go too much into detail because we know then you have to talk about domains and things like that. If you're talking about numbers, then the domains are larger whereas when you're talking about a Boolean circuit then domains are 0 and 1 which is easier to tabulate. But anyway the basic idea is the same. So this is how you model a multiplier. You say that if the multiplier is ok then the output must be the product of the inputs essentially.

Now if you see this input, you can see that p1, p2, p3 should be 6 each and 6+6 should be 12 each essentially. But what you see here are 10 and 12. It's a very standard example. You can find it in many places. This output is not predicted.

So the first phase is fault detection. Fault detection means if you can somehow figure out that there is some fault essentially. But the more important phase is fault identification.

So where does connectivity come in here? So if I call these inputs for the adders as, so I've got four inputs so let's say they are i5, i6, i7 and i8. Structure is defined by having constraints like p1 = i8. So that simply says that it's a connector. The output of the first multiplier is going as one of the inputs to the adder essentially. So you have equations like this all over the place. And we assume that there is nothing which can go wrong with that. The only thing that can go wrong is with one of the 5 components essentially.

Now you can think about this as a problem. I will not go into the algorithms which people use but essentially what diagnosis involves is candidate generation. And there is a whole algorithm which says that you write things like this implies that. One of the set of candidates

is a1 or m1. So this is saying a1 or m1 is faulty. So you do this by looking at the structure and connectivity and say that the output of o1 is influenced by the output of adder a1 and also its influenced by adder m1 and m2 essentially.

So this is one set of candidates. It's not necessary that you talk about only one set. There can be many combinations of candidates essentially. And from the set of candidates, so if you have such sets of hypotheses or candidates, from that you have to extract which is likely to be a faulty essentially.

And so what would you say if you look at this current data? So we're just looking at a snapshot. You're looking at four inputs and you're looking at two outputs and we can see that the output of the adder a1 is not matching. So obviously the logical thing to come to a conclusion is that a1 is broken. Or you could say that m1 is broken essentially.

How could m1 be broken? For example, if it is multiplying 2 and 3 and its giving you 4 as an answer and m2 is giving you 6 and then 4+6 is giving you 10. So that's an explanation. That would make it consistent with the observations. So one of the candidates that we have is m1. So this means there was an or here. This is an intermediate representation from which people use something called the hitting set and you have to construct a hitting set and so on and that will tell you.

So what are the candidates? One candidate is a1. This is a diagnosis. Another candidate is a2 and m1. And you have actually no way of preferring one candidate over the other. So there are people who would attach probabilities here and say that multipliers are more likely to fail than adders and then in which case you say m1 is more likely than a1. But that's a separate thing all together essentially. But given this much information there is no way of distinguishing them.

So what do you do then? You take measurements. Now in a device, you can't see the internal variables. Internal variables are p1, i8, i7 and so on. But you can explicitly, and of course there is a cost involved with taking measurements, you can measure the value of say, p1. Now if you measure the value of p1, for example, then you could resolve between these two candidates. So for example if you measure p1 and p1 turned out to be 6 then you can eliminate m1 as a candidate because m1 is working as expected. It is producing an output of six and then you can just say that the adder is faulty essentially.

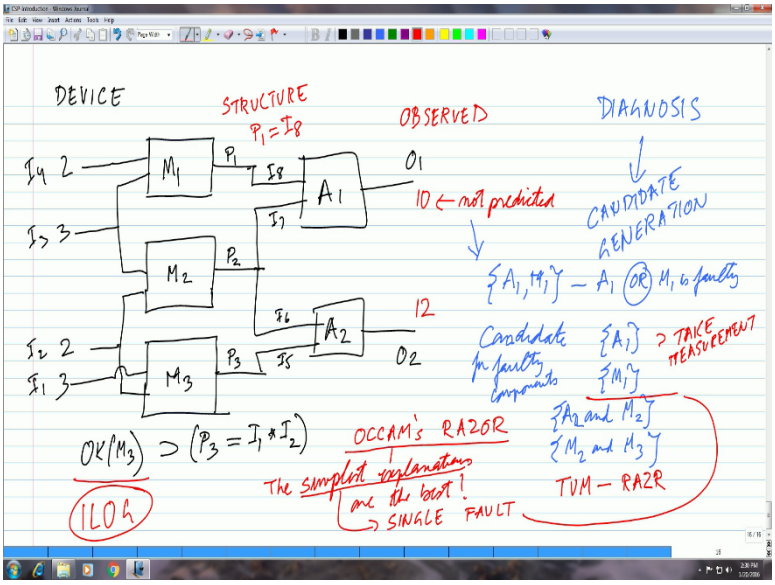
But life unfortunately is not so simple as this. I could say that there is another candidate set which is a2 and m2. I'm saying that two devices have gone bad essentially. So m2 is producing a wrong output. Let's say m2 is producing 4. And therefore 4 goes as an input to a1. So a1 does 4+6 which is 10 which is what we have. So that's explained. How do we explain the fact that we're seeing twelve as the output of a2? We're saying that a2 has also gone bad essentially. a2 is also broken. So a2 is taking 4+6 which is the output from third multiplier and adding up to 12. So that's also a candidate hypothesis. There is nothing to say that this is not the reality and the reality is something else. So you can imagine that you can even say m2 and m3. They are also possible candidate sets essentially.

So at this point most people who work in model based diagnosis, they use what is called as Occam's razor. So if you just look up Occam's razor on the website, it's by William of Ockham. He was a philosopher some time ago, a few hundred years ago, in Munich actually. So curiously these friends of mine who work in model based diagnosis are from Technical University, Munich and they have a tool for doing model based diagnosis which they have called as razr I think. It's a commercially available tool. So TUM stands for Technical University, Munich.

What is Occam's razor? Does anybody know? Or its Occam's principle essentially. It says that the simplest explanations are the best. And what is the simplest explanation here? The simplest explanation is that only one component has got faulty essentially. So the simplest explanation here is single fault. So if you make this assumption that single faults are more likely than double faults because when you say a2 is broken and m2 is broken it's a double fault, two components have broken essentially. So if you stick with the assumption that the simplest explanations are the best then you have to basically decide between a1 and m1 and so you can kind of draw a line and say that these are the only things I will investigate.

So anyways, so I wanted to bring this example as a very useful way in which constraints processing has been applied in the case of model based diagnosis. And constraint processing is quite a big area. There is a company in France called ILOG which produces constraints software and you can buy it if you have a lot of money essentially.

(Refer Slide Time 30:06)



Now we will move away from these examples and applications of constraints and we will try to first look at constraint networks in an abstract fashion and then look at algorithms for solving them essentially.

So I must remind you that from now onwards we will look at only binary finite domain constraints, not necessarily only binary constraints, but basically finite domain constraints which means that the domains of each variable will be finite and the relations that exist between variables, so for example, between p1 and i4 and i3 would be stated explicitly as tuples. So that's how mathematically we can look at relations and that how we will represent relations and the algorithms that we talk about will reason with that. So we'll take that up in the next class that we meet.