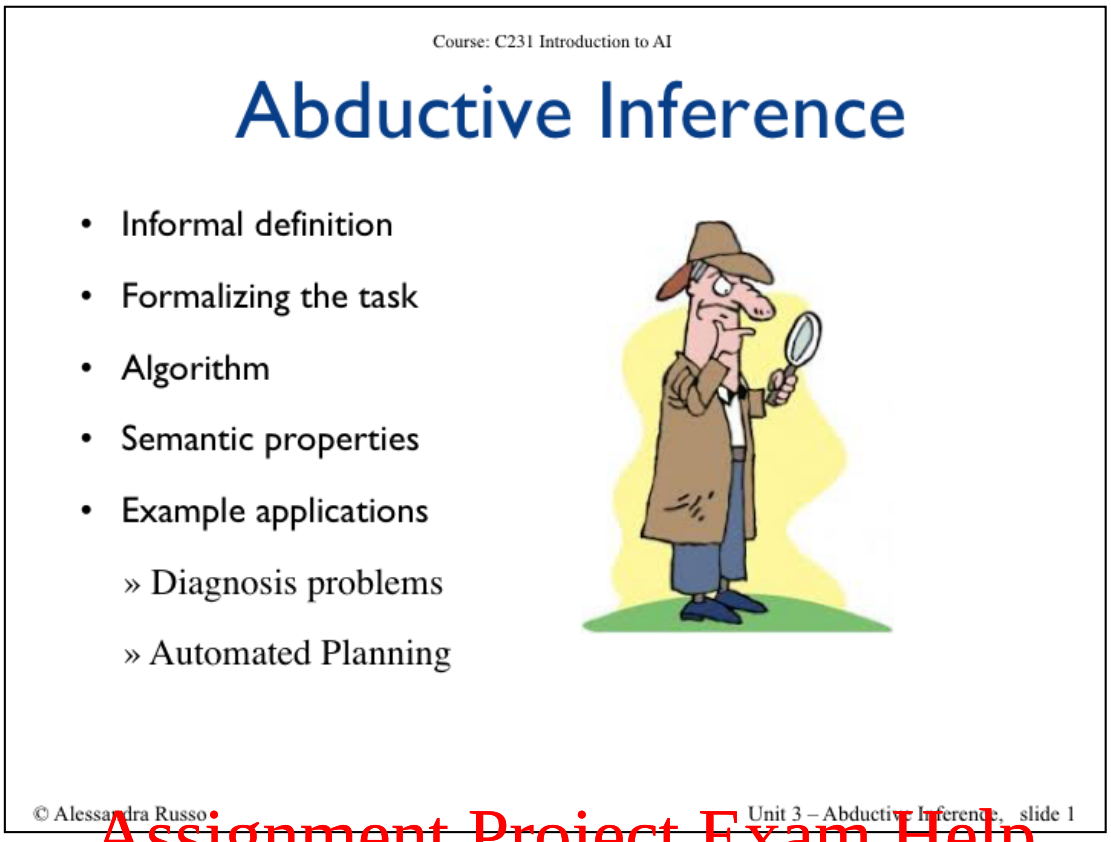
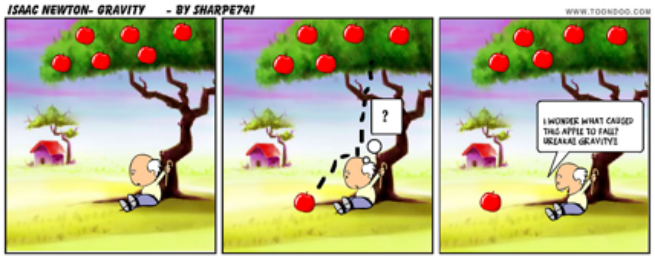


In the previous lecture, we have introduced three forms of reasoning: deductive, inductive and abductive. In this lecture we will look in more detail at the notion of abductive inference. We will give a brief introduction of what abductive inference is versus the other two forms of reasoning, and formalise the notion of abductive reasoning task in a rigorous manner. We will see one of the existing abductive inference algorithms for computing abductive solutions of a given abductive inference task using a goal-directed proof procedure and operational semantics. We will discuss pro and cons of this approach and in particular show the role that integrity constraints, or denials formulae, have during the computation of abductive solutions. We will then give few examples of applications of abductive inference, in particular in the area of fault diagnosis in systems and medical diagnosis. In the next unit we will instead look at goal-directed planning as one of the key applications of abductive inference.

Course: C231 Introduction to AI

Abductive Inference



When Newton saw the apple falling down, he must have done an abductive inference and came up with the theory of gravity.

- *Apple fell down.*
- *If earth pulled everything towards it, then of course, apple too would fall down.*
- *So earth is pulling everything towards it.*

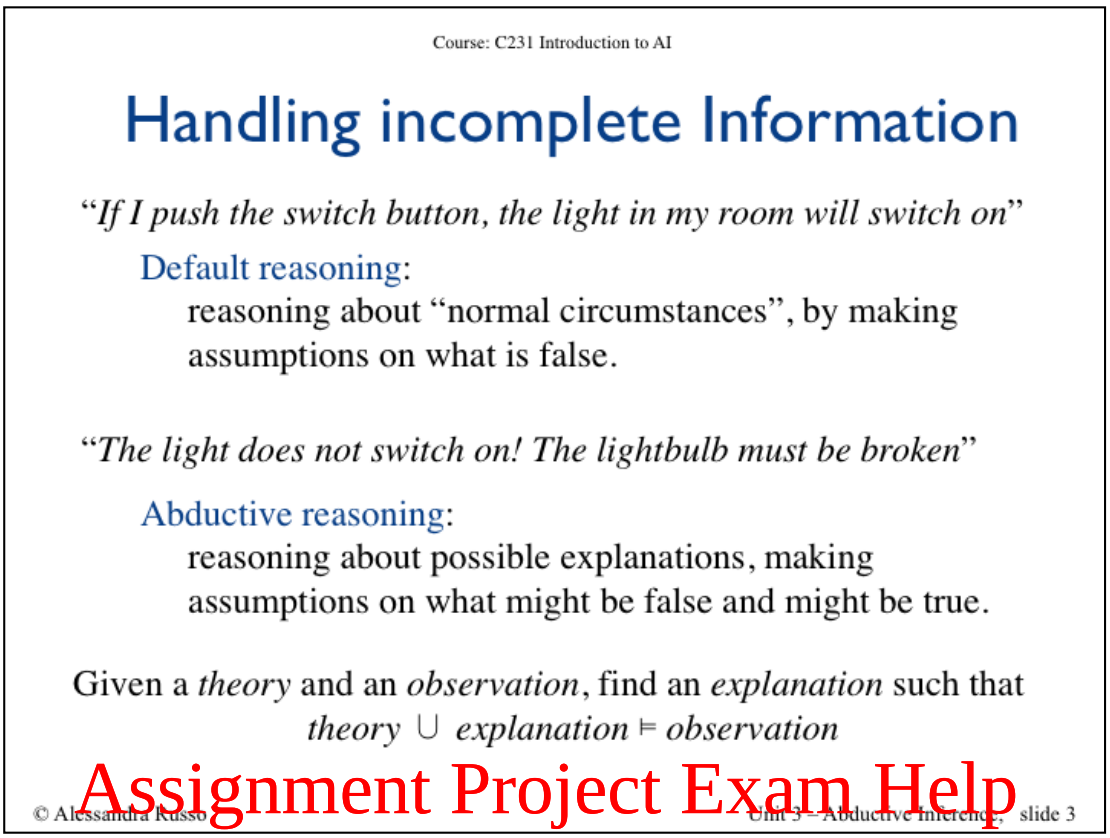
© Alessandra Russo Unit 3 – Abductive Inference, slide 2

As shown in the above example, abductive reasoning is a form of reasoning that is particularly relevant when solutions to a given problem have to be computed in the presence of incomplete knowledge. In the intuitive example illustrated above, for instance, the problem is how to explain the observation that the "apple has fallen on its own". The general commonsense principle used by Newton in his reasoning is that "anything, including also apples, would fall down on earth on their own if the earth pulled things towards it". The incomplete piece of information is whether "the earth pulls things towards it" or not. This is knowledge that at that time was not known to be true and false.

Abductive reasoning helps therefore make assumptions about incomplete information, which are needed in order to explain given observations, or in order to solve a given problem. The answer in the example above is clearly the explanation that "the earth does pull things towards it and therefore pulls also apples towards it".

In this lecture we aim to answer questions such as:

- What are the semantic properties of abductive reasoning?
- How do we compute abductive explanations?
- How to make the proof procedure for abduction satisfy the given semantic properties?
- What is a sound and complete algorithm for performing abductive reasoning?
- What are applications of abduction in real life problems?

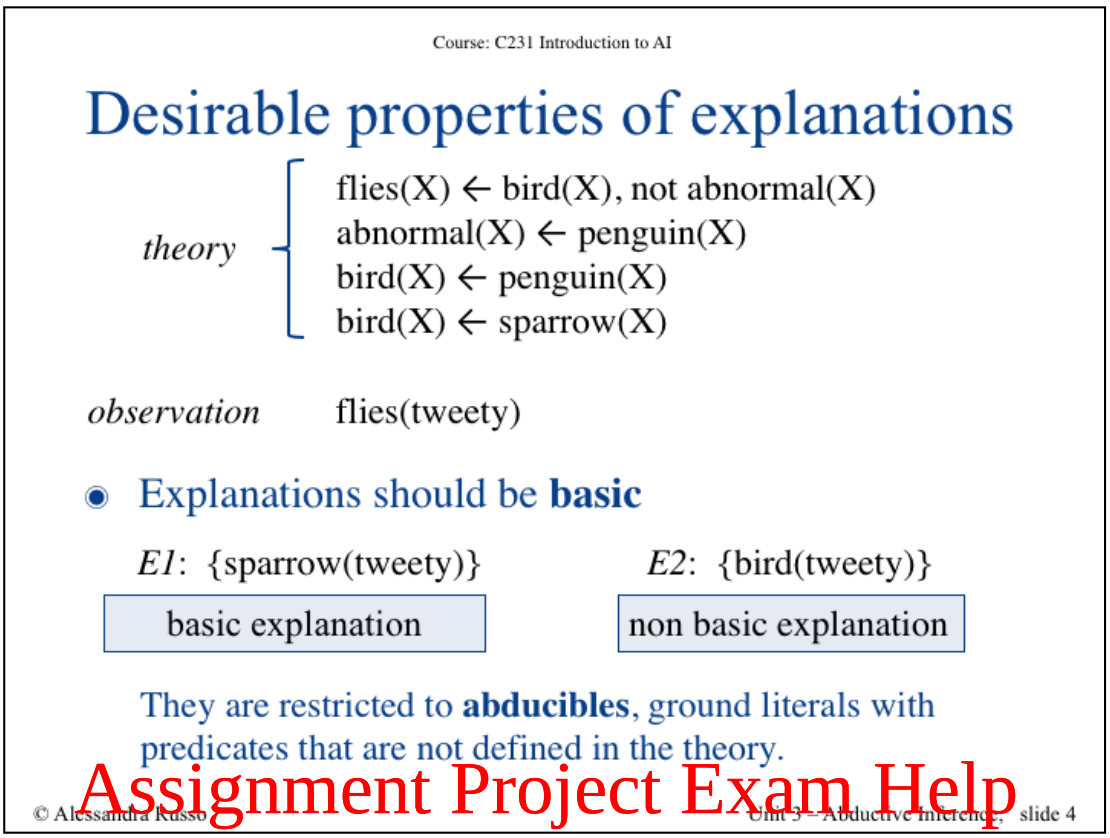


In everyday life, people can perform a number of different types of reasoning. Consider, for instance, the two forms of argument given in this slide.

The first argument seems perfectly alright, it assumes an implicit knowledge about how the switch is connected to the lightbulb and the power supply. But in practice we don't have this full complete knowledge about the state of the world, so we perform reasoning by making lots of implicit assumptions: e.g., the switch is not broken, the connections are in order, the lightbulb is not broken, and there is a supply of power. You have seen in the first part of this course the notion of *default reasoning*. Essentially one way of performing reasoning in the presence of incomplete information is assuming that, typically, things behave normally, unless there is evidence to the contrary. In the above argument, electric circuits are "normally" in order. The statement, however, is not in general true, as it describes just the "normal" case. Any evidence that contradicts the implicit assumptions would invalidate the given argument. For instance, any exceptional circumstance, such as a power cut in the area, or the lightbulb is broken, would make the above statement false.

The second argument extends the default reasoning expressed in the first argument by, not only making assumptions about what is false, but also **making assumptions about what is true**. We observe that the light bulb does not switch on, so a possible explanation (i.e., assumption about what might be true) is that the *lightbulb is broken*. But this is only one of the possible explanations, so it cannot be guaranteed to be true. For instance, there might be a problem with the power supply instead, or the switch might be broken.

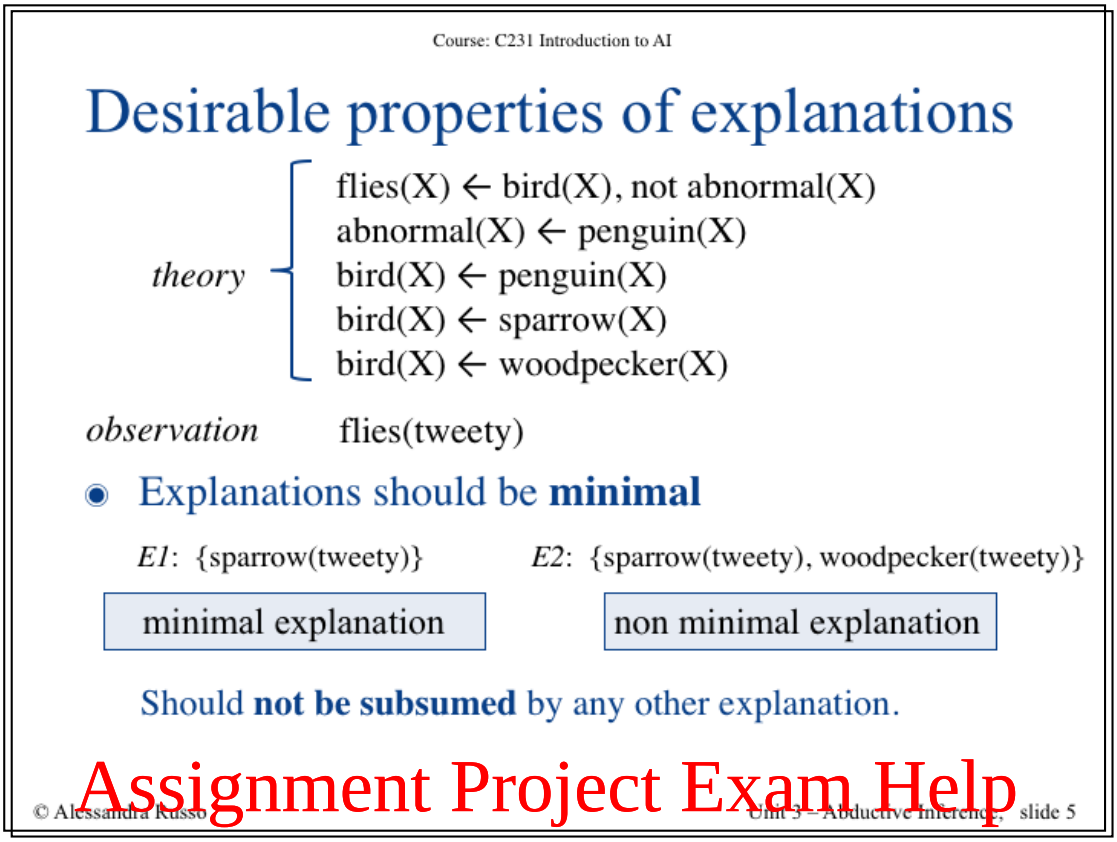
Hence, both default reasoning and abductive reasoning have the common characteristic that however plausible their conclusions may seem, they are not guaranteed to be true in the intended state of the world, because the information we have is incomplete. In default reasoning, the conclusion might be false because the state of affairs is not "normal" as it is assumed to be. In abduction, there might be several alternative explanations, and we do not know which one to choose.



Before going into the details of an abductive proof procedure, let's see what are the desirable properties that abductive explanations should have.

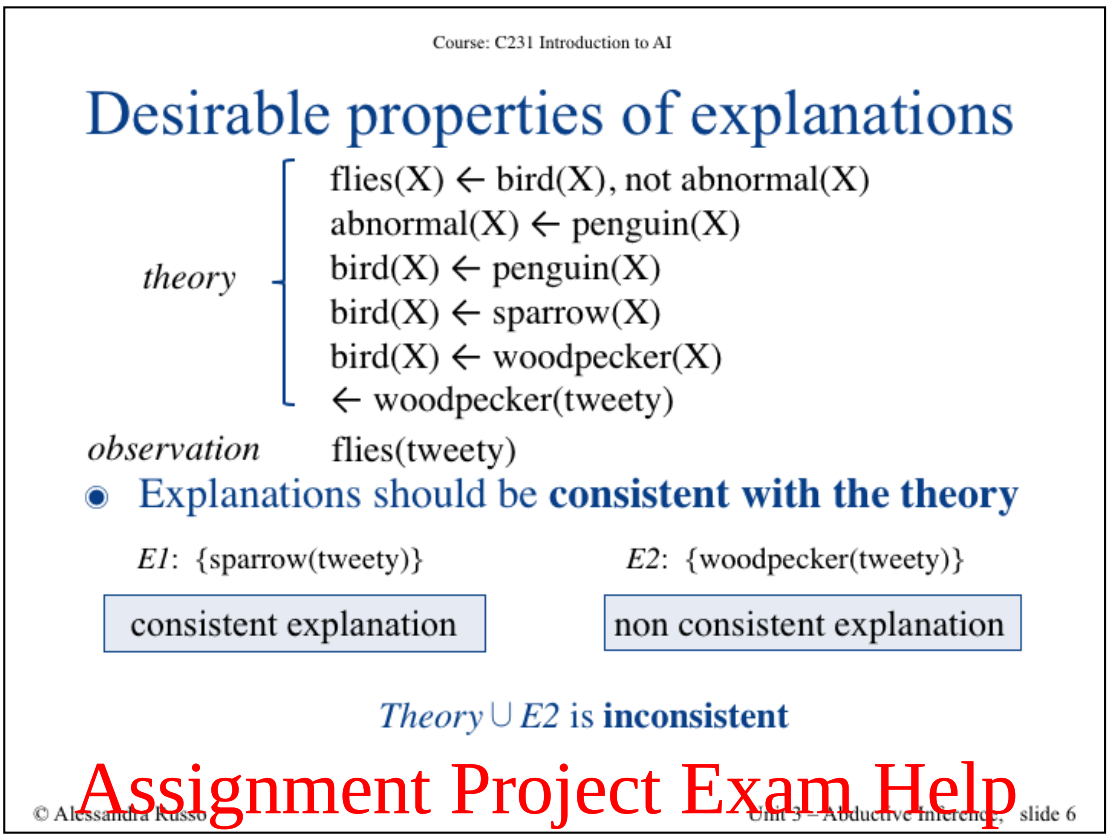
One desirable property is that explanations should be basic. They should be themselves explainable in terms of other explanations. For instance, in the slide above the positive assumption that tweety is a bird could for sure explain the observation that it flies, but being a bird can itself be explained in terms of the assumption that tweety is a sparrow. Given the above theory, the explanation that tweety is a sparrow constitutes a basic explanation, as we cannot identify in our knowledge any further assumption that would explain it.

Logically we say that explanation have to be ground literals expressed in terms of predicates that are not defined in the given theory. The predicate bird(.) for instance is defined in the theory as there is a rule with bird(.) in the head. Whereas the predicate sparrow(.) is not defined.



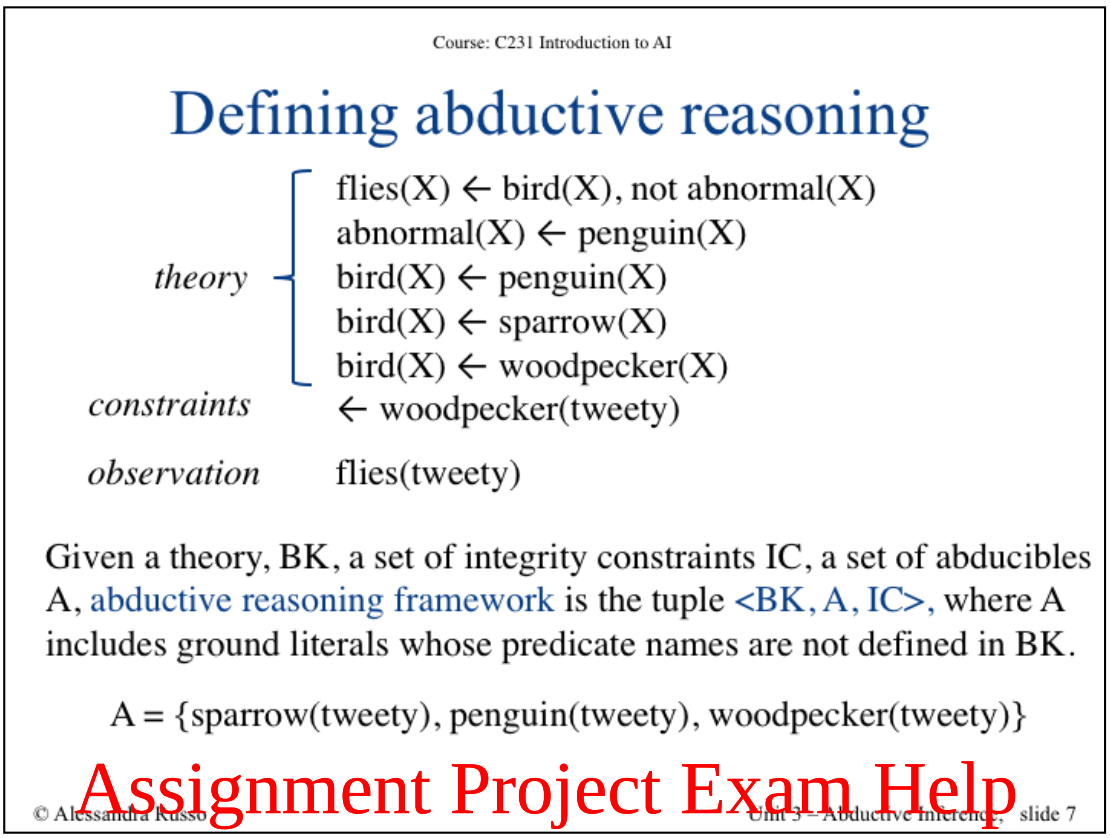
A second property of abductive explanations is **minimality**. Explanations should not be subsumed by other explanations. An example is given in this slide. Assuming, in fact, that the theory had other definitions of the notion of a bird, multiple explanations could be constructed. Although some of these would be minimal, other would not. The set of ground literals {sparrow(tweety), woodpecker(tweety)} is an alternative non minimal explanation, as there exists also the minimal explanation {sparrow(tweety)} that "subsumes it" (i.e. that is included in it) and that equally explains the given observation. A second alternative minimal explanation would be {woodpecker(tweety)}, which also subsumes the non minimal explanation given in the slide, whereas {sparrow(tweety)} and {woodpecker(tweety)} are both basic and minimal as they do not subsume each other. So both valid abductive explanations.

It is easy to see that an abductive task may have multiple basic and minimal explanations. We will see later that integrity constraints can be used to control the number of possible explanations.



A third property that an explanation must satisfy is the **consistency** with the given theory. For instance, the given theory may include some integrity constraints. The computation of the explanation has to take into account the integrity constraints in the given theory.

If the explanation violates any of the existing constraints, then it cannot be an abductive solution. In the example above, for instance, the possible minimal and basic explanation where tweety is a woodpecker cannot be a valid explanation because it violates the constraint that tweety is not a woodpecker.



Abductive reasoning requires three components <BK, A, IC>, where BK is a set of normal clauses, A is a set of ground literals, called abducibles, whose predicate names are not defined in BK, and IC is a set of integrity constrains in the form of denials, $\leftarrow A_1,..,A_n$, not $B_1,\ldots,$ not B_m. Such an integrity constraint means that it is not possible for all $A_1,..,A_n$ to be true and at the same time for all $B_1,\ldots,B_m$ to be false.

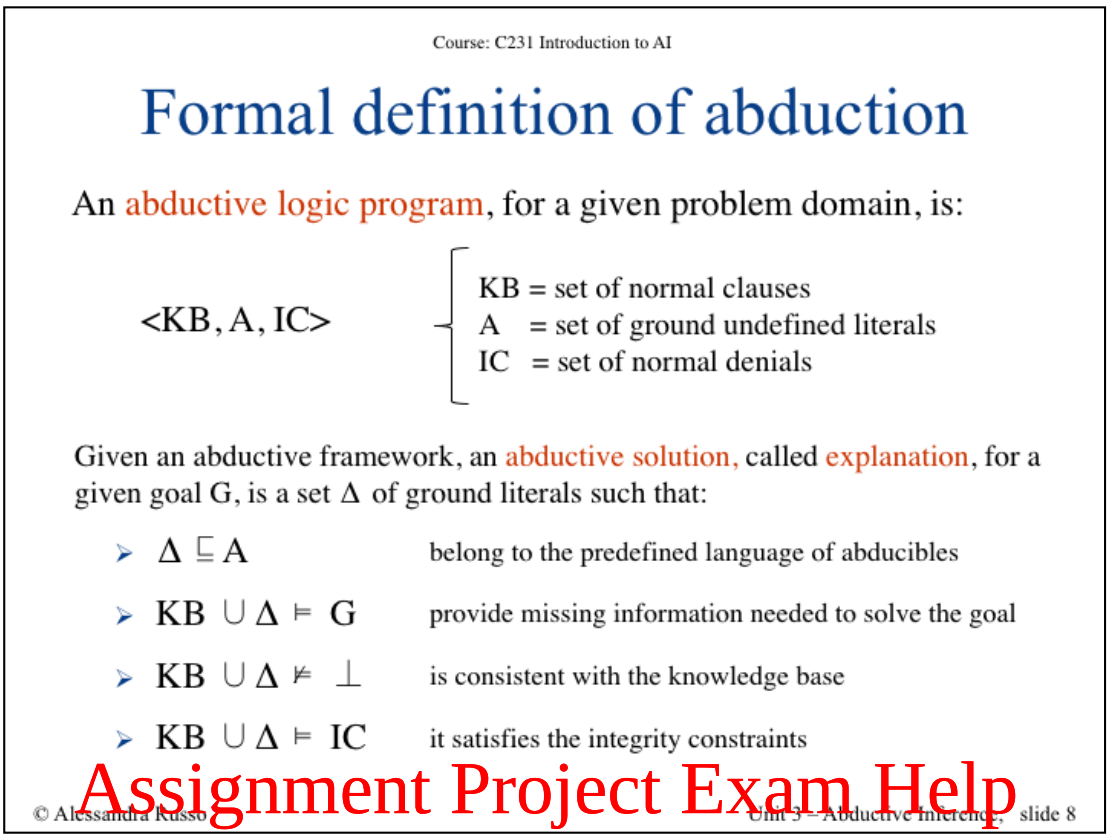
Informally, the normal clauses in BK provide the formal description (or model) of a given problem domain. The integrity constraints in IC specify general properties of the problem domain that need to be respected by any solution to our problem.

Given a specific abductive framework, an abductive task is defined in terms of a given goal (or observation) G that needs to be explained. G is represented by a conjunction of positive and negative (NAF) literals.

An abductive solution to a given abductive task G is essentially an **abductive explanations** of G. An abductive explanation of an abductive task G is a set of ground abducibles that, when added to the given theory BK, explains the observation G and satisfies the integrity constraints IC.

Thus, abductive explanations extend the given theory BK with assumptions about some of the abducible predicates. These assumptions form the solutions of the problem.

The extension or completion of the problem description BK with the abductive explanations provides new information, not contained in the initial problem description. In addition to the properties described before, quality criteria that help prefer one solution over another, can be often expressed via integrity constraints over abducible predicates.

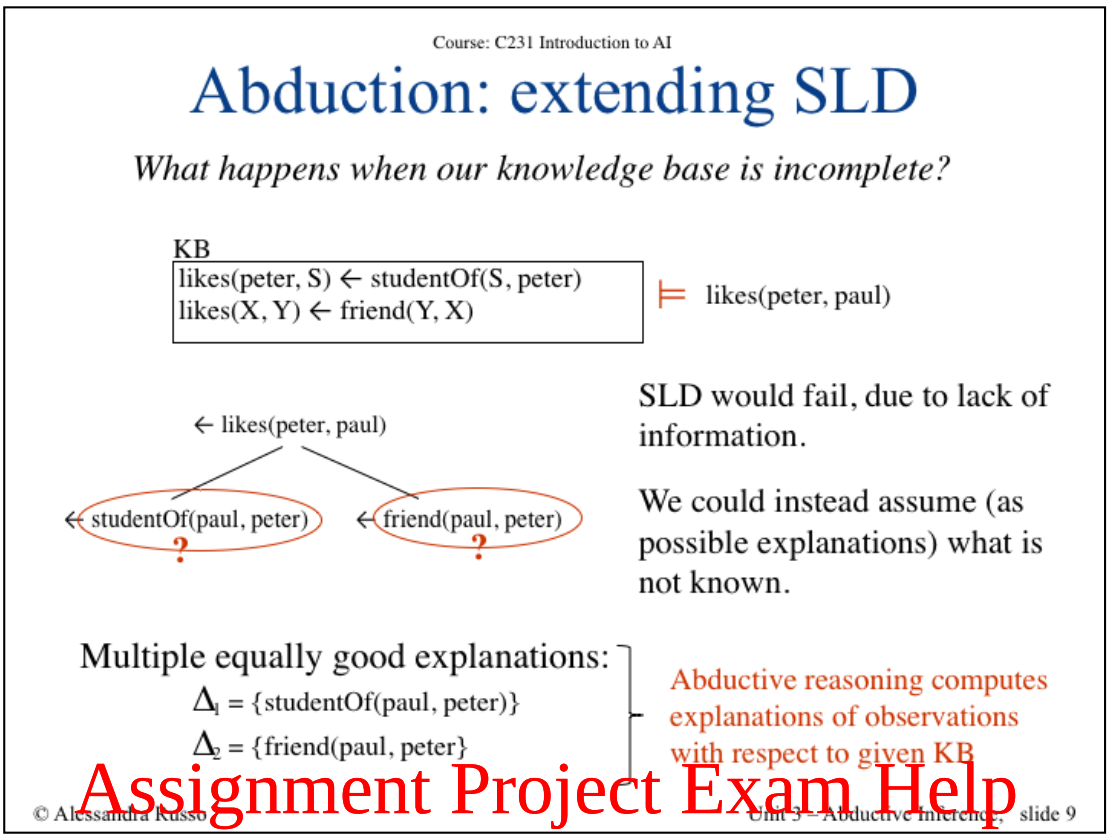


A problem domain can therefore be modelled as an abductive logic program by identifying a tuple of three elements: a theory (or set of normal clauses), known as the agent's knowledge based, KB, a set of abducibles (Ab), i.e. undefined predicates expressing the concepts which are unknown and therefore for which the KB is incomplete, and a set of integrity constraints (IC). The latter are assumed to be expressed as normal denials.

Goals G are conjunctions of ground literals and solving a given goal means computing explanations of G, in terms of the abducible predicates, that together with KB consistently satisfies the integrity constraints and derive the goal. We will see later on that integrity constraints are used to prune explanations that are not desirables but also to introduce in the solution additional assumptions needed to preserve the consistency of the solution.

The four conditions given above are, in principle, independent of the language in which KB, A, IC are formulated. They apply to problems represented as definite clauses, as well as to problems formalised as normal logic programs, and to problems formalised as answer set programs (as you will see later in this course). Depending on the type of language and formalisation you use to express the knowledge of the agent, the underlying semantics would differ. When the KB is a set of definite clauses, the augmented theory KB ∪ Δ will also be a set of definite clauses that accepts a unique minimal model, the Least Herbrand Model. The entailment relation, in this case, would be under the minimal model semantics. So the last three conditions will simply require that the given goal and the constraints **are satisfied** in the Least Herbrand Model of KB ∪ Δ.

The concept of integrity constraints (first arose in the database community to capture the fact that only certain states of the knowledge base are considered to be acceptable) and satisfiability of integrity constraint also depends on the chosen underlying semantics and its related notion of consistency. In general, the three main semantics of logic programming with negation as failure (completion, stable models and well-founded semantics) have been used to define different ALP frameworks. In the cases where KB ∪ Δ accepts a unique model, these three semantics coincide. We will assume for now this to be the case. So you can read the symbol ⊨ as entailment under the Least Herbrand Model semantics.



The above example is from Peter Flach's textbook "Simply Logical, intelligent reasoning by examples". As mentioned in the previous lecture, deduction proves information that is already implicit in the knowledge based (being this in the form of rules and/or facts). But what about if the knowledge base is not complete, for instance we don't know enough information about a certain situation (facts), but we know the general rules? A resolution derivation would fail as it would terminate with clauses (or denials) that cannot be resolved due to the incompleteness of the knowledge base.

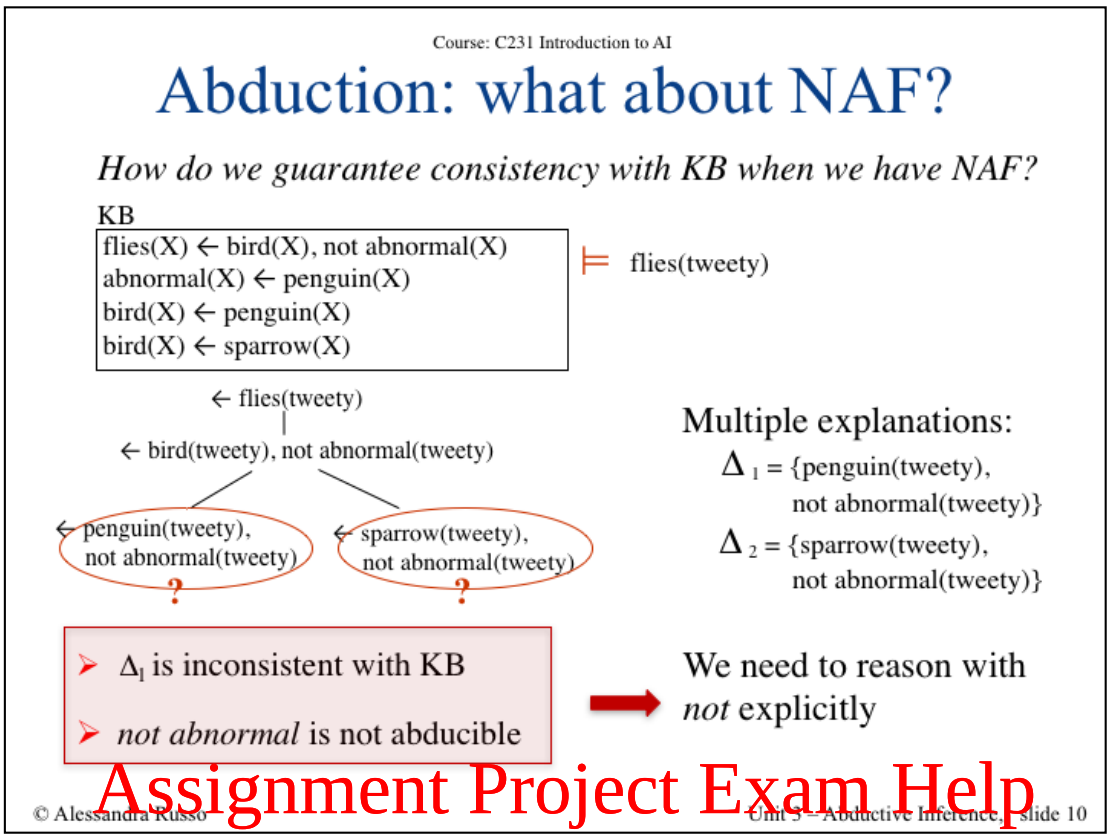
For instance, consider the simple example given in this slide with goal G = **peter likes paul**. There might be various reasons why somebody likes somebody else, and a couple of these principles are given in the knowledge base. When performing SLD on the given query we reach the sub-goals **friend(paul, peter)** in one possible derivation and the sub-goal **studentOf(paul, peter)**. Both these two sub-goals would respectively fail in SLD.

We can imagine to extend the proof procedure by using an accumulator Δ to which unproved sub-goals are added as assumptions. Such an accumulator Δ will basically contain, during the derivation process, explanations found so far in the proof. So, for instance, in the example above, in the left branch derivation the fact **studentOf(paul, peter)** would be assumed to be true and added to the accumulator Δ_1. In the alternative right branch derivation the fact **friend(paul, peter)** would instead be assumed to be true and added to the (alternative) accumulator Δ_2 . These two accumulators would, respectively, represent the abductive solutions for the given problem G=friend(peter, paul) given as initial goal.

As you can see, given to the existence of alternative derivations, alternative explanations could be generated. Note that explanations should not be unnecessarily strong (i.e. include more ground literals that what is needed) or unnecessarily weak (too few to prove the given goal in all circumstances). They have to be parsimonious and the proof procedure has to guarantee the computation of "minimal" abductive explanations that, together with the given theory, prove the goal.

But, is it sufficient to just include an accumulator and extend the SLD with the additional rule of adding literals that are not provable into the accumulator, as a procedure for performing abductive reasoning?

Let's consider the example given in the next slide.

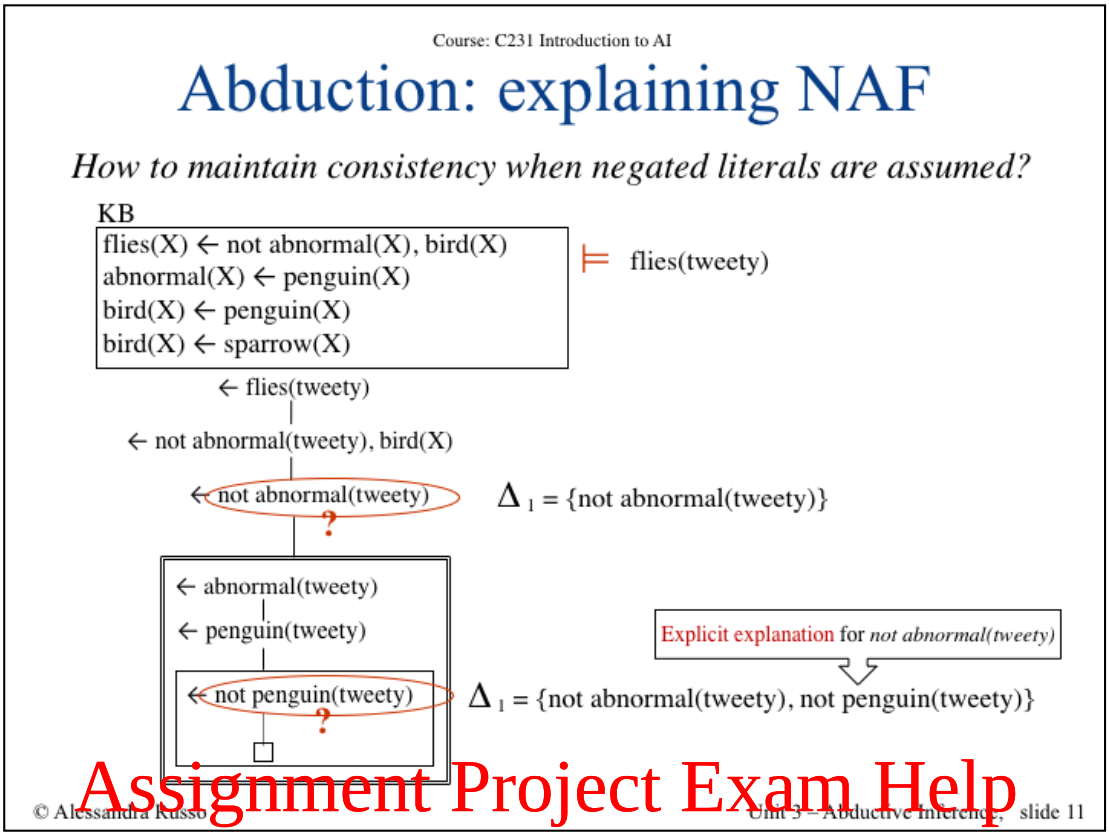


Let's assume now that our background knowledge includes negated conditions. For any negated condition, there will never be a rule with the negated literals in the head. So, we might on a first instance assume that it might be sufficient to also assume the negated literals that are needed in order to explain a given goal. Consider for instance the above background knowledge and let's assume that we want to find out if tweety flies.

If we use the same procedure we described in the previous slide we end up with two possible explanations, one stating that tweety flies because it is a penguin and it is not abnormal, the other explanation says that it flies because it is a sparrow and it is not abnormal. Bur, for both these two explanations there are two main problems:

1) In the first explanation the assumptions *not abnormal(tweety)* and *penguin(tweety)* are together inconsistent with the BK.
2) The assumption not abnormal(tweety) is a negated literal whose predicate name is defined in the background knowledge, so literally it is not an abducible.

So how do we compute explanations that are consistent with the background knowledge? We need to reason about negation explicitly in our abductive proof procedure, and not just assume relevant negated literals.



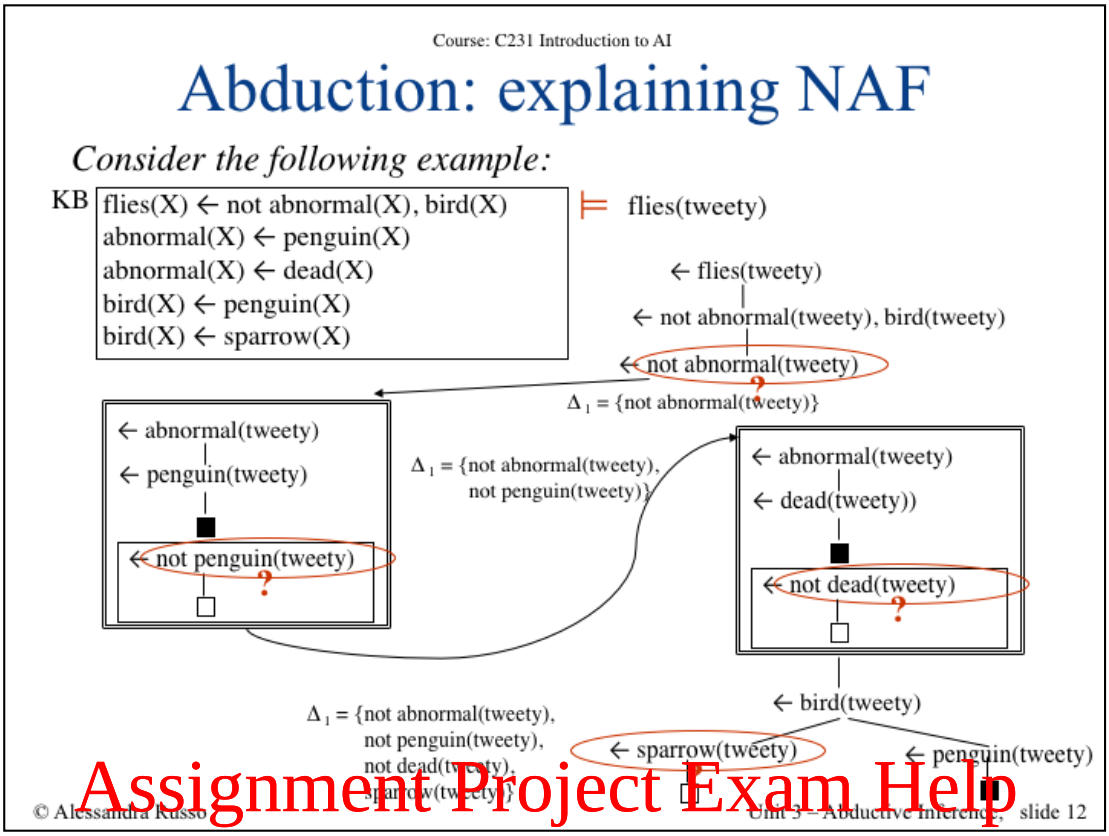
To maintain consistency when negated literals are assumed the abductive reasoning process has to perform a consistency reasoning phase. This in order to identify explicit explanations for the assumed negated literals. Accumulating these explicit explanations stops the reasoning process to later one assume additional information that may violate the assumed negated literals, as we have seen in the derivation shown in the previous slide, where the assumption of penguin(tweety) caused inconsistency with the previously assumed negated literal not abnormal(tweety).

How do we perform such a consistency reasoning phase? Consider the example given in this slide.

The abductive derivation tries to explain the goal **flies(tweety)**.The standard derivation is performed and when the negated condition *not abnormal(tweety)* is generated as sub-goal, we assume it to be true and add it to the accumulator Δ_1. But at this point a consistency phase is started (the proof in the double lined box) that tries to fail ←**abnormal(tweety).** This proof essentially looks for explanations, in terms of abducibles, of why abnormal(tweety) fails. The derivation identifies the sub-goal ←**penguin(tweety)**, which has to fail. This is now an abducible for which no assumption has been made so far. So try to fail it is equivalent to try to succeed its own negated form ←**not penguin(tweety)**. This is why in the slide a standard (abductive) derivation is started with sub-goal **not penguin(tweety).** Now we have to find an abducible that has to succeed but of which we have no information. Therefore, this is then assumed as true and added to the current accumulator Δ_1. Essentially, the assumption **not penguin(tweety)** is the explicit explanation of why **not abnormal(tweety)** can consistently be assumed to be true. The success of this inner branch makes the outer double boxed derivation also succeed in proving the failure of abnormal(tweety), and the initial derivation can continue with the next sub-goal ←**bird(tweety)** and the accumulator Δ_1 constructed so far.

The slide shows only part of the full derivation and the relevant accumulator Δ_1 . But, it is easy to see, although not shown in the slide that the next steps of this derivation would lead to two possible branches. On one hand, bird(tweety) can be proved if penguin(tweety) can be proved, but since we have already assumed **not penguin(tweety)** in Δ_1, this branch fails. The second branch tries to prove bird(tweety) by proving the sub-goal sparrow(tweety). This is again an abducible and can be assumed (added to Δ_1) so completing the proof. The final explanation would be the set

Δ_1 = {not abnormal(tweety), not penguin(tweety), sparrow(tweety)}



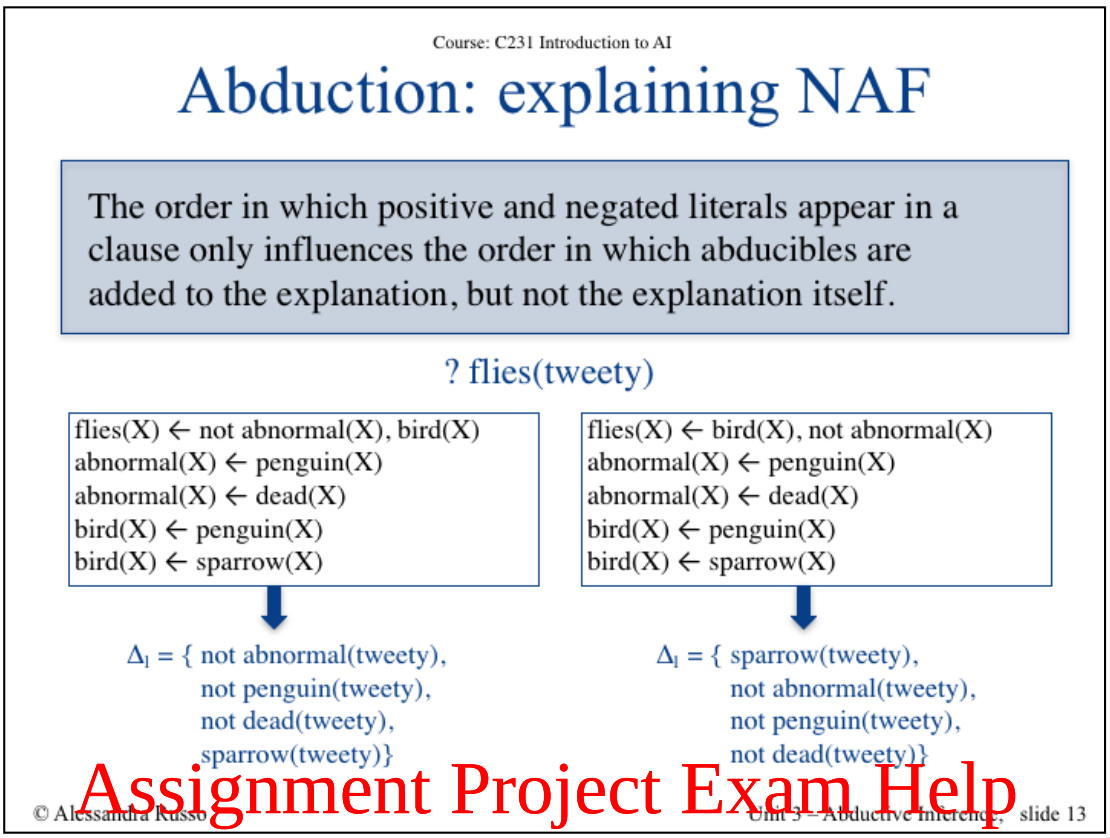
This slide shows a slightly more complex example where the literal **abnormal(tweety)** has multiple definitions. So in proving the negation of it, explicit explanations for every possible way of failing the goal **abnormal(tweety)** have to be computed. All these explanations are accumulated together as part of the same abductive solution to the given problem.

Please note that the above derivation is presented in an informal way, as we are trying to understand the key relevant aspects of an abductive reasoning process. We will see later one the formal definition of the full abductive proof procedure and the formal way of diagrammatically structuring the derivation.

The full abductive explanation would be

$$\Delta_1 = \{\text{not abnormal(tweety), not penguin(tweety), not dead(tweety), sparrow(tweety)}\}$$

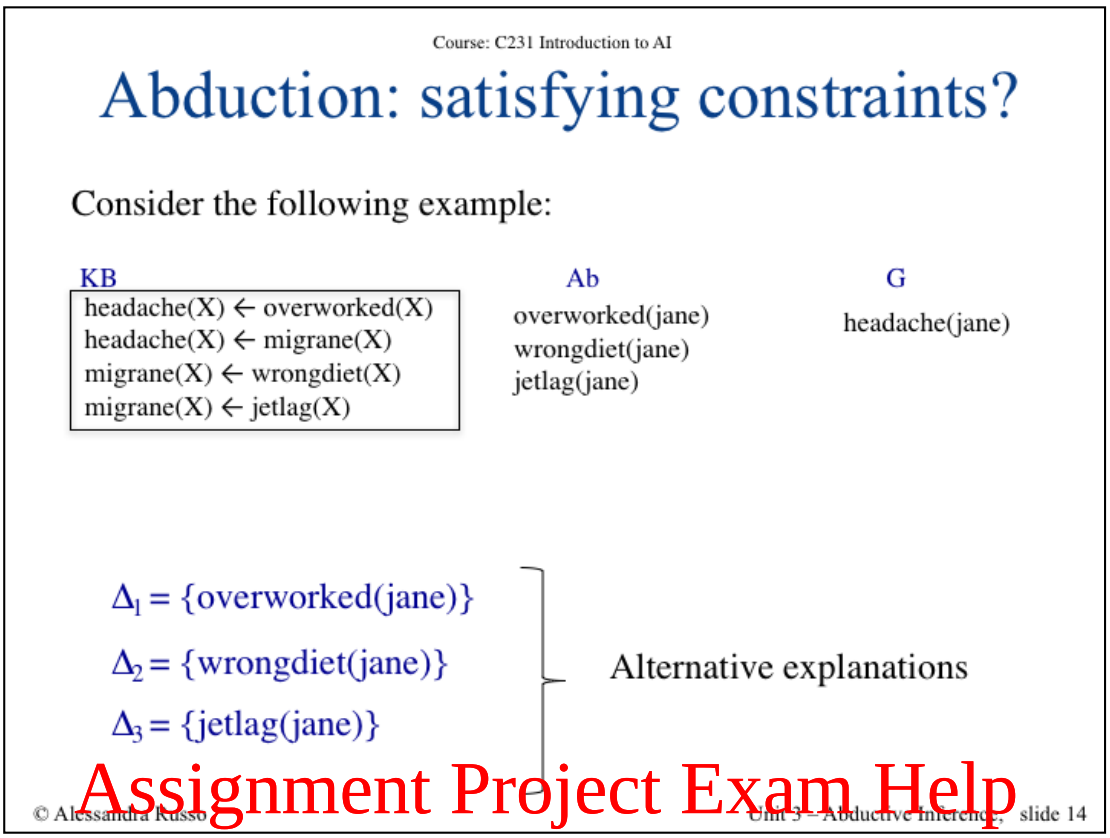
As you can see the set of abducibles (or explanations) computing during the derivation grows along the same derivation branch of the proof tree. Alternative branches would lead to alternative explanations as we have seen briefly in slide 10.



One of the important principle of abductive reasoning, is that the order in which the negative and positive literals appears in a given clause does not affect the abductive solution. Consider for instance the example given in this slide. The two background knowledge are identical except for the ordering of the body conditions in the first clause.

The abductive solution of the program on the left is as given in the slide, and shown in the derivation presented in the previous slide.

The abductive solution of the program on the right, is set theoretically identical, as it includes the same literals but just assumed during the computation in a different order. Would you be able to generate the derivation tree of the program on the right and show that it does indeed produces the set Δ_1 of explanations given in the slide?



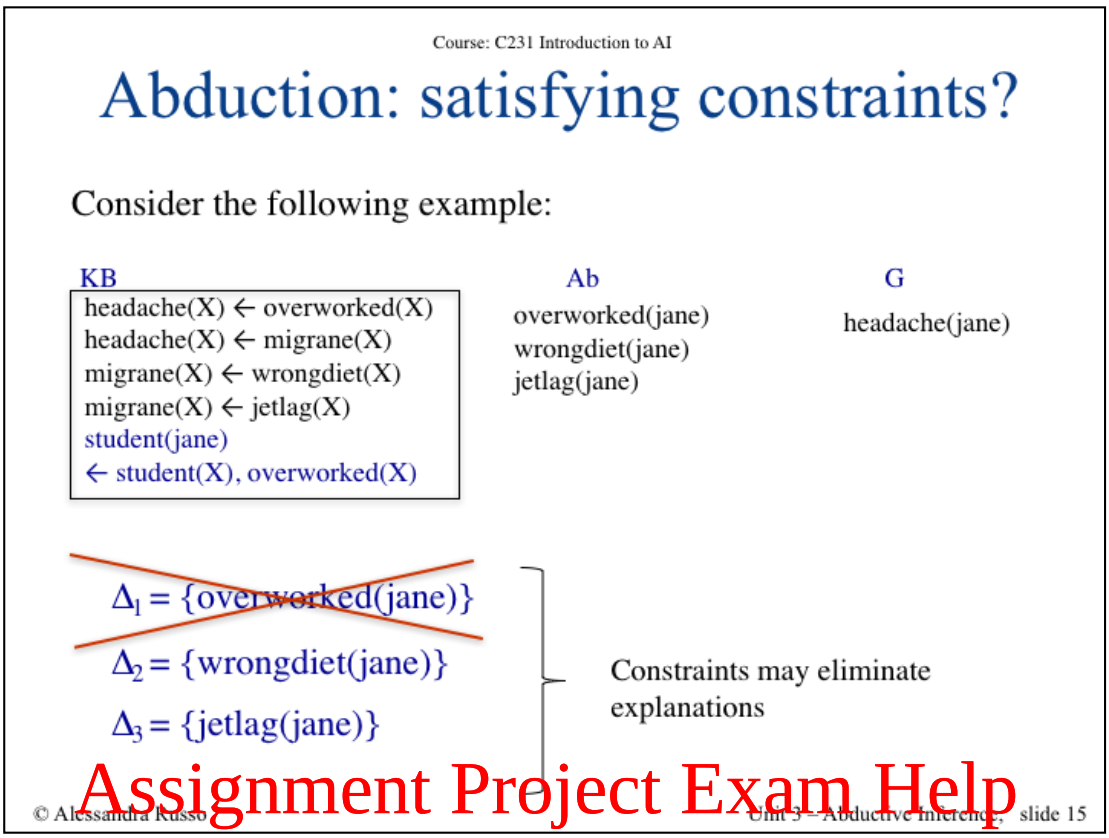
We have defined the notion of abductive framework in terms of a tuple <BK, A, IC>. We have seen what BK and A look like and how abductive solutions have to be consistent with the given background knowledge. Basically, we have seen what kind of inference steps we would need to use in order to generate solutions that satisfy the following three conditions of an abductive framework:

- $\Delta \subseteq A$
- $KB \cup \Delta \models G$
- $KB \cup \Delta \not\models \bot$

What we haven't see is how integrity constraints can be satisfied by an abductive solution. Essentially, how do we guarantee the following 4th condition:

- $KB \cup \Delta \models IC$

We consider a diagnostic example that consists of finding an explanation of why jane suffers of headaches. It is easy to see that given the above abductive framework, the goal "headache(jane)" has three possible alternative explanations. Let's assume now that our BK has some integrity constraints (see next slide).



The integrity constraint states that students cannot be overworked. It also happens to be the case that Jane is a student. When the abductive framework includes integrity constraints that involve abducibles (in this case the ground fact overworked(jane)}, making an assumption on these abducibles requires checking the satisfiability of the integrity constraints that involve such an assumption. For instance, assuming overworked(jane) needs to be followed by a phase of checking consistency with the integrity constraint that mentions overworked(jane).

Satisfying a denial clause (or integrity constraint) means proving that at least one of its literals is false, because clearly if all literals were true then the constraint would imply falsity, which is inconsistency. This computational process is also part of the consistency phase that we have briefly described before when dealing with assumptions of negated literals.

So, the abducible overworked(jane) can only be assumed (i.e. added to the accumulator Δ), if the constraint is satisfied, that is if student(jane) is not true. But in the given background knowledge we have the fact student(jane), so overworked(jane) cannot be assumed after all.

The effect of the integrity constraint, in this case, is to cut possible abductive solutions. Let's see how the derivation tree would look like.

Abduction: satisfying constraints?

KB

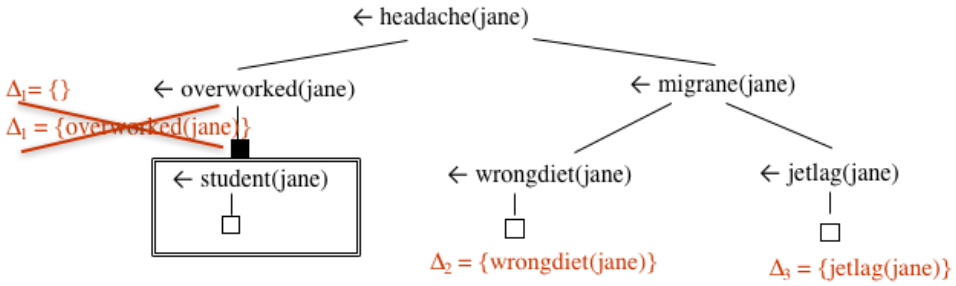
```
headache(X) ← overworked(X)
headache(X) ← migrane(X)
migrane(X) ← wrongdiet(X)
migrane(X) ← jetlag(X)
student(jane)
← student(X), overworked(X)
```

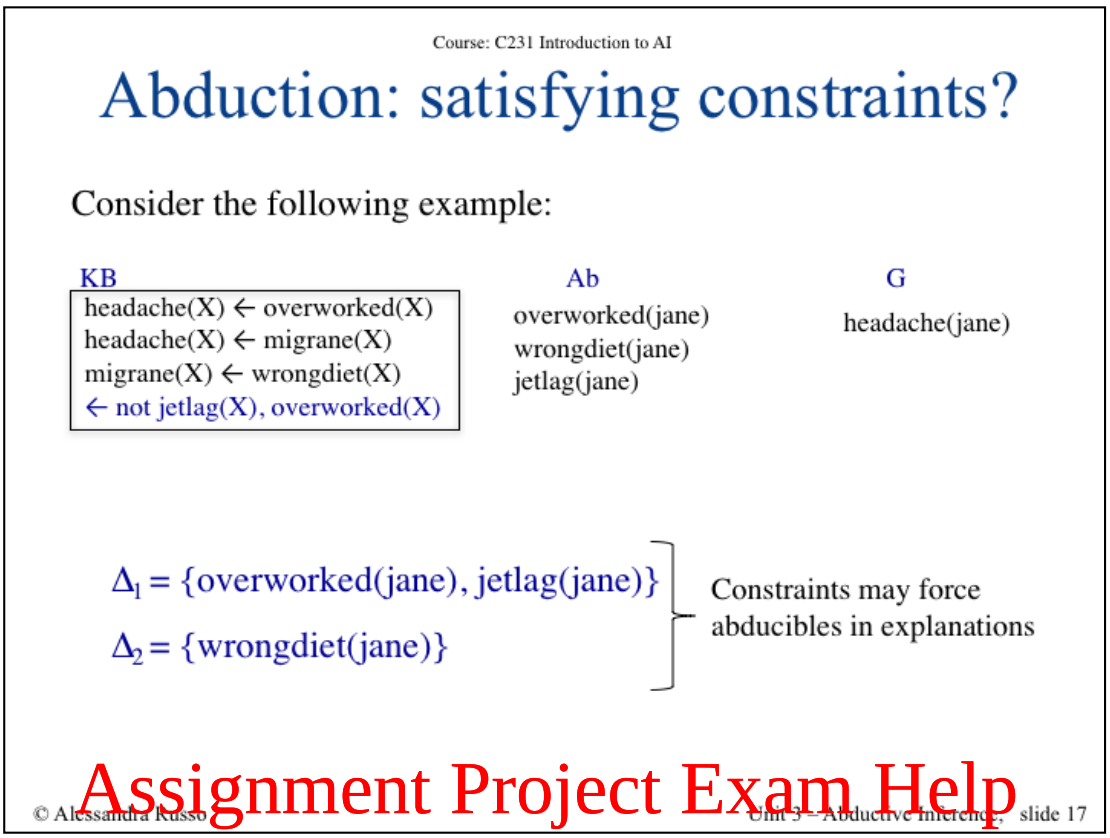
Ab

overworked(jane)
wrongdiet(jane)
jetlag(jane)

G

headache(jane)





But depending on whether literals in the integrity constraints are positive or negative, the effect on the final explanation may also be that of adding additional assumptions, for the sake of establishing satisfiability of the integrity constraints.

Consider this second example. The assumption of overworked(jane) requires the satisfiability of the integrity constraint. But under the assumption now that overworked(jane) is true, the constraint would only be satisfied if "not jetlag(jane)" fails, which essentially reduces to prove that "jetlag(jane) succeeds. Since jetlag(X) is an abducible itself the only way to succeed is to assume that it is true.

This is why explanation Δ_1 includes both overworked(jane) and jetlag(jane).

The process of satisfying the integrity constraints may therefore force additional assumptions in the final explanation.

Abduction: satisfying constraints?

KB

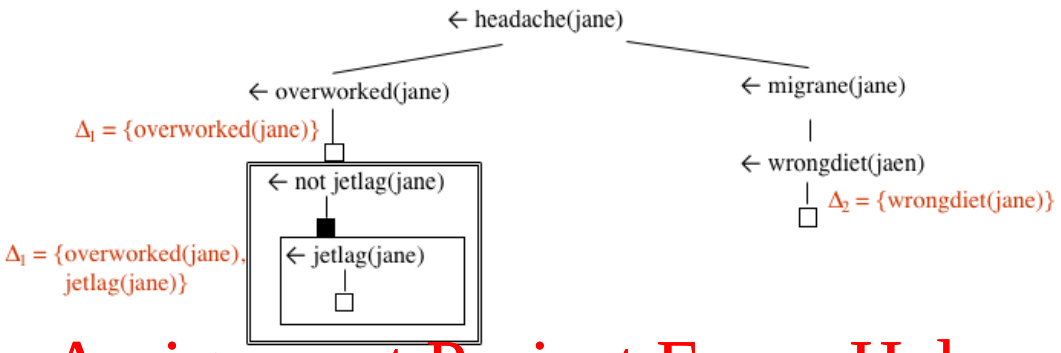
```
headache(X) ← overworked(X)
headache(X) ← migrane(X)
migrane(X) ← wrongdiet(X)
← not jetlag(jane)), overworked(X)
```

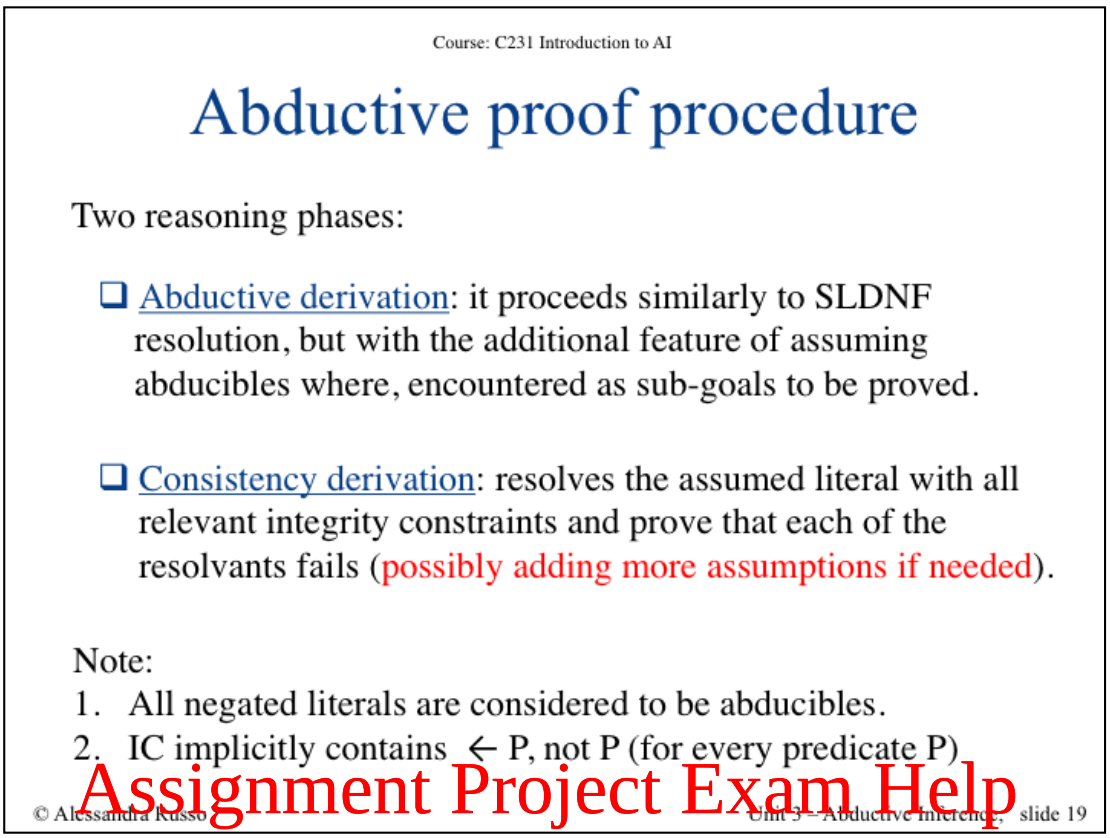
Ab

overworked(jane)
wrongdiet(jane)
jetlag(jane)

G

headache(jane)



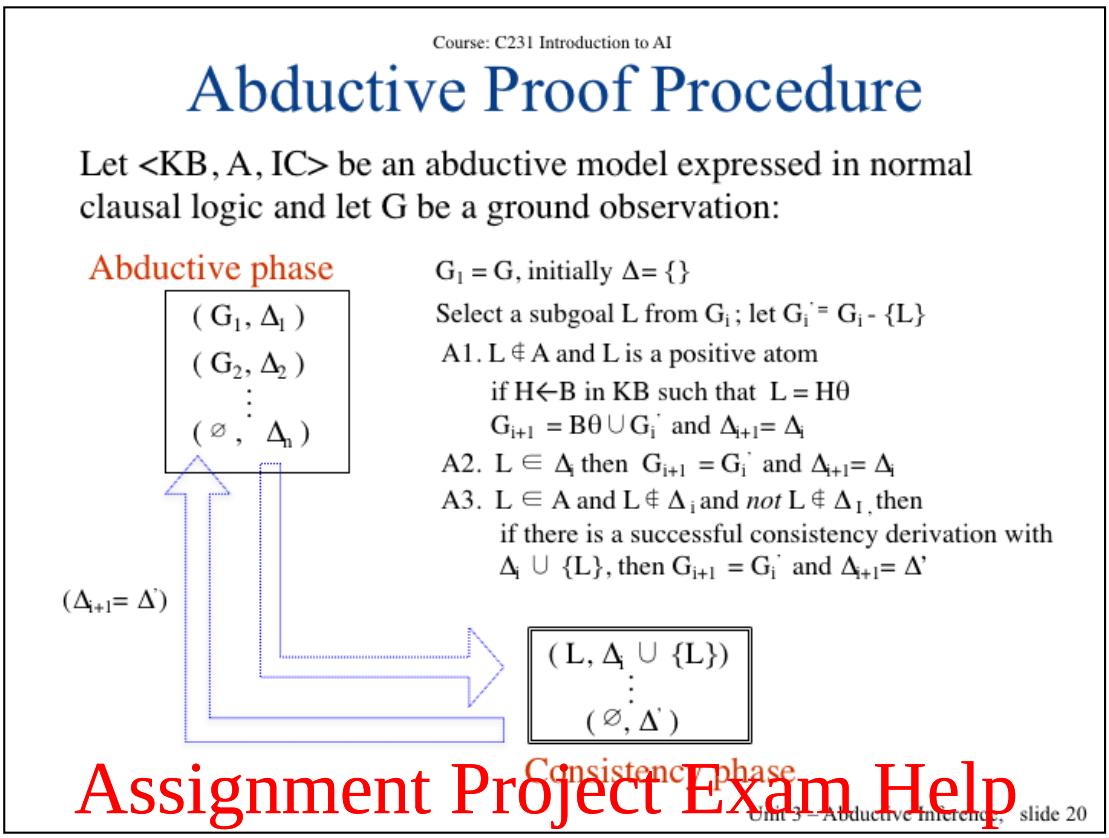


Various abductive inference systems have been developed. We consider here one of the basic but still effective systems. The main characteristic of this proof procedure is the interleave of two phases, called respectively abductive phase and consistency phase.

In simple terms the abductive phase extends the standard SLDNF resolution by allowing the assumption of abducibles when they are encountered during the resolution process. Note that all negated literals, even those whose base predicate names are not abducibles, are considered to be abducibles. This is because we need to assume them and check for consistency.

The consistency phase is called when an assumption is made and the idea is that constraints that mention this assumption need to be checked for satisfiability.

In each of the above phases difference cases may arise depending on whether the goal to prove is positive or negative, abducible or not abducible. The next two slides provide the definition of the two phases respectively and for each phase it gives the various cases that can occur.



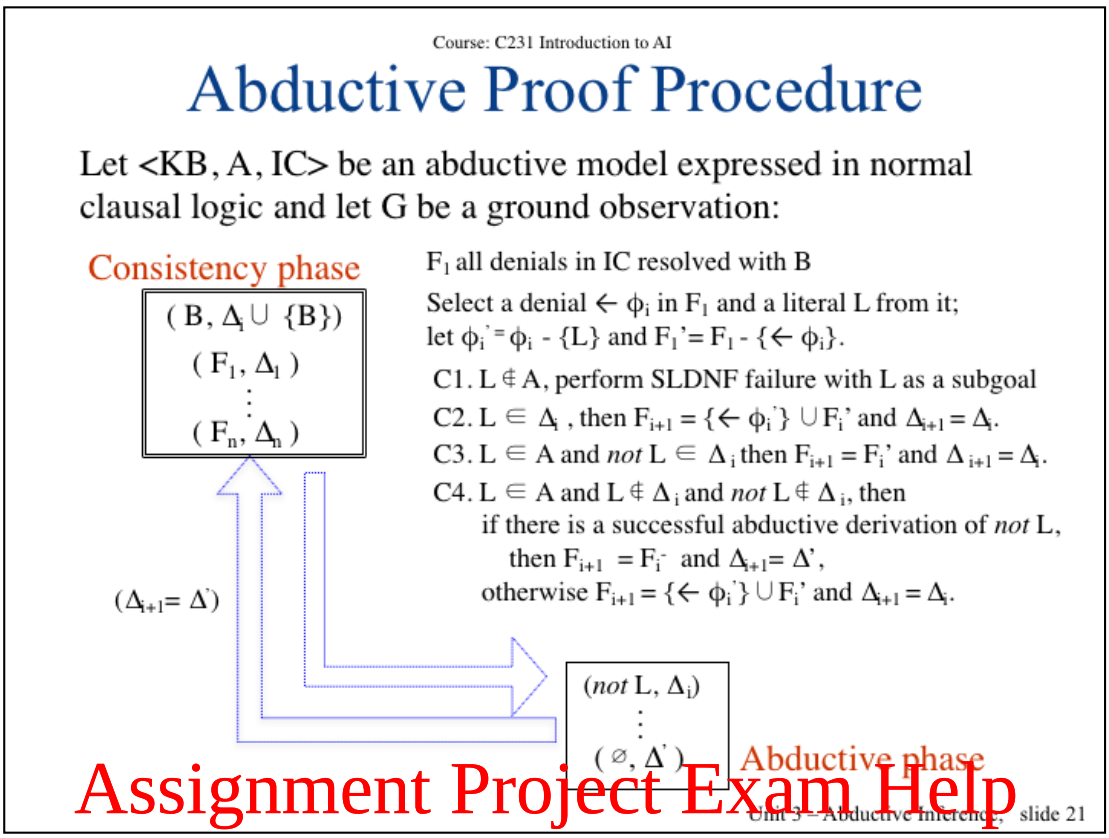
This and the next slide formalise the different cases that can occur during the abductive reasoning phase and consistency phase respectively. This is what is often referred to as operational semantics of abduction. Note that the two phases can interleave. One can call the other and vice-versa. The important thing to understand is, at which point each of these phases is called and why. As mentioned in the previous slide, to guarantee consistency between assumptions and negative literals, the set of abducibles A, of a given abductive framework, implicitly includes the negation of all defined predicates as abducibles; and a consistency constraint of the form ←A, *not* A, for any predicate A, is assumed to be part of IC.

Starting from a given observation (or goal), the first phase is always an abductive phase. So the observation becomes the goal of the abductive derivation that needs to be proved. Note that this goal may well be a conjunction of ground literals, not just a single literal. This is why the first thing to do is choose the literal to prove. The procedure follows the left to right strategy of prolog. At the beginning the set of explanations is empty as nothing has been assumed yet. The abductive phase is a derivation that succeeds when no further sub-goals are left to prove. This phase can be called at any point of the proof. So the current (goal) literal to prove could be of different types:

A1. The current goal is not an abducible. Then an SLDNF derivation is initiated with the current goal to be proved by refutation. If the literal is positive then standard resolution is applied. If the literal is negative then case three (iii) below is performed. (Note that this is because the negation of all defined predicates are implicitly part of the abducibles A, and they need to be assumed and checked for consistency.

A2. The current goal is an abducible (positive or negative). Since the abductive phase considers in each case the current KB $\cup \Delta_i$, if such a goal is already part of the current Δ, then by resolution the literal is proved and the abductive derivation proceeds with the rest of the pending sub-goals (G_i').

A3. The current goal is an abducible not yet assumed. (Remember this could also be the negation of a predicate defined in KB). In this case, the literal is assumed (i.e. added to the current Δ_i), but a consistency phase needs to be triggered to verify that such an assumption does not introduce inconsistencies. The consistent phase is a failure derivation. So its success requires the derivation to finish with a failure. If such consistency derivation succeeds, then the abductive derivation can continue with the remaining pending sub-goals (G_i') and the current accumulator Δ_i, is the one produced during the consistency proof.



In the consistency phase, the new assumption is added to the current set Δ and resolved with any constraint that includes it. The new set of resolvent denials have each of them to fail for the consistency phase to succeed. A literal is selected from the first resolvent denial. Different cases need to be considered here:

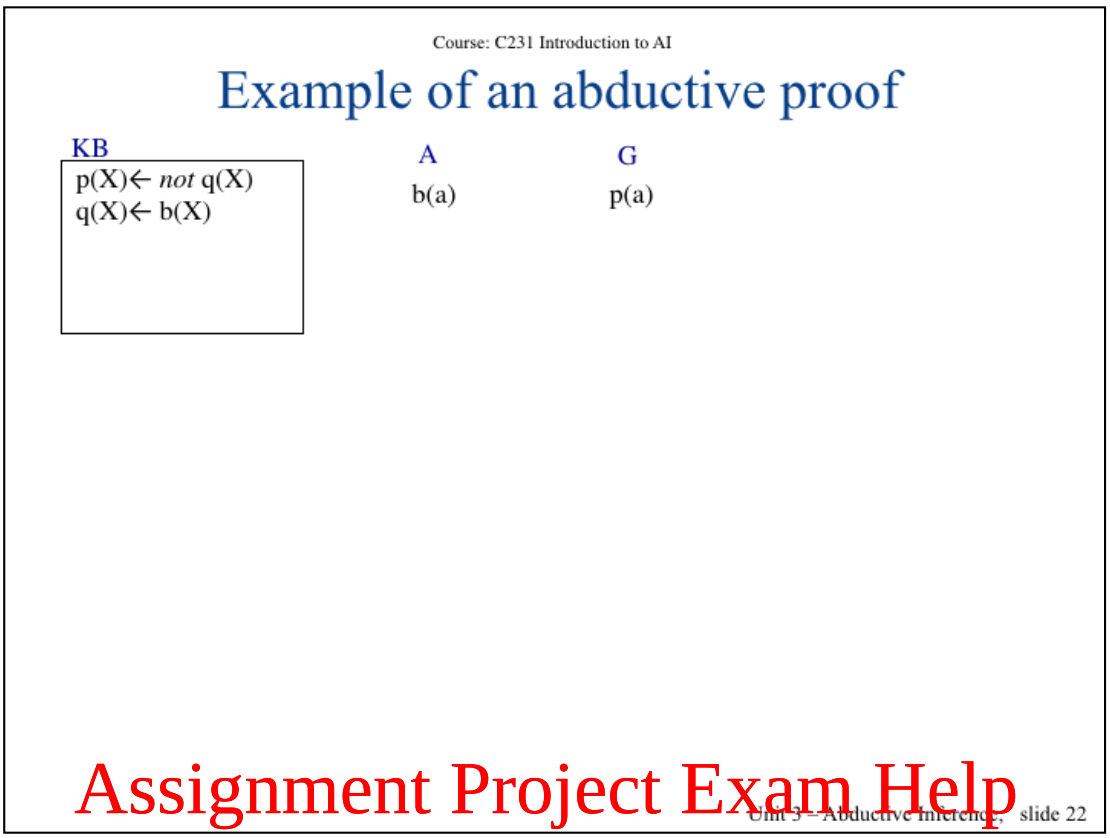
C1. The selected literal is not an abducible (this means is a positive defined literal). An SLDNF failure derivation of this literal is then applied.

C2. The selected literal is an abducible already assumed in Δ. Then this literal in the denial resolvent succeeds, so for the resolvent to be proved to fail the remaining literals will need to be checked for failure. So a new resolvent is generated by removing from it the selected literal, and adding it at the front of the set of pending resolvent denials left to check.

C3. The selected literal is an abducible whose negation is already assumed in Δ. In this case, the selected literal fails, and the denial is considered to be satisfied. A next denial is picked from the remaining pending resolvent denials left to check.

C4. The selected literal is an abducible who is not assumed and its negation is also not assumed. In this case, since the literal has to fail (because we are in a consistency phase derivation) an abductive derivation of its negation (as subgoal) is started. If this abductive derivation succeeds then the chosen literal fails and the consistency derivation of the initially chosen denial succeeds. The next pending denial is picked and the consistency derivation starts again.

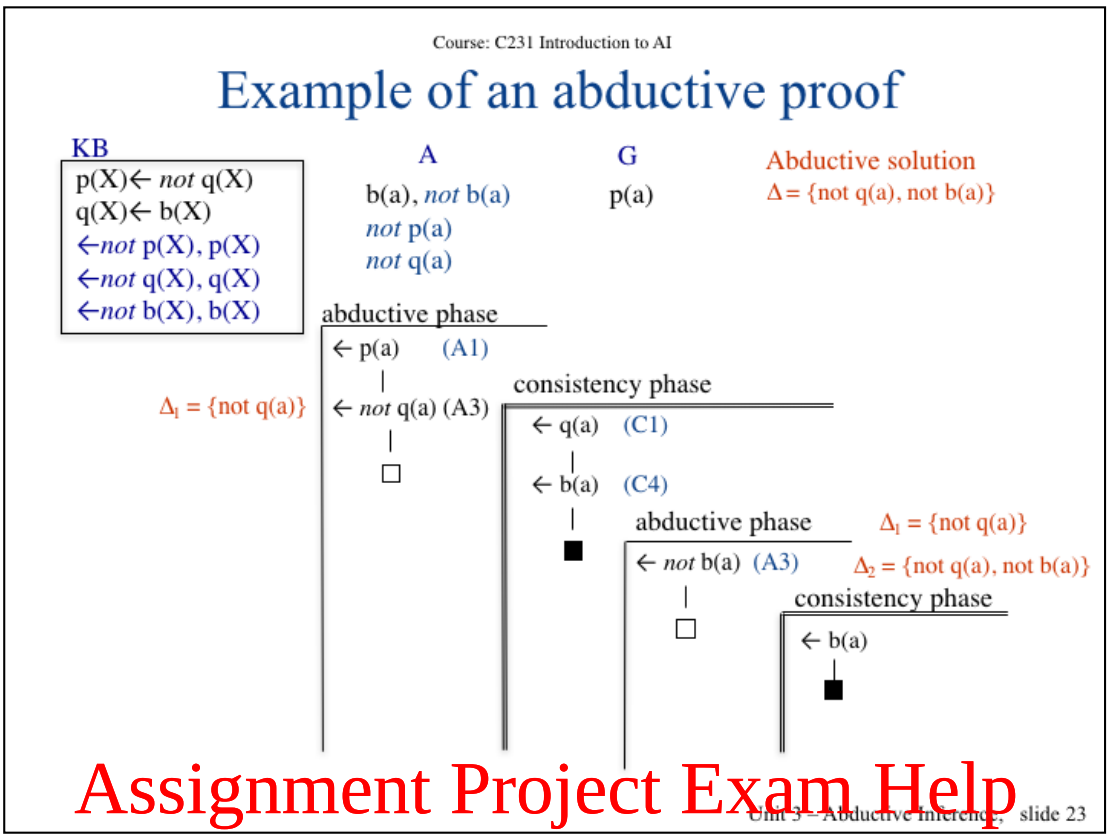
An example derivation is given in the next slide.



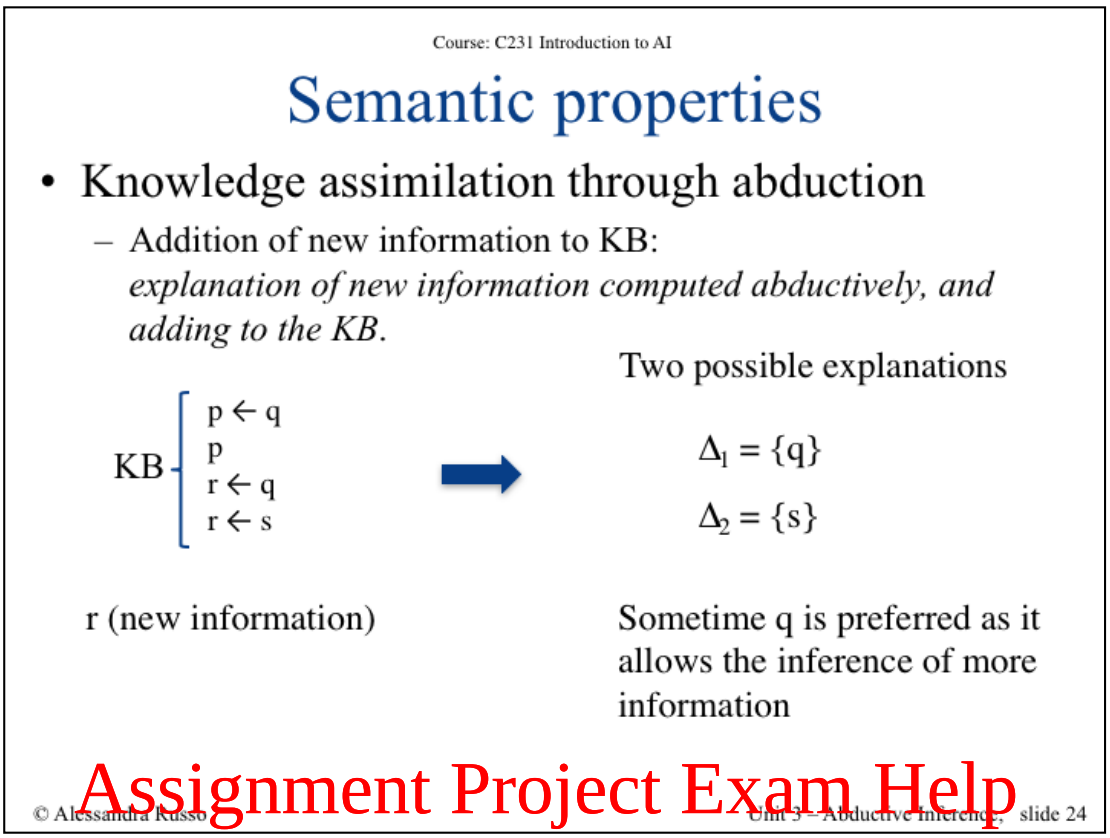
Let's consider a simple example and run through the various steps of the abductive proof procedure. The background knowledge is a normal logic program with no domain specific integrity constraints. The set of abducible includes b(a) as b is the only predicate that it is not defined in BK.

The negation of any predicate that appears in BK has to be added to the set of abducibles and denials of the form ← P(X), not P(X) has to be implicitly added to the set of ICs.

The full abductive framework is given in the next slide, where the implicit constraints and abducibles are colored in blue.



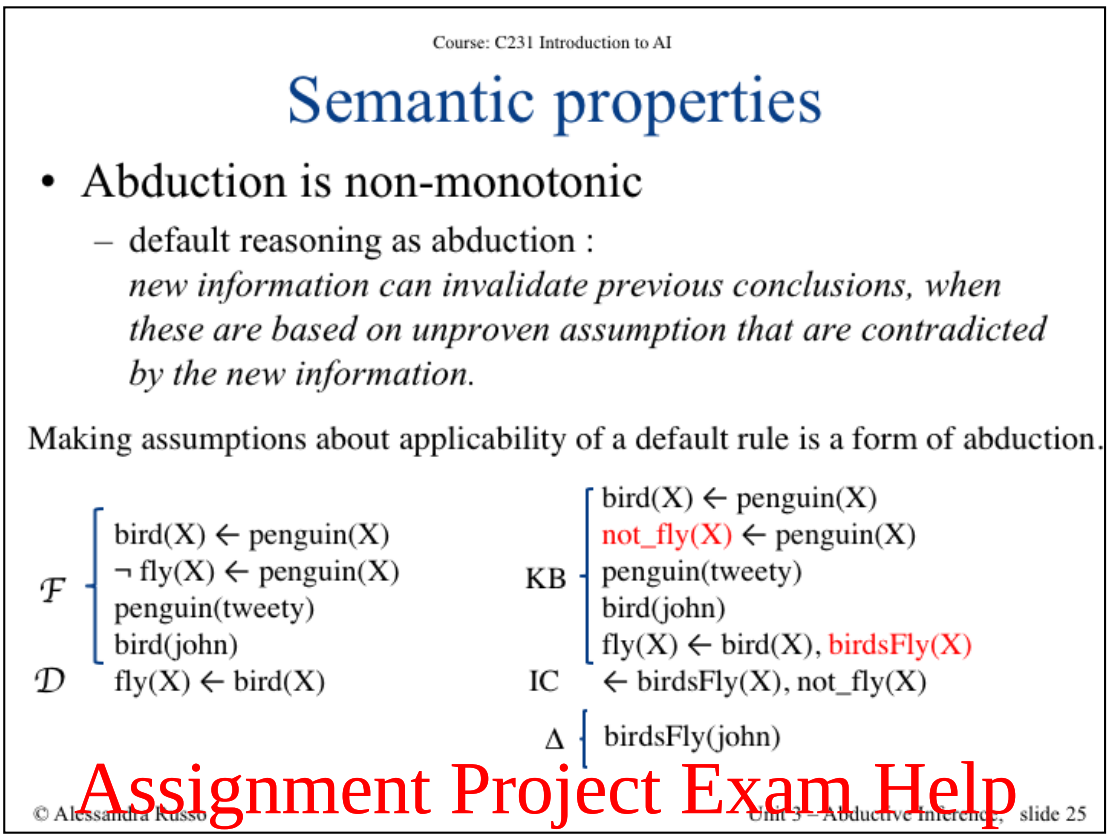
This example proof shows how abductive and consistency phase interleave during the derivation. The label of the specific case applied in the derivation is included next to the related sub-goal.



One of the simple uses of abduction is for knowledge assimilation. Databases are composed of an implicit "database view" and an extensional database of facts. The database view is used to infer further information from a given extensional database of facts. So, when we want to add information to a database view: it can either be the case that the information is already derivable by the database, or it is related to the existing database view through the given intentional database. Similar cases happen when we want to delete a piece of information from a database.

In the second case, we can perform addition/deletion of information from a database by first identify the fact(s) in the extensional database that are related to the new given information and then assimilate the latter by either adding or deleting these facts.

Similar mechanisms could be used to detect inconsistency in a given database and abductively identify the (minimal) causes for the inconsistencies, and resolving it by deleting these causes.



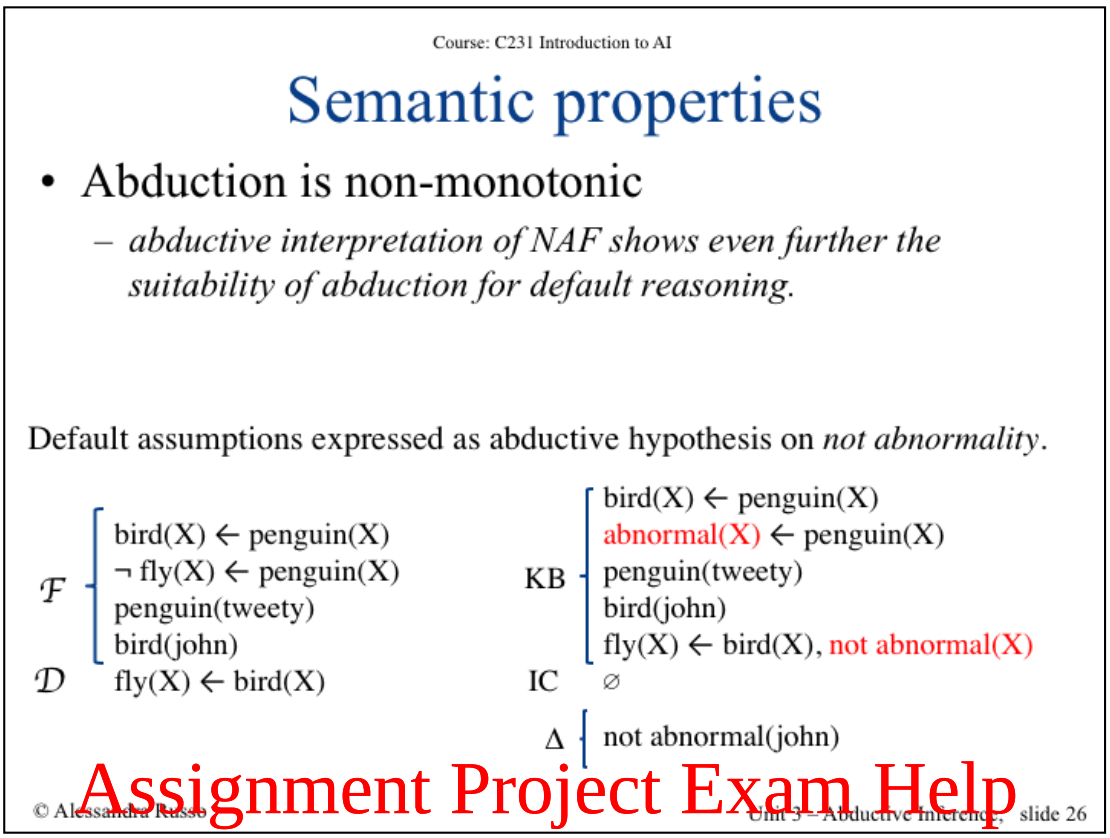
Slightly more elaborate is the case of using abduction for default reasoning.

In the first half of this course, you have since the notion of default reasoning as a mechanism for reasoning in the presence of incomplete information. Poole stated that abductive reasoning provides a natural mechanism for performing default reasoning. In default reasoning, default rules state what is "normally" derivable, namely conditions upon which we expect by default some conclusions unless evidence is provided that shows the opposite of the conclusion. For instance, in the example given on the left of this slide, the default rule (D) states that if for a particular value b of X it is possible to prove bird(b) then by default fly(b) can also be proved as long as this default cannot be contradicted for the same particular value b.

Given the facts in $\mathcal{F}$ the object "john" is a bird, so therefore john flies. But the object tweety, is also a bird, but the default rule instance cannot be applied because it is possible to prove that tweety does not fly (using the second rule).

It is possible to reformulate a default reasoning task in terms of an abductive task with the advantage that the abductive proof procedure can be used to compute the relevant explanations for the defaults made during the inference process. The transformation mechanism consists of naming the default rule and add this label as condition to the default rule. In the example above the default label is "birdsFly(X)". Constraints need then to be added that capture the notion of contradiction with the default assumption. In the example above the contradiction with the default is the possibility of proving not_fly(X). So the integrity constraint ← birdsFly(X), not_fly(X) is added to the program. The default label is then considered to be abducible.

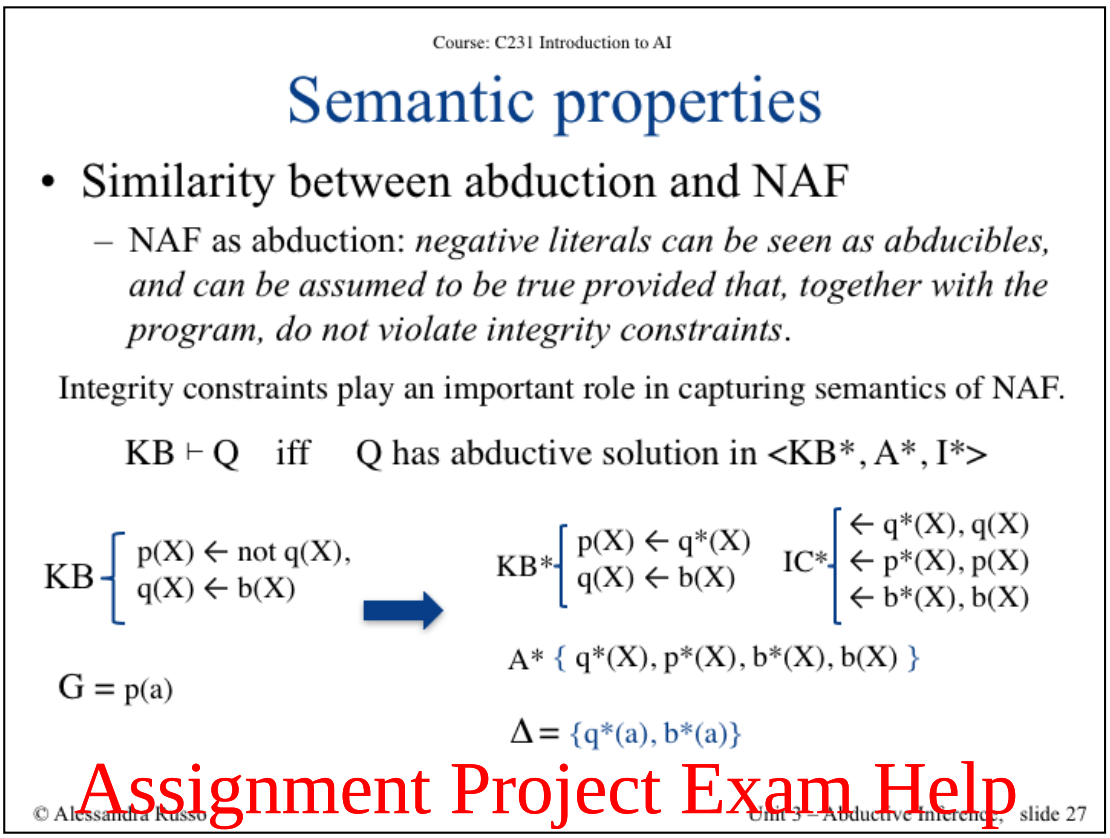
It is easy to see that solutions to the abductive problem given in the right-hand side of this slide are the same as the extensions computed by the default reasoning to prove the same set of conclusions.



In the previous slide, we have shown that default reasoning can be performed by means of abduction by introducing abducible predicates explicitly into the rules. These predicates express explicitly the default assumptions that we make when we compute default consequences from a given default theory. In this slide we show that these assumptions can be expressed in terms of negation as failure, also known to be a powerful mechanism for performing non-monotonic reasoning. The main difference between the two approaches, a part from the fact that the abducibles are opposite in "sign", is that the abductive interpretation of negation as failure already implicitly includes the integrity constraint ← abnormal(X), not abnormal(X). So there is no need to have additional domain specific integrity constraints as we have introduced in the previous slide.

In this case, the default rule can be read as "X flies if it is a bird and is not abnormal". Instead in the previous slide the default rule was stating that "X flies if it is a bird that normally flies".

The abnormality rule given in this slide captures instead the exception cases.

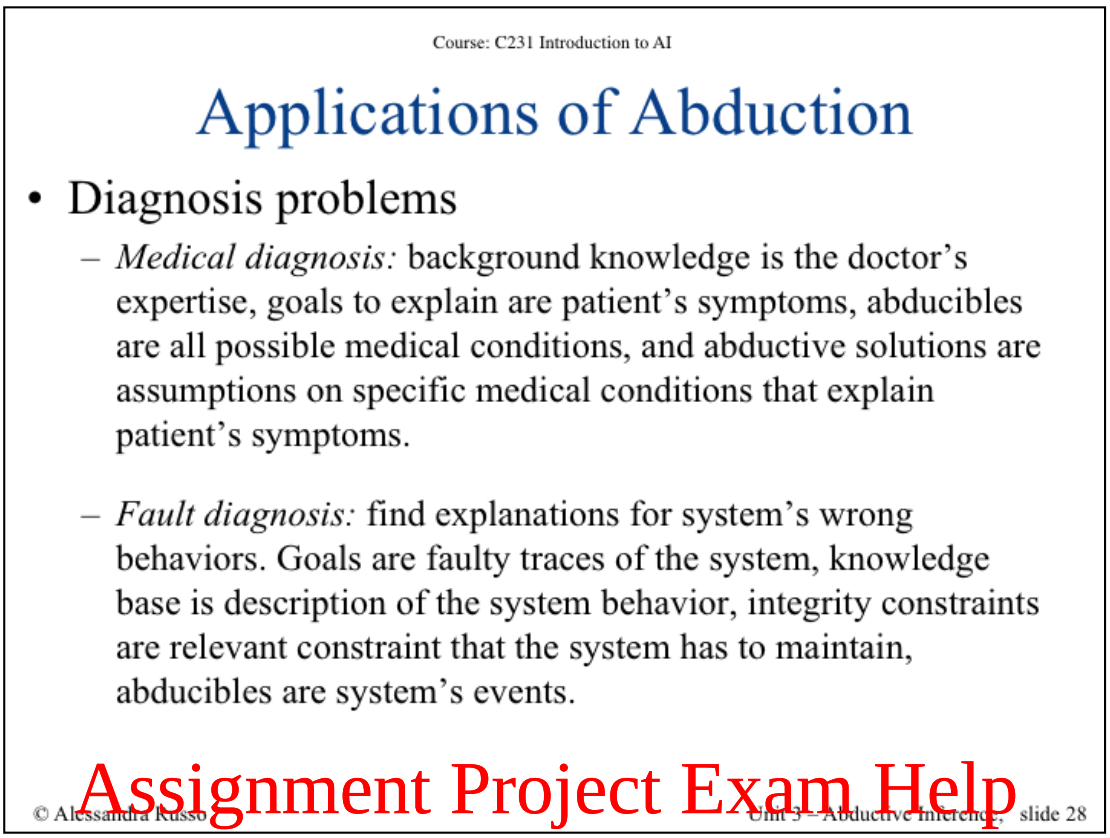


In the previous slide, we have mentioned the concept of abductive interpretation of NAF. In this slide, we show what we mean by this. Consider the background knowledge given on the left of the slide. The SLDNF proof procedure would be able to prove p(a) from the given KB, (i.e. KB $\vdash_{SLDNF}$ p(a)). Similar proof can be generated using abduction.

The idea is as follows: (i) transform the KB into a new KB* but replacing the negative literals that appear in KB with new predicate names (starred). In the example given here, the new predicate names are q*, p*, and b*. The knowledge base KB* is generated from KB by replacing the negative literal "not q(X)" with q*(X).

The important thing is the addition of ICs. For each newly introduced predicate name, an IC is added that captures the meaning of the "starred" predicate, i.e. semantically opposite to the non starred predicate.

Performing abductive proof of p(a), we get as abductive explanations { q*(a)}, which means that to prove p(a) we need to assume that q(a) and b(a) are false (i.e. their derivation fails), which is exactly what happens in the pure SLNF derivation on the left hand side.

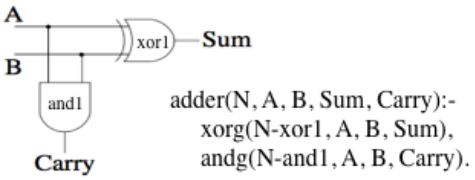


The word "diagnosis" is very general and can have slightly different connotations. In the case of medical diagnosis, it indicates specific medical conditions assumptions that a doctor makes in order to explain a given set of patient's symptoms. A patient describes his/her various symptoms, "I have pain here, and here, but not here ..." These are goals, the only "externally" observable information available to the doctor. The doctor's job is then to give a diagnosis, namely identify a particular disease that can explain the observed symptoms. Sometime, a combination of diseases must be suggested in order to explain the given symptoms.

This can be seen clearly as a case of abduction, whose tuple <KB, A, IC> is defined as the doctor' knowledge expressed as a large collection of rules about which disease and medical conditions may cause which symptoms (KB), the set of all possible medical conditions and diseases (A), and the set of relevant medical constraints (IC). The observations (goal) of an abductive task are the patient's symptoms. Abductive solutions are hypothesis on medical conditions. It is often the case that alternative hypothesis are available. The doctor may choose one, which may well turned up to be not the correct one. Further new observations and tests may confirm it to be the correct assumption or not. This is because abductive reasoning is itself not a sound reasoning process in the sense that even though an abductive assumption may explain a given observation with respect to the given background knowledge and satisfy the integrity constraints, it may be disproved once further evidence is collected. This potential unsound nature of abduction is clearly apparent in the case of medical diagnosis, as a doctor may choose the wrong (out of alternative) diagnoses.

A more general notion of diagnosis is for finding explanations for systems that shows certain wrong behavior. A system can be a structure of interconnected primitive components, and where possible malfunctioning is caused by the malfunctioning of one or more of the primitive components. This type of task can also be formulated as an abductive problem and solved using abductive inference.

Fault Diagnosis: example



adder(N, A, B, Sum, Carry):-
xorg(N-xor1, A, B, Sum),
andg(N-and1, A, B, Carry).

xor(0,1,1).
xor(1,0,1).
xor(1,1,0).
xor(0,0,0).
and(1,0,0).
and(0,1,0).
and(0,0,0).
and(1,1,1).

xorg(N, X,Y,Z):- xor(X,Y,Z).
xorg(N,0,0,1) :- fault(N, s1).
xorg(N,0,1,0) :- fault(N, s0).
xorg(N,1,0,0) :- fault(N, s0).
xorg(N,1,1,1) :- fault(N, s1).
andg(N, X,Y, Z):- and(X,Y,Z).
andg(N, 0,0, 1):- fault(N, s1).
andg(N, 1,0, 1):- fault(N, s1).
andg(N, 0,1, 1):- fault(N, s1).
andg(N, 1,1, 0):- fault(N, s0).

A = {fault(N, s0), fault(N, s1)}

G_1= adder(half_add,0,0,1,0) Δ_1 = {[fault(half_add, s1)]}

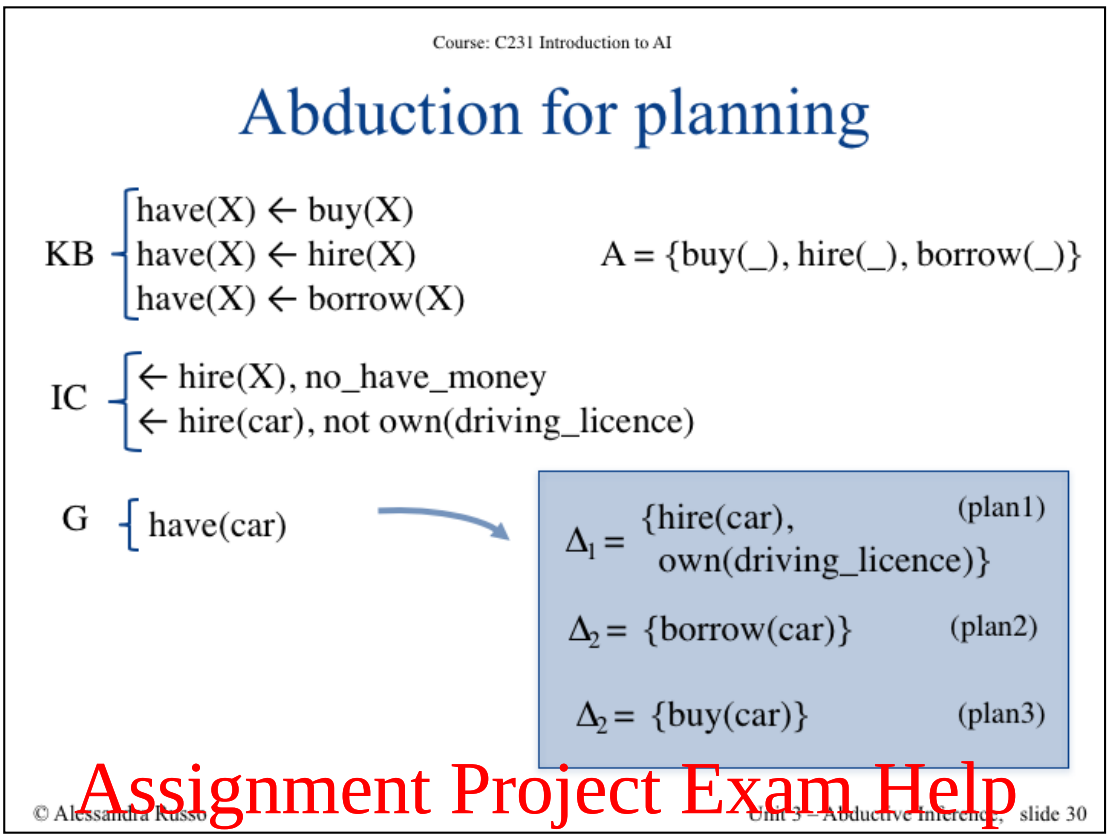
G_2= adder(half_add,0,1,0,1) Δ_2 = { [fault(half_add, s1), fault(half_add, s0)]) }

In this slide, we give an example of fault diagnosis. Consider the above electric circuit. It is composed of two components, an AND-gate and an XOR-gate. Each of the gates works in the usual logic gate manner. When all components work correctly, given certain inputs the circuit should generate certain output. In the case of faulty gates, the output of the gates and therefore of the circuit is not one that is expected. A faulty gate is either be blocked on the status 0 or status 1 independently on the input. This is expressed in this slide by using the predicate fault. A diagnosis task computes assumptions on error states of the gate that cause the circuit to generate wrong outputs. Abduction can be used to perform this task.

The background knowledge is the representation of the circuit given in the slide. The set of abducibles is given by all ground instances of the predicate “fault(.)”, a goal (or observation) is a ground instance of the predicate “adder”, which is the name of the circuit that takes in input two Boolean values (for A and B) and returns in output two Boolean values for Sum and Carry.

When a goal with wrong output is given, the abductive procedure generates explanations about wrong states of the gates that explain the wrong outputs. For instance, if the given goal states that the circuit generates Sum=1 and Carry=0 for A=0 and B=0, the abductive procedure diagnoses that there is a gate blocked in status 1. A richer representation can be also proposed in which each gate has a specific label used as argument of the fault predicate, so that the explanation identifies the wrong status and the

specific gate that is in that wrong status.



Planning is another key area of Artificial Intelligence, which refers to the computational task of finding a sequence of actions that achieves a given goal when executed from a given initial state. Artificial Intelligence planning is a very wide field of AI and a variety of different approaches and techniques have been proposed, starting from the initial STRIPS (Stanford Research Institute Problem Solver) automated planner developed in the 70s, and ranging through more recent approaches of regression planning, forward planning, partial-order planning, hierarchical planning, graph planning, etc.

Abduction can be used to perform regression (or backward) planning where the objective is to start from the goal state and identify relevant actions whose consequences satisfy the current (state) and progressively search the state space backwards continuing from the states in which identified relevant actions can be applied, etc, until a given initial state is reached.

This slides gives a simple planning example, where actions are ground instances of the abducible predicates given in A, goal state is the given goal, initial state can be any and integrity constraints can be used to express constraints on the action applications. These constraints are sometimes expressed as precondition of actions.

Such a simple planning problem can be then expressed as an abductive problem where abducibles are possible ground instances of actions and the background knowledge is a theory describing the domain of the planning problem. In the next lecture we will look at more specific types of planning and formalisms for representing the background knowledge needed by a planning agent.

Summary

- Introduced the notion of abductive reasoning
- Desirable properties
  - Consistency of assumptions
  - Minimality of Explanation
- Algorithm
- Semantic Properties
  - Default reasoning
  - Abductive interpretation of NAF
- Some applications of abduction