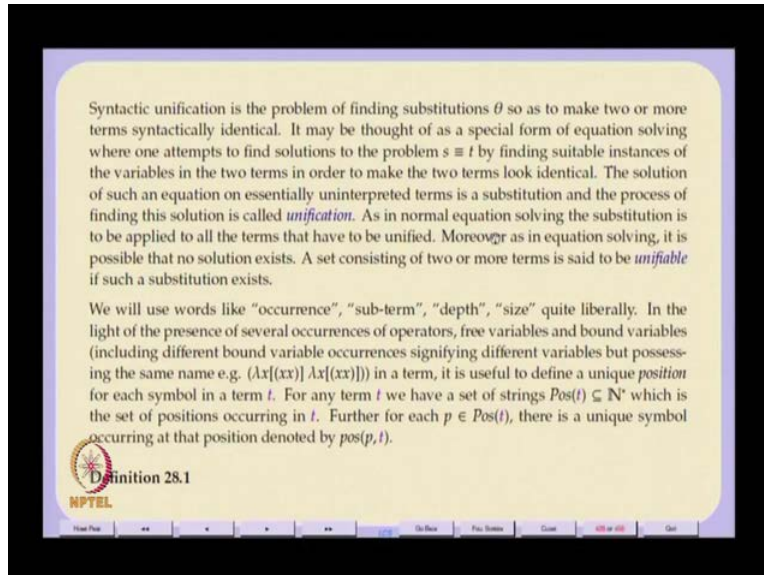


Logic for CS
Prof. Dr. S. Arun Kumar
Department of Computer Science
Indian Institute of Technology, Delhi

Lecture - 28
Unification

(Refer Slide Time: 00:27)



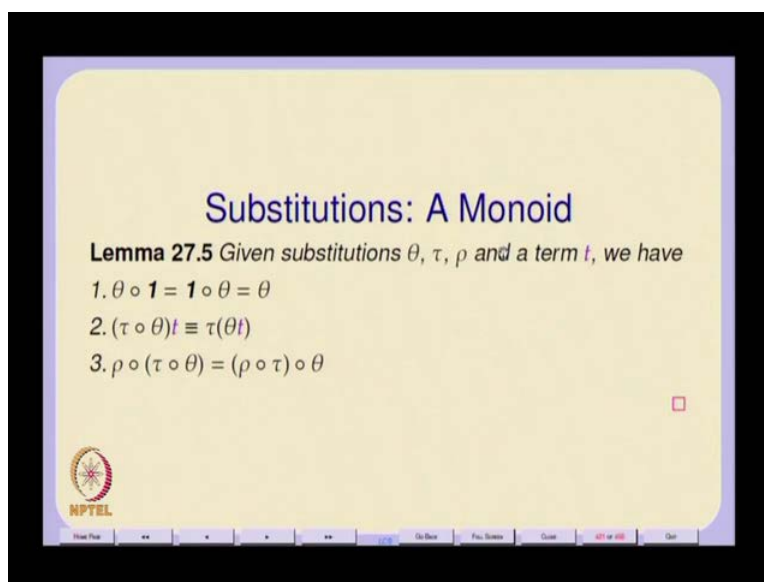
Syntactic unification is the problem of finding substitutions θ so as to make two or more terms syntactically identical. It may be thought of as a special form of equation solving where one attempts to find solutions to the problem $s \equiv t$ by finding suitable instances of the variables in the two terms in order to make the two terms look identical. The solution of such an equation on essentially uninterpreted terms is a substitution and the process of finding this solution is called *unification*. As in normal equation solving the substitution is to be applied to all the terms that have to be unified. Moreover as in equation solving, it is possible that no solution exists. A set consisting of two or more terms is said to be *unifiable* if such a substitution exists.

We will use words like "occurrence", "sub-term", "depth", "size" quite liberally. In the light of the presence of several occurrences of operators, free variables and bound variables (including different bound variable occurrences signifying different variables but possessing the same name e.g. $(\lambda x\{(xx)\} \lambda x\{(xx)\})$) in a term, it is useful to define a unique *position* for each symbol in a term t . For any term t we have a set of strings $Pos(t) \subseteq \mathbb{N}^*$ which is the set of positions occurring in t . Further for each $p \in Pos(t)$, there is a unique symbol occurring at that position denoted by $pos(p,t)$.

Definition 28.1

We start some start now so we were looking at various aspects of simultaneous substitutions.

(Refer Slide Time: 00:36)



Substitutions: A Monoid

Lemma 27.5 Given substitutions θ, τ, ρ and a term t , we have

1. $\theta \circ \mathbf{1} = \mathbf{1} \circ \theta = \theta$
2. $(\tau \circ \theta)t \equiv \tau(\theta t)$
3. $\rho \circ (\tau \circ \theta) = (\rho \circ \tau) \circ \theta$

□

(Refer Slide Time: 00:58)

Proof of lemma 27.5

Proof: We assume $\theta = \{s_1/x_1, \dots, s_m/x_m\}$ and $\tau = \{t_1/y_1, \dots, t_n/y_n\}$ and $\rho = \tau \circ \theta$ as in definition 27.4. Then

1. Trivial.
2. We prove this by induction on the structure of terms. The case of constants is trivial. The induction case will also follow once the cases of simple variables has been proven. So we simply prove this case for simple variables. For any variable x we have the following cases.
Case $x \notin \text{dom}(\theta)$. Then clearly $\theta x \equiv x$ and $\tau(\theta x) \equiv \tau x$. Since the first component of the union in the definition of ρ does not apply, we have $\rho x \equiv \tau x$.
Case $x \equiv x_i \in \text{dom}(\theta)$ for some $i, 1 \leq i \leq m$. In this case $\rho x_i \equiv \tau s_i$ and since $\theta x_i \equiv s_i$ we have $\tau(\theta x_i) \equiv \tau s_i$.
3. For any term t we have from the previous proof

$$(\rho \circ (\tau \circ \theta))t \equiv \rho(\tau \circ \theta)t \equiv \rho(\tau(\theta t)) \equiv (\rho \circ \tau)(\theta t) \equiv ((\rho \circ \tau) \circ \theta)t$$

■

And, we showed that Substitutions the set of substitutions which I think for given signature of omega I call it as S omega V I get somewhere some where I cannot immediately find it. But, so for in any algebraic system of course one of the most basic things is actually equation solving it does not matter in what branch of mathematics we are talking about. After the basic operations the next thing to do is to do equation solving.

(Refer Slide Time: 01:26)

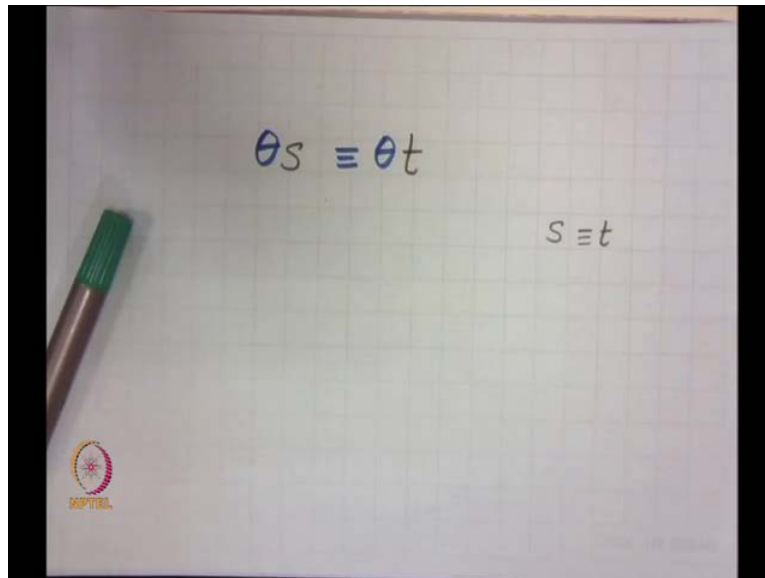
Syntactic unification is the problem of finding substitutions θ so as to make two or more terms syntactically identical. It may be thought of as a special form of equation solving where one attempts to find solutions to the problem $s \equiv t$ by finding suitable instances of the variables in the two terms in order to make the two terms look identical. The solution of such an equation on essentially uninterpreted terms is a substitution and the process of finding this solution is called *unification*. As in normal equation solving the substitution is to be applied to all the terms that have to be unified. Moreover as in equation solving, it is possible that no solution exists. A set consisting of two or more terms is said to be *unifiable* if such a substitution exists.

We will use words like "occurrence", "sub-term", "depth", "size" quite liberally. In the light of the presence of several occurrences of operators, free variables and bound variables (including different bound variable occurrences signifying different variables but possessing the same name e.g. $(\lambda x\{xx\}) \lambda x\{xx\}$) in a term, it is useful to define a unique *position* for each symbol in a term t . For any term t we have a set of strings $\text{Pos}(t) \subseteq \mathbb{N}^*$ which is the set of positions occurring in t . Further for each $p \in \text{Pos}(t)$, there is a unique symbol occurring at that position denoted by $\text{pos}(p, t)$.

Definition 28.1

And, unification is essentially like equation solving. I mean what we are saying is supposing I have two different terms s and t . Then I look at I want to find suitable substitutions which make those two terms syntactically identical.

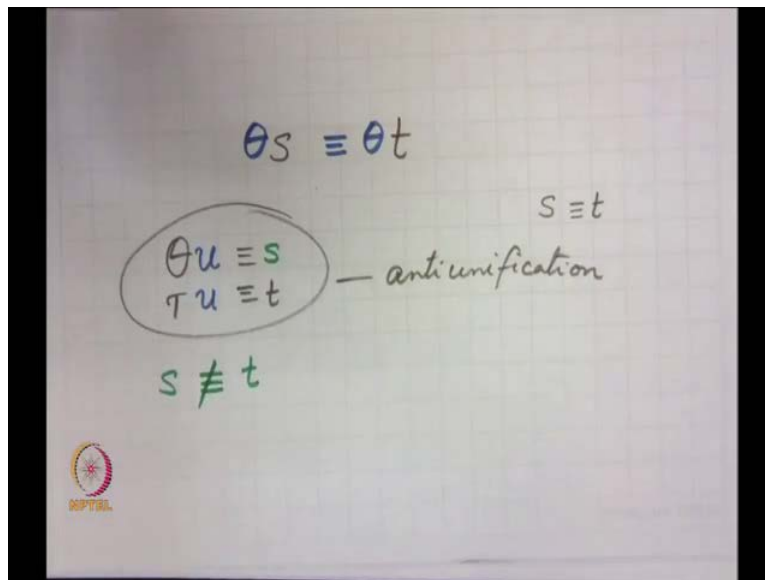
(Refer Slide Time: 01:53)



So, if I look at two terms and I am saying that I have two terms s and t and, when I say that I unify them. Then, I am essentially finding some substitution θ which will ensure that θs is syntactically identical to θt . That, is in the sense the same as trying to solve the equation s is syntactically equal to t . Where, you are looking for essentially term patterns for the variables so that when you substitute then the two sides become syntactically identical.

So, we will essentially talk about so this is form of if you if you take two terms and want to make them syntactically identical by just substituting terms for free variables I mean you are working all entirely within a universe I mean. So, the only the value domain is also just terms syntactic terms. So, you are saying what kinds of substitutions can, I perform for the variables and the two terms. Such that the two terms become look identical and there is a other process one can think of a process like anti unification. Which, says that two terms look different what could have been a term.

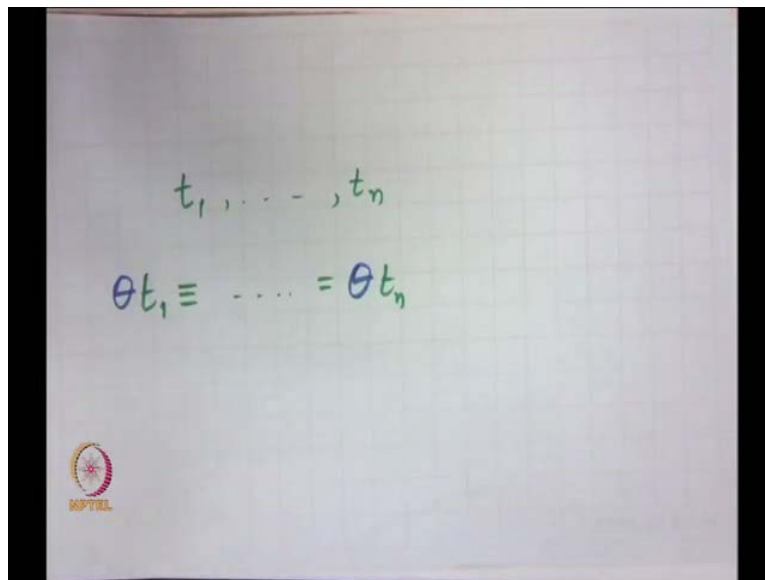
(Refer Slide Time: 01:53)



So, that if you gave separate substitutions to the two terms you get two different things. So, what I am saying is so there are two terms as in t let us say which are syntactically different. So, S is not syntactically different to t . What kinds of terms exist? Such that let us say a term u such that if I applied some substitution θ τu I get S . And, I apply some other substitution τ to u I get let us say t . So, this is supposing so this, is a process called anti unification one thing of course is that in the case of anti unification you can always just have a single variable x . Let us say that is that's an anti unifier you know you do an appropriate substitution for x you get S .

Some, other substitution for x you get y you get t I mean that is. But, you can ask a more subtle question, What is the most specific kind of term u that you can find which preserves all the commonality of structure of S and t ? And, then what is the most specific kind of term for which if you apply to some minimal substitutions θ and τ you will get S and t I means that is one thing. So, that is anti unification of course is more useful in the anti universe we, will worry about unification in the current universe. So, which is essentially like a equation solving but we will generalize it further.

(Refer Slide Time: 06:06)




So, let us just say you got n different terms and you are looking at some kind of substitution remember that free variable substitutions are length and depth in non decreasing. So, you are looking for some substitution θ such that, θt_1 is syntactically is equal to θt_2 is syntactically equal to θt_n . So, this is like generalizing the concept of equation solving to unify n terms. So, at the moment we will just regard this as simply an interesting problem to solve I, mean thus. So, and many of you may have actually come across these things before in a other areas like for which unification is very useful in addition to logic programming namely namely things like type assignment and functional programming languages. And, finding the most general or principal type scheme for a for a type for a lambda term in which the types are not pre declared principal polymorphic type scheme let us say here.

(Refer Slide Time: 07:37)

t	depth	size	ST	Pos
c	1	1	$\{t\}$	$\{e\}$
x	1	1	$\{t\}$	$\{e\}$
$o(t_1, \dots, t_n)$	$1 + \text{Max}_{1 \leq i \leq n} \text{depth}(t_i)$	$1 + \sum_{i=1}^n \text{size}(t_i)$	$\{t\} \cup \bigcup_{i=1}^n \text{ST}(t_i)$	$\{e\} \cup \bigcup_{i=1}^n i.\text{Pos}(t_i)$


- The functions given in the table above are defined by induction on the structure of term t . e is the empty word on strings, \cdot is the concatenation operator on strings and $i.\text{Pos}(t_i) = \{i.p \mid p \in \text{Pos}(t_i)\}$.
- $s \sqsubset t$ iff $s \in \text{ST}(t)$ is the **subterm relation** on terms. s is a **proper subterm** of t (denoted $s \sqsubset t$) iff $s \sqsubset t$ and $s \neq t$.
- For any t , the subterm at position $p \in \text{Pos}(t)$ is denoted $t|_p$ and defined by induction on p as follows: $t|_e \equiv t$, and for $t \equiv o(t_1, \dots, t_n)$, $t|_{ip} \equiv t_i|_p$ if $p = i.p' \in \text{Pos}(t)$
- For any term t and any position $p \in \text{Pos}(t)$, $\text{sym}(p, t)$ yields the symbol at position p in the term t .
- The position e is called the **root position** and the symbol at the root position is called the **root symbol**. Hence $\text{root\text{sym}}(t) = \text{sym}(e, t)$ and for any position $p \in \text{Pos}(t)$, $\text{sym}(p, t) = \text{root\text{sym}}(t|_p)$.



But, we let us look at this as a problem from which we can go on to logic programming. So, there are here are some basic things for terms their depths size sub terms. And, there is a peculiar thing that I have defined called positions I mean. So, we look at a term as a abstract syntax tree so you take any tree I can uniquely name the nodes by a position.

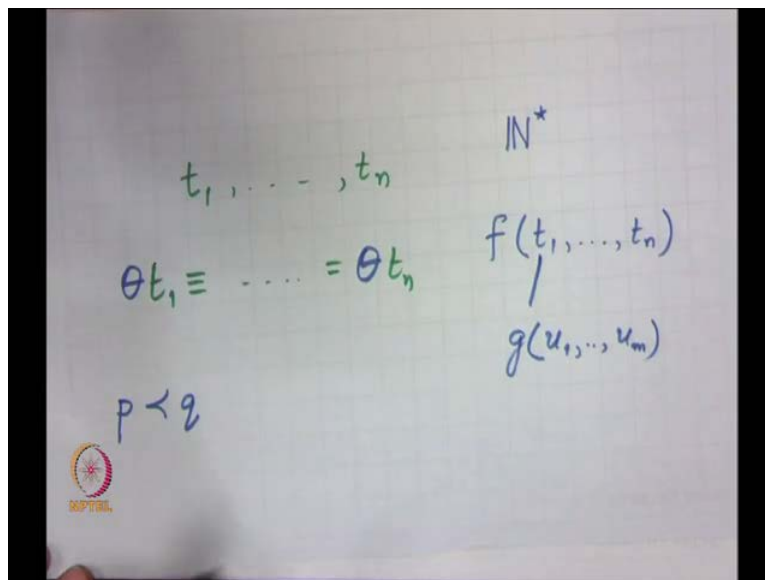
(Refer Slide Time: 08:06)

$$t_1, \dots, t_n \quad \mathbb{N}^*$$

$$\theta t_1 \equiv \dots \equiv \theta t_n \quad \begin{matrix} f(t_1, \dots, t_n) \\ | \\ g(u_1, \dots, u_m) \end{matrix}$$


Which, is a string drawn from N^* where the root position is the empty string and as you go down the tree you prefix current position. So, for example if I have a function like this then, if I have term like this then, at the root position I have f at position one I have t_1 at position n I have t_n . Now, supposing t_1 is of the form g of some u_1 to u_m . Then, at position 1 I have g at position 1,1 I have u_1 I have the root of u_1 at position 1m I have u_m and so on so on. So, we can define positions in a tree also in this we can as strings over the naturals based on the area t here. As, in there are symbols at the root position there is notion of sub term. So, the notion of sub term is of course obvious so our position this position has a string.

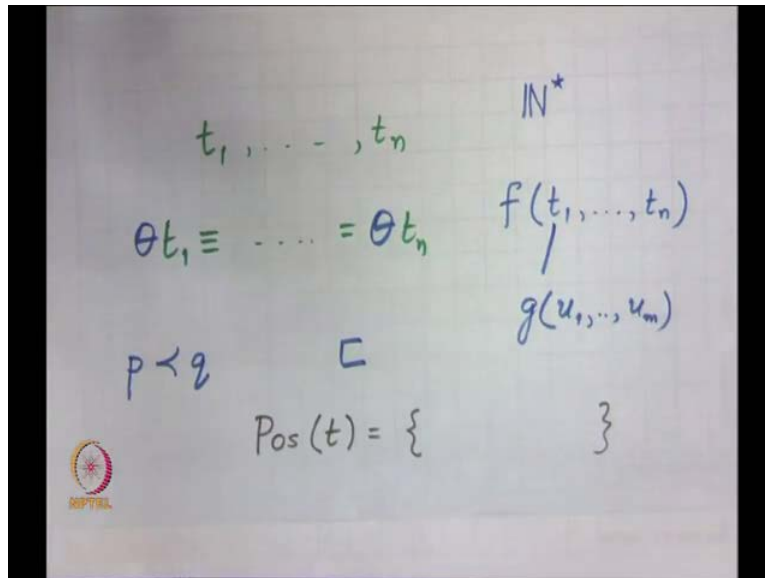
(Refer Slide Time: 09:32)



So, it is a, it has a prefix ordering so I can talk of position p which is prefix of a position q on a tree. And, essentially if you take this sub tree rooted at p the sub tree rooted at q is a sub tree of the sub tree rooted at p . So, I have sub tree relation like this, a proper sub tree relation like this which does that but I think I made a it is quite possible that I have made a mistake somewhere but we will when we come to that I will let you know. So, we will talk about let us define some basic things so we will say a nonempty.

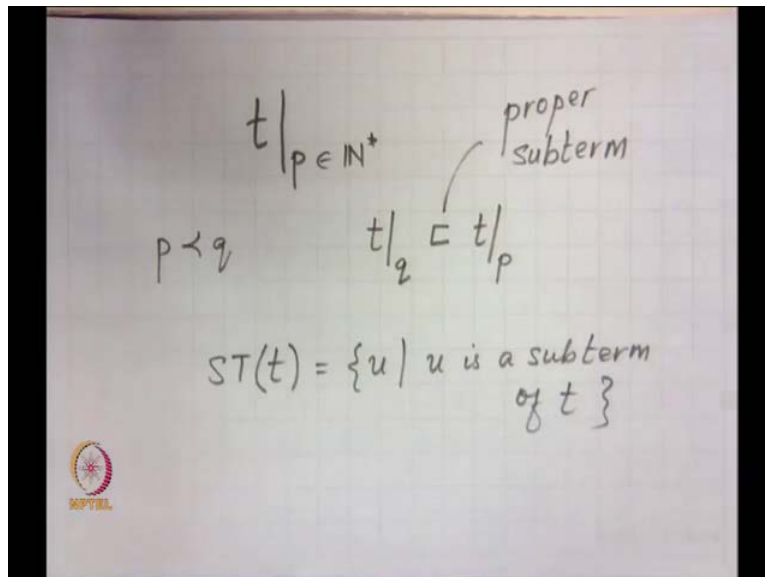
Student: When union (Refer Time: 10:20)

(Refer Slide Time: 10:31)



Which, you know here this is a, for any for any term t a pos of t is the set of all positions that are valid that, are that in that term that is all this here. So, the concatenation is here this i dot pos of t_i here, that that dot is a concatenation operation have I put the prefix here.

(Refer Slide Time: 11:15)



So, here is where I made the mistake it should be. So, t so for any term t and a valid position p which is a string belonging to n star t with this vertical thing subscripted with p is a sub tree

rooted at p here. So, it is so what is if P is a prefix of q then, essentially what we are saying and p and q are valid positions in t. Then, essentially what we are saying is t the sub tree rooted at the position q would be a proper sub term of the sub tree rooted at position p I mean that is the correction that is needs to this is the sub term. This is, the proper sub term relation here and ST of t is the set of all sub terms set of all u such that u is a sub term of t. In particular the sub term relation is reflexive so t itself is a sub term of itself here that is.

So, here is this mistake needs to be corrected. So, we will talk about unifiability so a nonempty finite set of terms t_1 to t_n is said to be unifiable if there exists a substitution θ . Such, that $\theta t_1 \theta$ makes all of the terms look alike. And, of course all substitutions apply only for free variables here. So, actually it is a good idea to if we had bound variables and so on in a even in our term language. Which, is quite possible if you consider things like summations and so on and so forth. Then, of course what we should say is it this is alpha equivalent to so this equivalence syntactically equivalence will be alpha equivalence and not pure syntactically equivalence. So, then we would say that θ is a unifier here.

(Refer Slide Time: 13:19)

Unification Examples:1

Example 28.4 Let f and g be distinct binary operators and $x, y, v, w \in V$.

1. The terms $f(x, y)$ and $f(v, w)$ may be unified by the substitution $\theta = \{x/v, y/w\}$ since $\theta f(v, w) \equiv f(x, y) \equiv \theta f(x, y)$. They may also be unified by $\theta^{-1} = \{v/x, w/y\}$.
2. Let r, s, t be any three terms. Then $f(x, y)$ and $f(v, w)$ may be unified by the substitution $\tau = \{g(s, t)/v, f(r, r)/w, g(s, t)/x, f(r, r)/y\}$.
3. The terms $f(x, y)$ and $f(y, x)$ may be unified by $\chi = \{x/y\}$ since $\chi f(x, y) \equiv f(x, x) \equiv \chi f(y, x)$.

NPTEL

There, are some simple examples let f and g be distinct binary operators and x, y, v and w be variables. You, say you take f of x, y and f of v, w they can be unified by this substitution θ which for example x for v and y for w is one possibility. They, can also be unified by θ

inverse in which in what the domain and the range. Which, is possible in this case when the substitutions are what are known as pure variables substitution here where the variable is replaced by variable and not by complex term.

So, then theta inverse is v for x and w for y that that also will work to unify them. So, in that sense theta f can be regarded as some kind of a solution of this equation f of x y syntactically equal to f of v , w . For any three terms r , s and t we can take f of x , y and f of v , w and you can unify them by this substitution. Which gives you more complicated substitutions I take g of s , t for v , g of s , t for x f of r , r for w and f of r , r for y and I can unify them. So, basically what you saying so the equation solving it is possible that there are there is more than one solution. There are there could be a large number of solutions. And, in certain cases of course like this is a peculiar case when you want to take when you want to make basically x , x to be the same as what kind of substitution will you get. So, if I take x for y then, I get the both syntactically identical. So, f of x , x is of course an instantiation of f of x , y it is like a particular case of f , x , y even though x is a variable.

(Refer Slide Time: 15:23)

Unification Examples:2

Example 28.5 Let f and g be distinct binary operators.

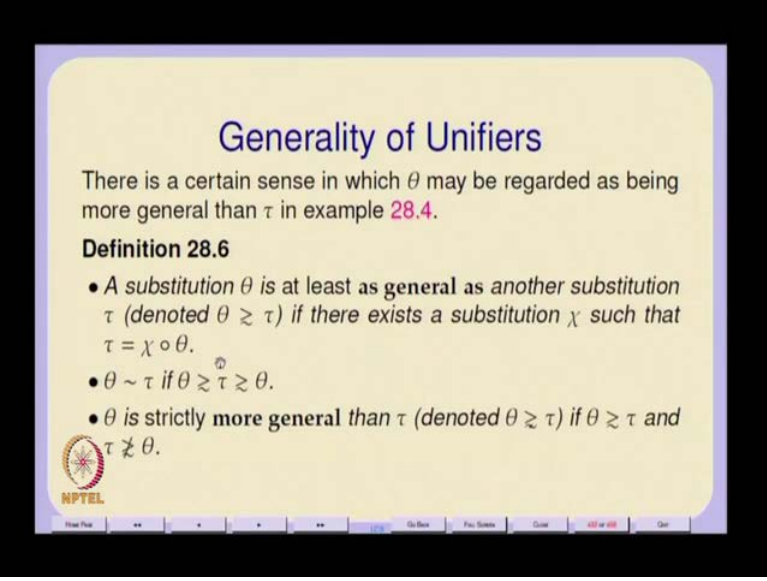
1. The terms $f(x, y)$ and $g(x, y)$ cannot be unified by any substitution.
2. The terms $f(x, y)$ and $f(y, x)$ cannot be unified by $\rho = \{x/y, y/x\}$ since $\rho f(x, y) \equiv f(y, x)$ and $\rho f(y, x) \equiv f(x, y)$. Hence $\rho f(x, y) \not\equiv \rho f(y, x)$.

NPTEL

So, and then if I take two terms f of x , y and g of x , y then they cannot be unified under any circumstances. So, what this means is that this equation solving is as doubtful I mean. So, it is possible that there are no solutions to this equation solving process here because of the fact that f

and g are distributing binary operators you can never make these two terms look the same by merely doing substitutions. If I take f of x, y and f of y, x they cannot be unified by a ρ . I mean we saw previously that they can be unified by x for y . But, they cannot be unified by ρ which switches which exchanges x and y for example. So, it does both x for y and y for x because each case when you apply this ρ to these two you will get different terms here. So, let us so this is the simplest way of looking at unification how you have to be sort of careful in unification. Now, so if you look at these unifiers so there is a certain sense in which this θ and θ inverse are more general than let us say this term here. So, what happens in the case of τ is, it makes a solution far more specific by using greater structure. Then, is a absolutely essential. I mean when you have a simple variable there you can allow for further substitutions to be made later which make it more specific. Where, as if that variable has to be replaced by some term unnecessarily then you are actually committing it to far more specific substitution than is absolutely essential. So, what we are so you were to take this so this for example this g, s, t for v as suppose to x for v is this g, s, t for v is very specific commitment you know which basically. Where, as x for v if you do the substitution x for v at some later stage if you have to do replace at x by g, s, t you could still do that with an, other substitution. So, there is some kind of minimum replacement that you do, to make the two terms equal that minimum replacement is such that it makes a minimum amount of commitments to a structure. So, in that sense this θ and this θ inverse are more general than let us say this g, s then this star. Similarly, f, r, r is a very specific commitment which makes it very specific. So, if it had been f, z, w f, y, z for example. Where, y and z are variables that would be more general than f, r, r or it will be more general than f, r, s for example.

(Refer Slide Time: 19:11)




Generality of Unifiers

There is a certain sense in which θ may be regarded as being more general than τ in example 28.4.

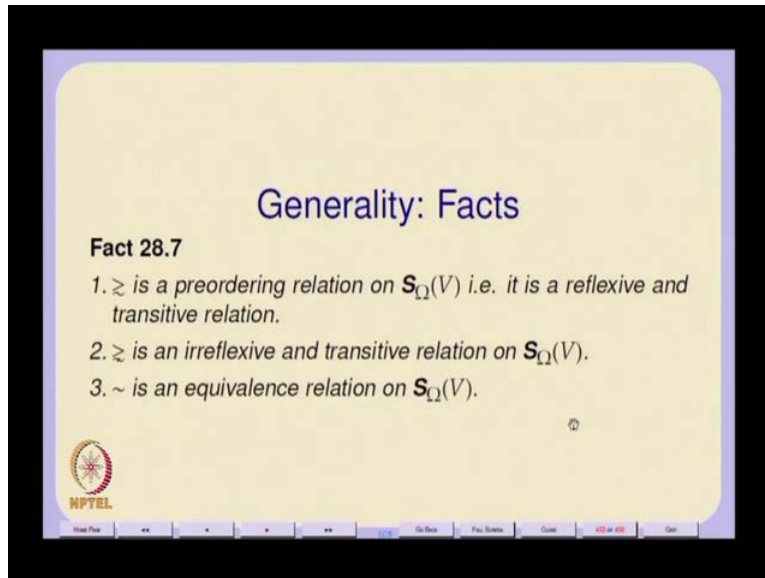
Definition 28.6

- A substitution θ is at least as general as another substitution τ (denoted $\theta \succeq \tau$) if there exists a substitution χ such that $\tau = \chi \circ \theta$.
- $\theta \sim \tau$ if $\theta \succeq \tau \succeq \theta$.
- θ is strictly more general than τ (denoted $\theta \succ \tau$) if $\theta \succeq \tau$ and $\tau \not\succeq \theta$.

 NPTEL

So, there is a notion of Generality of unification which essentially is comparative. So, a certain substitution theta is at least as general as another substitution tau. And, we will say that theta is at least as general as tau if there is exist a substitution chi such, that tau is obtained from theta by applying chi by doing a composition of substitutions. So, then as you do more and more substitutions you are likely to become more and more specific in the term. You, constrain the term structure more and more. We, would say that theta is as general as tau if, theta and tau are each as general as each other. So, this is something that typically happens can happen because of the fact that you are using different sets of variables in the substitution rather than. And, so therefore in general if, theta is at least as general as tau. What, you expect is that for any term t in which you apply theta and t the resulting terms theta t and tau t will differ at most in the names of variables. That is otherwise they will have the same structure they will differ the leaf nodes in terms of variables. We, would say theta is strictly more general then tau if, theta is more at least as general as tau but it is not equal to it is not as general as it.

(Refer Slide Time: 21:02)



Generality: Facts

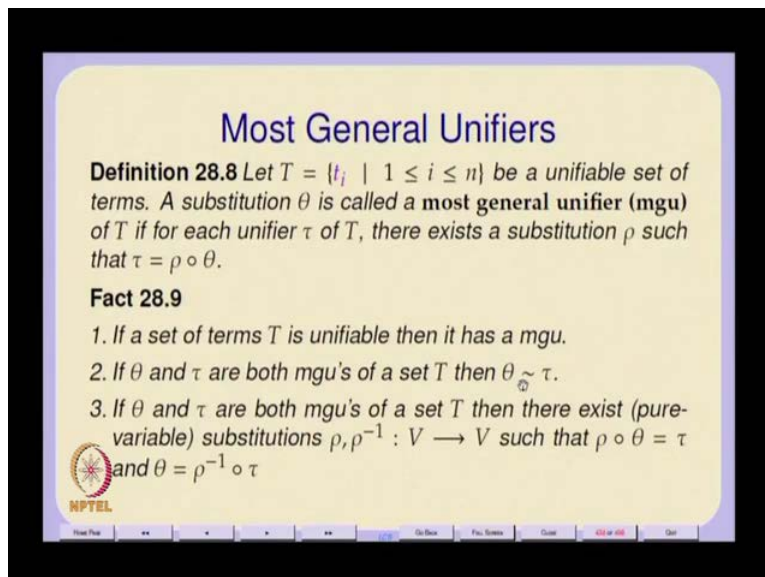
Fact 28.7

1. \succeq is a preordering relation on $S_{\Omega}(V)$ i.e. it is a reflexive and transitive relation.
2. \succ is an irreflexive and transitive relation on $S_{\Omega}(V)$.
3. \sim is an equivalence relation on $S_{\Omega}(V)$.

HPTTEL

So, this Generality relation is a preordering a so in the sensor it is a reflexive and transitive relation. And, this equally general at relation is an equivalence relation it is a reflexive transitive and symmetric. This, strictly more general is an irreflexive and transitive relation.

(Refer Slide Time: 21:35)



Most General Unifiers

Definition 28.8 Let $T = \{t_i \mid 1 \leq i \leq n\}$ be a unifiable set of terms. A substitution θ is called a **most general unifier (mgu)** of T if for each unifier τ of T , there exists a substitution ρ such that $\tau = \rho \circ \theta$.

Fact 28.9

1. If a set of terms T is unifiable then it has a mgu.
2. If θ and τ are both mgu's of a set T then $\theta \sim_{\theta} \tau$.
3. If θ and τ are both mgu's of a set T then there exist (pure-variable) substitutions $\rho, \rho^{-1} : V \rightarrow V$ such that $\rho \circ \theta = \tau$ and $\theta = \rho^{-1} \circ \tau$

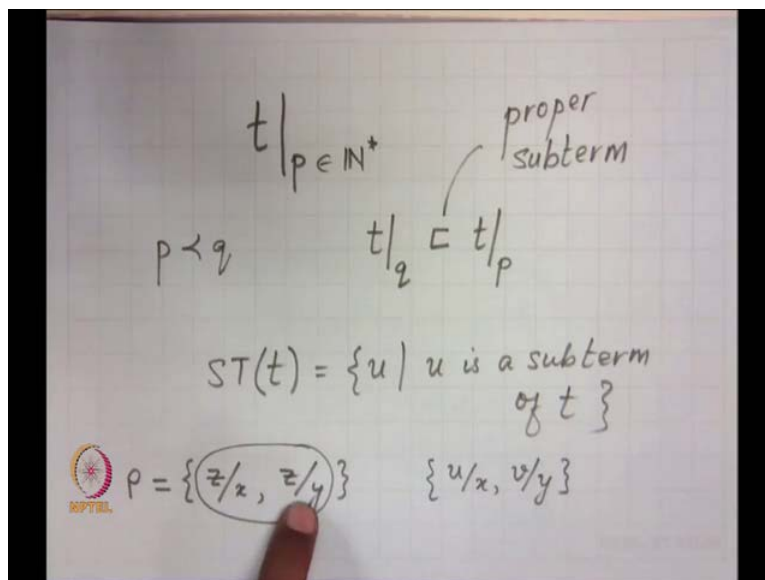
HPTTEL

So, that is we will look at it this way. So, now so what you have to have is you have this given some terms. And, given some substitutions you have a notion of more general which is

essentially if you quotient it on that equivalence basically you got a partial ordering relation on the set of all possible substitutions. So, there is an ordering where is a partial ordering on the set of all substitutions. And, we can always look for the highest element the most general element in this ordering that is what we are looking at. So, let T is consisting of a set of terms t_i be a unifiable set by the way we know that some of these sets of terms may not be unifiable. So, let us take a unifiable set and a substitution θ is called a most a most general do not know whether it is a most it is not unique. Because, there are other there could be others which are equally general and which are still different from θ .

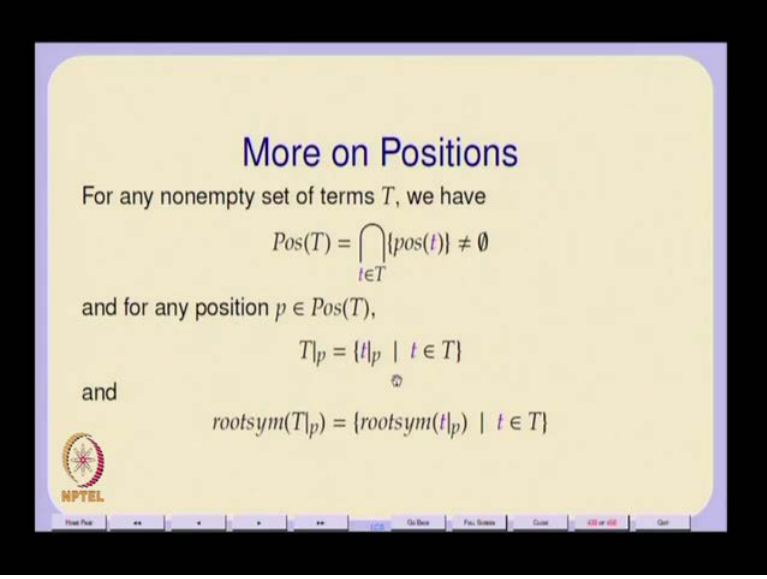
So, a substitution θ is called a most general unifier of T if for each unifier τ there exists a substitution ρ such that τ is ρ comes with θ . So, if it is possible to if every other substitution which unifies this set of terms can be obtained from θ by doing a further substitution then you would say that you got essentially a most general unifier. So, given a set of terms if it is unifiable then there is a most general unifier somewhere in the partial ordering of unifiers of substitutions you would find some unifier. And, it could be a most general unifier if there if there are two most general unifiers then there must be equally general. And, if they are both equally general if then they essentially differ in variable naming. So, I do a variable renaming I should be able to get one from the other that is. So, that is what this last property says. So, there is what is known as a pure variable substitution ρ in if, ρ is a substitution which is replaces variables by variables.

(Refer Slide Time: 24:26)



Then, there is also a rho inverse usually unless rho is of the form z for x z for y this should not happen if their domain and range are equally numerous then, a rho inverse might exist. And, notice that a substitution like this makes a makes a term more specific. Then, a substitution in which I have different variable names for each element of their domain this actually collapses to variables and therefore imposes a certain syntactic structure. Which, is more specific than this because this can be obtained from this by another substitution like this. So, I mean so the existence of rho inverse is not guaranteed even for pure variables substitutions firstly, secondly the generality is important. So, here I have so I have loosely said that basically there should be a variable renaming. What we are essentially saying is it that they are both equally general then they differ at most in the names of the variables. But, there is a one two one correspondence between the variables occurring in one and the other with respect to the positions. So, that positions are preserved that otherwise you would have you would have certain collapsing identifications like this z for x and z for y. So, what we are saying is that when if theta and tau are both equally general then not only do they differ only in the variables. But, variables basically means leaf nodes the labels on the leaf nodes of the abstract syntax trees. But, there is a one to one correspondence between the positions of the leaf nodes at which the variables occur between one tree and the other. And, that position is preserved so there is an isomorphism there between the leaves which, has to be preserved.

(Refer Slide Time: 27:10)



More on Positions

For any nonempty set of terms T , we have


$$Pos(T) = \bigcap_{t \in T} \{pos(t)\} \neq \emptyset$$

and for any position $p \in Pos(T)$,

$$T|_p = \{t|_p \mid t \in T\}$$

and

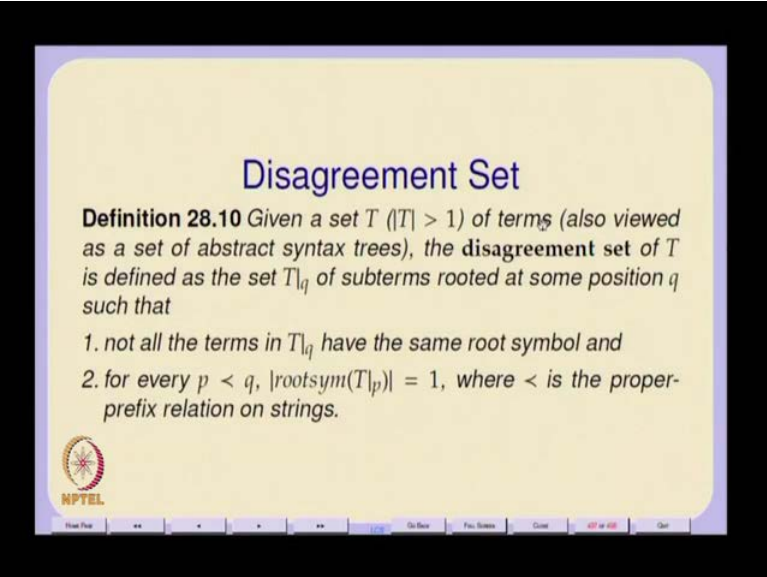
$$rootsym(T|_p) = \{rootsym(t|_p) \mid t \in T\}$$



Slide navigation controls: Home, Back, Forward, Stop, Full Screen, Close, 27 of 48, Quit

So, this is a Motion on Position and there is some notion of a root symbol at a position is a function some set we define. So, one thing of course is that the set of positions for any term is nonempty and there is always a root position for example. And, the sub tree rooted at that root position is it is a term itself. And, this is just to lift it from individual terms to set sub terms. So, this for a set of terms T I am looking at all the positions that are common to all the terms in there in tree.


(Refer Slide Time: 27:56)



Disagreement Set

Definition 28.10 Given a set T ($|T| > 1$) of terms (also viewed as a set of abstract syntax trees), the **disagreement set** of T is defined as the set $T|_q$ of subterms rooted at some position q such that

1. not all the terms in $T|_q$ have the same root symbol and
2. for every $p < q$, $|rootsym(T|_p)| = 1$, where $<$ is the proper-prefix relation on strings.



Slide navigation controls: Home, Back, Forward, Stop, Full Screen, Close, 27 of 48, Quit

So, the process of unification essentially requires that we compare these two trees in some order we traverse from the root and compare them. So if the root symbol of the two trees is different in the sense that there is some function symbol their not a variable symbol. But, a function symbol there you can then and if two function symbol are different of course the two trees are not unifiable. And, if the two trees are not unifiable in a set consisting of many trees then that set is not unifiable. So, given a set T of terms also we would a set of abstract syntax trees the disagreement set of T is defined as the sub terms rooted at some position q . Such, that for all prefixes of q the two trees the thus all the trees look identical. For, all prefixes presiding q the trees look identical. And, it there is a disagreement if at position q there are at least two trees with have different symbols rooted at q at position q . So, this is the proper prefix relation.

(Refer Slide Time: 29:33)

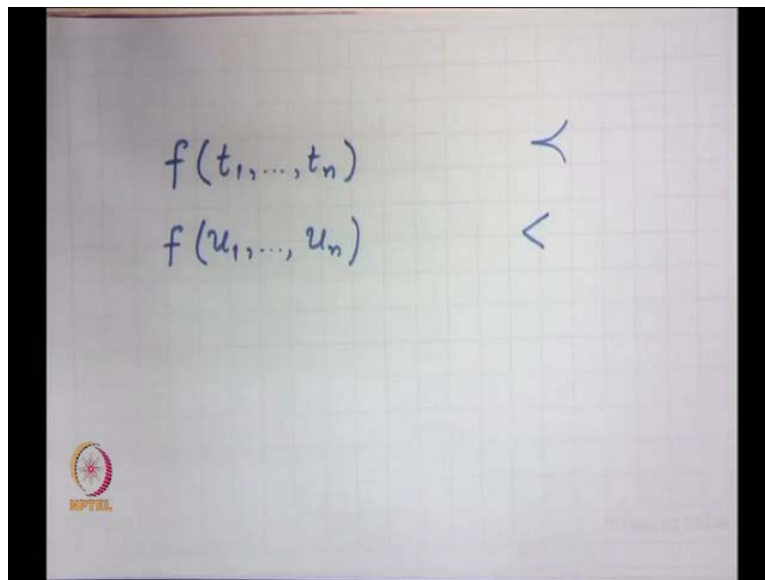
We have seen that $Pos(t)$ for any term t is partially ordered by the relation $<$ which is faithful to the proper sub-term ordering on $ST(t)$. For the purpose of specifying the unification algorithm, it is useful to define a *total order* on the positions of terms which is consistent with $<$. Intuitively if $u = o(t_1, t_2, \dots, t_n)$ we would like to specify *recursively* that

- the root position u precedes the root positions of all the subterms t_1, \dots, t_n , (which is taken care of by the prefix ordering $prec$ on positions) and
- for each i, j such that $1 \leq i < j \leq n$, the position of the root of t_i precedes that of t_j in the total ordering.
- If $i < j$, then the position of the root of any proper subterm of t_i precedes the position of any subterm of t_j (including the root).

If all operators in Ω are always used in prefix form then each term may also be regarded as a string in $(\Sigma \cup \{(\cdot)\})^*$. The ordering $<$ on $Pos(t)$ simply becomes the prefix ordering on the well-formed terms of $\mathcal{T}_0(V)$.

So, one thing one standard thing that most logic books use is that they do not actually at them as abstract syntax trees. But, they look at them as terms involving terms involving parenthesis I mean you can think of.

(Refer Slide Time: 29:52)



Supposing I have this let us say t have a set consisting of just these two terms. Where, I have not specified what the terms t_1 to t_n are what these terms u_1 to u_n are. But, basically the idea is what you can do is, they since they represent it always in prefix form with parenthesis so on so forth. You, can just look upon it as doing a traverse from left to right of the symbols. And, looking at the essentially left most disagreement a left. So, but what this uses is something that goes beyond our, this prefix ordering. Because our prefix ordering is not a total order but actually any order which is consistent with the prefix ordering is sufficient to get a left most disagreement. But, I prefer to use this prefix ordering because there is absolutely no reason why we should look at it as through a left most disagreement. We, could even start looking at it from the and look at a most disagreement. Sometimes, that might actually be more beneficial but I would not do that. But, in principle at least that possibility does exist so then what it means is that you take any total ordering so if you want to take the get the left most left most disagreement basically you take any total ordering consistent with the prefix ordering. And, such that the left sibling precede the sibling in any at any level.

(Refer Slide Time: 31:56)

```
Algorithm: Computing a Disagreement Set
Require: |T| > 1 {At least two different terms}
1 DISAGREEMENT(T) ≜ DISAGREE(T, ε, Pos(T) - {ε}) where
2 Pos(T) = ∩_{t ∈ T} Pos(t) {At least ε ∈ Pos(T)} and
3 DISAGREE(T, p, P) ≜
4 if |rootsym(T|_p)| = 1 then
5 let p' = Min(P); P' = P - {p'} in
6 DISAGREE(T, p', P')
7 end let
8 else {A disagreement has been found at position p}
9 return T|_p
10 end if
```

So, if you have this if you have this total ordering then you can compute what is known as a, Disagreement set. And, in particular this will compute the a left most disagreement set this is actually. So, I have written this in essentially some kind of functional style but, basically what we are saying is you start at a position a start at the root position. And, start take the set of all positions that are common to all the trees all the terms. And, if the root positions are all the same which is what this is sub carnality of the set of all root symbols at position p. If, that set is has a carnality of 1 which means they are all the root symbols are identical. Then, what you do is you take the next minimal position that is available in the set of common positions. And, look at all the trees and look at their look at whether they have the they disagree. And, so this is essentially recursive on this it is essentially looping on this till you find a minimum position when there is disagreement. Or, which will give a position p at which there is a disagreement. So, you take the entire set of terms at that position rooted at position you take all the sub terms rooted at that position p. And, that is your disagreement set and because you are using a left to ordering essentially you will get the left most disagreement. When, you look at in this fashion so this just computes the disagreement set.

(Refer Slide Time: 33:56)


Example: Disagreement 1

Example 28.11 Consider the set of terms

$$S_1 = \{f(a, x, h(g(z))), f(z, h(y), h(y))\}$$

where a is a constant, f is a ternary operator and g and h are unary operators. In this case, reading the terms from left to right we get a disagreement set $D_1 = \{a, z\}$. On the other hand, reading from right to left we obtain the disagreement set $D'_1 = \{g(z), y\}$ which requires going down one level deeper.

The algorithm however will compute the leftmost disagreement D_1 always.

 NPTEL

So, just let us look at these terms so I have taken a set of two terms here a , is constant z is a variable. So, there is a disagreement of course the root at the position ϵ there is no disagreement. So, the left most disagreement occurs when between a , and z where let us assume a , is constant and z is a variable. So, that is a disagreement set and the sub tree rooted at that is just the sub tree is rooted at that adjusts this a and z . So, this is what the algorithm actually we will give you could of course change this ordering and still keep it consistent with the prefix ordering. And, do a most or some other kind of disagreement if you do if you try to look at it from the most then of course what the disagreement you will get is somewhat deeper. So, at ϵ they are both the same at 3 which is the next position they are both the same. And, at 3 dot 1 is the first disagreement. So, you get your disagreement set then is $g(z), y$. So, that is this at a position 3 dot 1 this string 3 1 position string 3 1.


(Refer Slide Time: 35:26)

Example: Occurs Check

Example 28.12 Consider the set

$$S_2 = \{f(g(z), x, h(g(z))), f(z, h(y), h(y))\}$$

The disagreement set $\{g(z), z\}$ is such that S_2 is not unifiable, because for any substitution θ of θz can never be syntactically identical with $\theta g(z)$. This is an example of the notorious **occurs check problem**. Hence S_2 is not unifiable.

 NPTEL

Slide Show ** * * ** Go Back Prev. Section Close 1/17 of 4/18 Quit

Let us, look at this is an important example let us look at left most disagreements that is convenient. So, of course so these two are so there is a disagreement at position 1. So, this 1 has the term g of z this 1 has a term z and, this is the disagreement set. But, the important point is that if I take any substitution trying to unify these two terms i need to provide a replacement for z which, will make it equal to g of z . But, the same substitution is applied to all the terms. So, therefore there is going to be no possible substitution θ for z . Which, will make θz the same as θ of $g z$ and, that is.

So, this is what is known as occurs check problem. And, so in such a case actually this set is not unifiable at all. So, this set of terms is not unifiable and this is this is serious problem. In fact this is, a problem that makes all prolong implementations logically unsound. What, does happen is that for the sake of efficiency no prolong implementation actually checks before doing a, unification. Whether, there is an, occurs check problem they proceed ahead anyway. And, then there is and then actually you can get unsound so you can get unsound results because of that the occurs check actually makes it unsound. There, is another problem in prolong which also makes it unsound and that is the cut problem. But we will not worry about that at the moment that is another efficiency kind of problem.


(Refer Slide Time: 37:40)

Example: Disagreement 3

Example 28.13 Consider the set

$$S_3 = \{f(a, x, h(g(z))), f(b, h(y), h(y))\}$$

where a and b are both constant symbols. Here a disagreement set is $D_3 = \{a, b\}$. Again it is clear that S_3 is not unifiable.

 NPTEL

So, most prolong implementations actually do not do an, occurs check simply because it means going to deep down into the tree to find out possibility of occurrence of a free variable. Let us look at this set f of a, x, h or g z f of b h,y h,y here the disagreement is a position1 and a and b are both constants there's absolutely no way this place this set can be unified.


(Refer Slide Time: 38:23)

Example: Disagreement 4

Example 28.14 Consider the set

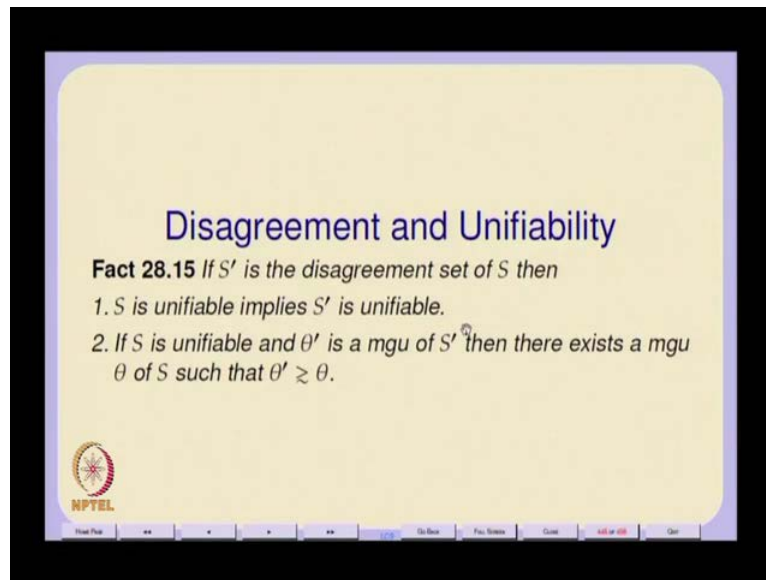
$$S_4 = \{f(h(z), x, h(g(z))), f(g(x), h(y), h(y))\}$$

Here we have a disagreement set $D_4 = \{h(z), g(x)\}$. Since $h(z)$ cannot be unified with $g(x)$ under any substitution of free variables, S_4 is not unifiable.

 NPTEL

So, S_3 this set is not unifiable here, S_4 f of h z, x, h of g z f of g x h, y h, y again at position 1 there is a disagreement but, then h and g are not the same. So this is also again not unifiable under any substitution of free variables.

(Refer Slide Time: 38:40)



So, here are some facts if S' is a Disagreement set of S , of a set S then if S is unifiable. Then, S' must also be unifiable unless you can unify S' you really cannot unify S . If S is unifiable and θ' is a most general unifier of S' . Then, there exists a most general unifier θ of S such that θ' is at least as general as θ . Basically, θ' gives you the least number of constraints required to make the disagreement set look the same. And, therefore in that sense it is likely to be more general than the entire set S . And, in fact what it means is that what this actually is saying is that we will start from θ' unify with the minimum number of constraints the first disagreements. And, then traverse down this is a partial order you traverse down this partial order. And, keep getting more and more specific as in when you require to be specific. So, your final mgu will be related to any initial mgu of a disagreement set by this generality relation.

(Refer Slide Time: 40:24)

The above facts reduce the problem of finding a unifier if it exists, to that of systematically finding disagreement sets and unifying them.

Finding a unifier for a disagreement set is a pre-requisite for finding a unifier for the original set of terms. A disagreement set consists of subterms of the the original set of terms at a particular position such that at least two distinct terms exist in the set. Further a disagreement set is unifiable only if there is at most one non-variable term in it. By choosing a substitution θ/x where both t and x are terms in the disagreement set satisfying the condition $x \notin FV(t)$, there is a possibility of unifying the disagreement set. Our **algorithm** constructs a sequence of singleton substitutions whose composition yields a unifier if it exists.

NPTEL

(Refer Slide Time: 40:25)

Algorithm: Unification
Require: $S \subseteq_f \mathcal{T}_\Omega(V)$ and $|S| > 1$
Ensure: If S is not unifiable then fail else $\exists \theta \in \mathcal{S}_\Omega(V) : |\theta S| = 1$ and θ is a mgu of S

- 1 UNIFY(S) $\stackrel{df}{=} \text{PARTIALUNIFY}(1, S)$ where
- 2 PARTIALUNIFY(θ, S) $\stackrel{df}{=} \dots$
- 3 let $T = \theta S$ in
- 4 if $|T| = 1$ then
- 5 return θ { θ is a mgu of S }
- 6 else {There is a position at which at least two terms are different}
- let $D = \text{DISAGREEMENT}(T)$ in

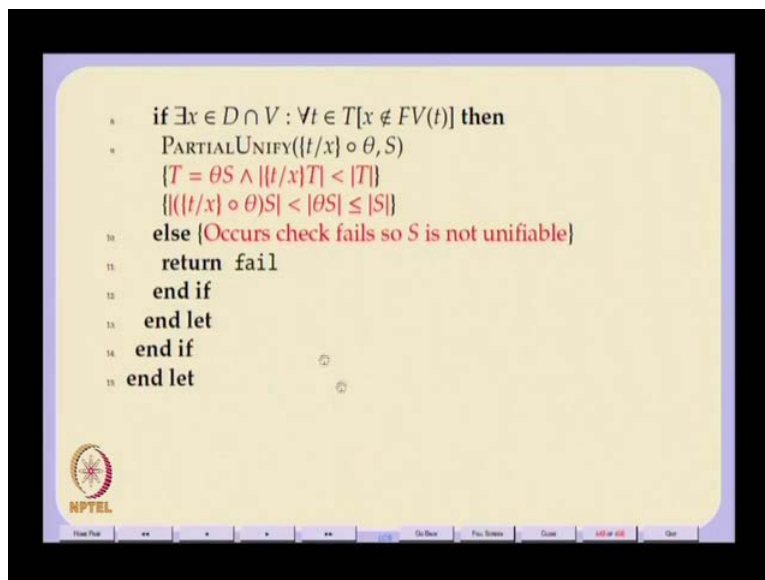
NPTEL

So, here is a Unification Algorithm which is actually quite a inefficient algorithm. But, it is for a enough to know that it is there is an effective procedure for unification. So, we start with the identity of the empty substitution. Which, is this 1 you are given a set a finite set of terms to be unify. So, there are two possibilities either the set is unifiable in which case you find a most general unifier or the set is not unifiable in which case you report failures that is. So, I do a so this theta prime here, is actually a partial unifier. So, you found a disagreement set you unify that

disagreement set now you apply that theta prime over the entire set S then, some part of it begins to look the same compared to S itself. So, theta prime applied to S is partially unified version of S. And, you have to do further you have to find further disagreements before you can unify the whole of S.

So, here again we I look at this even though it is an effective procedure I look at it in terms of sets so, and I look at sets and cardinality of sets. So, basically at any point you have got a certain substitution which is a partial substitution to be applied uniformly over the entire set T. And, having applied at a you will check the cardinality of the set so, obtained if it is 1 then it is been unified and, the unifier is theta. So, you start with some theta initially of course you start with a theta which is the identity substitution. So, if theta S has a cardinality of 1 that means it is being unified and theta is your most general unifier otherwise find it disagreement in theta S. So, what so essentially if you going to traverse this path then you should apply this theta prime to S and then look for a new disagreement. So, you apply you find the disagreement set of theta S

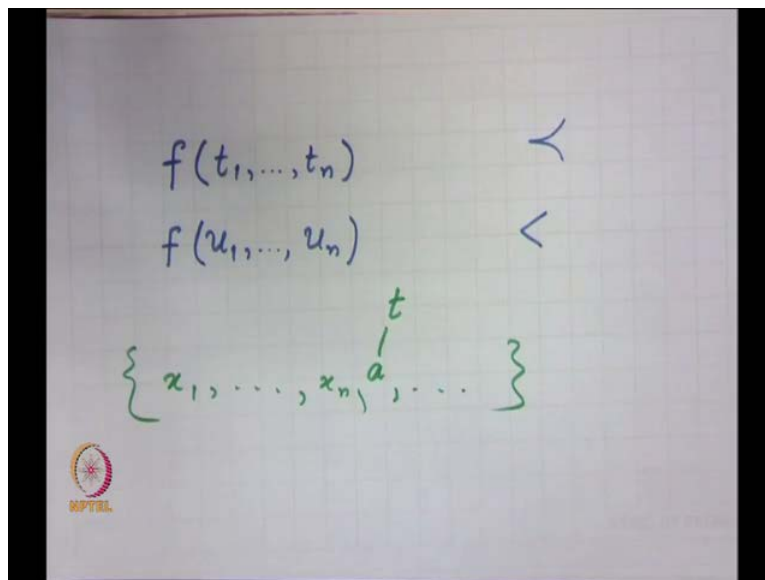
(Refer Slide Time: 43:03)



And then, this portion so is the occurs check problem you check whether there exists a variable in the disagreement set such that that variable is not free in any term. You remember that zgz the occurs check was because z occurs freely in g z. So, this x that you choose should be a variable that does not occur freely in any of the other terms unless the other terms are also x then which

case there is no disagreement. So, a disagreement is a set so you cannot you will have only one occurrence of a variable x . So, if they all differ only in variables then you will just have as many variables as many distinguished variables as there are in the set. So, if you can find a variable which does which is there in the disagreement set but which does not occur free in any of the other terms of the disagreement set. Then, you perform a substitution the other thing is that in this notice the other thing is that, this partial unification is possible only if this disagreement set consists of at most one complex term. All other entities in the disagreement set can only be variables I mean you take a disagreement set.

(Refer Slide Time: 45:09)

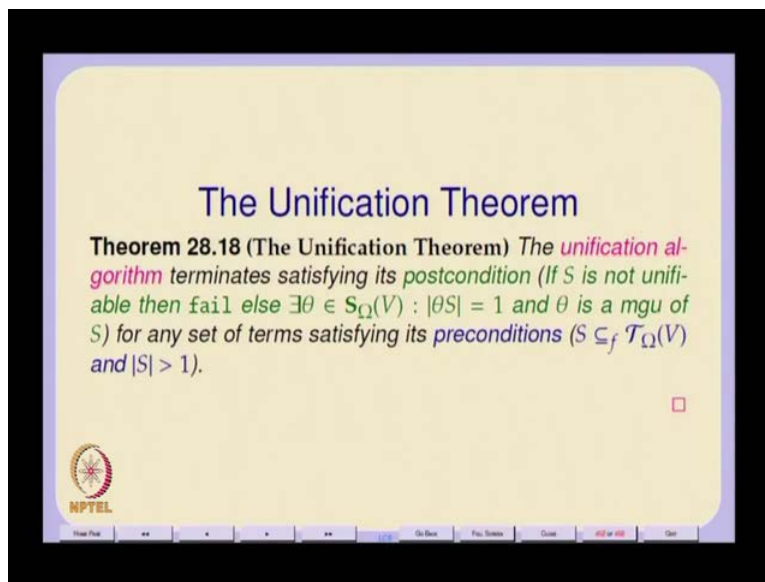


Basically what we are saying is that it can have a whole lot of variables and it can have maybe at most one constant or some complex term. But, it cannot have more than, one constant or complex term if it has is a set if it has more than, one constant or complex term. Then, they occur only because they are different then you cannot unify. So, basically the disagreement set you are making some progress towards unification only if everything there except one possible term is variable. If, you have two different constants or two different other terms which are not variables then your unification is going to fail because the terms are not going to be unifiable. So, in fact this term actually will be unique if it is unifiable this term T if it is just a variable then there is no problem.

But, if it is something other than a variable then that can be only one term otherwise it is not going to be unifiable. And, which means that you can choose that T for x essentially and in fact what you are going to choose is T for all the variables that occur in that is what you are going to choose it is possible to do that here. But what we what I have done is that I have made it one substitution at a time choice. So, if there are two variables x1 and x2 and a term and a single term T then what will happen is it will first do a T for x1 in this iteration. In the next iteration it will find the disagreement T for x2 and then make that another substitution. So, it is sort of does to it does it to sequentially even within the set where as it can be done in more efficient way.

So, what you are going to do now is you going to just take the existing substitution and compose this in your substitution. And, if this condition is false then essentially you have an occur check failure and that set is not unifiable and then you return failure and so on and so forth. But, the important thing and the hard thing which I am not going to do is this which I am going to leave it for you to study.

(Refer Slide Time: 48:10)



This is the unification theorem every algorithm has to be justified by a corresponding theorem and the unification theorem essentially says. So, the unification algorithm always terminates firstly and it satisfies its post condition. And, the post condition is that if S is not unifiable then it reports failure and if S is unifiable then it gives a most general unifier. So, I have got the proof

here this put in various claims but I am not going to go in detail because you can go and study it. Especially the last case where you are actually checking where if it does if it is unifiable then you do get an mgu you do not get an arbitrary substitution. You, do not get an arbitrary unifier you get an unifier which is most general that is an important aspect to be proven here.