Relative Clauses, Variables, Variable Binding (Part I)

Heim and Kratzer

Chapter 5

- Relative clauses are another kind of noun modifier, and a good way to introduce variables and variable binding.

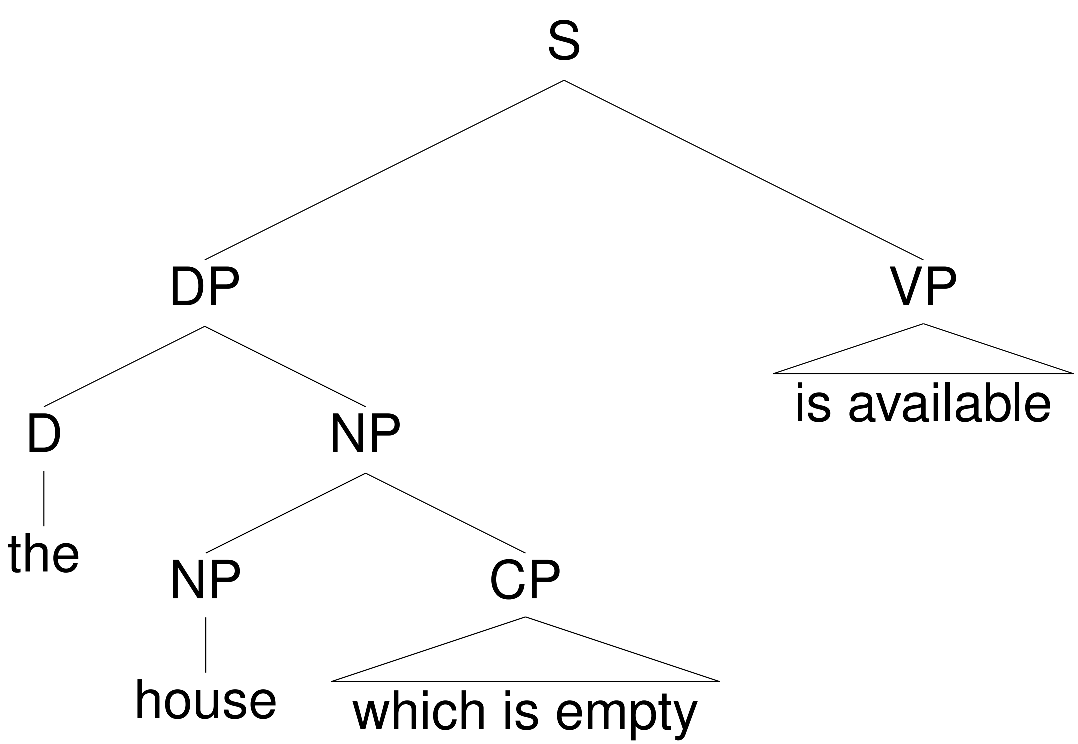
- We will start with a simplified version of variable assignments, and then present a more complicated, full version later that will enable us to deal with multiple variables.

5.1 Relative clauses as predicates

- Read the quote from Quine 1960 in the text. Relative clauses are just another kind of noun modifier and are of type $< e, t >$.

- They combine with the head noun via PM and then combine with the determiner by FA. Check out the semantic types of the nodes in the following tree.

(1′)



5.1 Relative clauses as predicates (cont.)

- Restrictive relative clauses, like (1), are thus different from nonrestrictive relative clauses, like (2).

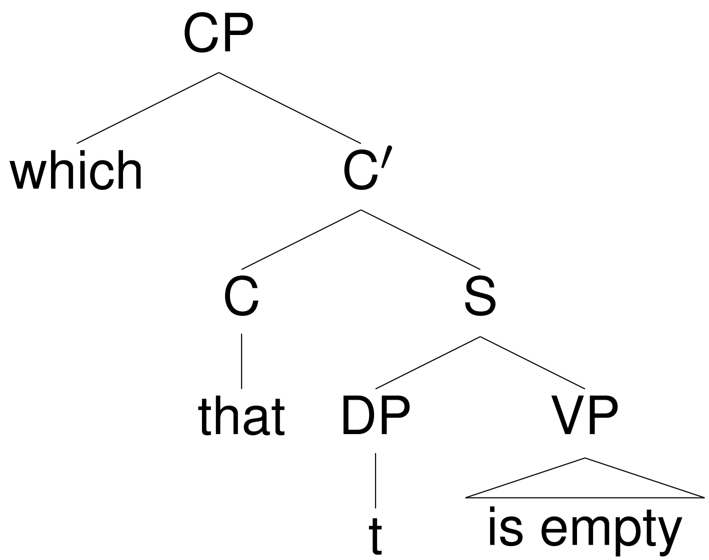
- Only (2) presupposes that there is only one house.

- We will only talk about restrictive relative clauses here.

(1) **The house which is empty is available**.
(2) **The house, which is empty, is available**.

5.2 Semantic composition within the relative clause

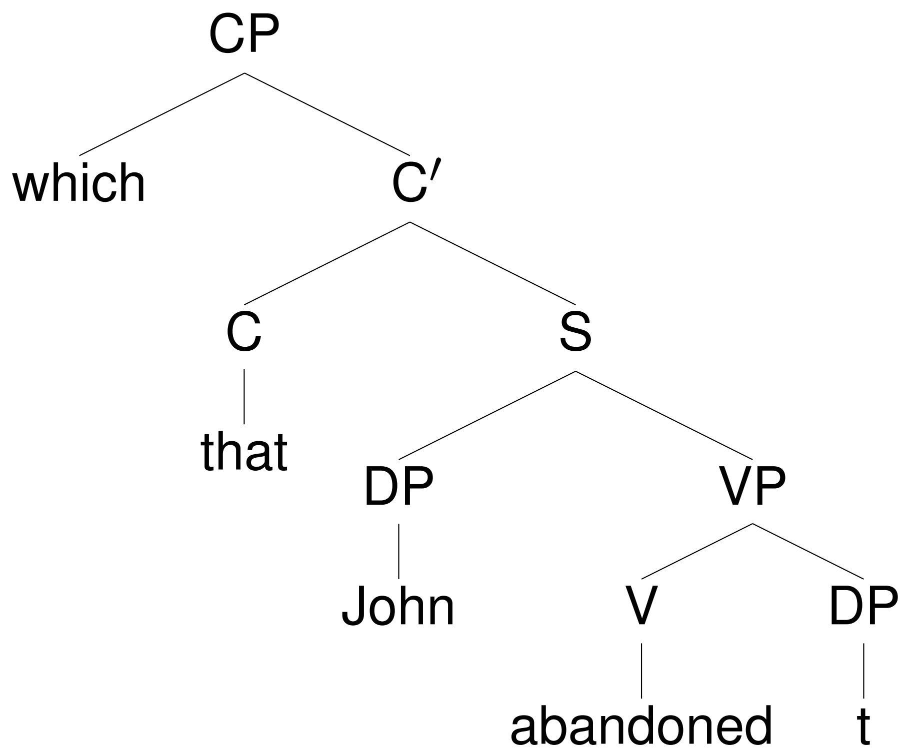
(1)



- We assume that either “which” or “that” is deleted on the surface.
- We will henceforth assume that phrases with determiners, proper names, pronouns and traces are DPs.
- We can’t just assume that the VP sends its semantic value all the way up through the tree because there are also object relatives.

5.2 Semantic composition within the relative clause (cont.)

(2)

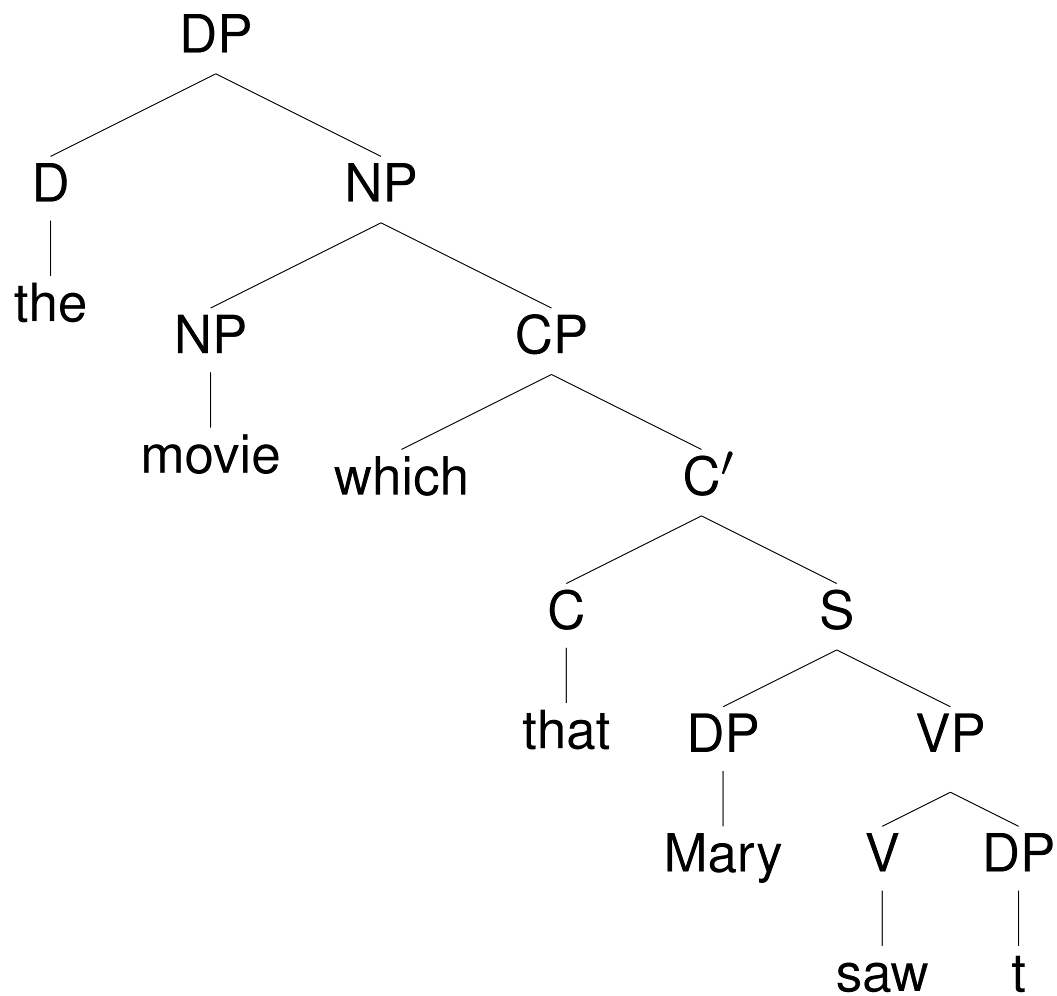


- (2) should mean: λx $\in$ D$_e$. John abandoned x.

- What are the semantic values of traces? We want traces to be of type e, but does it make sense to assign a referent to the trace?

5.2.1. Does the trace pick up a referent?

(3)



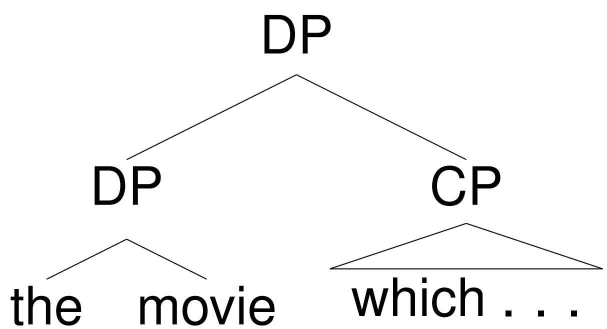
- It is sometimes thought that the relative pronoun is anaphorically related and picks up the referent from its head, but where is there a DP antecedent?

- It can’t be the DP as a whole because the denotation of the whole is supposed to be built up from the denotations of its parts, so we need the denotation of the part first.

5.2.1. Does the trace pick up a referent? (cont.)

- The alternative phrase structure in (4) also will not work because if the bottom DP is of type e and the relative clause is of type $< e, t >$, then the whole DP will be of type t, which is not correct.

(4)



- We need a new semantic construct: the variable.

5.2.2 Variables

- A variable denotes an individual, but only *relative to a choice of an assignment of a value*.

(5) Preliminary definition: An *assignment* is an individual (that is, an element of D (= D_e)).

(6) The denotation of “t” under the assignment Texas is Texas.

(7) $[\![\mathbf{t}]\!]^{\text{Texas}}$ = Texas.

(8) If α is a trace, then for any assignment a, $[\![\alpha]\!]^{\text{a}}$ = a.

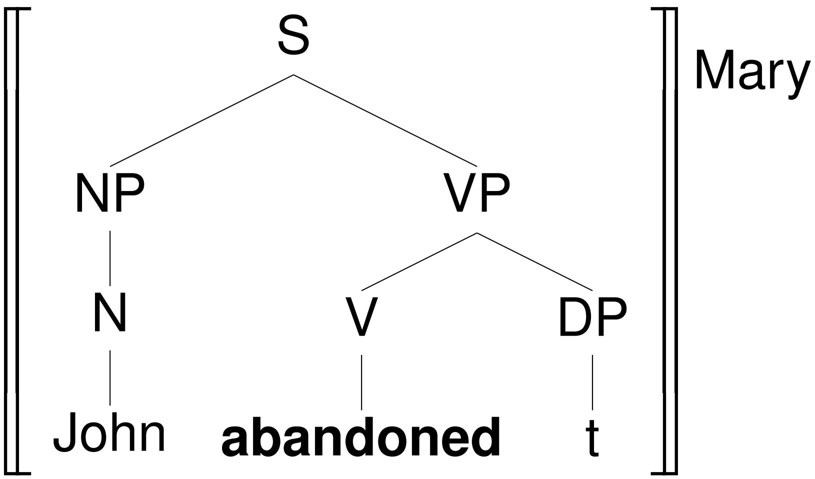
- We must allow the denotations of large phrases that contain traces to be assignment-relative as well.

5.2.2 Variables (cont.)

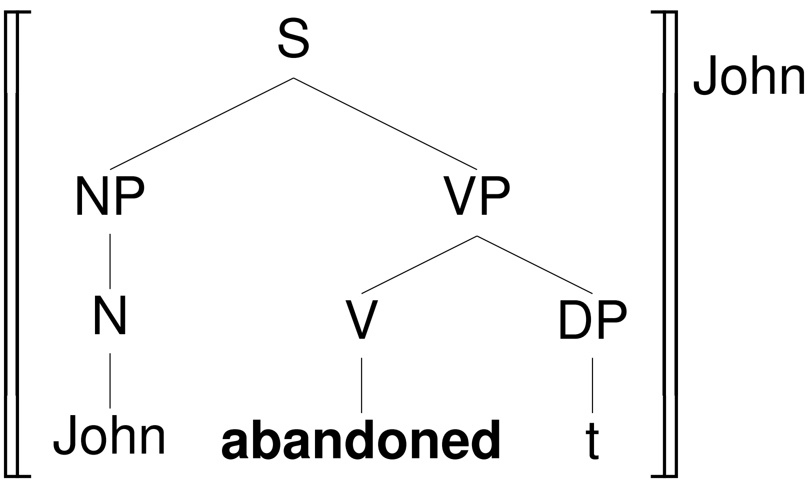
$$[\![\, [_{VP} \; [_{V} \; \textbf{abandoned}] \; [_{DP} \; t] \,] \,]\!]^{\text{Mary}} = \lambda x \in D_e \,.\, x \text{ abandoned Mary}$$

$$[\![\, [_{VP} \; [_{V} \; \textbf{abandoned}] \; [_{DP} \; t] \,] \,]\!]^{\text{John}} = \lambda x \in D_e \,.\, x \text{ abandoned John}$$

5.2.2 Variables (cont.)



= 1 iff John abandoned Mary.



= 1 iff John abandoned John.

5.2.2 Variables (cont.)

- Now we have sentences that have truth conditions *per se* and also sentences (in relative clauses) that need assignment-relative denotations. This will affect all the composition principles.

- Instead of duplicating composition principles to deal with each case, it is simpler to formulate all our composition principles for assignment-dependent denotations and introduce assignment-independent denotations through a definition:

(9) For any tree α, α is the domain of $[\![\ ]\!]$ iff for all assignments a and b, $[\![\alpha]\!]^{a} = [\![\alpha]\!]^{b}$.
If α is in the domain of $[\![\ ]\!]$, then for all assignments a, $[\![\alpha]\!] = [\![\alpha]\!]^{a}$.

(10) For any assignment a, $[\![\textbf{laugh}]\!]^{a} = [\![\textbf{laugh}]\!] = \lambda x \in D_e$. x laughs.

5.2.2 Variables (cont.)

(11) *Lexical Terminals*

If α is a terminal node occupied by a lexical item, then $[\![\alpha]\!]$ is specified in the lexicon.

(12) *Non-Branching Nodes* (NN

If α is a non-branching node and β its daughter, then, for any assignment a, $[\![\alpha]\!]^a = [\![\beta]\!]^a$.

(13) *Functional Application* (FA)

If α is a branching node and $\{\beta, \gamma\}$ the set of its daughters, then, for any assignment a, if $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$, then $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

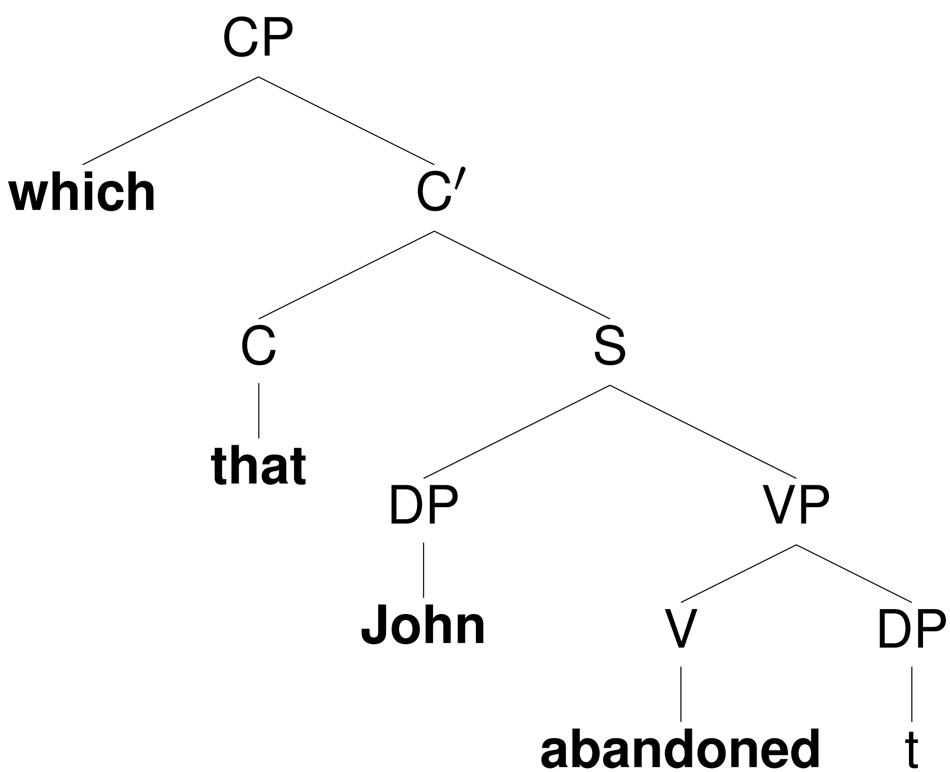
(14) *Predicate Modification* (PM

If α is a branching node and $\{\beta, \gamma\}$ the set of its daughters, then, for any assignment a, if $[\![\beta]\!]^a$ and $[\![\gamma]\!]^a$ are both functions of type $< e, t >$, then $[\![\alpha]\!]^a = \lambda x \in D$. $[\![\beta]\!]^a(x) = [\![\gamma]\!]^a(x) = 1$.

5.2.3 Predicate abstraction

- Now for the semantic principle that determines the denotation of a relative clause.

(2)



5.2.3 Predicate abstraction (cont.)

- We treat the complementizer “that” as semantically vacuous, so the C’ inherits the value of the S below it.

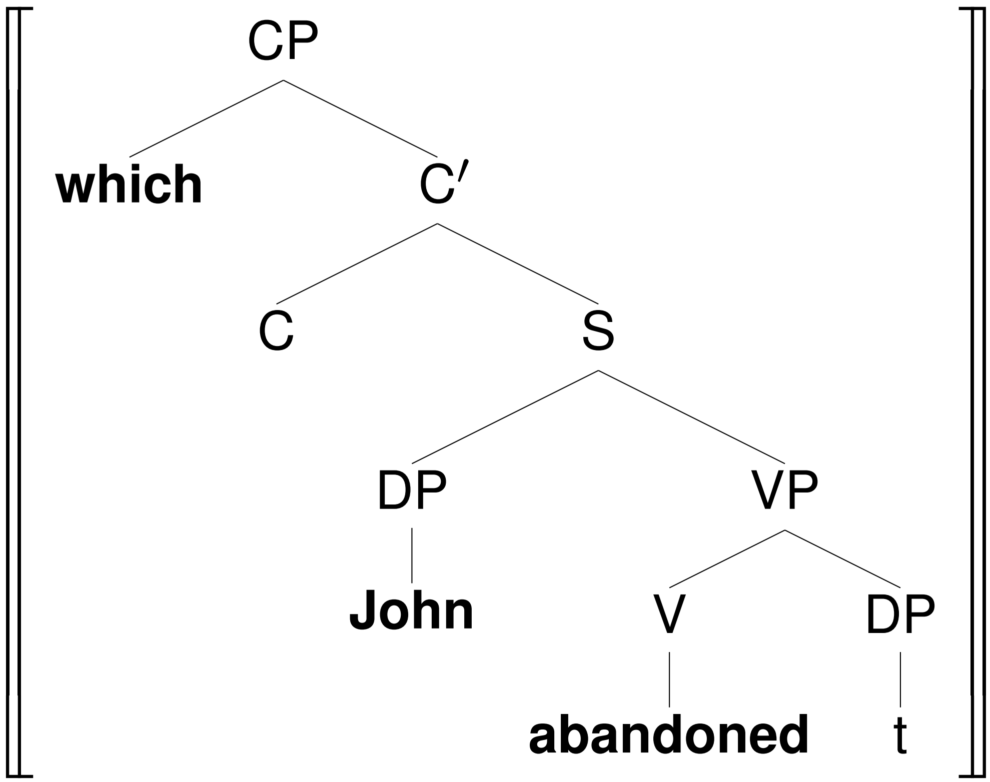
- The relative pronoun within CP is also not assigned any denotation of its own. But it is not simply vacuous; its presence will be required to meet the structural description of the composition principle applying to the CP above it.

(15) *Predicate Abstraction* (PA)

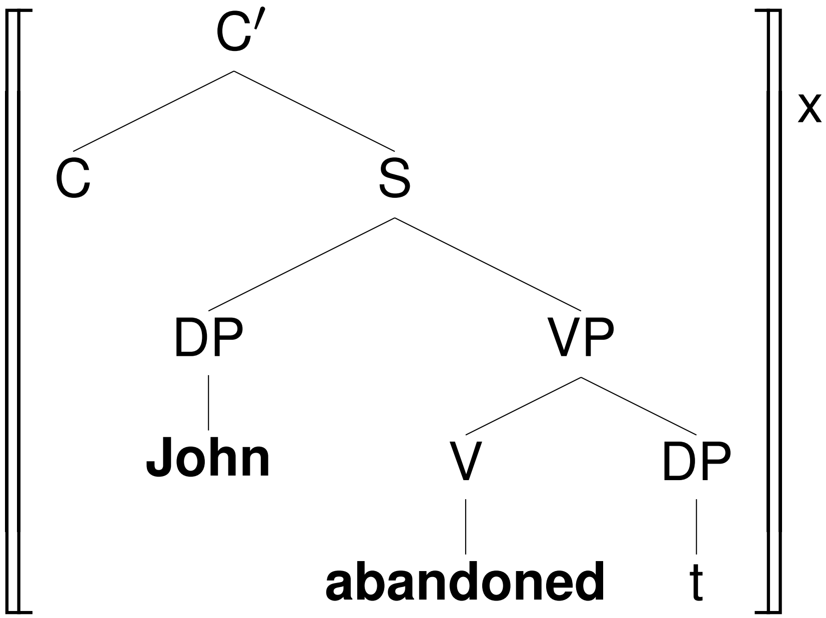
If α is a branching node whose daughters are a relative pronoun and β, then $[\![\alpha]\!] = \lambda x \in D \, . \, [\![\beta]\!]^{x}$.

- Being a relative clause, (2) should have an assignment-independent interpretation and it should denote the function $\lambda x \in D$. John abandoned x. Here is a proof.

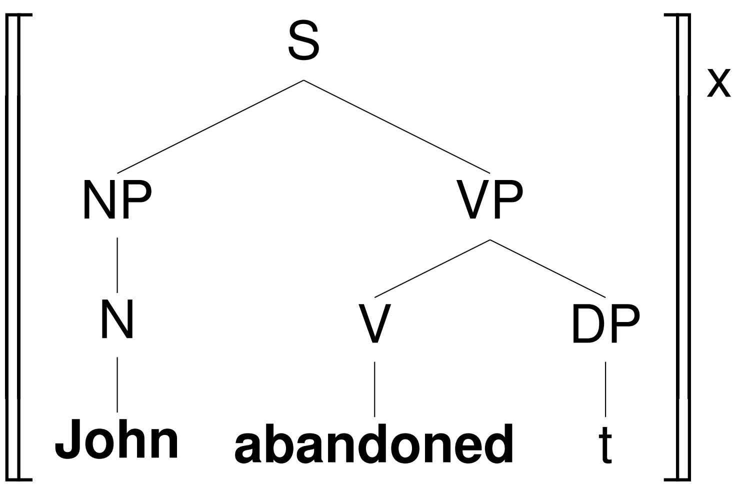
5.2.3 Predicate abstraction (cont.)



= (by PA)

$\lambda x \in D$. 

= (by vacuity of C)

$\lambda x \in D$. 

= (by FA)

$$\lambda x \in D \,.\, [\![[_{VP}\ [_{V}\ \textbf{abandoned}]\ [_{DP}\ t]]]\!]^{x} ([\![\textbf{John}]\!]^{x}) = \text{(by definition (9))}$$

$$\lambda x \in D \,.\, [\![[_{VP}\ [_{V}\ \textbf{abandoned}]\ [_{DP}\ t]]]\!]^{x} ([\![\textbf{John}]\!]) = \text{(lexical entry of } \textbf{John}\text{)}$$

$$\lambda x \in D \,.\, [\![[_{VP}\ [_{V}\ \textbf{abandoned}]\ [_{DP}\ t]]]\!]^{x} (\text{John}) = \text{(by FA)}$$

$$\lambda x \in D \,.\, [\![\textbf{abandoned}]\!]^{x} ([\![t]\!]^{x}) (\text{John}) = \text{(by Traces Rule)}$$

$$\lambda x \in D \,.\, [\![\textbf{abandoned}]\!]^{x} (x) (\text{John}) = \text{(by definition (9))}$$

λx $\in$ D . ⟦**abandoned**⟧ (x) (John) = (by lexical entry of **abandoned**)

λx $\in$ D . [λy $\in$ D . [λz $\in$ D . [z abandoned y]]] (x) (John) = (by definition of λ-notation)

λx $\in$ D . John abandoned x. QED.

5.2.3 Predicate abstraction (cont.)

- The Predicate Abstraction Rule gives the moved relative pronoun what is called a *syncategorematic* treatment. Syncategorematic items don't have semantic values of their own, but their presence affects the calculation of the semantic value for the next higher constituent.

- A syncategorematic treatment of relative pronouns goes against the concept of type-driven interpretation that we argued for earlier, and we eventually want to abolish rules of this kind. However, we will keep it for now.

5.2.3 Predicate abstraction (cont.)

- When you work with assignments, do not confuse denotations *under* assignments with denotations *applied to* assignments. There is a big difference between $[\![\text{a}]\!]^{\text{x}}$ and $[\![\text{a}]\!](\text{x})$.

$[\![\textbf{whom John abandoned t}]\!]^{\text{Charles}} \neq [\![\textbf{whom John abandoned t}]\!](\text{Charles})$
$[\![\textbf{sleeps}]\!]^{\text{Ann}} \neq [\![\textbf{sleeps}]\!](\text{Ann})$

- On the left is a function of type $< e, t >$, but what's on the right is the result of applying such a function to an individual – in other words, a truth value.

5.2.3 Predicate abstraction (cont.)

- In many instances, one of the two notations isn't even well-defined.

$[\![$**John abandoned t**$]\!]^{x} \neq [\![$**John abandoned t**$]\!](x)$

- The lefthand side makes sense. It stands for a truth value; that is, it equals 1 or 0, depending on what individual "x" stands for.

- But the right side is nonsense for two reasons. First, it falsely presumes that "John abandoned t" is in the domain of $[\![\ \ ]\!]$; that is, that it has a semantic value independently of a specified assignment. This is not the case.

- Second, even if "John abandoned t" did have an assignment-independent denotation, its denotation would be a truth value, not a function. So, it would be as though we had written "1(x)" or "0(x)".

5.2.4 A note on proof strategy: bottom up or top down?

- When you are asked to calculate the semantic value of a given tree, you have a choice between two strategies: to work from the bottom up or from the top down.

- Beginners often find the bottom-up strategy easier, but there is no reason to prefer it.

- However, for derivations involving Predicate Abstraction, the top-down strategy is preferable.

- See derivations in the book for illustration and explanation.