# Variables, pronouns, and scope

## 1 Notes on variables

(1) **Some math "sentences":**

a. $1 = 2 - 1$ — a sentence with no variables; not context-sensitive

b. $n = 2 - 1$ — a sentence with a variable; context-sensitive

c. $\forall n\ (2(n+1) = 2n + 2)$ — a sentence with a variable; *not* context-sensitive

- We say (1b) contains a *free variable* because the truth of the sentence depends on the context. In particular, the sentence is true iff the variable "$n$" is interpreted as 1.
- The truth of sentence (1c), like (1a), does not depend on the context at all.

(2) **Some terminology, using (1c) as an example:**

$$\underset{binder}{\forall n} \underbrace{\left(2(\underset{bound}{n}+1) = 2\underset{bound}{n}+2\right)}_{scope}$$

- *Binders* control the interpretation of a particular variable within a certain part of its structure, which we call its *scope*. Here, $\forall$ *binds* the variable $n$ in its scope.
- We call variables that are in the scope of a matching binder *bound variables*.

Let's call the mapping between free variables and their values *assignment*.

## 2 Pronouns

This free/bound terminology is useful for natural language as well:

(3) a. John likes Mary. — a sentence with no variables; not assignment-sensitive

b. John likes him. — a sentence with a variable; assignment-sensitive

c. Every boy likes himself. — a sentence with a variable; *not* assignment-sensitive

We'll formalize this by giving each pronoun a numerical *index*. We'll compute denotations relative to an *assignment function*, which is a function from the set of indices ($\mathbb{N}$) to $D_e$.

(4) **Pronouns Rule (to be replaced later):**
If $\alpha$ is a pronoun, $g$ is a variable assignment, and $g(i)$ is defined, then $[\![\alpha_i]\!]^g = g(i)$.

(5) Suppose $g$ is a function and $g(3) = \text{Sam} \in D_e$.

a. $[\![\text{him}_3]\!]^g = \text{Sam}$

b. $[\![\text{John likes him}_3]\!]^g = 1$ iff John likes Sam

**Q:** Does it matter what $g$ returns for other values in (5)?

**A:** No. It might even be undefined for other values.

**Q:** Why did we use 3? Does the number matter?

**A:** The choice of number was arbitrary, but it is important whether or not we reuse numbers:

(6) a. $\text{He}_2$ thinks that $\text{he}_2$ is smart.

b. $\text{He}_2$ thinks that $\text{he}_7$ is smart.

**Q:** Does the assignment function affect other parts of the sentence?

**A:** No. "John" and "likes" are *constants*, meaning their values are the same no matter the assignment: for any assignment function $f$, $[\![\text{John}]\!]^f = \text{John}$.

**Warning:** There's a section of H&K (pp. 92–109) where they just use notation like $[\![\text{him}]\!]^{\text{John}}$ = John, which only accommodates one variable at a time, but then they introduce their actual notation on page 110, which we use here.

## 3 Rules with assignments

In order to work with assignment functions, we need to modify all our existing rules so that they pass assignment functions. These definitions are based on H&K p. 95:

(7) **Terminal Nodes (TN):** (unchanged)
If $\alpha$ is a terminal node, $[\![\alpha]\!]$ is specified in the lexicon.[1]

(8) **Non-branching Nodes (NN):**
If $\alpha$ is a non-branching node, and $\beta$ is its daughter node, then, for any assignment $g$, $[\![\alpha]\!]^g = [\![\beta]\!]^g$.

(9) **Functional Application (FA):**
If $\alpha$ is a branching node, $\{\beta, \gamma\}$ is the set of $\alpha$'s daughters, then, for any assignment $g$, if $[\![\beta]\!]^g$ is a function whose domain contains $[\![\gamma]\!]^g$, then $[\![\alpha]\!]^g = [\![\beta]\!]^g([\![\gamma]\!]^g)$.

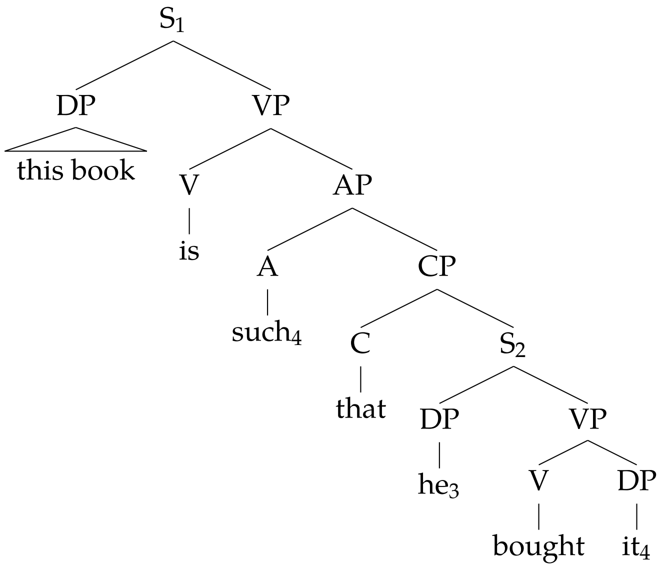
(10) **Predicate Modification (PM):**
If $\alpha$ is a branching node, $\{\beta, \gamma\}$ is the set of $\alpha$'s daughters, then, for any assignment $g$, if $[\![\beta]\!]^g$ and $[\![\gamma]\!]^g$ are both of type $\langle e, t\rangle$, then $[\![\alpha]\!]^g = \lambda x \in D_e \,.\, [\![\beta]\!]^g\,(x) = 1$ and $[\![\gamma]\!]^g = 1$ .

[1] H&K proposes (p. 94) to still use $[\![\alpha]\!]$ without an assignment function superscript for *constants*, i.e. if $[\![\alpha]\!]^g$ is the same value for all assignment functions $g$.

## 4 *Such that* relatives

The English expression *such that* allows us to construct relative clauses without movement.[2]

(11) $^?$ This book is such$_4$ that he$_3$ bought it$_4$. ($g(3)$ = John)




Here, (11) does not seem assignment-dependent. But the Principle of Compositionality states that $[\![\text{S}_1]\!]$ be computed based on the meaning of $[\![\text{S}_2]\!]$, which contains a pronoun and *is* assignment-dependent.

**Idea:** *Such* binds *it*, doing the work of creating a *predicate* out of the assignment-dependent sentence "John bought it."

(12) ***Such* Rule (temporary):**[3]
$[\![\text{such}_i\ \gamma]\!]^g = \lambda x_e\ .\ [\![\gamma]\!]^{[i \mapsto x] || g}$

$[i \mapsto x] \,||\, g$ is the *combination* of functions $[i \mapsto x]$ and $g$:

(13) **Definition: function combination**
$$f \,||\, g \equiv \lambda x\ .\ \begin{cases} f(x) & \text{if } x \in \text{domain}(f) \\ g(x) & \text{otherwise} \end{cases}$$
Read "$f$ or else $g$."

Let's compute $[\![\text{S}_1]\!]^g$ with the following global assignment function: $g = \begin{bmatrix} 3 \mapsto \text{John} \\ 11 \mapsto \text{Tama} \end{bmatrix}$.
Assume $[\![\text{that}]\!]$ = Id.

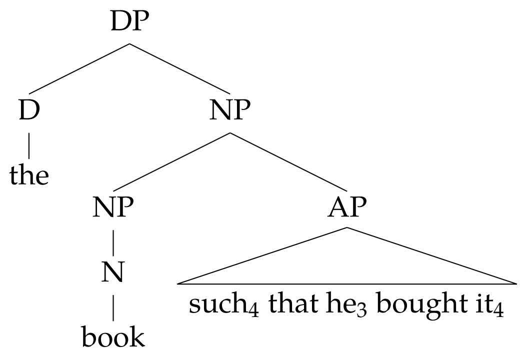
**Warning:** H&K uses $g^{x/i}$ notation for $[i \mapsto x] \,||\, g$, but I think it's confusing so I don't use it.[4]

[2]Unfortunately, the use of *such that* sounds "unlyrical" (Quine, 1960, §23)... but we'll ignore that here.
[3]"Such" does not have a type. That's why it can only be interpreted using the *Such* Rule.
[4]For one, I've also seen very similar notation "$g(x/a)$" for a function that maps $x$ to $a$, which is the reverse of

We can also use *such that* to construct (slightly awkward) *relative clauses*:

(14) $^?$ the book such$_4$ that he$_3$ bought it$_4$

The semantics for *such* above works perfectly fine here.




**Binding multiple variables:**

(15) $^?$ This book is such$_4$ that he$_3$ bought it$_4$ and then gave it$_4$ to Sarah.

(16) $^?$ every book such$_4$ that he$_3$ bought it$_4$ and then gave it$_4$ to Sarah

**Binding no variables (vacuous binding):**

(17) * This book is such$_4$ that today is Monday.

(18) * every book such$_4$ that today is Monday

The ungrammaticality of these examples shows that binding *no* variables is disallowed by the grammar. This is called *vacuous binding*.

## 5 Traces & Pronouns

(19) **The interpretation of movement (revised):** replaces last week's movement rule
Pick an arbitrary index $i$.

a. The base position of movement is replaced with a *trace* with index $i$: $t_i$.

b. A *binder index* $i$ is adjoined right under the target position of the movement chain.

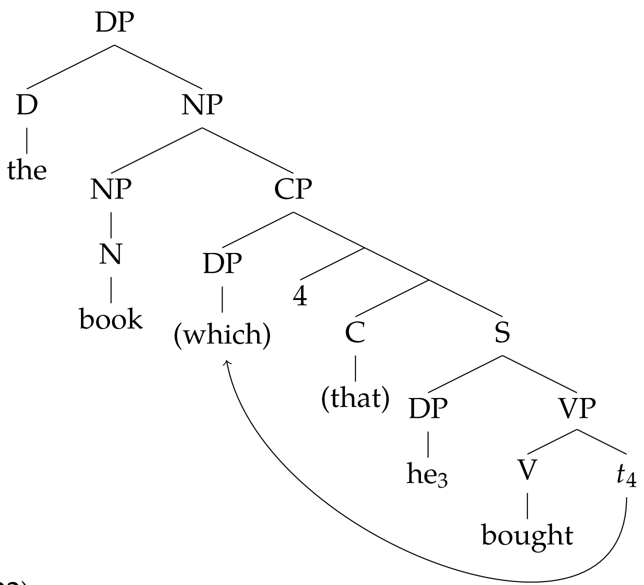
(20) **Traces and Pronouns Rule (T&P):** replaces Pronouns Rule in (4)
If $\alpha$ is a pronoun or trace, $g$ is a variable assignment, and $g(i)$ is defined, then $[\![\alpha_i]\!]^g = g(i)$.

(21) **Predicate Abstraction (PA):** (H&K p. 186 version)
replaces last week's rule for $\lambda$ nodes in the tree and the *Such* Rule in (12)[5]
Let $\alpha$ be a branching node with daughters $\beta$ and $\gamma$, where $\beta$ dominates only a numerical index $i$. Then, for any assignment $g$, $[\![\alpha]\!]^g = \lambda x \,.\, [\![\gamma]\!]^{[i \mapsto x]||g}$ .

what H&K mean in their $x/i$.

[5]We can think of "such" as the pronunciation of a lexicalized binder index, not generated through movement.

(22) the book that $he_3$ bought ___




**Exercise:** Compute (22).

## 6 Quantifier scope

(23) Everyone does not sleep (during class).

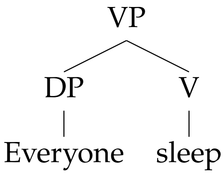
a. $$1 \text{ iff } \forall x \in D_e \underbrace{\left[ x \text{ is animate} \rightarrow \text{it's not that } \underbrace{[x \text{ sleeps (during class)}]}_{\text{scope of } \textit{not}} \right]}_{\text{scope of } \forall} \qquad (\forall > \textit{not})$$

b. $$1 \text{ iff it's not that } \underbrace{\left[ \forall x \in D_e \underbrace{[x \text{ is animate} \rightarrow x \text{ sleeps (during class)}]}_{\text{scope of } \forall} \right]}_{\text{scope of } \textit{not}} \qquad (\textit{not} > \forall)$$

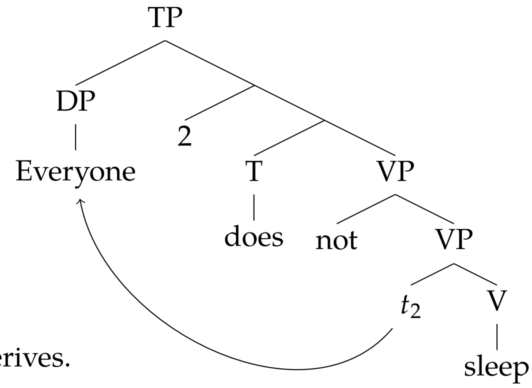
The two readings in (23) represent a *scope ambiguity*. There are two operators that "take scope"—∀ and negation—and one scope contains the other. We say ∀ in (23a) takes *wider* scope, and write ∀ > *not* to indicate this.

Recall from the problem set that there are advantages to adopting a VP-internal subject, interpreted through movement. We will adopt this here.

Step 1: Build subject in Spec,VP




Step 2: Add *not* + T, move subject DP to Spec,TP

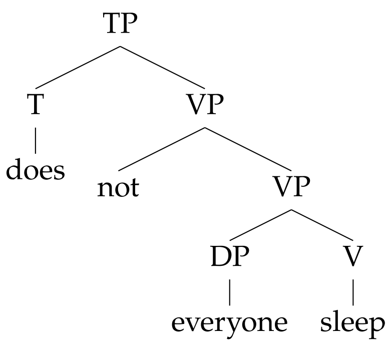



**Exercise:** Let's see what meaning this tree derives.

We call the meaning that is reflected on the surface form—here, (23a)—a *surface scope* reading.

How do we get reading (23b)? One option: *pretend the movement didn't take place*.

At Logical Form (LF): Pretend the movement didn't happen




**Exercise:** Interpret this tree.

We call this the *inverse scope* interpretation. The process of "ignoring" movement at LF is called *syntactic reconstruction*.

## References

Quine, Willard Van Orman. 1960. *Word and object*. Cambridge.