Merge

CAS LX 422 ∼ GRS LX 722 Intermediate Syntax

Lecture 3

Knowledge of language

Knowing a language is:

- knowing the “words”
- knowing how to put them together
- knowing how to pronounce them
- knowing what they mean in combination

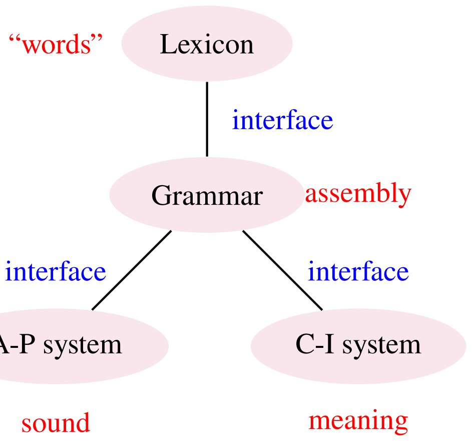
Lexicon

To construct a sentence, we start with the “words” and put them together.

We can describe the knowledge of the words of the language as being a list, a mental *lexicon*.

We can view a “word” as being a bundle of features, defined by the properties it has. The grammar then assembles the words into sentences. The sentences are *interpreted* and *pronounced*. Any given lexical item has semantic features, phonological features, and syntactic features. The hypothesisis that syntax can only see the syntactic features.

System architecture



Interfaces

The assembly system is the grammar proper.

The system that interprets sentences is another cognitive module ("conceptual-intensional system") concerned with meaning, reasoning, etc. It interprets the constructed sentence at the interface.

The system that determines the pronuncation of sentences is yet another cognitive module ("articulatory-perceptual system"), interpreting the constructed sentence at its interface.

Verbs and substitution

One of the ways we know a verb is a verb (category) is by observing that it can substitute in for other verbs.

(1) Pat likes to sing.

(2) Pat likes to drive.

(3) Pat bought a book

(4) * Pat bought (a) sing.

(5) Pat likes to eat sandwiches.

(6) * Pat unpleasant to eat sandwiches.

But, wait, so is *eat sandwiches* a verb?

Kind of, yes. It's a constituent, a phrase, that has the properties that a verb does. It's a verb phrase.

The making of a phrase

We're trying to characterize our knowledge of syntactic structure.

Our grammatical knowledge is a system (we can judge new sentences).

All things being equal, a theory in which the system is simpler (needed fewer assumptions) is to be preferred over a theory that entails a more complex one.

The making of a phrase

In that spirit, we know that a phrase differs from a word in that it *contains* words (or other phrases).

We've seen that when words are combined into a phrase, the phrase inherits the properties of one of the things we combined. The phrase, you might say, has a head.

Suppose: a *phrase* can arise from *merging* two words together, with one taking priority. In a way, attaching one word to another.

The making of a phrase

What will Pat do?

- sing
- eat sandwiches

What does Pat like?

- to sing
- to eat sandwiches

So the constituency here is: [to [eat sandwiches]]

A phrase can also arise from combining *to* and a verb phrase, to make a still bigger phrase.

Merge

Let’s go for the simplest theory of structure we can (and only move away from it if the simplest theory won’t work).

Phrase

A **phrase** is a syntactic object formed by combining (**merging**) two syntactic objects, with the properties inherited from one of them (the **head** of the phrase).

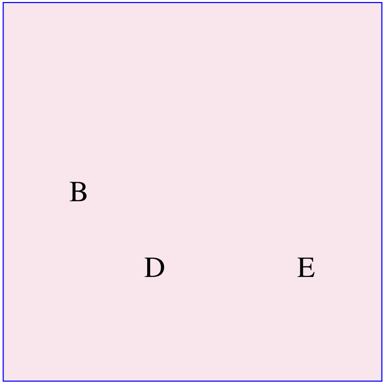
Syntactic object

A word is a syntactic object.

Merge

A good way to think about this is that we have a number of syntactic objects lying around on a workbench of sorts.

We use the operation **Merge** to assemble them into one syntactic object.



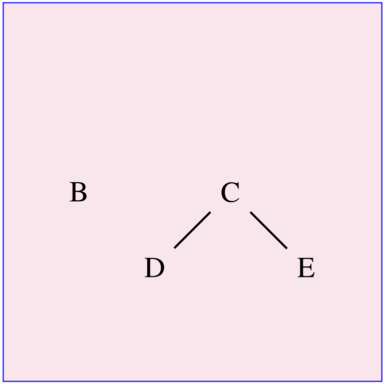
Merge

We combine D and E using Merge, forming a new syntactic object.

We need to call our new object something, so we call it C.

C is now a syntactic object (containing D & E).

D and E are now “off the table”—we can’t now Merge D with anything because it’s inside C. Merge only combines objects at their root nodes.

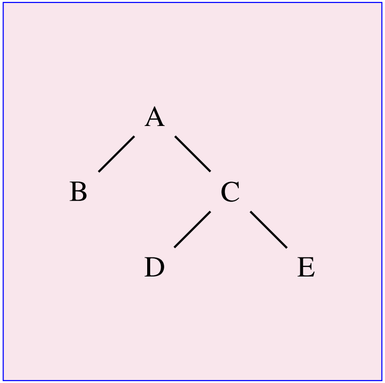


Merge

Since C is now a syntactic object, we can combine C with the other sytactic object, B, to form a new syntactic object we'll call A.

Now, all we're left with is the single syntactic object A.

When two objects are Merged, one of them is the **head**. The head determines the properties of the constituent—that is, the features of the head **project** to become the features of the whole combined object.



Trees and constituency

Pat will eat that.

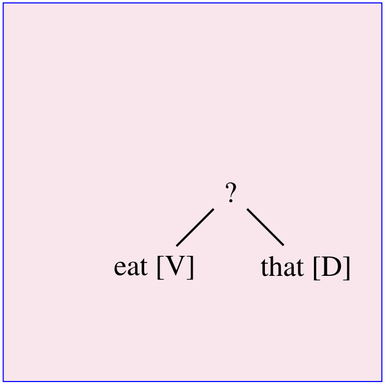
Pat will dine.

eat [V]   that [D]

Trees and constituency

Pat will eat that.

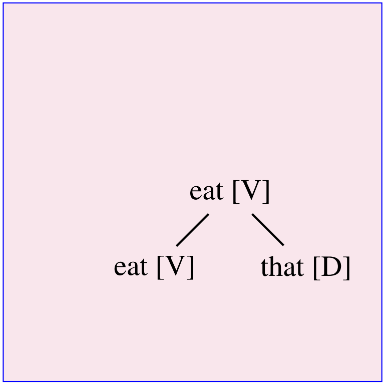
Pat will dine.



Trees and constituency

Pat will eat that.

Pat will dine.



So how do we know which is the head?

When we Merge two things, one is the head, and determines the properties of the resulting syntactic object. But how do we know which?

A primary—and perhaps the most important—type of sentence is that which represents a **proposition**.

A proposition is the kind of thing that can be true or false (basically).

Truth and verbs, predicates and arguments

(7) Michael swam.

- Michael: refers to an individual; it is a name, a label. It is complete.
- swam: describes an action that can be undertaken by someone, or a property someone can have. Someone. *Swam* can't be true—it needs an individual, then it can be true (or false).

Suppose the construction of a proposition to be the end result of a (common kind of) sentence construction.

Swam needs an individual to be true or false. Fortunately, *Michael* is an individual. So combining *swam* (predicate) and *Michael* (argument) gives us a proposition, that can be true or false.

Verbs and participants

Intransitive (1-place)

(8) Bill slept

(9) * Bill slept the book

Transitive (2-place)

(10) * Bill hit

(11) Bill hit the pillow

Ditransitive (3-place)

(12) * Bill put

(13) * Bill put the book

(14) Bill put the book on the table

Weather (0-place)

(15) It rained

Predicates

The “participants” in an event denoted by the verb are the **arguments** of that verb. Some verbs require one argument, some require two, some require three, some require none. Intuitively, the number of arguments is the number of things that a verb needs in order to make a proposition (something that can be either true or false).

We will call verbs the **predicates**. They define properties of and/or relations between the arguments.

(16) Bill hit the ball

- There was a hitting, Bill did the hitting, the ball was affected by the hitting.

Different arguments have different **roles** in the event (e.g., the hitter, the hittee).

Thematic relations

The **thematic relation** that the argument has to the verb—the role it plays in the event—will prove useful in describing the behaviors of different classes of verb.

One thematic relation is **agent** of an action, like *Bill* in:

(17) Bill kicked the ball

Common thematic relations

Agent

initiator or doer in the event
(*Sue kicked the ball*)

Experiencer

feel or perceive the event
(*Pat likes pizza*)

Goal/Recipient

Chris ran to Copley Square
Pat gave the book to Tracy

Instrument

Ed cut the pie with a spoon

Theme/Patient

affected by the event, or undergoes the action
(*Sue kicked the ball*)

Proposition

a statement, can be true/false
(*Bill said that he likes pizza*)

Source

Mary took pens from the pile

Benefactive

Pat cooked dinner for Chris

Thematic relations

Armed with these terms, we can describe the semantic connection between the verb and its participants.

(18) Ray gave a grape to Bill

- Ray: Agent, Source, …
- A grape: Theme
- Bill: Goal, Recipient, …

Required vs. optional

Things that have certain thematic relations don’t seem to be *needed* by a given verb, but can be there. E.g., Location.

(19) Pat screamed (in the library)

Others, like Theme/Patient, Goal, or Agent, often do seem to be required. (“Required” means that even if left out, there is something assumed.)

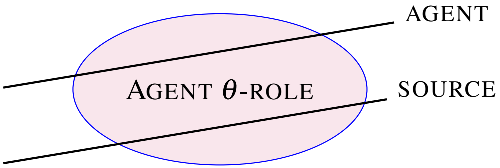
(20) Chris gave a book to Pat.

θ-roles

A participant can be in several thematic relations with the verb simultaneously (e.g., Agent, Goal).

In the syntax, we assign a sepcial connection to the verb called a "θ-role," which is a *collection* of thematic relations.

For the purposes of syntax, the θ-role (the collection of relations) is much more central than the actual relations in the collection. Generally referred to by the most prominent relation in the collection ("AGENT θ-ROLE").



Unique θ Generalization

Unique θ Generalization

Each θ-role must be assigned to a constituent, but a constituent cannot be assigned more than one θ-role.

Verbs, as part of their meaning (that is, whatever is recorded in the lexicon), are often “selective” about what kinds of arguments, θ-roles they have. They “select for” certain things.

Verbs have a certain number of θ-roles to assign (e.g., *eat* has two), and each of those must be assigned to a distinct argument. The θ-role does not necessarily map directly onto a specific syntactic category (the same θ-role seems in some cases to be able to be assigned to objects with different syntactic categories in different sentences. We will return to this.)

Kickings

The verb *kick* seems to require a nominal (category D) argument. Verbs differ, so we need this to be recorded in the lexicon.

Kick is a verb. It has a [V] feature.

It "needs" a DP, and DPs have a [D] feature. But we need to distinguish between being and needing.

Being a verb, the [V] feature of *kick* is **interpretable**—it is part of interpreting the content of *kick*. Needing a DP, *kick* has an **uninterpretable** [D] feature. The name tells us why the DP is required—the uninterpretable [D] feature is dangerous. It must be eliminated. Otherwise, there will be something that can't be interpreted at the interface.

Feature checking

If a syntactic object has an uninterpretable feature, it must Merge with a syntactic object that has a matching feature.

Once a uninterpretable feature has been Merged with a matching feature, the uninterpretable feature is **checked**. Neutralized, deleted.

Feature checking

Full Interpretation

The structure to which the semantic interface rules apply contains no uninterpretable features.

Checking Requirement

Uninterpretable features must be checked (and once checked, they are deleted)

Checking (under sisterhood)

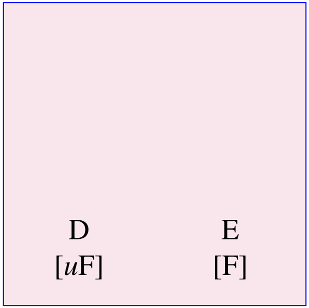
An uninterpretable feature F on a syntactic object Y is checked when Y is sister to another syntactic object Z which bears a matching feature F.

Feature checking

To distinguish interpretable features from uninterpretable features, we will write uninterpretable features with a *u* in front of them.

- D has uninterpretable feature F
- E has interpretable feature F

If we Merge them, the uninterpretable feature can be checked (under sisterhood).

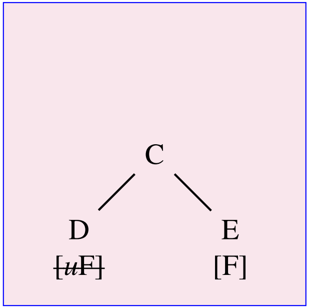


Feature checking

To distinguish interpretable features from uninterpretable features, we will write uninterpretable features with a *u* in front of them.

- D has uninterpretable feature F
- E has interpretable feature F

If we Merge them, the uninterpretable feature can be checked (under sisterhood).



Feature checking

More concrete example.

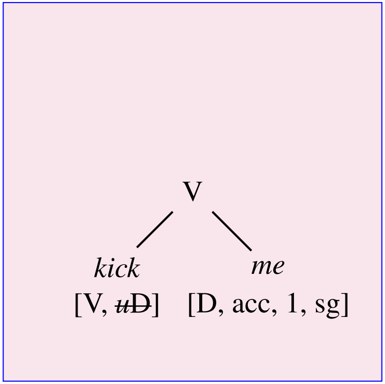
- *kick* is a verb (has an interpretable V feature) and requires a DP (has an uninterpretable D feature).
- *me* is a DP (a pronoun), has an interpretable D feature, as well as others like accusative case, first person, singular.

kick	*me*
[V, *u*D]	[D, acc, 1, sg]

Feature checking

More concrete example.

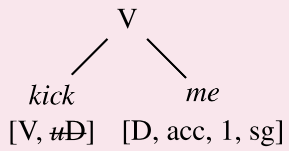
- *kick* is a verb (has an interpretable V feature) and requires a DP (has an uninterpretable D feature).
- *me* is a DP (a pronoun), has an interpretable D feature, as well as others like accusative case, first person, singular.



Feature checking

The head is the "needy" one. The one that had the uninterpretable feature that was checked by Merge.

The combination has the features of the verb *kick* and so its distribution would be like a verb's distribution would be.



Syntactic operations

- **Merge** is a syntactic operation. It takes two syntactic objects and creates a new one out of them.
- The new syntactic object created by Merge inherits the features of one of the components (the **head projects its features**).
- Merge cannot "tinker" inside a syntactic object. Syntactic objects are only combined at the root.

The Extension Condition

A syntactic derivation can only be continued by applying operations to the root projection of a tree.

Feature checking

Syntactic objects have features. Lexical items (syntactic objects) are bundles of features.

Some features are **interpretable**, others are **uninterpretable**.

By the time the derivation is finished, there must be no uninterpretable features left (*Full Interpretation*).

Uninterpretable features are eliminated by **checking** them against matching features. This happens as a result of Merge: Features of sisters can check against one another.

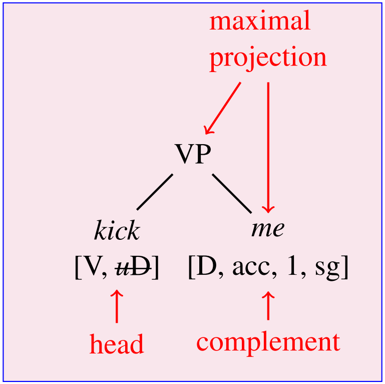
Merge doesn't just happen. It *has* to happen.

Heads and complements

When Merge combines two syntactic objects, one projects its features, one does not.

When a lexical item projects its features to the combined syntactic object, it is generally called the **head**, and the thing it combined with is generally called the **complement**.

A syntactic object that projects no further is called a **maximal projection**. Where X is the category, this is alternatively called "XP." The complement is necessarily a maximal projection.

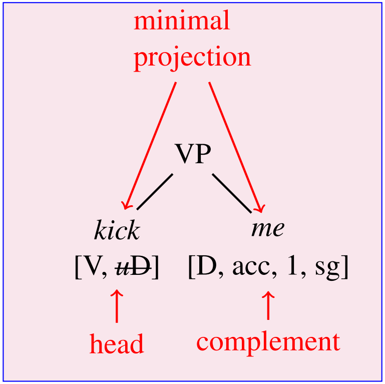


Heads and complements

A syntactic object that has not projected at all (that is, a lexical item) is sometimes called a **minimal projection**.

A **head**. Where X is the category, this is alternatively called “X” or “X^0.”

The head is a minimal projection.



Linear order

Merge takes two syntactic objects and combines them into a new syntactic object.

Merge does not specify linear order (which of the two combined objects comes first in pronuncation).

In the English VP, heads always precede complements. But languages differ on this. Languages generally have something like a basic word order for "neutral" sentences. Japanese SOV, Engish SVO. In our terms, this amounts to a (often language-wide choice) as to whether heads are pronounced before complements or vice-versa.

Second Merge

Merge occurs when there is a selectional feature that needs to be satisfied. If there is more than one such feature, Merge must happen more than once.

As always, the node that projects is the one whose selectional feature was satisfied by the Merge.

- The sister of the head (that projects) after the first Merge involving the head is called the **complement**.
- The nonprojecting sister of a syntactic object that has already projected once from a head is called a **specifer**.

In the English VP, heads always precede complements. But languages differ on this. Languages generally have something like a basic word order for “neutral” sentences. Japanese SOV, English SVO. In our terms, this amounts to a (often language-wide choice) as to whether heads are pronounced before complements or vice-versa.

Heads and complements

A transitive verb like *called* (or, even *kick*) really needs two arguments (the caller and the callee). We encode this knowledge by hypothesizing two selectional features for D. The first will be checked by the callee, and the second will be checked by the caller.

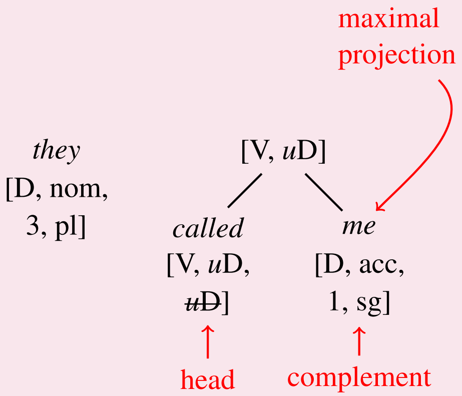
they
[D, nom, 3, pl]

called
[V, *u*D, *u*D]

me
[D, acc, 1, sg]

Heads and complements

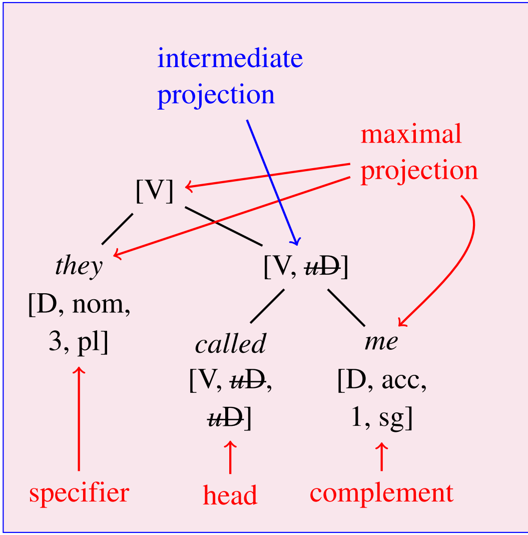
Merge *called* and *me*, checking one of the *u*D features. The features of *called* project.



Heads and complements and specifiers

Merge *they* and *called me*, checking the remaining *u*D feature. The features of *called* project again.

The nonprojecting sister in this second Merge is the **specifier**. A node that has been projected and then projects further is neither maximal nor minimal, and is called an **intermediate projection**.



Node labeling conventions

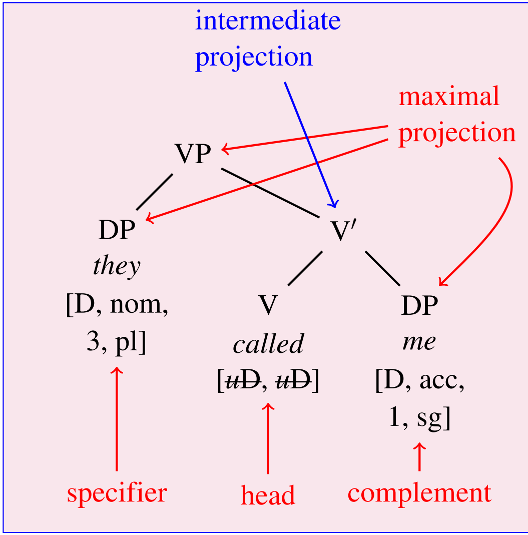
When we Merge two objects, the features of one of them project to become the features of the new object.

The label for the new node comes in two pieces.

- The **category** (projected from the head).
- The **projection "level"**:
  - P = maximal projection (or both maximal and minimal)
  - $^{\text{o}}$ or nothing = minimal projection
  - $'$ = intermediate projection

Heads and complements and specifiers and labels

An XP is any node that does not project its features up. An X (or X^o) node comes from the lexicon. We can write the category feature as the label rather than in the feature list. When projected features are checked, we check them off on the original object as well—so for simplicity, we can just do the accounting all under the head.



Adjuncts

(21) * Pat put the book.

(22) Pat put the book on the shelf.

(23) Pat put the book on the shelf dramatically.

(24) Pat put the book on the shelf dramatically on Tuesday.

(25) Pat put the book on the shelf dramatically on Tuesday before several witnesses.

Some things are required, some things are not.

- **Arguments** get θ-roles and are **required**.
- **Adjuncts** are modificational and are **optional**.

Adjuncts and distribution

Adjuncts are relatively "transparent"—having an adjunct does not seem to change the distributional characteristics of what it is attached to.

(26) Pat wants to eat lunch (quickly).

(27) Pat wants to dine.

(28) * I like to draw eat lunch (quickly).

(29) I like to draw (happy) elephants.

(30) * Pat wants to (happy) elephants.

Idea: A verb (phrase) with an adjunct is still a verb (phrase) just as if it didn't have an adjunct.

Adjoin

The operations **Merge** and **Adjoin** are two different ways to combine two objects from the workbench.

Merge takes two objects and creates a new object (with the label/features inherited from one of them).

Adjoin attaches one object on top of another one. The linear order of adjuncts does not appear to be set parametrically, so they can (often) appear either before or after the object they attach to.

The luxury of adjunction

We will also assume that **Adjoin only applies to maximal projections**.

That is: if a syntactic object still has a selectional feature, Adjoin cannot attach something to it. Merge must happen first. Once all the things that *need* to happen are taken care of, *then* you have the luxury of adjoining something.

Any number of adjuncts can be added, generally in any order. Adjuncts come in many different categories—“adjunct” is not a category but rather a structural description.

Representation of adjuncts

At a technical level, it might be worth thinking a moment about how these things are represented. The distinction between Merge and Adjoin is pretty subtle. Both combine two things into one, and results in something that has a label/features.

Adjunction is more asymmetrical than Merge. You Adjoin something to something else; you Merge two things together. In either case you need to designate which of the two constituent objects is providing the label, the features of the combined structure.

Representation of adjuncts

There are a couple of mathematical ways to combine two objects but highlight one of them. Merge of X, Y has usually been represented as doing this: {X, {X, Y}}. The outer X is the label, always one of the two elements of the internal set.

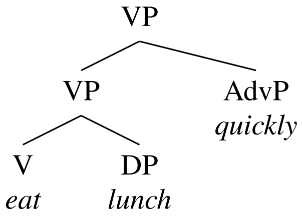
The other way we can create a set but make one element more prominent is to introduce order, to create an ordered pair: <X, Y>. In such a structure, X is first, and we can take the first element to be the label. This is nearly information-equivalent to the set representation above. But it feels emotionally like a better candidate for adjunction just due to its fundamentally asymmetric nature.

Representation of adjuncts

So, if we were going to form *eat lunch quickly*, where *eat lunch* forms a VP and *quickly* is adjoined to it, we would have something like this:

{eat, {eat, lunch}}

<{eat, {eat, lunch}}, quickly>



Representation of adjuncts

Also: it feels like maybe {X, {X, Y}} is still too complicated a representation for the output of Merge. The ideal would be if Merge just formed {X, Y} and Adjoin formed <X, Y>. That would fit even better into this feeling that Merge is fundamentally symmetrical whereas Adjoin is fundamentally asymmetrical.

We could get there if we could always predict, based on the properties of X and Y considered together, which of the two would be the label. We're kind of working toward that. If we get there, then Merge creates things that are as simple as they could be, and Adjoin creates things that are only one step more complex.

<{eat, lunch}, quickly>

Node labeling conventions (revised)

When we Merge two objects, the features of one of them project to become the features of the new object. When we Adjoin one object to another, the node being adjoined to does not change, it labels the combined object the same as itself.

The label for the new node comes in two pieces.

- The **category** (projected from the head).
- The **projection "level"**:
  - P = maximal projection (or both maximal and minimal)
  - ^{o} or nothing = minimal projection
  - $'$ = intermediate projection
- Adjuncts do not change projection level, so the labels of the combined object and the target of adjunction match exactly, including in projection level.