Chain Formation I

Robin Clark
Département de linguistique - LATL
Université de Genève

Janvier 1991

In this note, I will consider some relatively simple cases of $\overline{A}$-chain formation. By simple, I mean cases where the head of the $\overline{A}$-chain is phonologically overt as opposed to, for example, a non-overt operator. In addition, since the conditions for initiating an $\overline{A}$-chain and for licensing intermediate links in the chain are rather different from those governing A-chains, I will set aside the latter.

$\overline{A}$-chains are implicated in the following structures:

(1)a. $[_{DP}$ who$]_i$ t_i saw John?

b. $[_{DP}$ what$]_i$ did John see t_i

c. $[_{DP}$ who$]_i$ did John give the book to t_i

d. $[_{PP}$ to whom$]_i$ did John give the book t_i

e. $[_{DP/AdvP}$ when$]_i$ did John give the book to Bill t_i

f. $[_{AdvP}$ why$]_i$ did John give the book to Bill t_i

g. $[_{AdvP}$ how$]_i$ did John give the book to Bill t_i

h. $[_{DP}$ which day$]_i$ did John give the book to Bill t_i

i. $[_{AdjP}$ how raw$]_i$ did John eat the meat t_i

(2)a. the man who$_i$ t_i saw John?

b. the book that John saw t_i

c. the man who$_i$ John gave the book to t_i

d. the man to whom$_i$, John gave the book t_i

e. the day John gave the book to Bill t_i

f. the reason why$_i$ John gave the book to Bill t_i

g. the way John gave the book to Bill t_i

h. the degree of rawness that John ate the meat t_i

(3) This violin is easy to play this sonata on t_i

(4)a. $[_{DP}$ this book$]_i$, John saw t_i

b. $[_{DP}$ this man$]_i$, John gave the book to t_i

c. $[_{PP}$ to this man$]_i$, John gave the book t_i

d. John promised he would run away and $[_{VP}$ run away$]_i$ he will t_i

As can be seen from the above, $\overline{A}$-chains are implicated in the formation of *wh*-questions, relative clauses, *tough* movement and topicalization. Furthermore, all categories are capable of being displaced to an $\overline{A}$-position.

An algorithm for constructing well-formed $\overline{A}$-chains must take into account the following reuirements:

1. Conditions for initiating an $\overline{A}$-chain, including, by definition, the search for its tail (gap-positing).
2. The information contained in chain.
3. Conditions for terminating the chain by positing a tail.
4. Conditions for licensing intermediate links in the chain.
5. Locality and other structural conditions on chain links.
6. Data structures for representing and performing calculations on chains.

The first requirement can be satisfied by stipulating that the chain-formation mechanism goes into effect just in case an item in an $\overline{A}$-position is encountered. Given that we are restricting our attention to overt *wh*-phrases, we need only say (for the moment) that when the parser stipulates the presence of a phrase in the Spec-CP, it begins to form an $\overline{A}$-chain. More generally, whenever an element is hypothesized by the parser to be in an $\overline{A}$-position (Spec-CP or TP-adjoined, in the case of topicalization), it initiates the formation of an $\overline{A}$-chain.

I will assume that a chain (either an A- or an $\overline{A}$-chain) consists of an ordered list of structural positions. The first position, the *head*, is the elements that initiates the chain-formation algorithm. The last position, the *tail*, is the A-position which terminates the chain-formation algorithm relative to a representation:

(5) $(\text{pos}_1, \ldots, \text{pos}_n)$ is an $\overline{A}$-chain in phrase marker P if:
pos_1 is an $\overline{A}$-position in P.
pos_n is an A-position in P.
If $i \neq n$, the pos_i is an $\overline{A}$-position in P.
If $i < j$, then pos_i c-commands pos_j in P.
pos_{i+1} is subjacent to pos_i in P.

Intermediate positions are posited to satisfy locality conditions on chain-links. Note that, on this view, there is no need to represent gaps explicitly in the phrase-marker since information about the position of the gaps can be derived from the chain, which I assume to be independent of the phrase marker. Thus, explicitly representing the gaps both in the phrase marker and the chain would be redundant.

In the case of DPs, the search for a well-formed $\overline{A}$-chain terminates when a gap in Case position is found. Note that this condition is distinct from terminating a chain when an appropriate θ-position is found. Consider, for example, passives or raising structures. In these cases, the $\overline{A}$-chain terminates in a position that is [+Case, $-\theta$]. The requirement that chains terminate in a θ-position holds of A-chains headed by a true arguement (as opposed to an expletive) and not for $\overline{A}$-chains. However, in order for an $\overline{A}$-chain to be well-formed, its tail must head a well-formed nonexpletive A-chain. Hence, the requirement that $\overline{A}$-chains be associated with a θ-role will hold. In the case of categories other than DP, the search for a well-formed $\overline{A}$-chain terminates when that position is licensed either by lexical requirements (subcategorization requirements of a head) or by positing a gap in an adjunct position. As a general rule of thumb, the former (subcategorization) takes precedence over the latter (adjunction).

A syntactic trick for terminating the chain formation can be stated as follows:

(6) Terminate $\overline{A}$-chain formation by positing a gap in:
a. the Spec position of a [+tense] TP (for DPs); or
b. sister position of an X^0 (if X^0 can take the head of the chain as a complement); or
c. an adjoined position.

The third condition is, however, somewhat more subtle than the preceding statement implies in that one must carefully distinguish between terminating a particular $\overline{A}$-chain in an A-position and terminating the search for the best $\overline{A}$-chain relative to the sentence being parsed. Consider the following examples:

(7)a. John warned Bill that the students should avoid Mary.

b. John warned that the students should avoid Mary.

c. Who$_i$ did John warn t_i that the students should avoid Mary?

d. Who$_i$ did John warn that the students should avoid t_i?

As shown by (7a) and (7b), *warn* may take as its complement a *DP* followed by a *CP* or simply by a *CP*:

(8)a. *warn*$_1$; [__ *DP CP*]

b. *warn*$_2$; [__ *CP*]

As shown in (7c) and (7d), an $\overline{A}$-chain might terminate in the A-position that is θ-governed by *warn* or it might terminate inside the clausal complement to *warn*. Thus, consider the parse up to the point where the following fragment has been encountered:

(9) Who$_i$ did John warn that the ...

Suppose that the parser has decided that it is now entering a CP which is attached as complement to *warn*. It can either posit a gap as a DP complement to *warn*$_1$, thus terminating the $\overline{A}$-chain headed by *who*$_i$, or it can assume that the tail of the chain is located within the CP complement of *warn*$_2$. If it selects only one of the two possibilities, it stands a good chance

of failing to parse the input correctly. For example, if it posits a gap in the complement position of *warn*, as in (10a), there is some possibility that the continuation will be as in (10b). Positing an additional gap, as in (10c), would violate either the binding conditions or the θ-Criterion, and so cannot be permitted:

(10)a. Who$_i$ did John warn t_i that the ...

b. Who$_i$ did John warn t_i that the students should avoid

c. Who$_i$ did John warn t_i that the students should avoid t_i

Similarly, failing to posit a gap after *warn*, as in (11a) will result in an error when the continuation is as in (11b) since the *wh*-operator will fail to bind a variable:

(11)a. Who$_i$ did John warn that the ...

b. Who$_i$ did John warn that the students should avoid Mary.

The above considerations indicate that the parser should pursue hypotheses about the form of $\overline{A}$-chains in parallel in order to be fault-tolerant. In particular, the ambiguity of *warn* indicates that the parser has reached a choice point and must continue in parallel. This will be the case whenever lexical information and the Extended Projection Principle underdetermine the locus of the tail of an $\overline{A}$-chain. This will occur whenever there is a lexical ambiguity of the same structure as the one illustrated above or when the $\overline{A}$-chain is associated with an adjunct as in (12):

(12)a. Why did John think that Mary left?

b. [$_{CP}$ why$_i$ did [$_{TP}$ John think [$_{CP}$ that [$_{TP}$ Mary left]] t_i]]

c. [$_{CP}$ why$_i$ did [$_{TP}$ John think [$_{CP}$ t_i that [$_{TP}$ Mary left t_i]]]]

In the above example, the adverbial *why* is not a complement of any lexical head and, as a result, subcategorization information cannot be used to recover the locus of the tail of the $\overline{A}$-chain headed by it. In principle, *why* may be taken as either modifying the matrix *TP* ("what reason does John have for thinking that Mary left?") or it may be taken as modifying the embedded *TP* ("what reason is such that John thinks that Mary left for that reason?"), although the former is a more accessible reading than the latter. Ambiguous structures like (12a) show that the chain formation algorithm must be capable of going in parallel when it encounters a possible, but not obligatory locus for the tail of an $\overline{A}$-chain.

The above parallelism can be treated in a serial fashion if chains themselves are as a list containing (1) the name of a position and (2) a pointer to the next position.[1] The tail of the chain would, then, contain only the name of a position and no pointer to a following position. A general well-formedness condition on chains would be that once a tail has been found, that position cannot point to further positions. Notice that the chain formation algorithm need never see the entire list of positions on the list while it is running; it needs only have access to the category of the element occupying the head position of the chain and the most recent

[1]This treats a chain in much the same way that LISP treats a list, ie., as a *cons* cell containing the name of an element and the address of the next *cons* cell on the list.

link in the chain. If multiple pointers are allowed, then the various possible chains can be represented parsimoniously.

This can be represented in terms of a list, where the head of the list is a position, pos_i, and the following elements are themselves lists headed by potential next elements of a chain. Thus, an unambiguous chain would be represented as:

(13) $(\text{pos}_1\ (\text{pos}_2\ (\text{pos}_3\ ())))$

In the above, pos_1 is the head and pos_3 is the tail since it points to nothing.[2] A parallel set of chains is shown in (14):

(14) $(\text{pos}_1\ (\text{pos}_2\ (\text{pos}_3\ (\text{pos}_4\ ()))\ (\text{pos}'_3\ (\text{pos}'_4\ ()))))$

In (14), the list headed by pos_2 contains two other elements, the list headed by pos_3 and the list headed by pos'_3. Notice that if a possible chain has no legal continuation, it can be abandoned by going back to the most recent three element list and pruning the offending element.

Consider, next, locality conditions on chains. In general, we want the following facts to follow from the chain formation algorithm (putting aside parasitic gap constuctions):

1. Subparts of a left branch may not contain a gap.
2. Adjuncts cannot contain a gap.
3. Complex NPs (DPs with relative clauses) and *wh*-islands may not contain gaps bound outside them.

The above conditions follow by placing the following conditions on the positing of intermediate gaps:

1. In the configuration $[_{\overline{X}}\ \text{X}^0\ YP]$, place YP as the last position in the chain just in case XP is the current last position in the chain.
2. If YP is the most recent element in a chain and $YP \neq TP$, the Spec of YP must be structurally absent.
3. If the tail of a chain occurs as a daughter of a projection of Y, then YP is the most recent position on the chain.

By condition (1), a link in a chain will always c-command its successor. By conditions (1) and (3), a gap may occur neither within an constituent which is in Spec position (not a sister to an X^0) nor as a subpart of an adjunct (again, not a sister to an X^0). Furthermore, the *wh*-island condition and CNPC follow from conditions (1), (2) and (3) since the three conditions

[2] This makes it evident that I'm a LISP programmer.

effectively conspire to block gap positing from an external chain in embedded questions and relative clauses.[3]

Notice that this method successfully distinguishes between:

(15)a. *who$_i$ does John believe [the rumor [which$_j$ [Bill told t_j to t_i]]]

b. ?Who$_i$ does John believe [the rumor [that [Bill loves t_i]]]

In (15a), *rumor* is modified by a relative clause which is not a lexical complement and which contains material in its Spec (namely, *which*). Thus, by the above set of conditions, the chain headed by *who* cannot have a continuation within this relative clause. In (15b), on the other hand, *rumor* takes a *CP* as its complement and this *CP* has no material in its Spec. Thus, the chain headed by *who* can have a continuation inside the *CP*.

Before considering particular examples, another condition on gaps should be mentioned:

> If the choice is between attaching a gap as a complement (or subject) and attaching the gap as an adjunct, attach the gap as a complement (or subject).

Consider a sentence like:

(16) Tomorrow, John dreads.

This condition guarantees that *tomorrow* will be interpreted as the object of *dread* and not as an adjunct.

Finally, ranking the possible chains is contingent on finding the shortest chain in a group. Consider an example like:

(17) Who did John promise Mary would leave?

This example is ambiguous depending on whether *Mary* is taken as the object of *promise* or the subject of *would leave*. The former reading is, however, strongly preferred over the latter.

Consider the progression of steps involved in the formation of chains in the examples discussed earlier. Since the structures are so close, I will discuss them in parallel:

(18)a. Who$_i$ did John warn t_i that the students should avoid Mary?

b. Who$_i$ did John warn that the students should avoid t_i?

[3] I assume here that all relative clauses have an operator in the Spec of *CP*, even when that operator is null as in:

(i) the man Op$_i$ John saw t_i

Note that no explicit material marks the left-edge of the relative clause in (i). In this case, chain formation should be triggered by stipulation. This problem is, of course, independent of the general problem of chain formation.

The parser initially encounters *who did* which can receive an analysis where *who* is in the Spec of *CP* and *did* is in C^0. The presence of *who* in the Spec-*CP* triggers the chain formation mechanism with the position occupied by *who* as the head of an $\overline{A}$-chain. For both (18a) and (18b), the parser will then enter a TP. The TP is then entered on the list structure representing the chain; hence, for both examples, the chain structure will be:

(19) (Spec-CP_1 (TP_1))

In both cases, the algorithm will attempt to posit a gap in the Spec of TP. This will fail since a lexical DP is present which can be attached in this position. The parser next encounters the VP headed by *warn*. Since the Spec of this VP is null, it is added to the list:

(20) (Spec-CP_1 (TP_1 (VP_1)))

Recall that *warn* is ambiguous between taking a complement DP and CP and taking a complement CP alone. Thus, the possible chains must be pursued in parallel. One structure posits a gap as complement of the verb *warn*, thus terminating the chain, and the other does not:

(21) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ())) (VP_1)))

At this point, the parser encounters the complementizer, *that*, and projects a CP. Since the CP has a null Spec, it too can be added to the list:

(22) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ())) (VP_1 (CP_2))))

When an embedded TP is projected, it is appended to the list:

(23) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ())) (VP_1 (CP_2 (TP_2)))))

Again, gap positing in the Spec of the TP is blocked by the presence of a lexical DP, *the students*.[4] Finally, the parser enters the embedded VP. Again, since the VP has a null Spec, it can be added to the list:

(24) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ())) (VP_1 (CP_2 (TP_2 (VP_2))))))

At this point, the structures become non-parallel. In the case of (25), gap positing after avoid is blocked, due to the presence of *Mary*:

(25) Who$_i$ did John warn t_i that the students should avoid Mary?

When the parser hits the end of sentence marker, the chain fragment entering the embedded CP is pruned since it does not terminate in a legal tail. This leaves the chain in (26):[5]

[4]Recall that gap positing within the DP is also blocked since it is not a sister to an X^0.

[5]I ignore the adjunction possibility, for simplicity. This is a syntactic possibility for DPs since temporal DPs like *yesterday* can adjoin.

(26) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ()))))

The above correctly indicates that there is a legal chain extending from the Spec of the matrix CP to the complement of *warn*.

In the case of (27), gap positing can occur after *avoid*, since *avoid* is (strongly) transitive and no lexical DP is available to block attachment of the gap:

(27) Who$_i$ did John warn that the students should avoid t_i?

Thus the chains will appear as in (28):

(28) (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ())) (VP_1 (CP_2 (TP_2 (VP_2 (Compl-V_2 ()))))))))

The above unpacks as two chains:

(29)a. (Spec-CP_1 (TP_1 (VP_1 (Compl-V_1 ()))))

b. (Spec-CP_1 (TP_1 (VP_1 (CP_2 (TP_2 (VP_2 (Compl-V_2 ()))))))))

Since V_1 (*warn*) only optionally takes a DP complement while V_2 obligatorily takes one, only (29b) is correct since only in this representation are the lexical requirements of all the heads satisfied. Thus, the Projection Principle must act as a filter on the output of the chain component.

A final note. Consider:

(30) To whom did John shout to throw the ball.

In the above, both *shout* and *throw* can optionally take a PP headed by *to*. The above algorithm correctly predicts that (30) is ambiguous between the reading where *to whom* is a complement to *shout* and the reading where it is a complement to *throw*. Note that it does so from only one parse tree, given the convention that traces are only represented in chains. Furthermore, given the packed representation for chains, the algorithm avoids copying identical positions across hypothesized chains. This is about as parsimonious as one is entitled to expect.