1 Changelog

- 27.05.2021: Fixed Eq. 1, RE case
- 27.05.2021: Fixed Eq. 3 (the equation in the lecture slides is incorrect)
- 31.05.2021: Added an extended description of the equations for the dynamic oracle (for the arc-eager system) in Sec. 2.2
- 19.07.2021: Fixed two instances where it said "arc-standard" but should say "arc-eager".

2 Arc-eager parser

We focus on the *arc-eager parser*, for which a *static oracle* is defined in Sec. 2.1 and a *dynamic oracle* is defined in Sec. 2.2. We assume the version of the arc-eager parser based on the following rules (or transitions):

$$\begin{aligned}
\text{LA}_k: &\quad \frac{(\sigma|i, j|\beta, A)}{(\sigma, j|\beta, A \cup \{(j, i, k')\})} && i \neq 0 \wedge \neg\exists_{i',k'}(i', i, k') \in A \\
\text{RA}_k: &\quad \frac{(\sigma|i, j|\beta, A)}{(\sigma|i|j, \beta, A \cup \{(i, j, k')\})} && \neg\exists_{i',k'}(i', j, k') \in A \\
\text{RE}: &\quad \frac{(\sigma|i, \beta, A)}{(\sigma, \beta, A)} && \exists_{i',k'}(i', i, k') \in A \\
\text{SH}: &\quad \frac{(\sigma, i|\beta, A)}{(\sigma|i, \beta, A)}
\end{aligned}$$

The horizontal line is used instead of $\Rightarrow$ to separate the premise from the conclusion for the sake of readability. The left-arc (LA) transition has the precondition that $i \neq 0$ because the first word of the input sentence is assumed to be the dummy root. It also requires that $\neg\exists_{i',k'}(i', i, k') \in A$, i.e., that the dependent i of the arc $i \leftarrow j$ being created does not yet have a head specified in A (if it had, applying LA would break the single-head property of dependency trees). The right-arc (RA) transition has a similar precondition ($\neg\exists_{i',k'}(i', j, k') \in A$), which is actually redundant because no words in the buffer (and j is in the buffer) can have a head specified.[1] Finally, the reduce transition (RE) has the precondition that the word being reduced from the stack already has a head specified ($\exists_{i',k'}(i', i, k') \in A$). The shift transition (SH) apparently has no preconditions, but in reality it has one implicit precondition: the buffer must be not empty ($i|\beta$). Other rules similarly have such implicit preconditions: LA and RA require that both stack and buffer are non-empty, and RE that the stack is non-empty.

2.1 Static oracle

A static oracle for the arc-eager parser specifies the next transition to take for a given parsing configuration. Let $c = (\sigma|i, j|\beta, A)$ be a configuration[2] and T be the gold tree. The oracle is then defined as follows:

$$o(c, T) = \begin{cases} \text{LA}_k & i \leftarrow j \text{ in } T \\ \text{RA}_k & i \rightarrow j \text{ in } T \\ \text{RE} & \exists_{v<i}\ v \leftrightarrow j \text{ in } T \wedge \exists_{i',k'}(i', i, k') \in A \\ \text{SH} & \text{otherwise} \end{cases} \tag{1}$$

[1] When a word is attached as dependent to another word it is either discarded completely (LA) or moved from the buffer to the stack (RA). Moreover, none of the rules allows to move a word from the stack back to the buffer (in contrast with e.g. the online reordering system, where swap makes this possible). Hence, a word which is still in the buffer cannot have a head specified. Similarly, the precondition of the target word not having a head is redundant in the arc-standard transition system for both LA and RA.

[2] In a slight abuse of notation, we assume that i is undefined if the stack is empty and that j is undefined if the buffer is empty. Note that for the sake of consistency with the rest of this document we follow a slightly different notation than in the lecture (slide 6).

So, for instance, LA should be selected if there is a left arc $i \leftarrow j$ in the gold tree, and RE if there is a word v on the left of i directly connected with j in T (provided that neither LA nor RA applies – the individual cases are considered in order from LA to SH).

The above defined oracle is *correct* in the sense that, given projective dependency tree T, it allows to derive a sequence of configurations and transitions (starting from the initial configuration) which reconstructs T precisely. Additionally, it is *deterministic*, i.e., it only allows to derive a single gold transition sequence (even when many exists which reconstruct T!), and *incomplete*. The latter means that the oracle only ,,makes sense" for the configurations which make part of the gold transition sequence. This leads to suboptimal behavior which is exemplified below and contrasted with the behavior of a dynamic oracle.

2.1.1 Example

Let the input sentence be *ROOT He sent her a letter .* and the corresponding gold tree be the one from Fig. 1. In what follows we ignore the labels of the arcs for simplity. The transition sequence determined by the

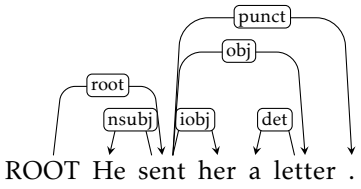

Figure 1: Example tree

static oracle (Eq. 1) is then:

ROW	TRANSITION	STACK	BUFFER	ARCS
1		[ROOT]	[He sent her a letter .]	$\emptyset$
2	SH	[ROOT He]	[sent her a letter .]	
3	LA	[ROOT]	[sent her a letter .]	+(He ← sent)
4	RA	[ROOT sent]	[her a letter .]	+(ROOT → sent)
5	RA	[ROOT sent her]	[a letter .]	+(sent → her)
6	SH	[ROOT sent her a]	[letter .]	
7	LA	[ROOT sent her]	[letter .]	+(a ← letter)
8	RE	[ROOT sent]	[letter .]	
9	RA	[ROOT sent letter]	[.]	+(sent → letter)
10	RE	[ROOT sent]	[.]	
11	RA	[ROOT sent .]	[]	+(sent → .)

While it allows to retrieve all the arcs, it is not the only such sequence:

ROW	TRANSITION	STACK	BUFFER	ARCS
1		[ROOT]	[He sent her a letter .]	$\emptyset$
2	SH	[ROOT He]	[sent her a letter .]	
3	LA	[ROOT]	[sent her a letter .]	+(He ← sent)
4	RA	[ROOT sent]	[her a letter .]	+(ROOT → sent)
5	RA	[ROOT sent her]	[a letter .]	+(sent → her)
6	**RE**	[ROOT sent]	[a letter .]	
7	**SH**	[ROOT sent a]	[letter .]	
8	**LA**	[ROOT sent]	[letter .]	+(a ← letter)
9	RA	[ROOT sent letter]	[.]	+(sent → letter)
10	RE	[ROOT sent]	[.]	
11	RA	[ROOT sent .]	[]	+(sent → .)

In this sequence, *her* is reduced earlier from the stack and the transitions differ in rows 6, 7 and 8. Yet the resulting set of arcs is the same (see the last column of the table). A static oracle does not account for such spurious ambiguity.

Moreover, a static oracle is suboptimal when we consider the possibility of the system performing erroneous transitions during parsing. Let's say the parser (incorrectly) decides to shift *her* instead of attaching it to *sent* as indirect object, as shown in Tab. 1. From this moment on, *her* cannot become a right-dependent

ROW	TRANSITION	STACK	BUFFER	ARCS
...	...	...	...	...
4	RA	[ROOT sent]	[her a letter .]	+(ROOT → sent)
5	SH	[ROOT sent her]	[a letter .]	

Table 1: Incorrect transition

of any other word on the stack anymore. We could still make it a left-dependent of some word in the buffer, but this is not licensed by the static oracle, which only allows to make an arc between two words if it is present in the gold tree. Moreover, it is not possible to reduce it anymore either, since it doesn't have a head specified, which is a precondition of applying the RE transition in the first place! All this leads to an arguably very poor continuation, with only 3 our of 6 arcs recovered (see Tab. 2). Note in particular the two

ROW	TRANSITION	STACK	BUFFER	ARCS
...	...	...	...	...
5	SH	[ROOT sent her]	[a letter .]	
6	SH	[ROOT sent her a]	[letter .]	
7	LA	[ROOT sent her]	[letter .]	+(a ← letter)
8	SH	[ROOT sent her letter]	[.]	
9	SH	[ROOT sent her letter .]	[]	

Table 2: Error propagation

last transitions, where RE is not possible since the top element of the stack (resp. *her* and *letter*) is headless (cf. the precondition of RE in Sec. 2).

It is thus questionable if training the classifier to always model the static oracle is the best idea, as this can make the system more susceptible to error propagation.

2.2 Dynamic oracle

A dynamic oracle allows to alleviate both issues mentioned within the context of the static oracle (see Sec. 2.1). On the one hand, a dynamic oracle allows to account for the spurious ambiguity, i.e., the fact that there may be many different transition sequences which allow to reconstruct the gold tree. On the other hand, it's complete, i.e. it provides the best (or optimal) transition(s) to follow for every configuration reachable from the initial configuration, whether it's a part of the gold transition sequence or not. A transition t is *optimal* for a configuration c if (at least) one of the best dependency trees (i.e. the trees most similar to the gold tree) reachable from c is still reachable from $t(c)$. In particular, if the gold tree is reachable from c, t is optimal iff it is still possible to reconstruct the gold tree after applying t to c.

The concept of dynamic oracle is general and applies to other transition systems as well, but we focus here on the dynamic oracle for the arc-eager system only.

For example, let $c = ([\text{ROOT sent her}], [\text{a letter .}], \{\text{He} \leftarrow \text{sent}, \text{ROOT} \rightarrow \text{sent}, \text{sent} \rightarrow \text{her}\})$ and T be the tree from Fig. 1. While, according to the static oracle, SH should be performed at this point, both SH and RE are clearly optimal as both allow to reconstruct the entire gold dependency tree, as shown in Sec. 2.1. Now let's consider $c = ([\text{ROOT sent her}], [\text{a letter .}], \{\text{He} \leftarrow \text{sent}, \text{ROOT} \rightarrow \text{sent}\})$ from Table 1, where *her* is incorrectly shifted to the stack. As mentioned above, by that point it is no longer possible to restore the arc *sent* → *her*, but this doesn't mean that the transition sequence from Table 2 (determined by the static oracle) is the best we can do. Indeed, an alternative sequence of transitions presented in Tab. 3 allows to restore 5 out of 6 arcs from the tree in Fig. 1. Since this is the best we can do, the subsequent transitions LA SH LA RA RE RA are all optimal. But is that the only option? Not really! Another one is shown in Tab. 4,

ROW	TRANSITION	STACK	BUFFER	ARCS
...	...	...	...	...
5	SH	[ROOT sent her]	[a letter .]	
6	LA	[ROOT sent]	[a letter .]	+(her ← a)
7	SH	[ROOT sent a]	[letter .]	
8	LA	[ROOT sent]	[letter .]	+(a ← letter)
9	RA	[ROOT sent letter]	[.]	+(sent → letter)
10	RE	[ROOT sent]	[.]	
11	RA	[ROOT sent .]	[]	+(sent → .)

Table 3: Error recovery

where *her* gets attached to *letter* instead. Therefore, both LA and SH are optimal w.r.t. the configuration in

ROW	TRANSITION	STACK	BUFFER	ARCS
...	...	...	...	...
5	SH	[ROOT sent her]	[a letter .]	
6	SH	[ROOT sent her a]	[letter .]	
7	LA	[ROOT sent her]	[letter .]	+(a ← letter)
8	**LA**	[ROOT sent]	[letter .]	+(her ← letter)
9	RA	[ROOT sent letter]	[.]	+(sent → letter)
10	RE	[ROOT sent]	[.]	
11	RA	[ROOT sent .]	[]	+(sent → .)

Table 4: Error recovery – alternative

row 5.

But how do we determine what are the optimal transitions without considering all the possible continuations and checking if a best possible tree is still reachable? Fortunately this is not too difficult in case of the arc-eager system. First of all, thanks to arc-decomposability of the system (see Sec. 2.3), we can consider all the arcs separately, without having to look at entire trees. More precisely, it is sufficient to verify that every arc reachable from the current configuration c is still reachable from $t(c)$ to tell that t is optimal. Furthermore, there is actually a closed formula which makes it easier determine whether a given transition is optimal or not for a given $c = (\sigma|i, j|\beta, A)$ and gold tree T in the arc-eager system:[3]

$$o(c, \mathrm{LA}, T) = \begin{cases} \textbf{false} & \exists_{w \in j|\beta}\ i \rightarrow w \ \vee\ \exists_{w \in \beta}\ i \leftarrow w \\ \textbf{true} & \text{otherwise} \end{cases} \tag{2}$$

$$o(c, \mathrm{RA}, T) = \begin{cases} \textbf{false} & \exists_{w \in \sigma|i}\ w \leftarrow j \ \vee\ \exists_{w \in \sigma}\ w \rightarrow j \ \vee\ \exists_{w \in \beta}\ j \leftarrow w \\ \textbf{true} & \text{otherwise} \end{cases} \tag{3}$$

$$o(c, \mathrm{RE}, T) = \begin{cases} \textbf{false} & \exists_{w \in j|\beta}\ i \rightarrow w \\ \textbf{true} & \text{otherwise} \end{cases} \tag{4}$$

$$o(c, \mathrm{SH}, T) = \begin{cases} \textbf{false} & \exists_{w \in \sigma|i}\ w \leftrightarrow j \\ \textbf{true} & \text{otherwise} \end{cases} \tag{5}$$

The formulas above can be interpreted as follows:

- Eq. 2: Creating the left arc $i \leftarrow j$ from the first buffer element j to the top of the stack i is not optimal if there's a right arc from i to some word w in the buffer $j|\beta$ in the gold tree ($\exists_{w \in j|\beta}\ i \rightarrow w$). Creating

[3]Note that a dynamic oracle is defined as a function of 3 arguments – configuration, transition, and dependency tree – and returns **true** iff the transition is optimal within the context of the configuration and the tree (a static oracle, in contrast, takes on input a configuration and a tree and returns a transition).

$i \leftarrow j$ would make it impossible to restore $i \rightarrow w$, since LA removes i from the stack.[4] Similarly, creating $i \leftarrow j$ is not optimal if $i \leftarrow w$ is in the gold tree for some $w \in \beta$ ($\exists_{w \in \beta}\ i \leftarrow w$) – not only would i be removed from the stack, but also it would have a head different than w (and in dependency forests one word cannot have more than one head).[5]

- Eq. 3: Creating the right arc $i \rightarrow j$ is not optimal if some word w in the stack $\sigma|i$ is a dependent of j in the gold tree ($\exists_{w \in \sigma|i}\ w \leftarrow j$). This is because, once $i \rightarrow j$ is created, j is moved to the stack (and cannot ever go back to the buffer), and it's not possible to create an arc between two words in the stack, hence it's not possible to restore $w \leftarrow j$ anymore. Besides, $i \rightarrow j$ is not optimal if j has its head somewhere else in the stack ($\exists_{w \in \sigma}\ w \rightarrow j$) or the buffer ($\exists_{w \in \beta}\ j \leftarrow w$), since a word can only have one head (either i or w, but not both).

- Eq. 4: Reducing i from the stack is not optimal if i has a dependent somewhere in the buffer ($\exists_{w \in j|\beta}\ i \rightarrow w$) because, once i is reduced, $i \rightarrow w$ cannot be restored anymore. We don't have to verify if i's head is determined because by the precondition of RE (see Sec. 2) it must be. We don't have to check if all i's left dependents are already identified either because, even if there's some $w \in \sigma$ such that $w \leftarrow i$ is in the gold tree but not in the predicted set of arcs, it cannot be restored anymore anyway (both w and i are in the stack in this case), regardless of choosing RE or not.

- Eq. 5: Shifting j to the stack is not optimal if there's some word in the stack which is connected with j by an arc ($\exists_{w \in \sigma|i}\ w \leftrightarrow j$). On the one hand, it can be shown that such an arc cannot be in the set of predicted arcs yet. On the other hand, shifting j to the stack makes restoring any such arc impossible. It's not a problem if there are words in the buffer connected with j in gold tree, though, as it will still be possible to restore them after performing the shift.

Importantly, all these equations rely on the property of the arc-eager system that none of the transitions makes it impossible to restore an arc $v \leftrightarrow w$ such that $v \in \sigma$ and $w \in \beta$. Put differently, if $v \leftrightarrow w$ is possible to reach from c, it is possible to reach it from $t(c)$ regardless of the selected transition t.

Let's get back to the examples in Tab. 3 and Tab. 4. Let c = ([ROOT sent her], [a letter .], {He $\leftarrow$ sent, ROOT $\rightarrow$ sent) and T be the tree from Fig. 1:

- There's no arc from *her* to any of the words in *a letter .* in Fig. 1, nor is there an arc from any of the words in *letter .* to *her*. Hence, LA is optimal.

- There's an arc from *letter* to *a*, thus $\exists_{w \in \beta}\ j \leftarrow w$. Hence, RA is not optimal.

- RE is not possible because *her* does not have a head specified.

- There's no arc between any of the words *ROOT sent her* on the one hand and *a* on the other hand. Hence, SH is optimal.

As a result, we know that LA and SH are the only two optimal transitions for configuration c.

2.3 Arc-decomposability

The arc-eager system is so-called *arc-decomposable*, which means that the following two properties are equivalent for a given configuration c and projective tree T:

- There exists a transition sequence $t_1, \ldots, t_k$ such that $t_k(\ldots(t_1(c)))$ reconstructs[6] all the arcs in T.

- For every arc a in T, there exists $t_1, \ldots, t_k$ such that $t_k(\ldots(t_1(c)))$ contains a.

[4] And $i \rightarrow w$ cannot be in the set of predicted arcs yet either, since the words in the buffer do not have their heads determined. Once a head of a word is determined, the word is moved to the stack or removed, and it cannot ever get back to the buffer again.

[5] We don't have to check if $\exists_{w \in \sigma}\ w \rightarrow i$, because both i and w are in the stack in this case and, therefore, it's not possible to restore $w \rightarrow i$ anyway, regardless of whether $i \leftarrow j$ is created or not.

[6] Recall that $t_k(\ldots(t_1(c)))$ is a configuration itself, and its third component is a set of arcs.

This simplifies the task of determining the optimal transition(s) for a given configuration c. Namely, if we check that every arc reachable from c is still reachable from $t(c)$, then we know that t is optimal.[7]

[7] Even if the gold tree T is not reachable from c, there's some projective tree T' among the set of all reachable trees that is most similar to T. If we make sure that all the arcs reachable from c are reachable from $t(c)$, we also make sure that all the arcs in T' are reachable from $t(c)$, which means (thanks to arc-decomposability) that T' itself is reachable from $t(c)$, hence t is optimal. Note also that there is always at least one optimal transition.