Dependency Parsing

lecture 6

Jakub Waszczuk, Kilian Evang

Heinrich Heine Universität

Summer semester 2021

Today

- The arc-eager transition system
- Non-projective parsing

(slides from ESSLII 2007 course and EACL 2014 tutorial by McDonald & Nivre)

Shift-Reduce Dependency Parsing

- Transitions:
  - Left-Arc$_k$:
    $$(\sigma|i,j|\beta,A) \Rightarrow (\sigma,j|\beta,A\cup\{(j,i,k)\})$$
  - Right-Arc$_k$:
    $$(\sigma|i,j|\beta,A) \Rightarrow (\sigma,i|\beta,A\cup\{(i,j,k)\})$$
  - Shift:
    $$(\sigma,i|\beta,A) \Rightarrow (\sigma|i,\beta,A)$$
- Preconditions:
  - Left-Arc$_k$:
    $$\neg[i=0]$$
    $$\neg\exists i'\exists k'[(i',i,k')\in A]$$
  - Right-Arc$_k$:
    $$\neg\exists i'\exists k'[(i',j,k')\in A]$$

Arc-Eager Parsing

- Transitions:
  - Left-Arc$_k$:
    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})$$
  - Right-Arc$_k$:
    $$(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, j, k)\})$$
  - Reduce:
    $$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$$
  - Shift:
    $$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$$
- Preconditions:
  - Left-Arc$_k$:
    $$\neg[i = 0]$$
    $$\neg\exists i'\exists k'[(i', i, k') \in A]$$
  - Right-Arc$_k$:
    $$\neg\exists i'\exists k'[(i', j, k') \in A]$$
  - Reduce:
    $$\exists i'\exists k'[(i', i, k') \in A]$$

Example: Arc-Eager Parsing

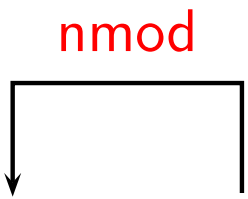
$[\text{root}_0]_\sigma$ $[\text{Economic}_1$ news_2 had_3 little_4 effect_5 on_6 financial_7 markets_8 $.9]_\beta$

Example: Arc-Eager Parsing

$[\text{root}_0\ \text{Economic}_1]_\sigma\ [\text{news}_2\ \text{had}_3\ \text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ .9]_\beta$

Shift

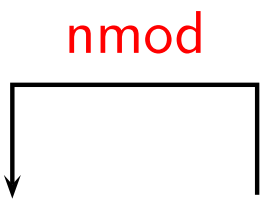
Example: Arc-Eager Parsing



$[\text{root}_0]_\sigma$ Economic_1 $[\text{news}_2$ had_3 little_4 effect_5 on_6 financial_7 markets_8 $.9]_\beta$

Left-Arc$_{nmod}$

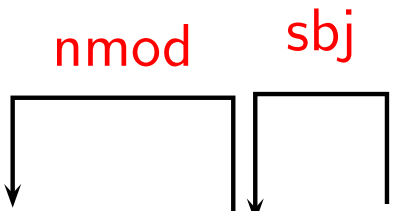
Example: Arc-Eager Parsing



$[\text{root}_0\ \text{Economic}_1\ \text{news}_2]_\sigma\ [\text{had}_3\ \text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ .9]_\beta$

Shift

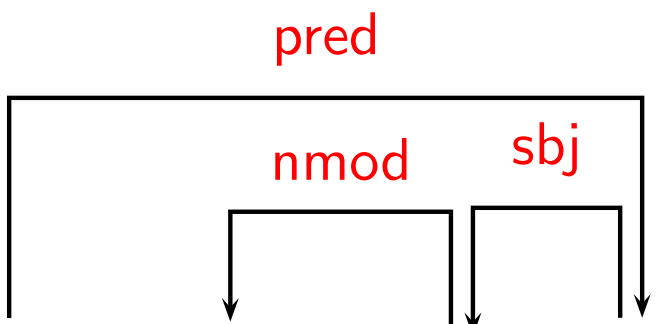
Example: Arc-Eager Parsing



$[\text{root}_0]_\sigma$ Economic_1 news_2 $[\text{had}_3$ little_4 effect_5 on_6 financial_7 markets_8 $.9]_\beta$

Left-Arc$_{sbj}$

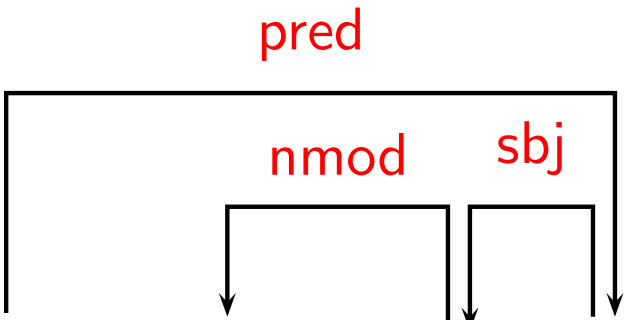
Example: Arc-Eager Parsing



$[\text{root}_0\ \text{Economic}_1\ \text{news}_2\ \text{had}_3]_\sigma\ [\text{little}_4\ \text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ .9]_\beta$

Right-Arc$_{pred}$

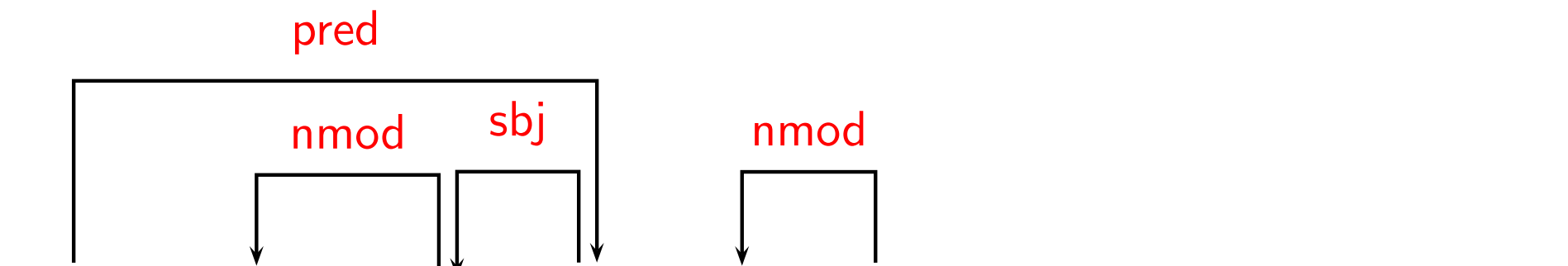
Example: Arc-Eager Parsing



$[\text{root}_0\ \text{Economic}_1\ \text{news}_2\ \text{had}_3\ \text{little}_4]_\sigma\ [\text{effect}_5\ \text{on}_6\ \text{financial}_7\ \text{markets}_8\ .9]_\beta$

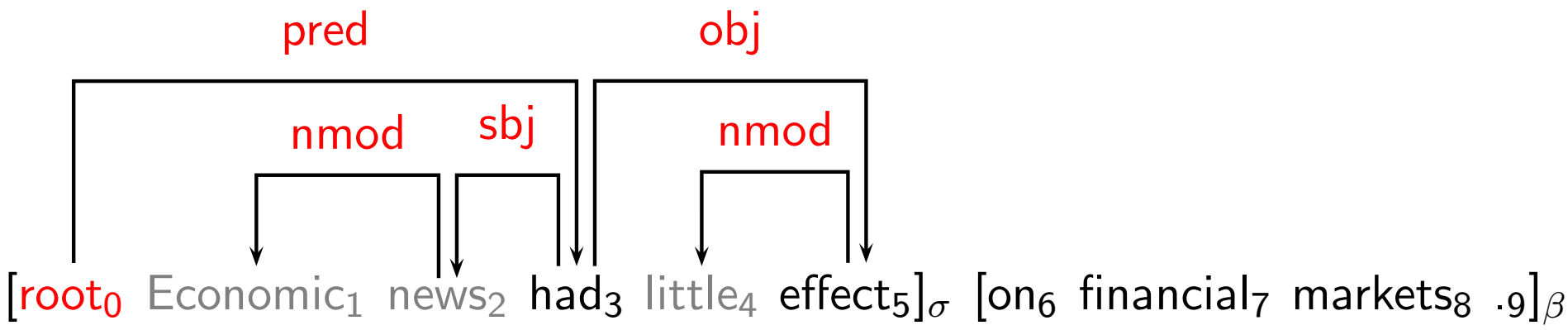
Shift

Example: Arc-Eager Parsing



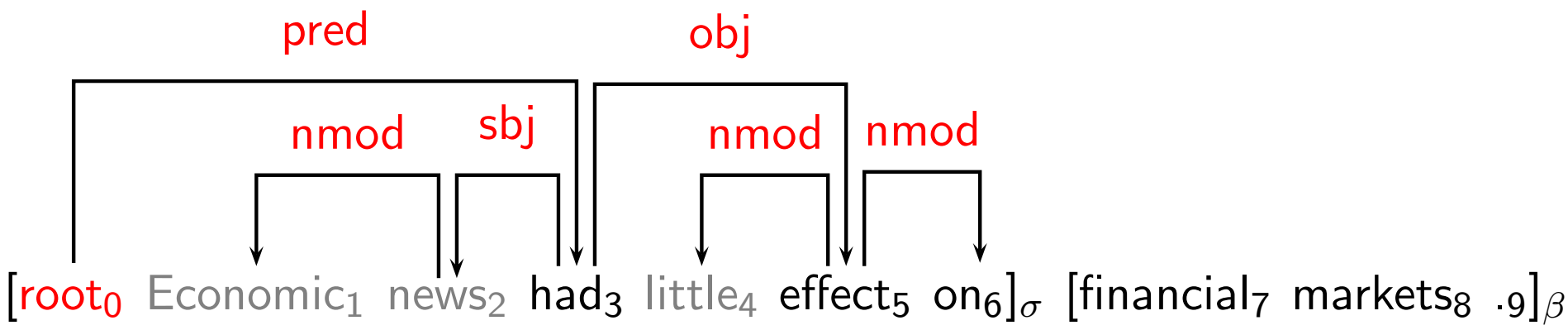
Left-Arc$_{nmod}$

Example: Arc-Eager Parsing



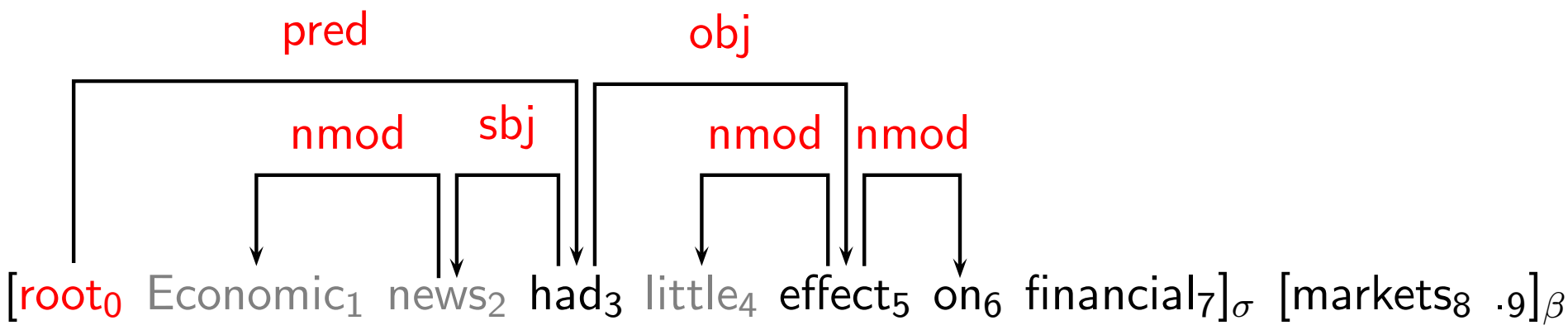
Right-Arc$_{obj}$

Example: Arc-Eager Parsing



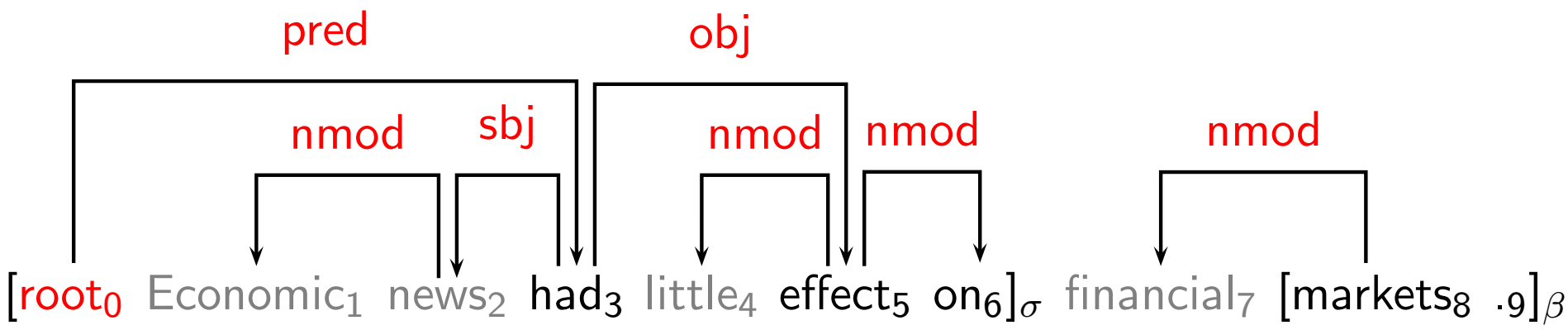
Right-Arc$_{nmod}$

Example: Arc-Eager Parsing



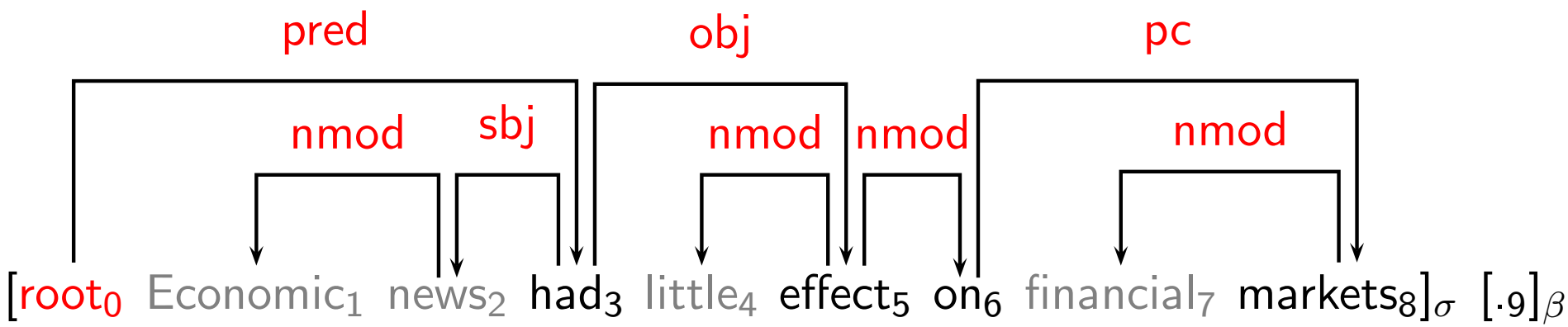
Shift

Example: Arc-Eager Parsing



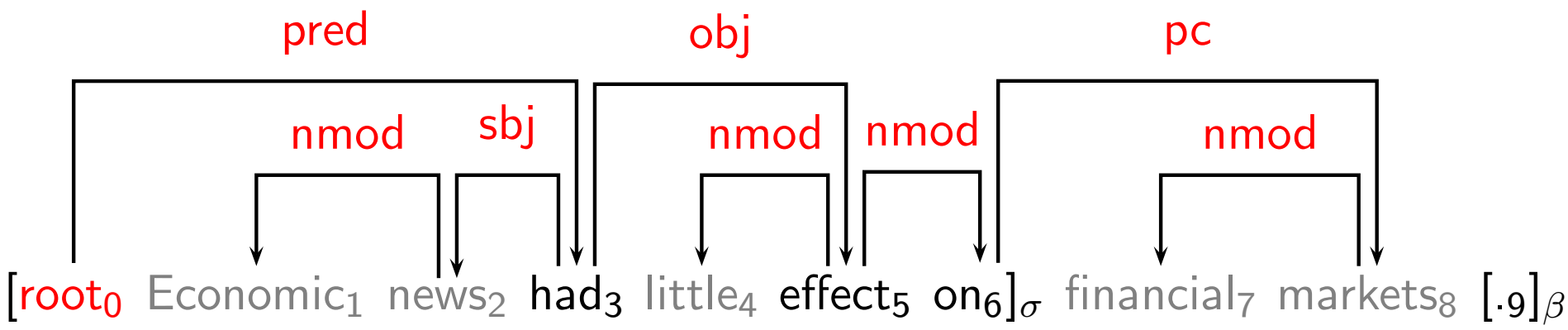
Left-Arc$_{nmod}$

Example: Arc-Eager Parsing



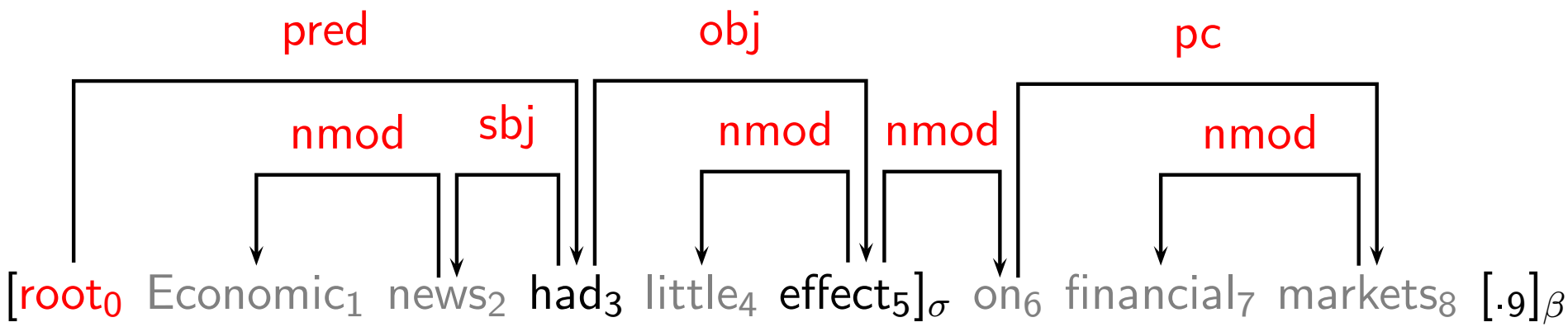
Right-Arc$_{pc}$

Example: Arc-Eager Parsing



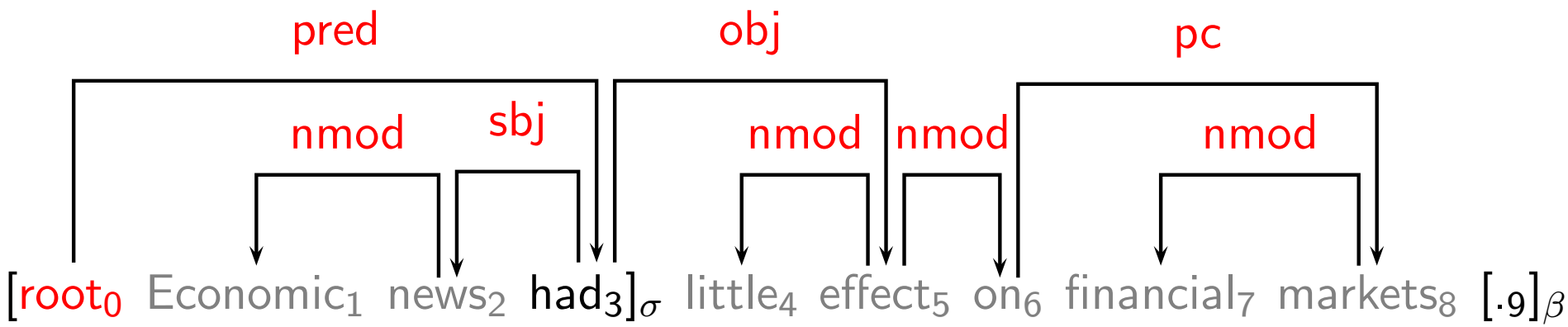
Reduce

Example: Arc-Eager Parsing



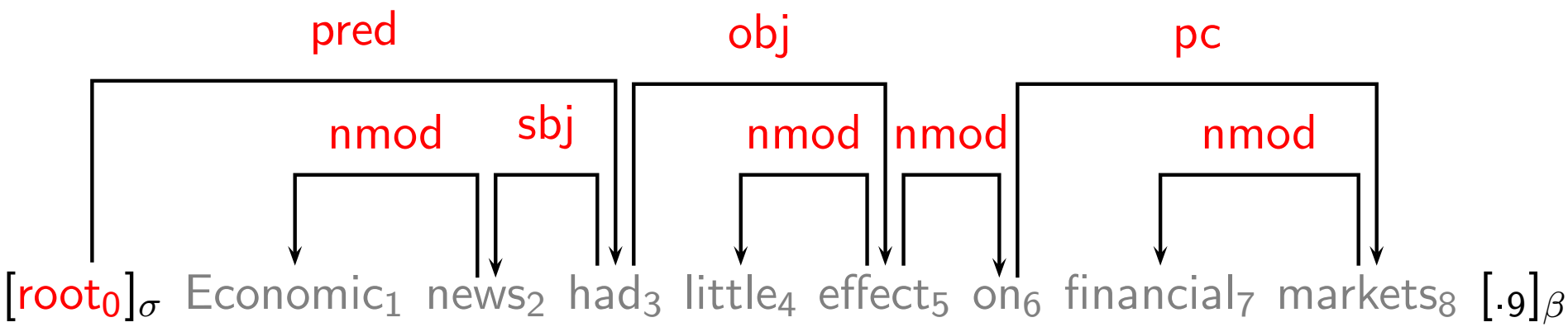
Reduce

Example: Arc-Eager Parsing



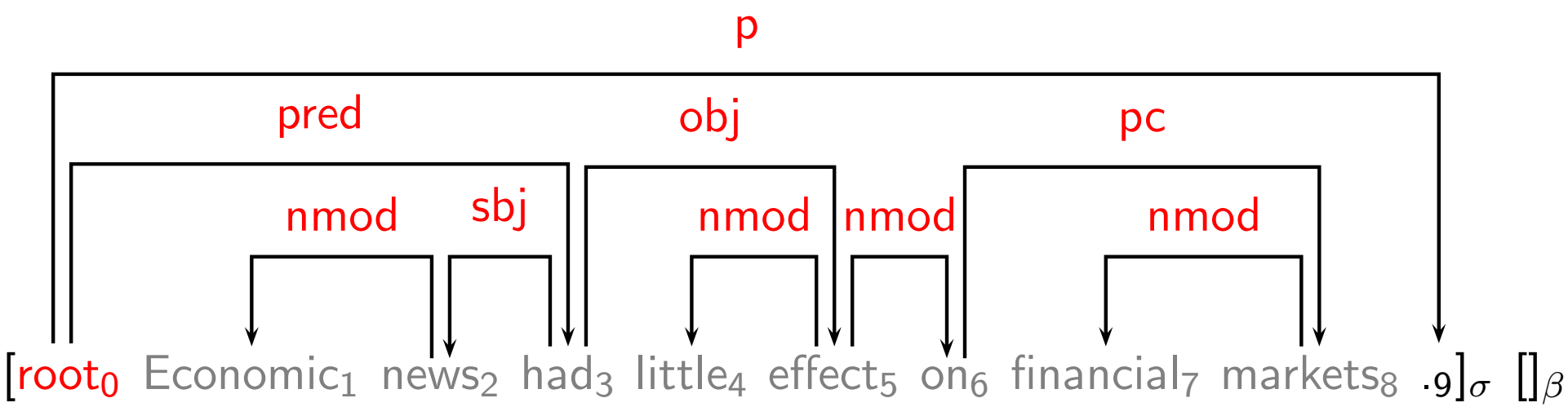
Reduce

Example: Arc-Eager Parsing



Reduce

Example: Arc-Eager Parsing



Right-Arc$_p$

To sum up:

- Add right arc immediately, add further dependents later
- Do not remove the dependent of the right arc immediately
- Remove it (*reduce*) once all its right dependents have been determined

To sum up:

- Add right arc immediately, add further dependents later
- Do not remove the dependent of the right arc immediately
- Remove it (*reduce*) once all its right dependents have been determined

Which works best: arc-standard or arc-eager?

Non-Projective Parsing

- So far only projective parsing models
- Non-projective parsing harder even with greedy inference
  - Non-projective: $n(n-1)$ arcs to consider – $O(n^2)$
  - Projective: at most $2(n-1)$ arcs to consider – $O(n)$
- Also harder to construct dynamic oracles
  - Conjecture: arc-decomposability presupposes projectivity

Previous Approaches

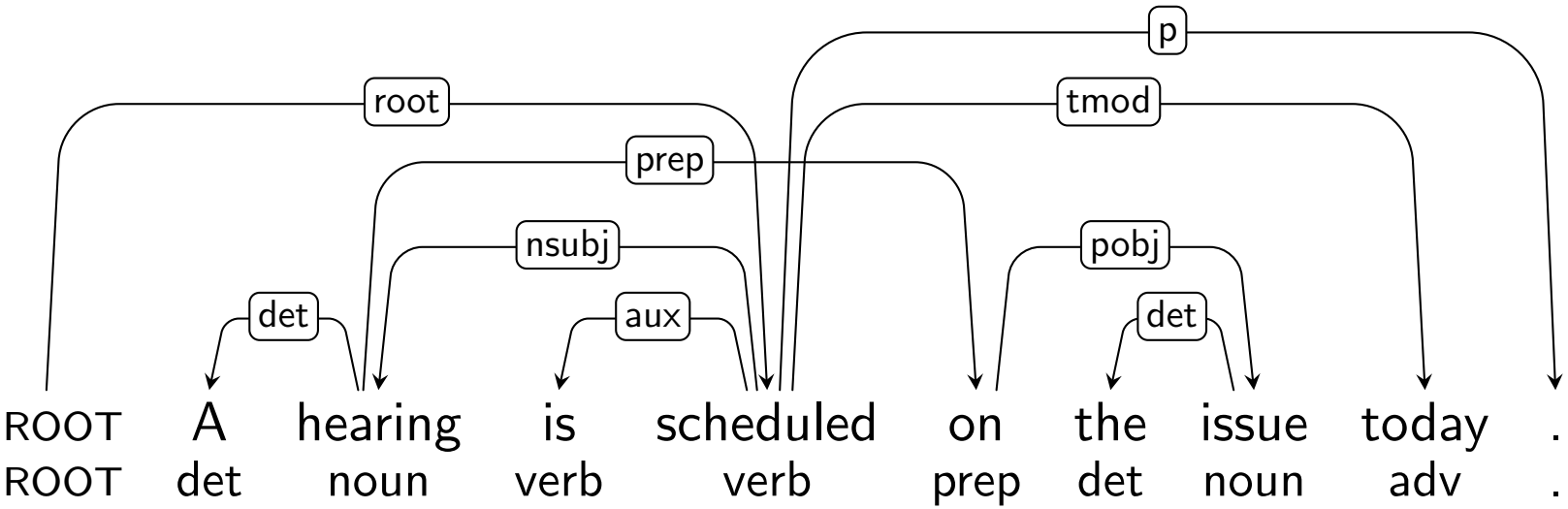
- Pseudo-projective parsing [Nivre and Nilsson 2005]
  - Preprocess training data, post-process parser output
  - Approximate encoding with incomplete coverage
  - Relatively high precision but low recall
- Extended arc transitions [Attardi 2006]
  - Transitions that add arcs between non-adjacent subtrees
  - Upper bound on arc degree (limited to local relations)
  - Exact dynamic programming algorithm [Cohen et al. 2011]
- List-based algorithms [Covington 2001, Nivre 2007]
  - Consider all word pairs instead of adjacent subtrees
  - Increases parsing complexity (and training time)
  - Improved accuracy and efficiency by adding "projective transitions" [Choi and Palmer 2011]

Novel Approaches

- Online reordering [Nivre 2009, Nivre et al. 2009]:
  - Reorder words during parsing to make tree projective
  - Add a special transition for swapping adjacent words
  - Quadratic time in the worst case but linear in the best case
- Multiplanar parsing [Gómez-Rodríguez and Nivre 2010]:
  - Factor dependency trees into k planes without crossing arcs
  - Use k stacks to parse each plane separately
  - Linear time parsing with constant k

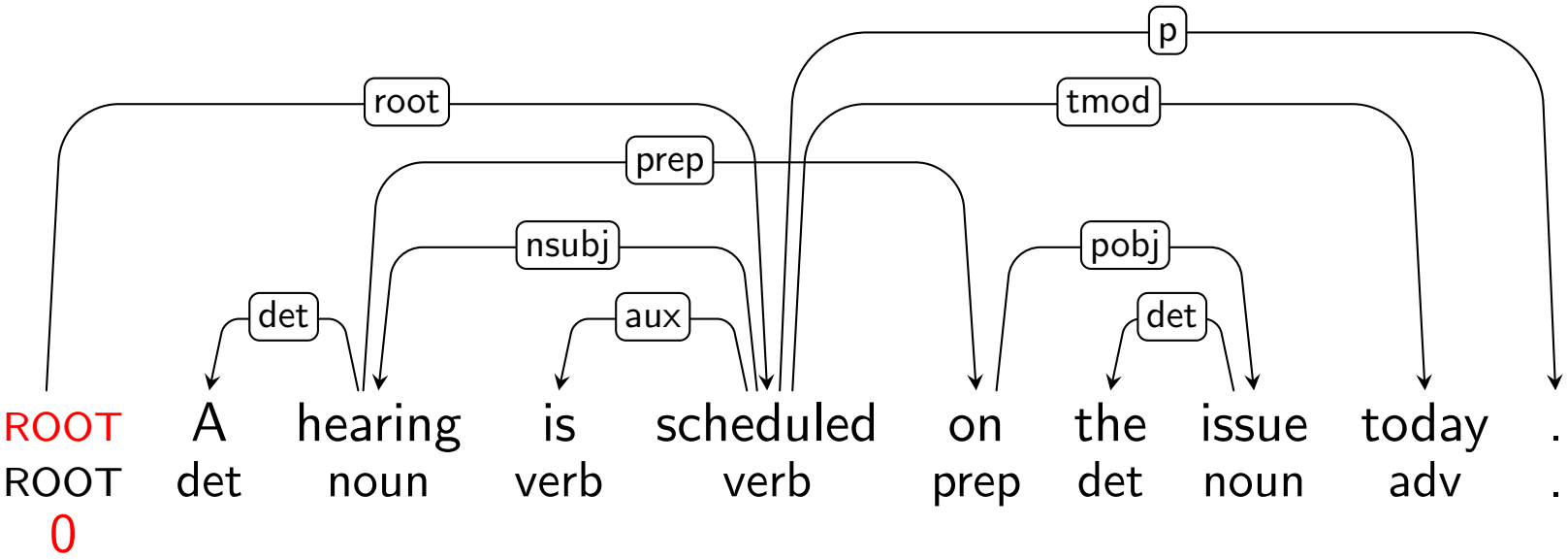
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



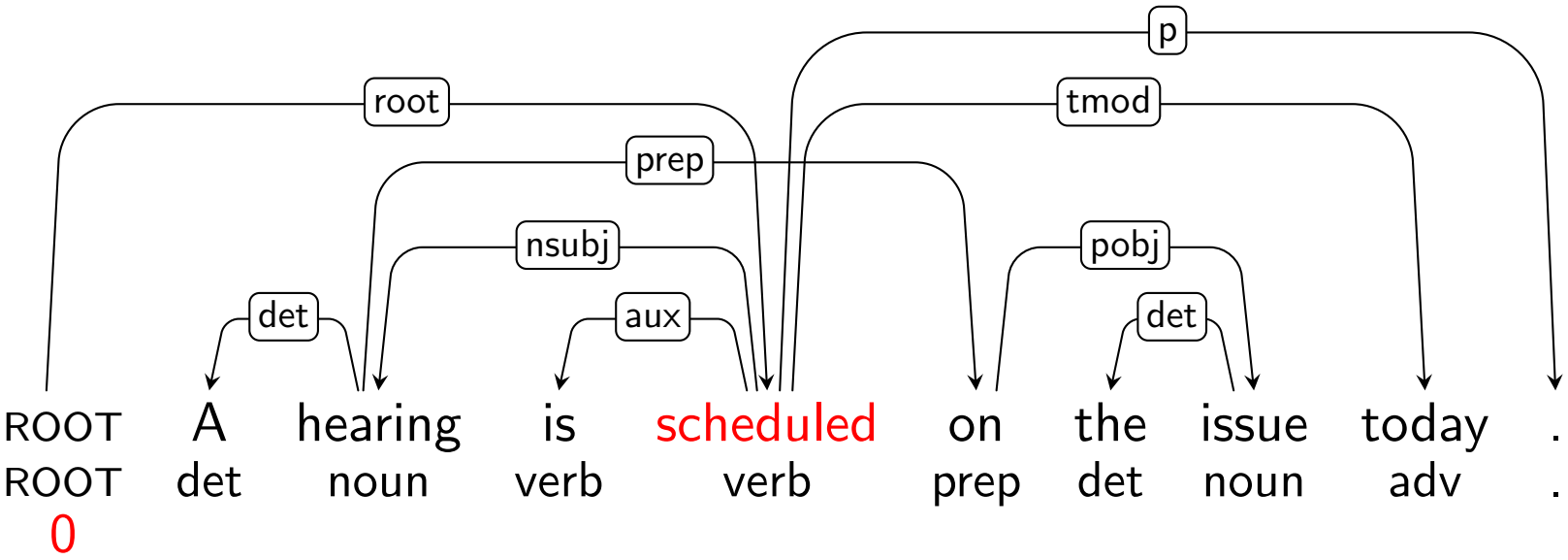
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



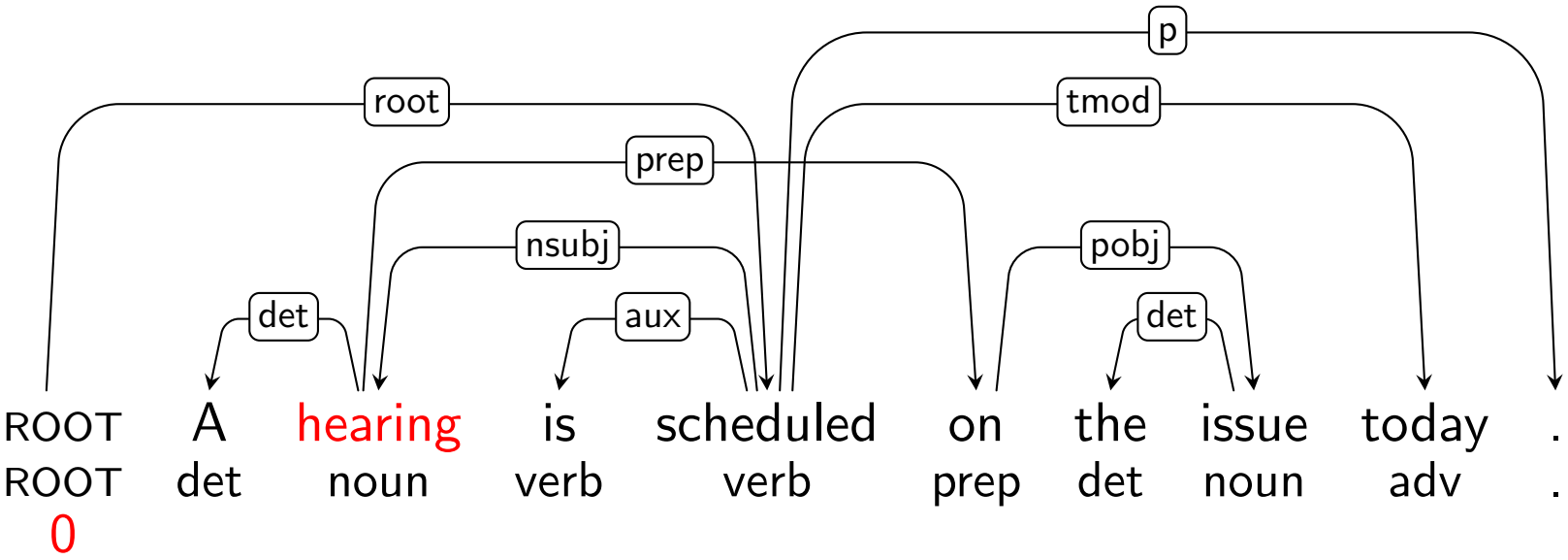
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



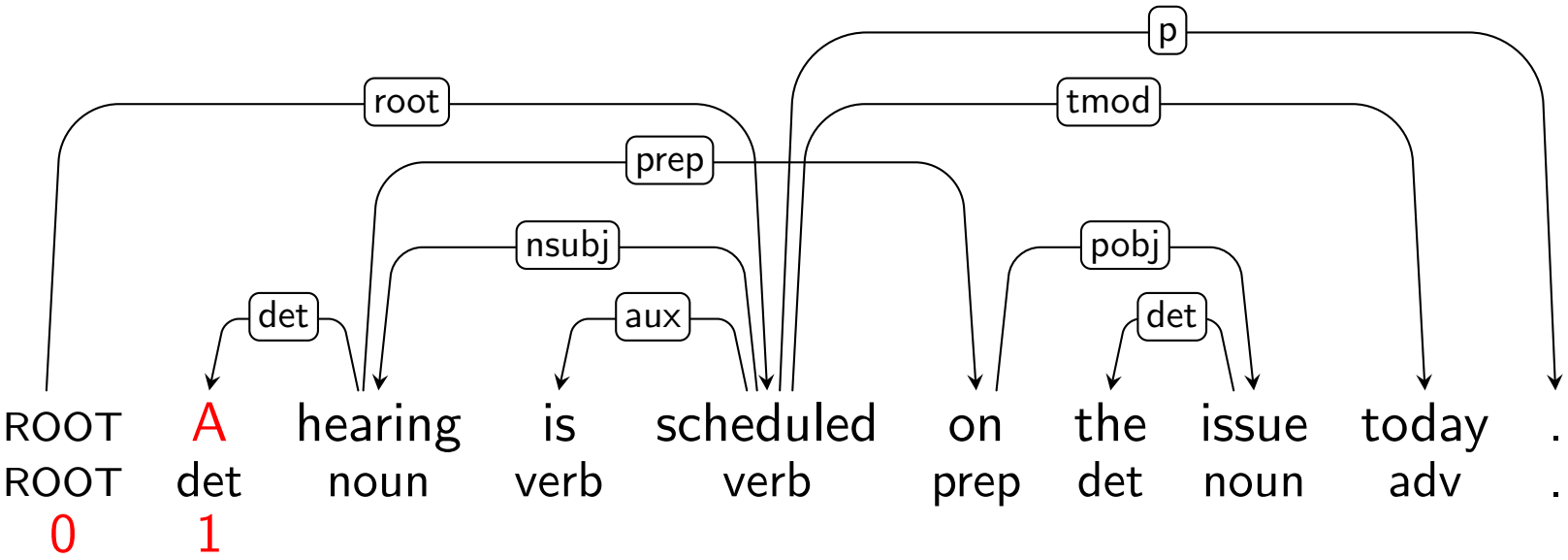
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



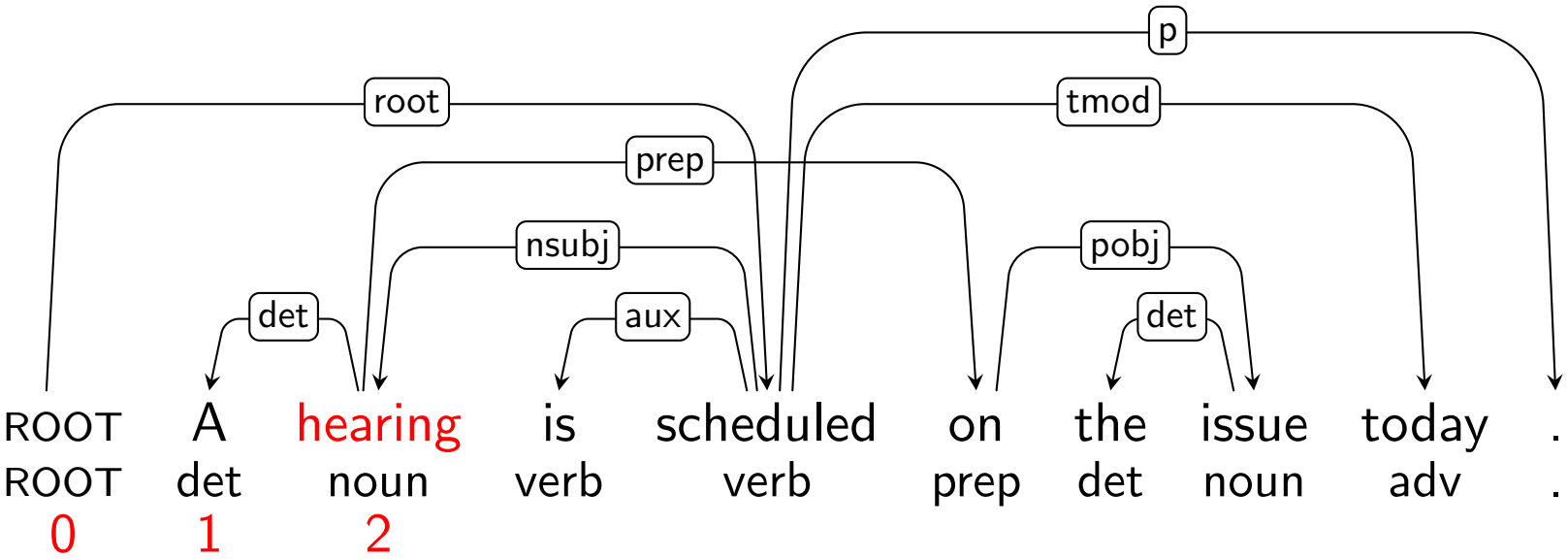
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



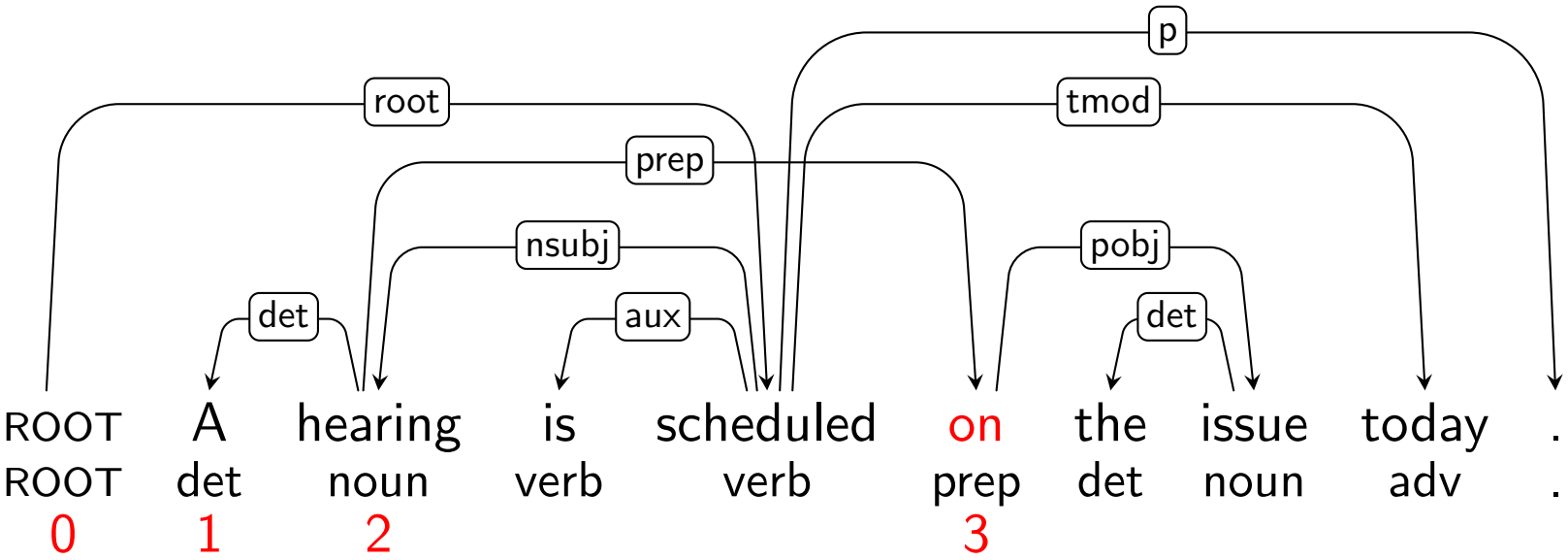
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



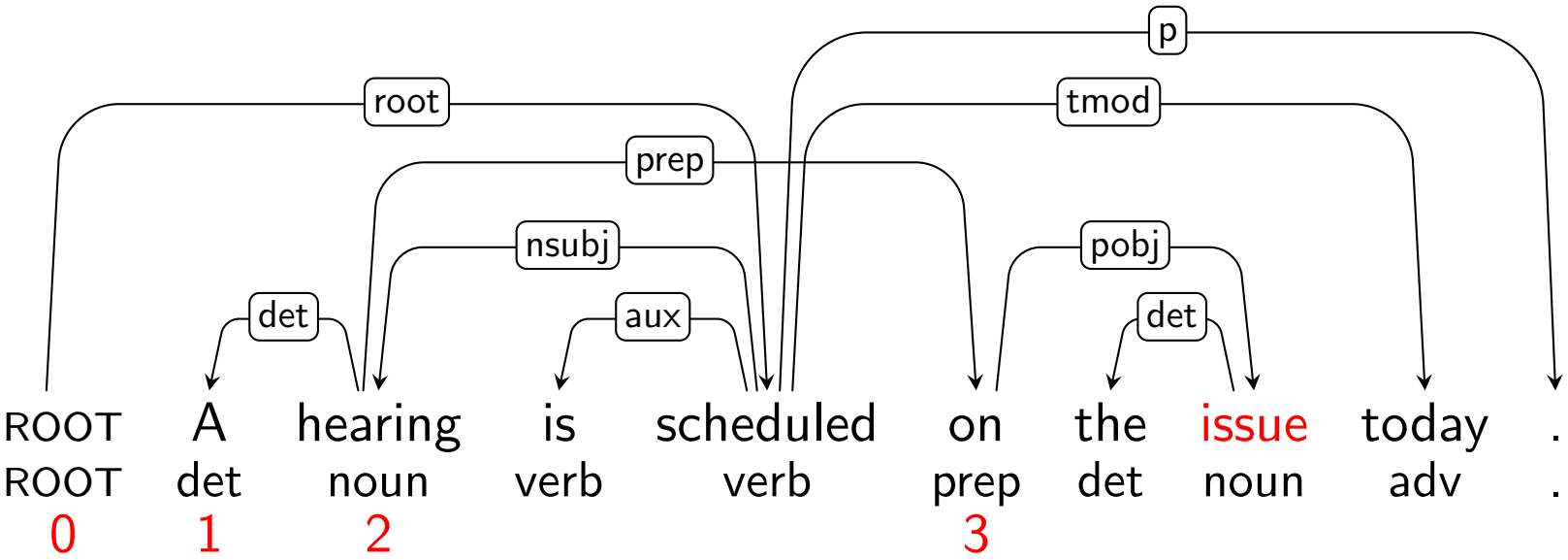
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



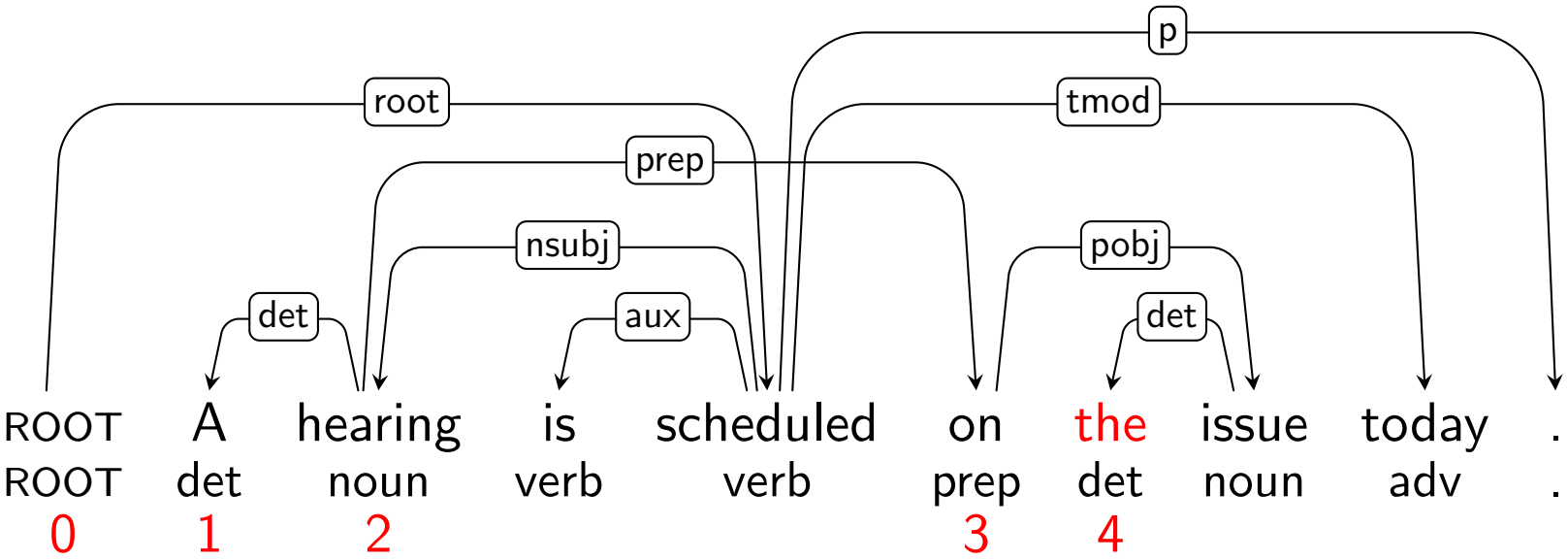
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



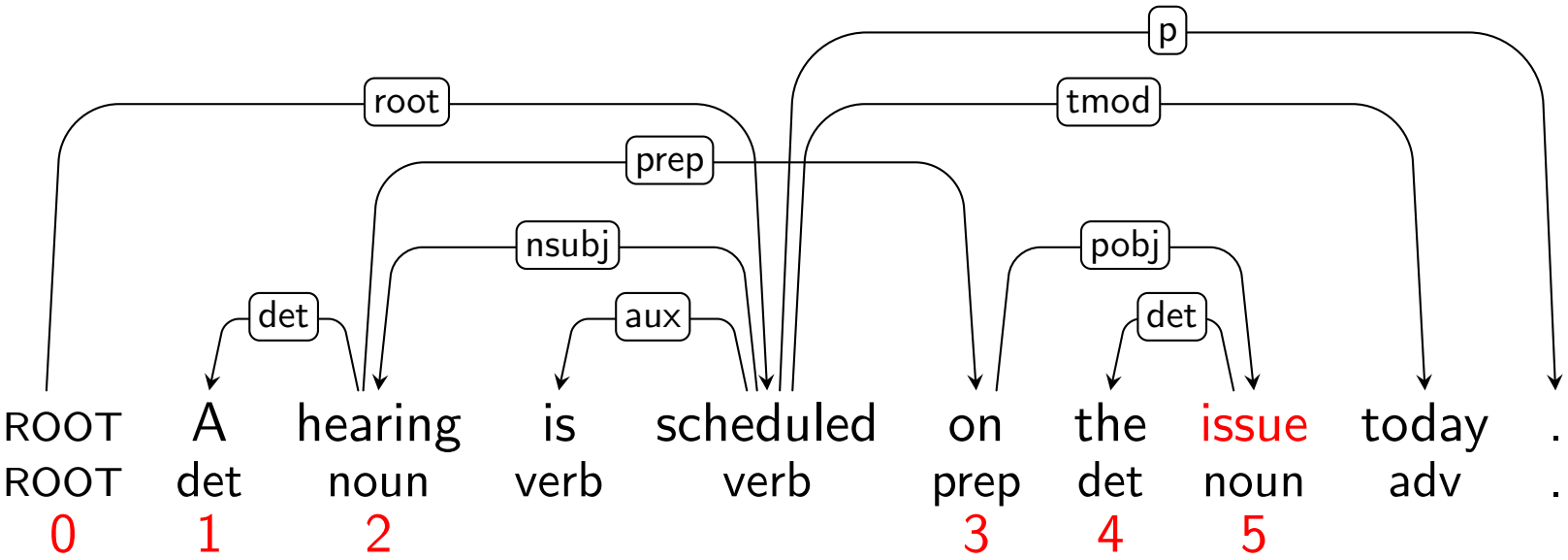
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



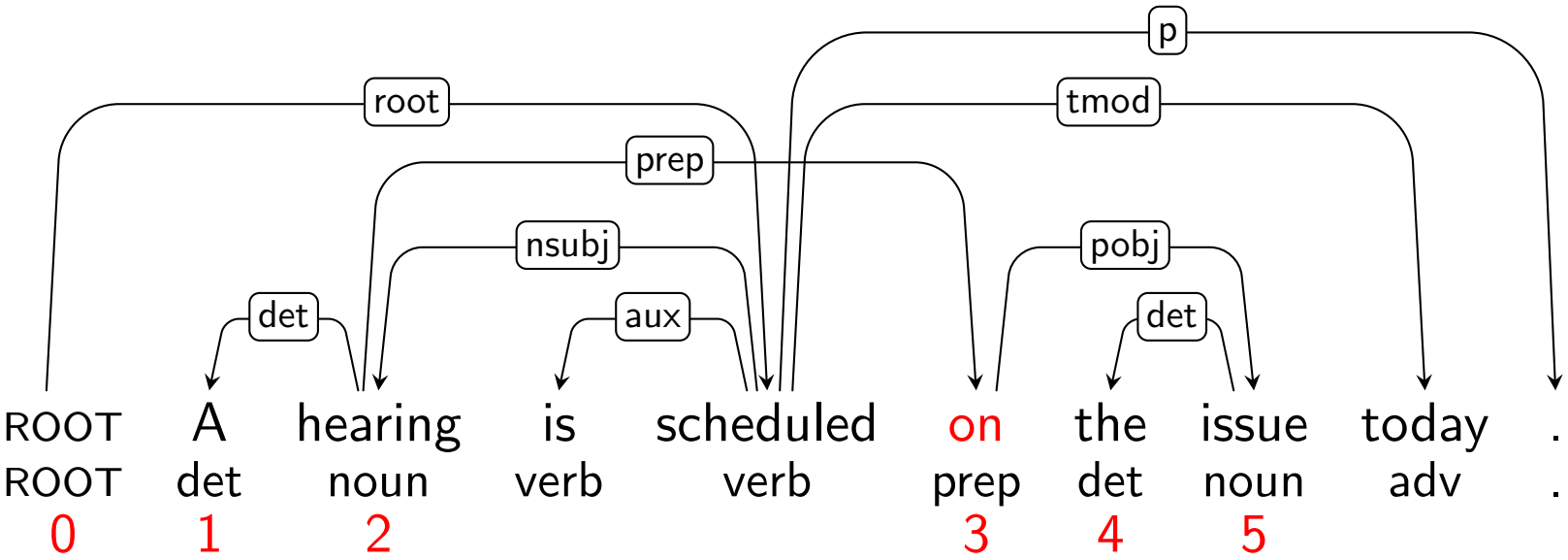
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



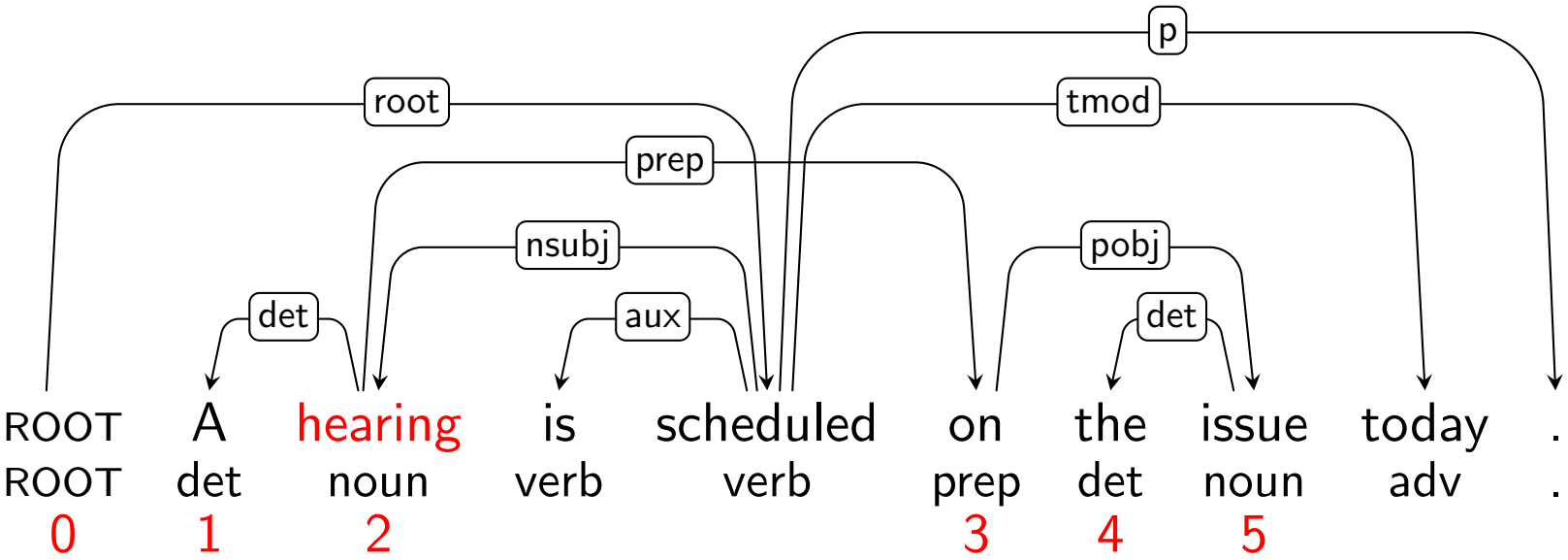
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



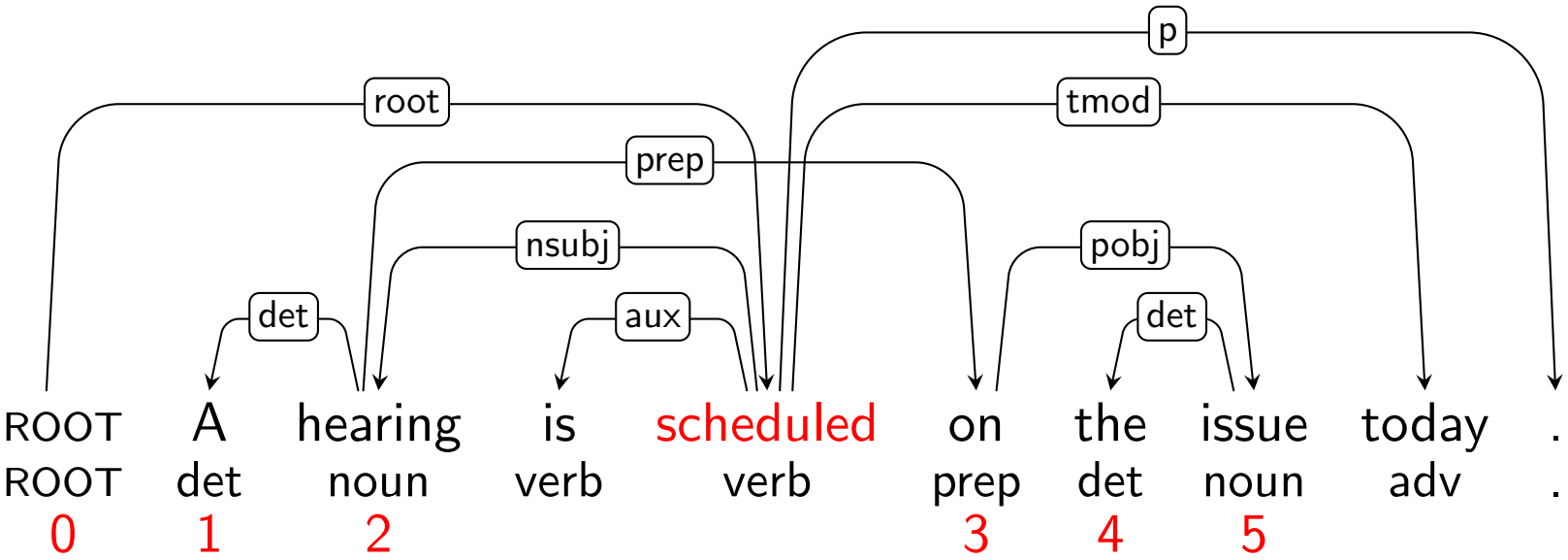
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



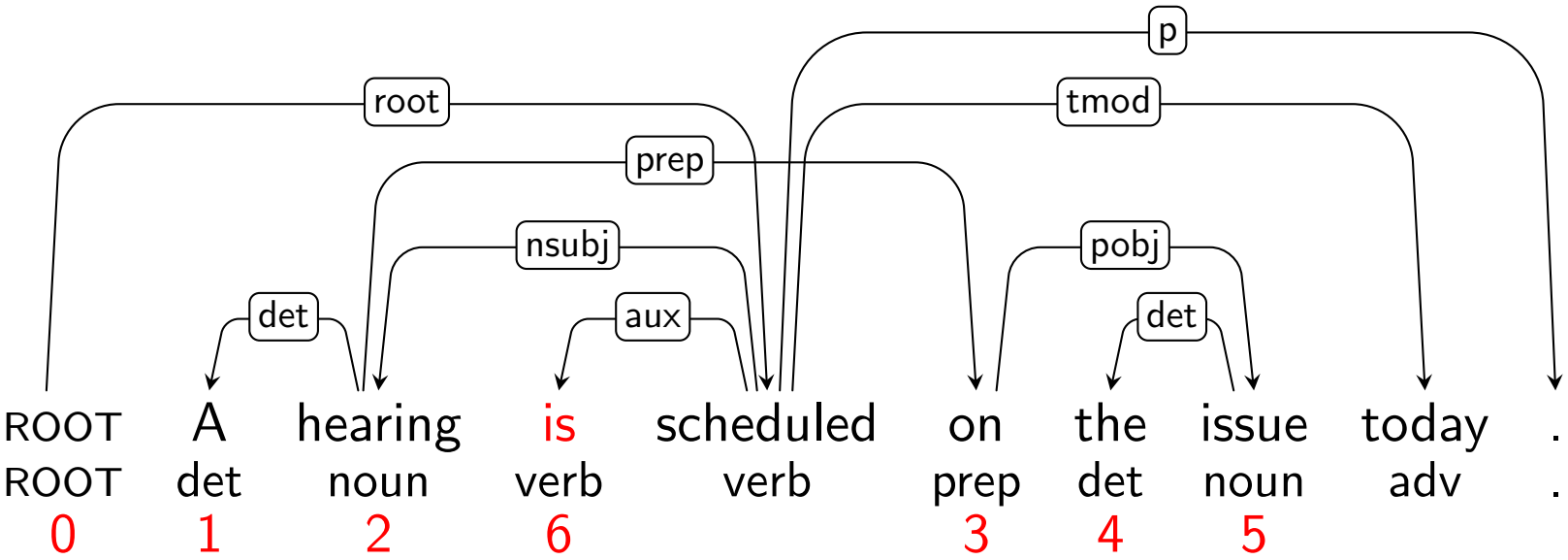
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



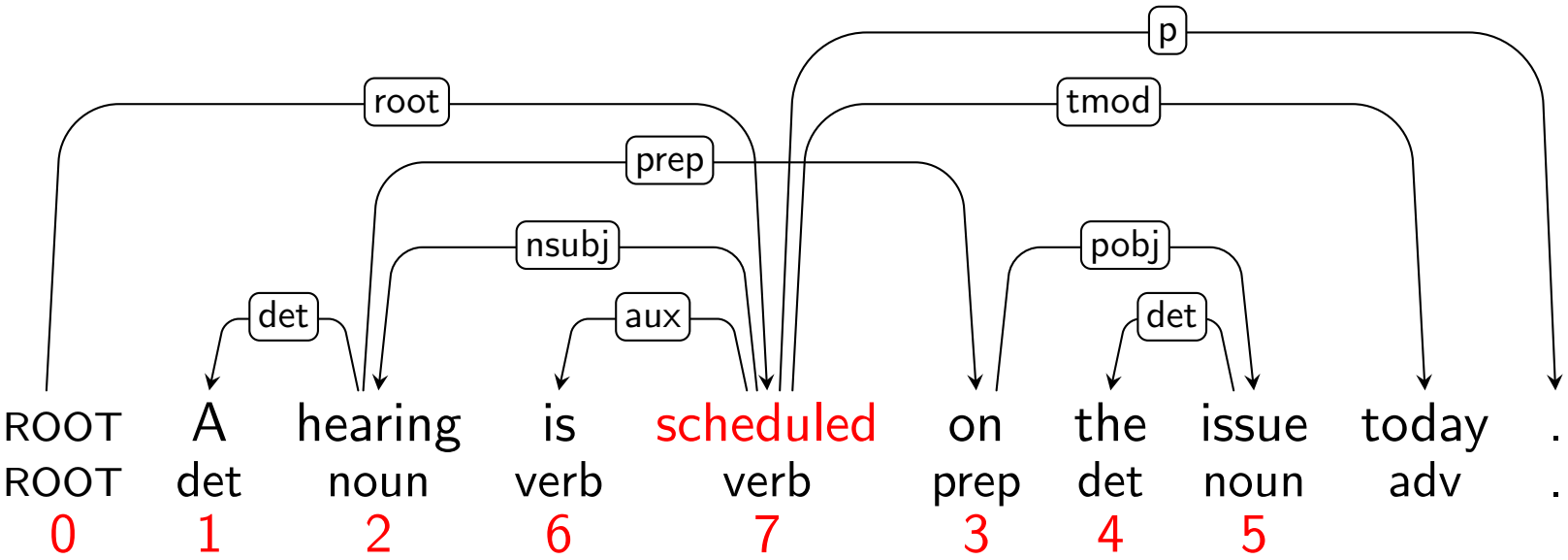
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



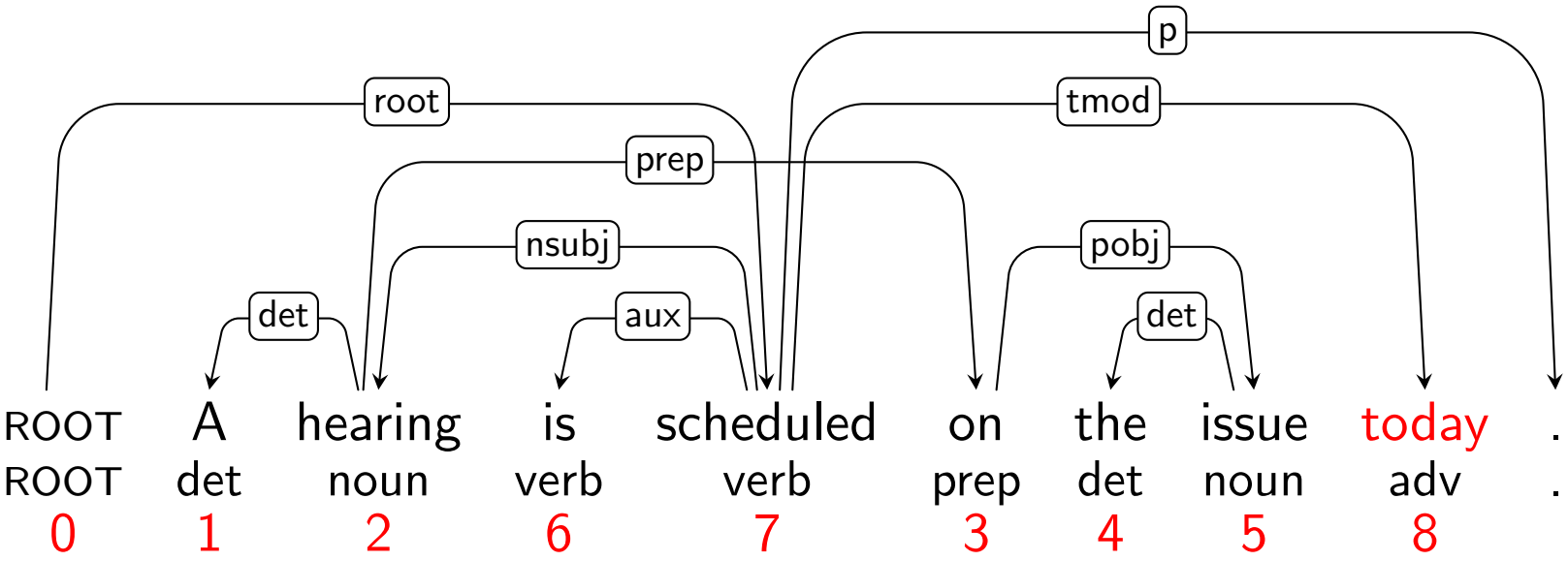
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



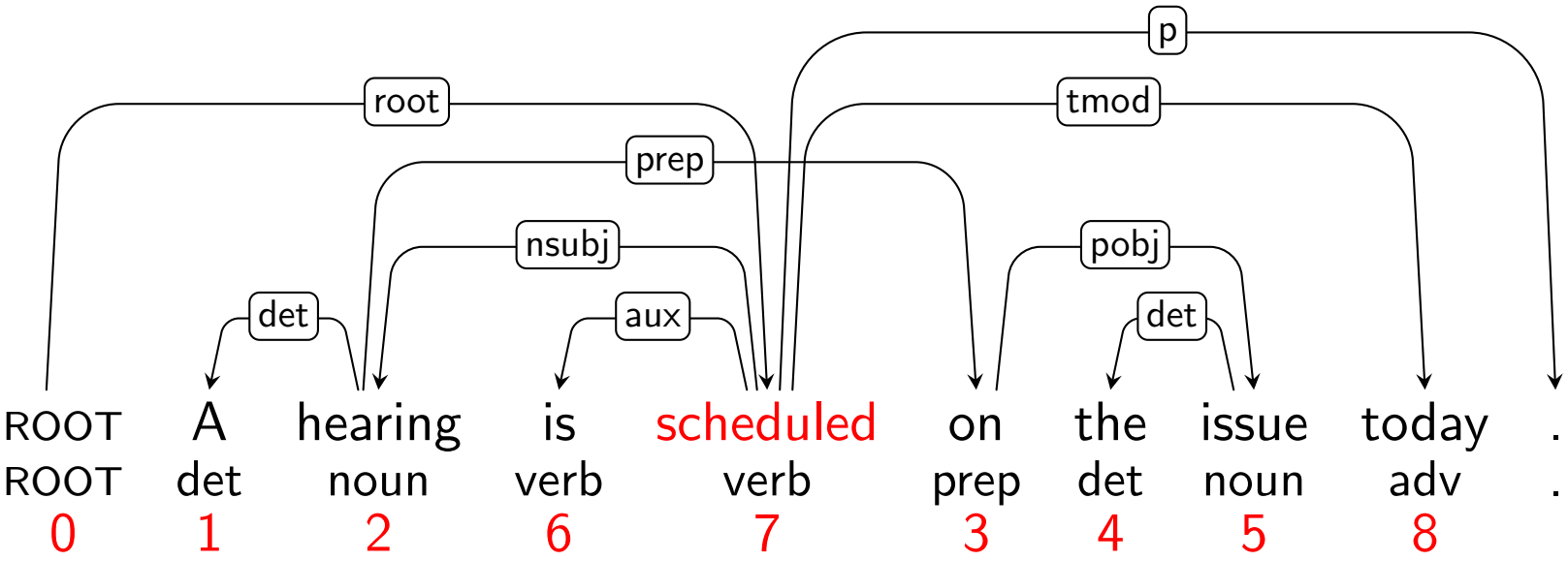
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



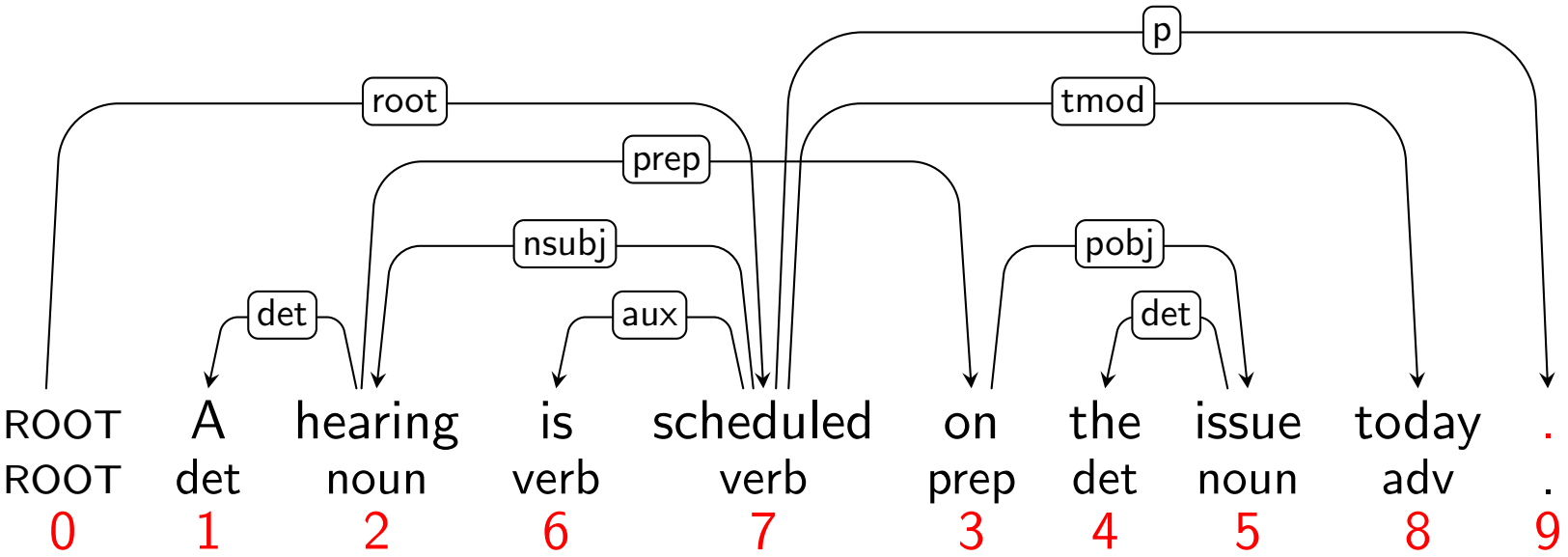
Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



Projectivity and Word Order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an inorder traversal of T with respect to $<$ [Veselá et al. 2004]



Transition System for Online Reordering

Configuration: (S, B, A) $[S = \text{Stack},\ B = \text{Buffer},\ A = \text{Arcs}]$

Initial: $([\,], [0, 1, \ldots, n], \{\,\})$

Terminal: $([0], [\,], A)$

Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$	
Right-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|i, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, B, A \cup \{(j, i, k)\})$	$i \neq 0$
Swap:	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, i\|B, A)$	$0 < i < j$

Transition System for Online Reordering

Configuration: (S, B, A) $[S = \text{Stack}, B = \text{Buffer}, A = \text{Arcs}]$

Initial: $([\,], [0, 1, \ldots, n], \{\,\})$

Terminal: $([0], [\,], A)$

Shift:	$(S, i\|B, A)$	$\Rightarrow$	$(S\|i, B, A)$	
Right-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|i, B, A \cup \{(i, j, k)\})$	
Left-Arc(k):	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, B, A \cup \{(j, i, k)\})$	$i \neq 0$
Swap:	$(S\|i\|j, B, A)$	$\Rightarrow$	$(S\|j, i\|B, A)$	$0 < i < j$

- Transition-based parsing with two interleaved processes:
  1. Sort words into projective order $<_p$
  2. Build tree T by connecting adjacent subtrees
- T is projective with respect to $<_p$ but not (necessarily) $<$

Example Transition Sequence

$[\ ]_S$ $[\text{ROOT}, \text{A}, \text{hearing}, \text{is}, \text{scheduled}, \text{on}, \text{the}, \text{issue}, \text{today}, .]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

$[\text{ROOT}]_S$ $[\text{A, hearing, is, scheduled, on, the, issue, today, .}]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

Example Transition Sequence

$[\text{ROOT}, \text{A}]_S$ $[\text{hearing}, \text{is}, \text{scheduled}, \text{on}, \text{the}, \text{issue}, \text{today}, .]_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

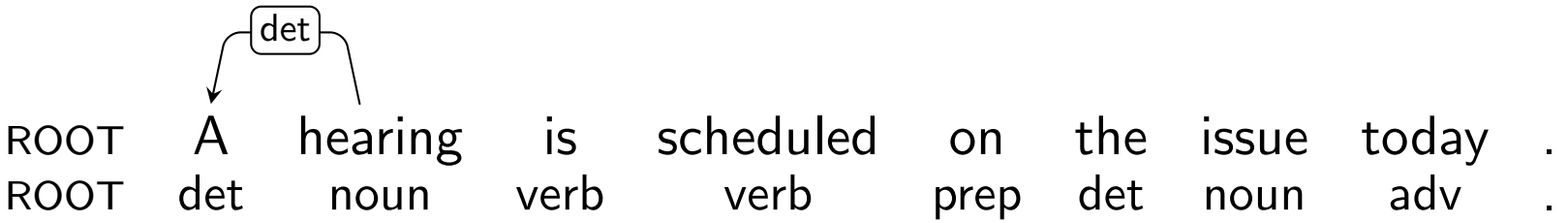
Example Transition Sequence

[ROOT, A, hearing]$_S$ [is, scheduled, on, the, issue, today, .]$_B$

ROOT	A	hearing	is	scheduled	on	the	issue	today	.
ROOT	det	noun	verb	verb	prep	det	noun	adv	.

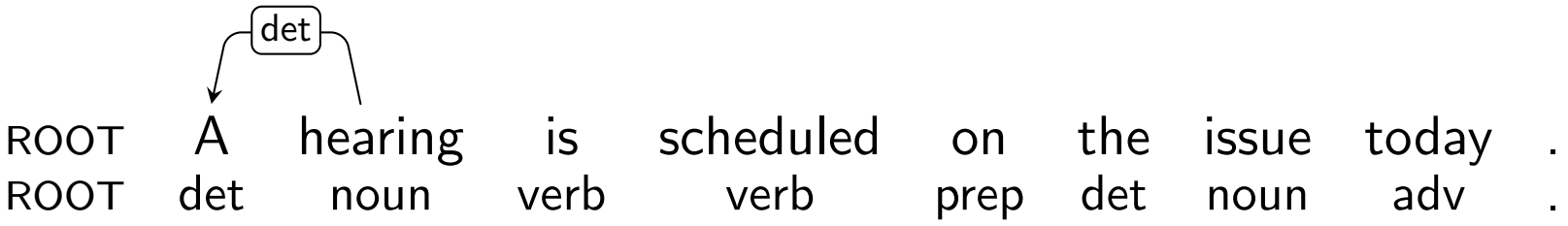
Example Transition Sequence

[ROOT, hearing]$_S$ [is, scheduled, on, the, issue, today, .]$_B$



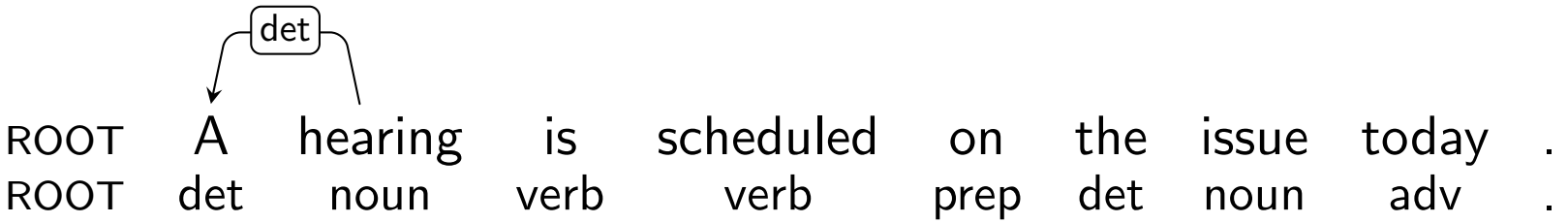
Example Transition Sequence

[ROOT, hearing, is]$_S$ [scheduled, on, the, issue, today, .]$_B$



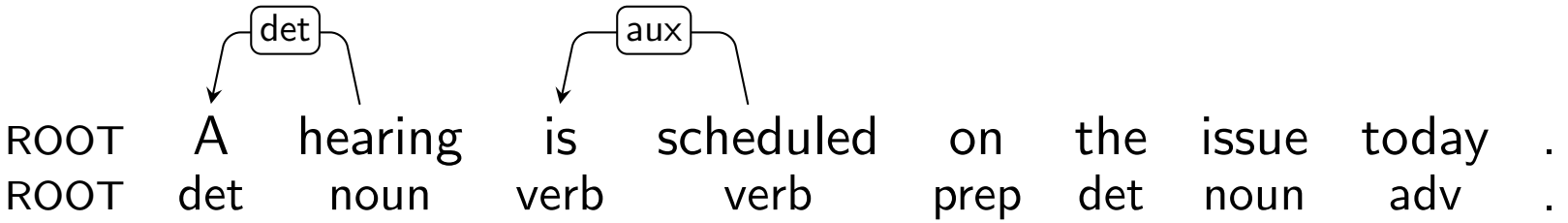
Example Transition Sequence

[ROOT, hearing, is, scheduled]$_S$ [on, the, issue, today, .]$_B$



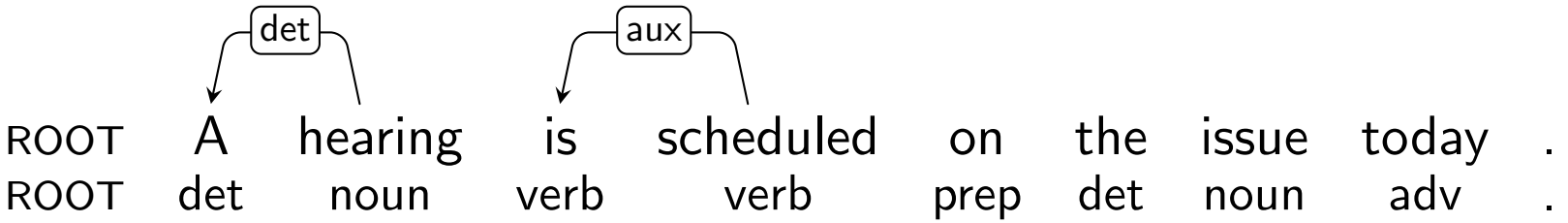
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [on, the, issue, today, .]$_B$



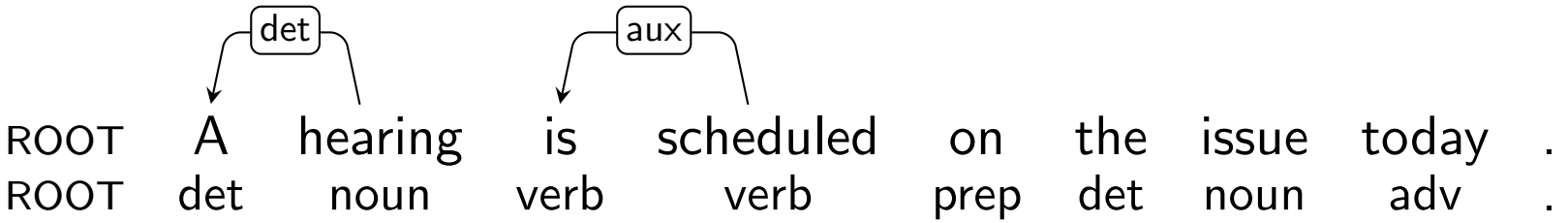
Example Transition Sequence

[ROOT, hearing, scheduled, on]$_S$ [the, issue, today, .]$_B$



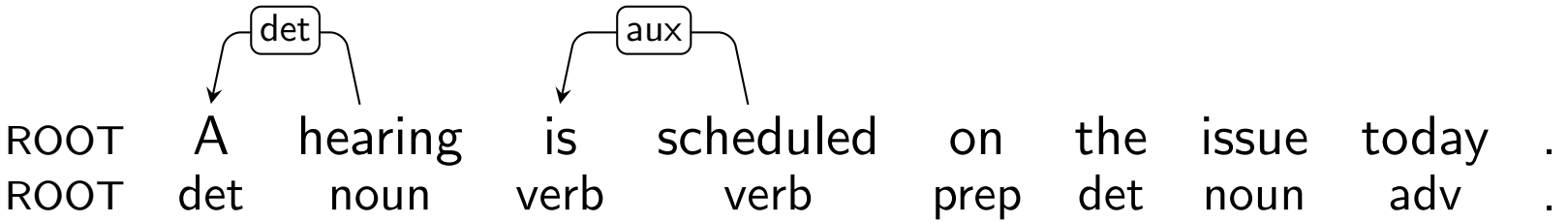
Example Transition Sequence

[ROOT, hearing, scheduled, on, the]$_S$ [issue, today, .]$_B$



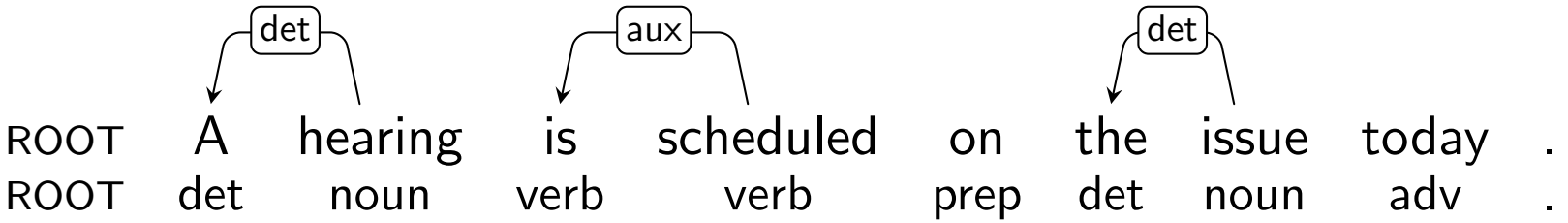
Example Transition Sequence

[ROOT, hearing, scheduled, on, the, issue]$_S$ [today, .]$_B$



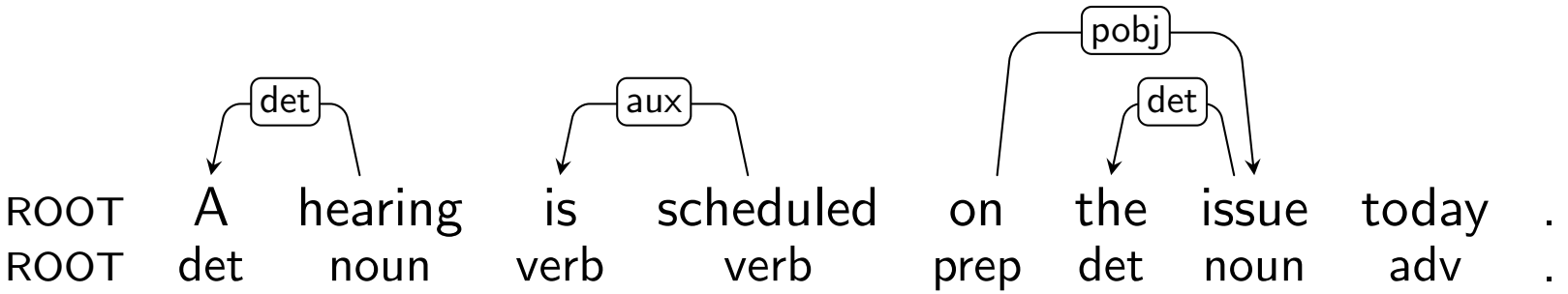
Example Transition Sequence

[ROOT, hearing, scheduled, on, issue]$_S$ [today, .]$_B$



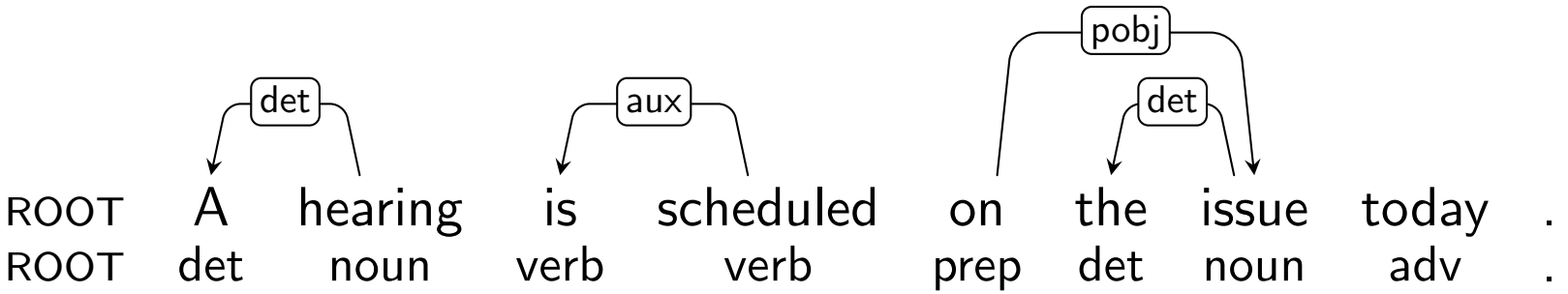
Example Transition Sequence

[ROOT, hearing, scheduled, on]$_S$ [today, .]$_B$



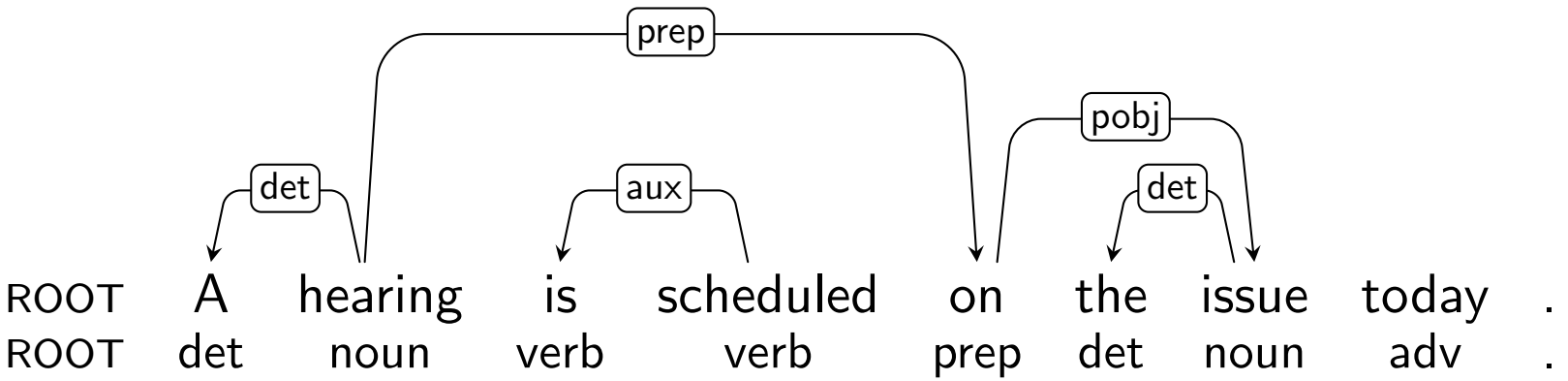
Example Transition Sequence

[ROOT, hearing, on]$_S$ [scheduled, today, .]$_B$



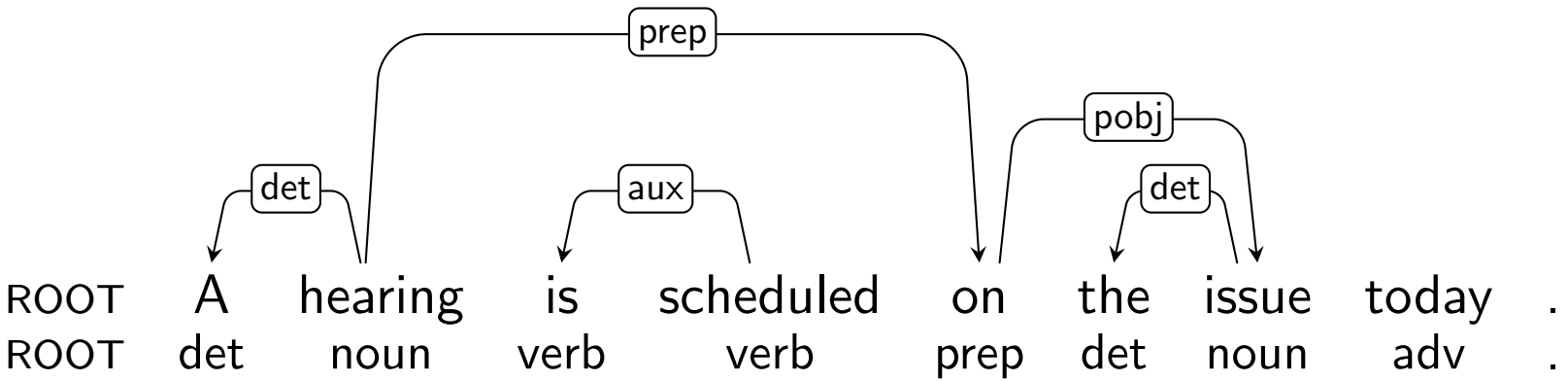
Example Transition Sequence

[ROOT, hearing]$_S$ [scheduled, today, .]$_B$



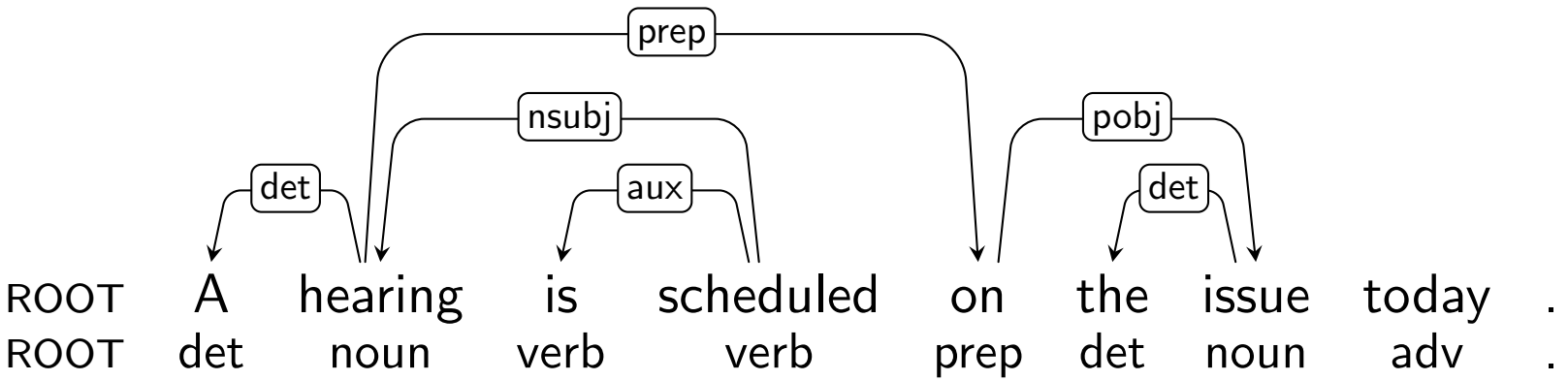
Example Transition Sequence

[ROOT, hearing, scheduled]$_S$ [today, .]$_B$



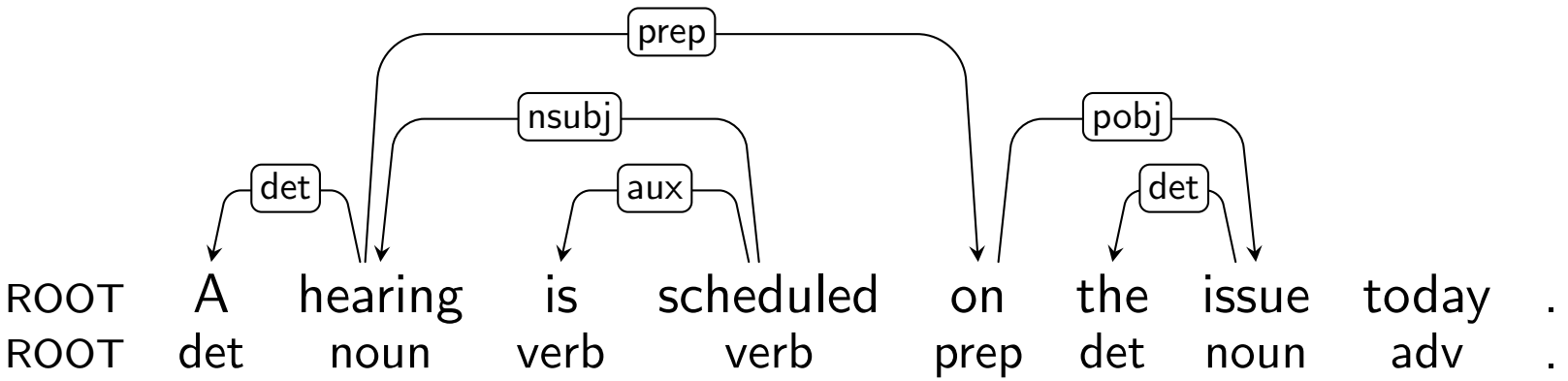
Example Transition Sequence

[ROOT, scheduled]$_S$ [today, .]$_B$



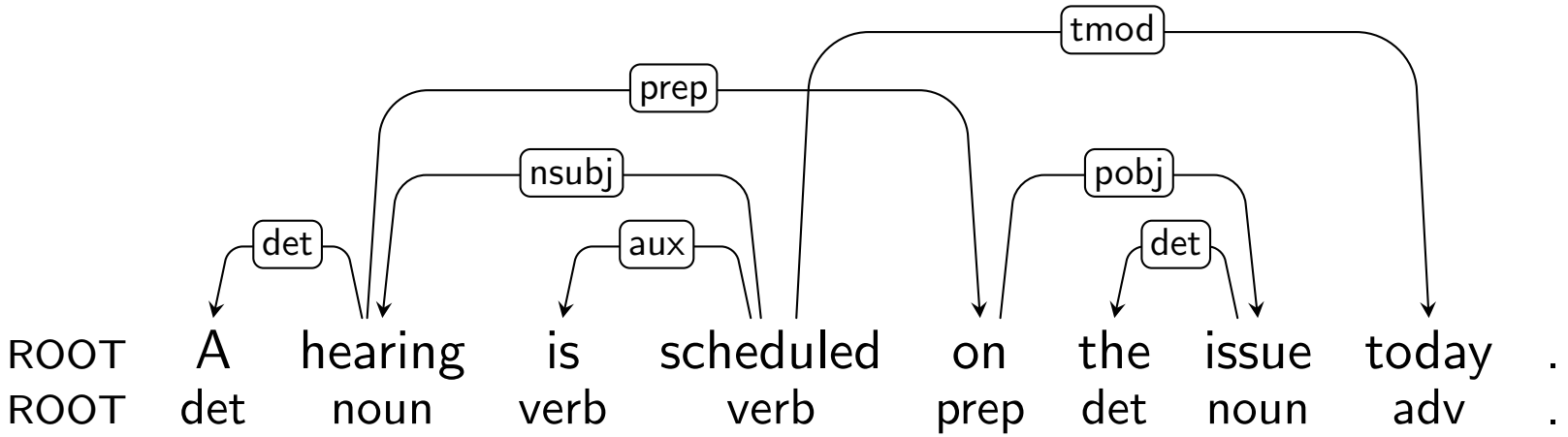
Example Transition Sequence

[ROOT, scheduled, today]$_S$ [.]$_B$



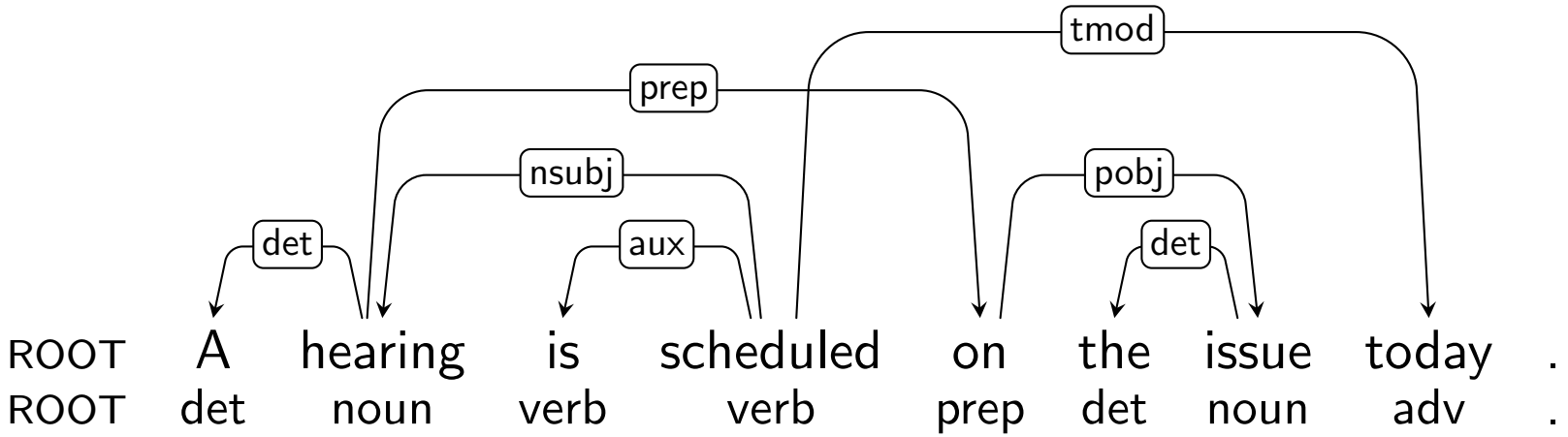
Example Transition Sequence

[ROOT, scheduled]$_S$ [.]$_B$



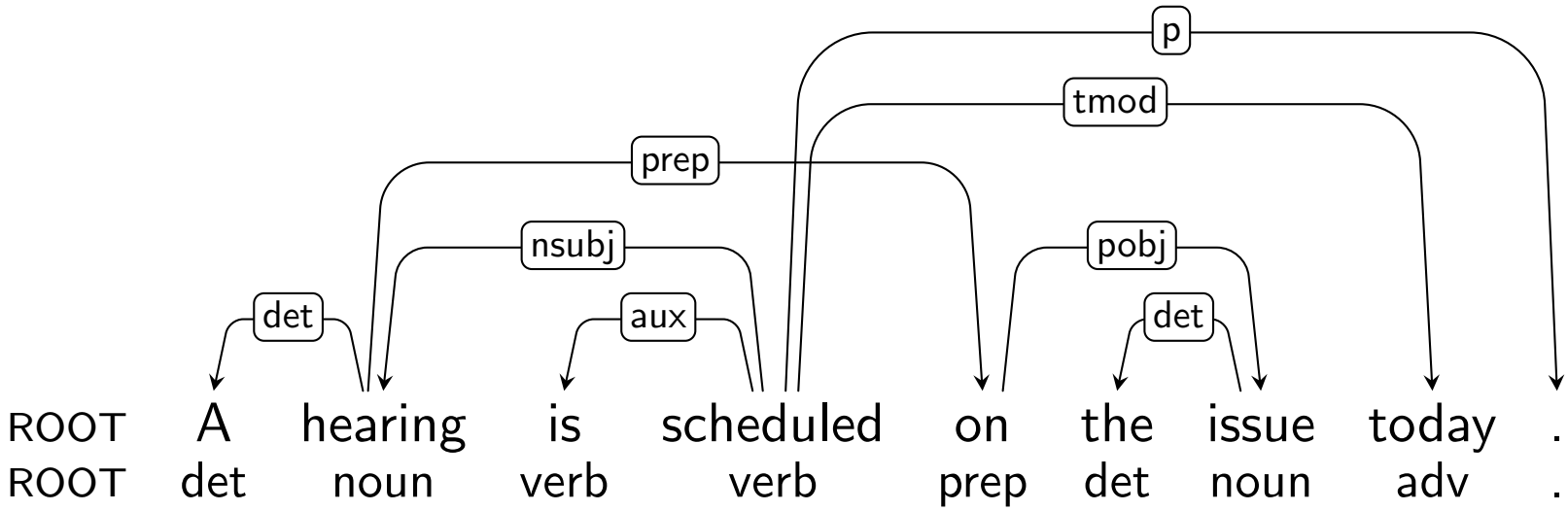
Example Transition Sequence

[ROOT, scheduled, .]$_S$ [ ]$_B$



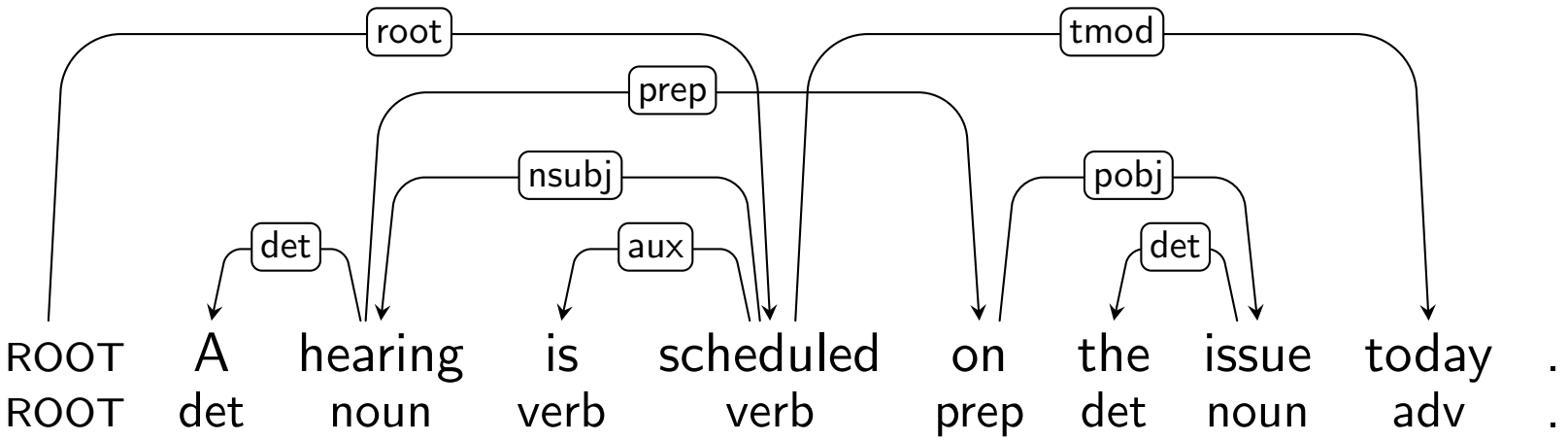
Example Transition Sequence

[ROOT, scheduled]$_S$ [ ]$_B$



Example Transition Sequence

[ROOT]$_S$ [ ]$_B$



Summary:

- Different parsing strategy: Arc-eager parsing
- Non-projective parsing: Online reordering

Next week:

- Dynamic oracle: learn from errors at train time

THE

END

References and Further Reading

- Giuseppe Attardi. 2006.
  Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.
- Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2004.
  Deterministic dependency structure analyzer for Chinese. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP)*, pages 500–508.
- Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005.
  Machine learning-based dependency analyzer for Chinese. In *Proceedings of International Conference on Chinese Computing (ICCC)*, pages 66–73.
- Hideki Isozaki, Hideto Kazawa, and Tsutomu Hirao. 2004.
  A deterministic word dependency analyzer enhanced with preference learning. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 275–281.
- Taku Kudo and Yuji Matsumoto. 2002.
  Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pages 63–69.
- Joakim Nivre and Jens Nilsson. 2005.

Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

- Joakim Nivre and Mario Scholz. 2004.
  Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 64–70.
- Joakim Nivre, Johan Hall, and Jens Nilsson. 2004.
  Memory-based dependency parsing. In Hwee Tou Ng and Ellen Riloff, editors, *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.
- Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryi**u**git, and Svetoslav Marinov. 2006.
  Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.
- Joakim Nivre. 2003.
  An efficient algorithm for projective dependency parsing. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.
- Hiroyasu Yamada and Yuji Matsumoto. 2003.

Statistical dependency analysis with support vector machines. In Gertjan Van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.