Exact Decoding of Syntactic Translation Models Through Lagrangian Relaxation

Alexander M. Rush and Michael Collins

Syntactic Translation

Problem:

Decoding synchronous grammar for machine translation

Example:

<s> abarks le dug </s>

↓

<s> the dog barks loudly </s>

Goal:

$$y^* = \arg\max_y f(y)$$

where y is a parse derivation in a synchronous grammar

Hiero Example

Consider the input sentence

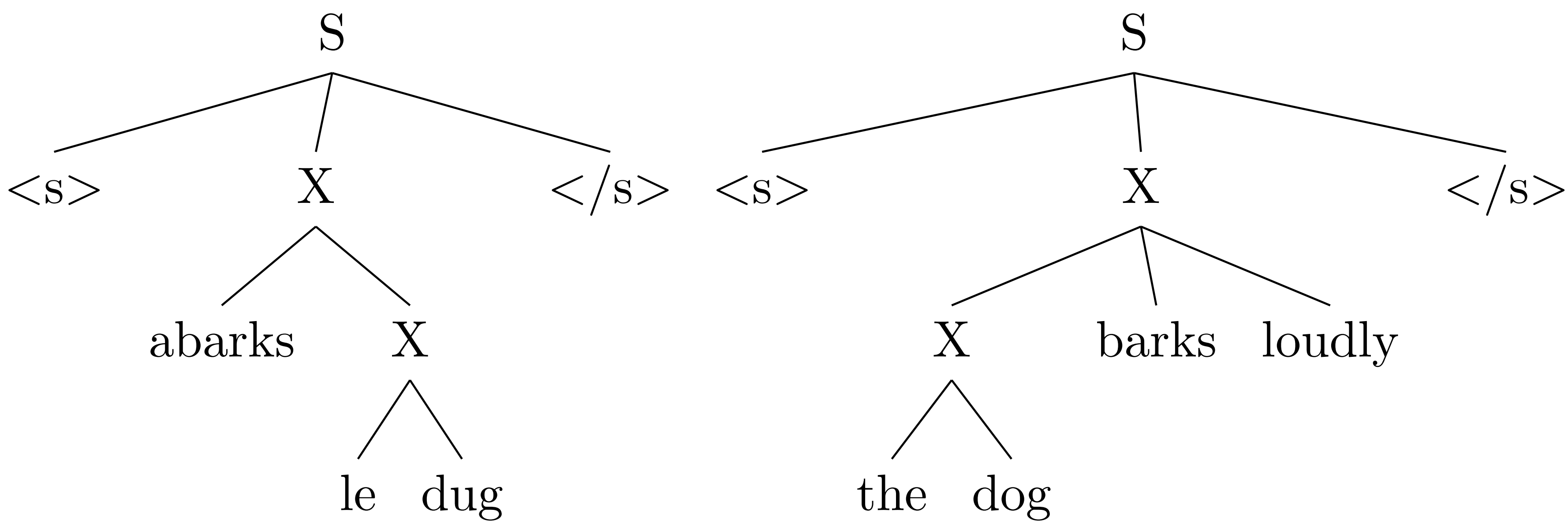
<s> abarks le dug </s>

And the synchronous grammar

S $\rightarrow$ <s> X </s>, <s> X </s>
X $\rightarrow$ abarks X, X barks loudly
X $\rightarrow$ abarks X, barks X
X $\rightarrow$ abarks X, barks X loudly
X $\rightarrow$ le dug, the dog
X $\rightarrow$ le dug, a cat

Hiero Example

Apply synchronous rules to map this sentence



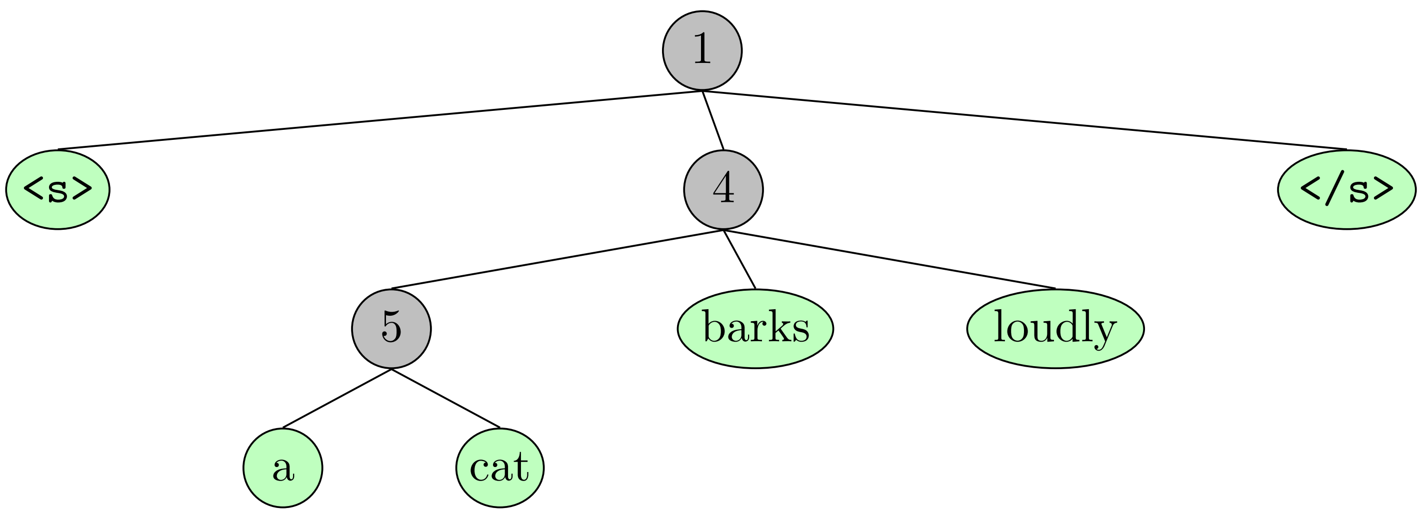
Many possible mappings:

<s> the dog barks loudly </s>
<s> a cat barks loudly </s>
<s> barks the dog </s>
<s> barks a cat </s>
<s> barks the dog loudly </s>
<s> barks a cat loudly </s>

Translation Forest

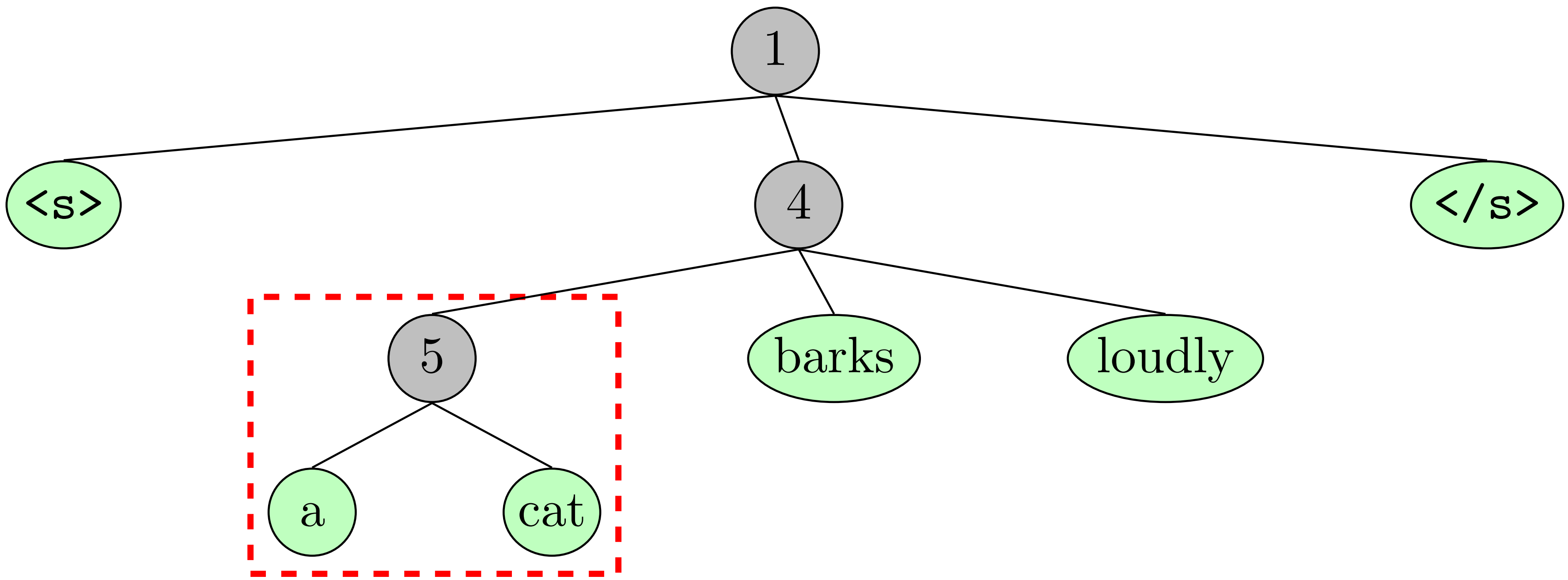
Rule	Score
1 → <s> 4 </s>	-1
4 → 5 barks loudly	2
4 → barks 5	0.5
4 → barks 5 loudly	3
5 → the dog	-4
5 → a cat	2.5

Example: a derivation in the translation forest



Scoring function

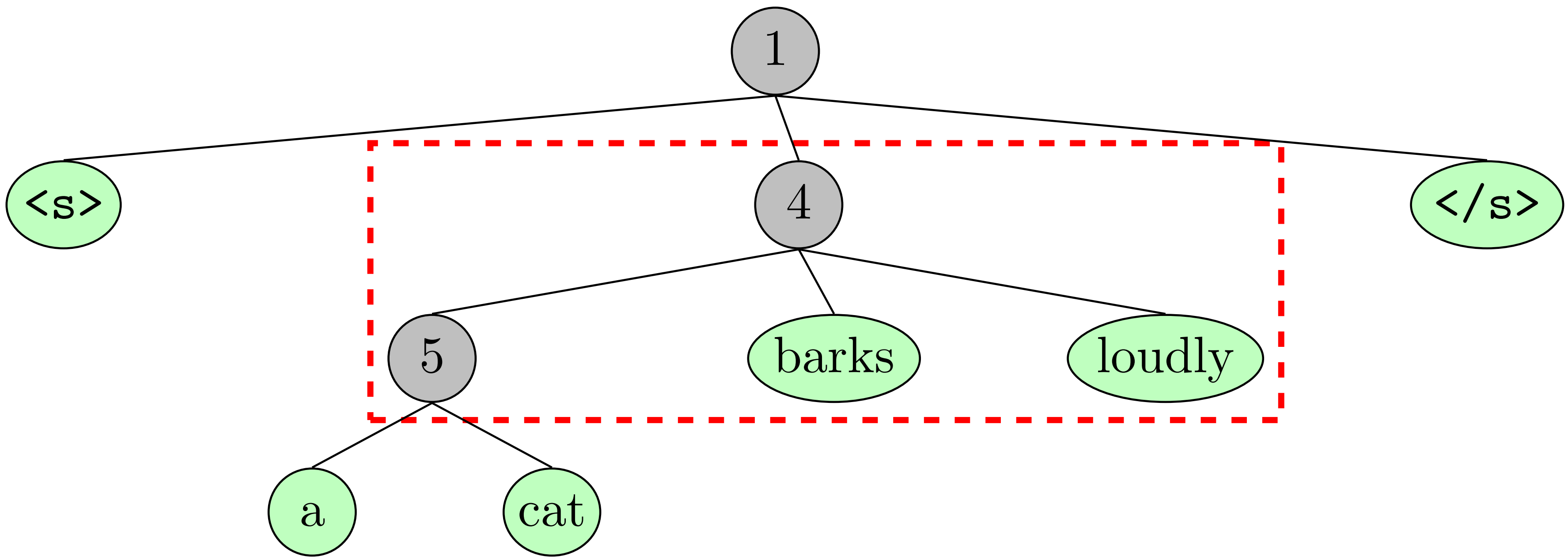
Score : sum of hypergraph derivation and language model



$f(y) = \textit{score}(5 \rightarrow \text{a cat})$

Scoring function

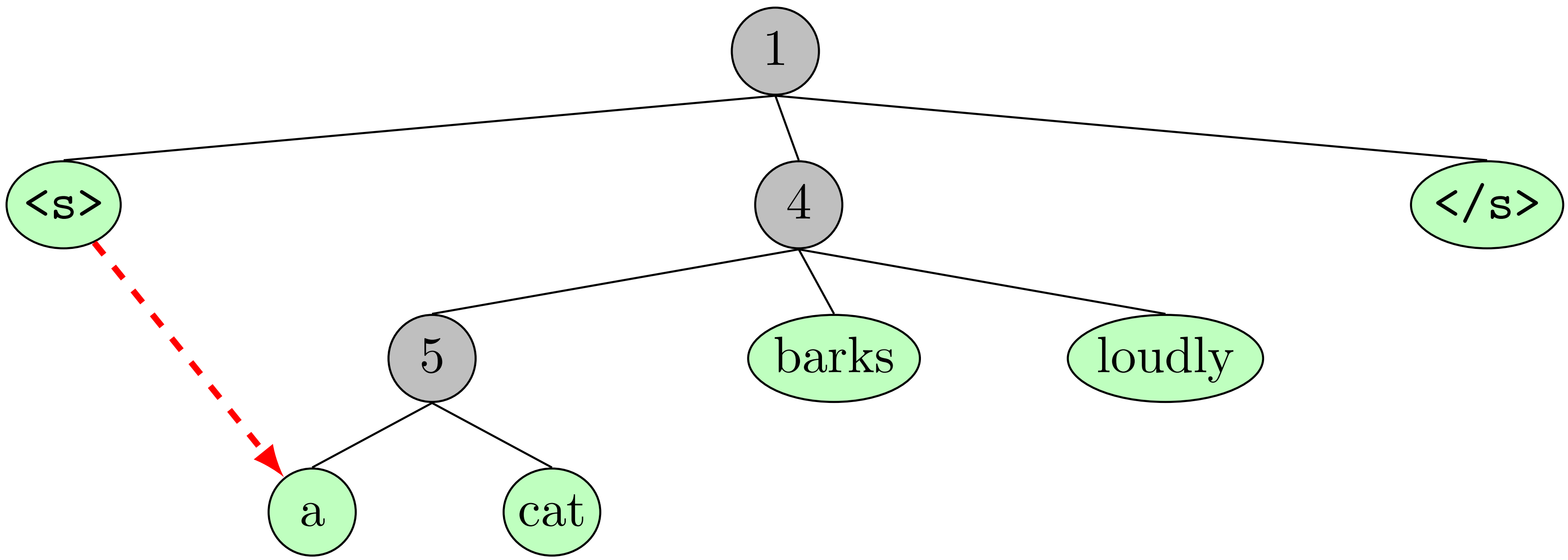
Score : sum of hypergraph derivation and language model



$$f(y) = \mathit{score}(5 \rightarrow \text{a cat}) + \mathit{score}(4 \rightarrow \text{5 barks loudly})$$

Scoring function

Score : sum of hypergraph derivation and language model

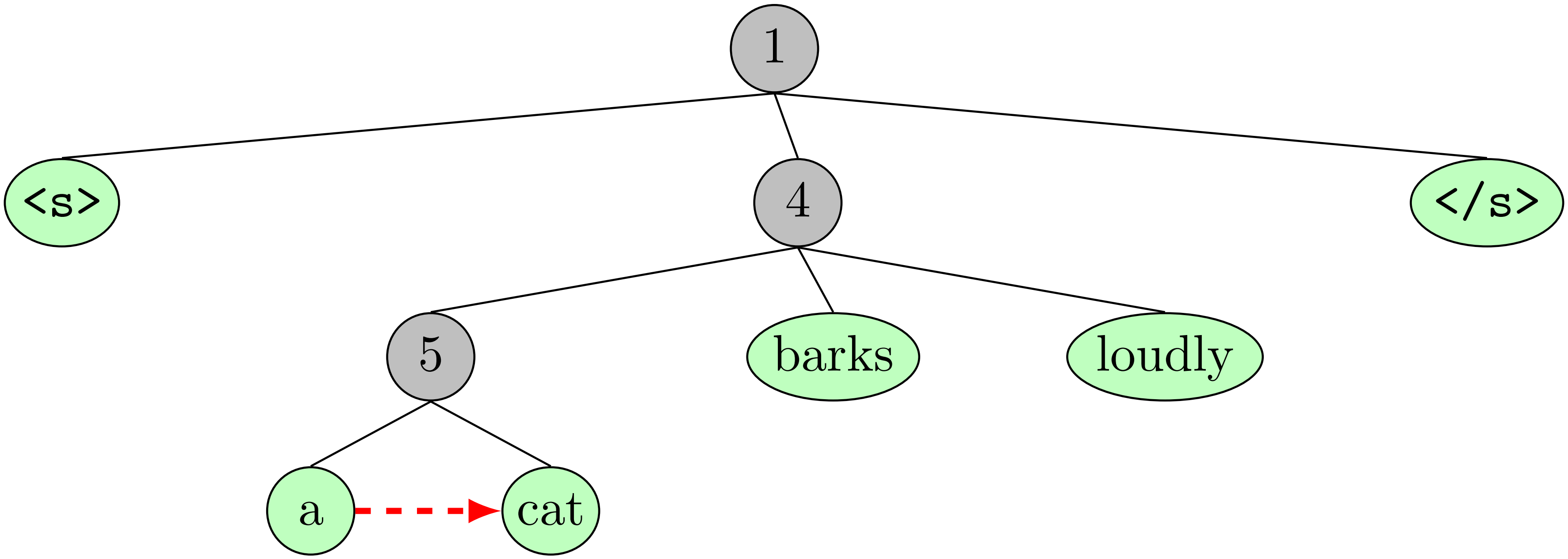


$$f(y) = \textit{score}(5 \rightarrow \text{a cat}) + \textit{score}(4 \rightarrow \text{5 barks loudly}) + \ldots$$

$$+\textit{score}(\texttt{<s>}, \text{the})$$

Scoring function

Score : sum of hypergraph derivation and language model

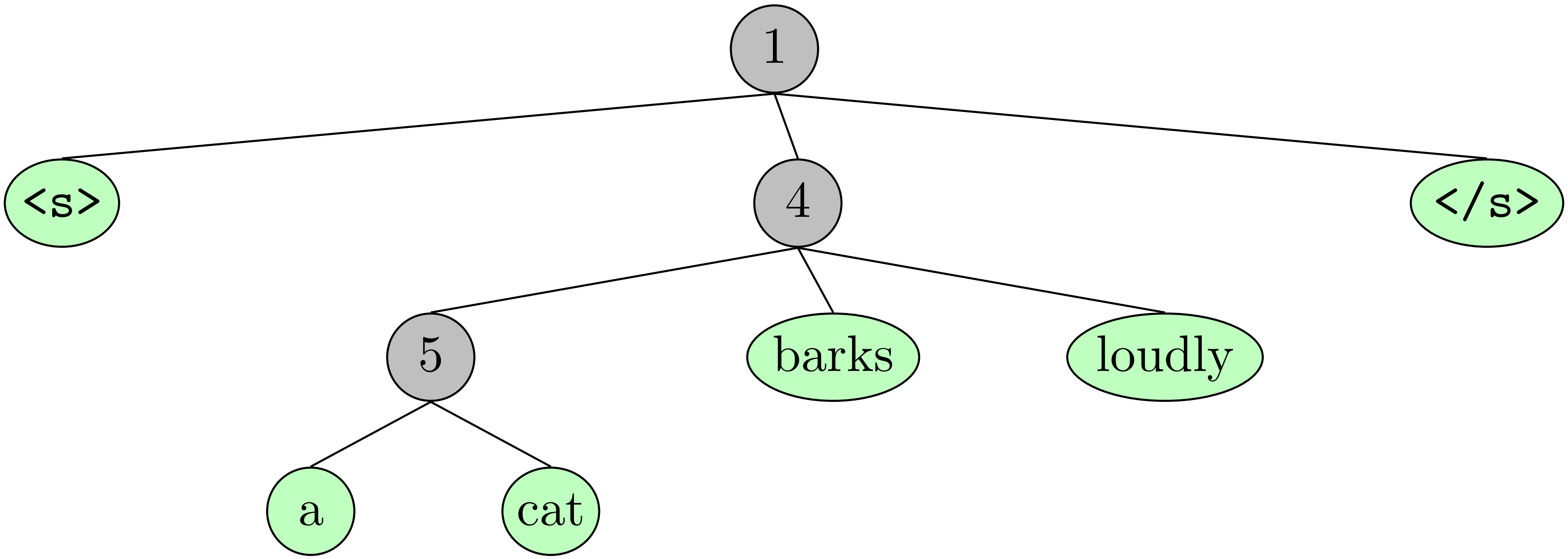


$$f(y) = score(5 \rightarrow \text{a cat}) + score(4 \rightarrow \text{5 barks loudly}) + \ldots$$

$$+score(\texttt{<s>}, \text{a}) + score(\text{a}, \text{cat})$$

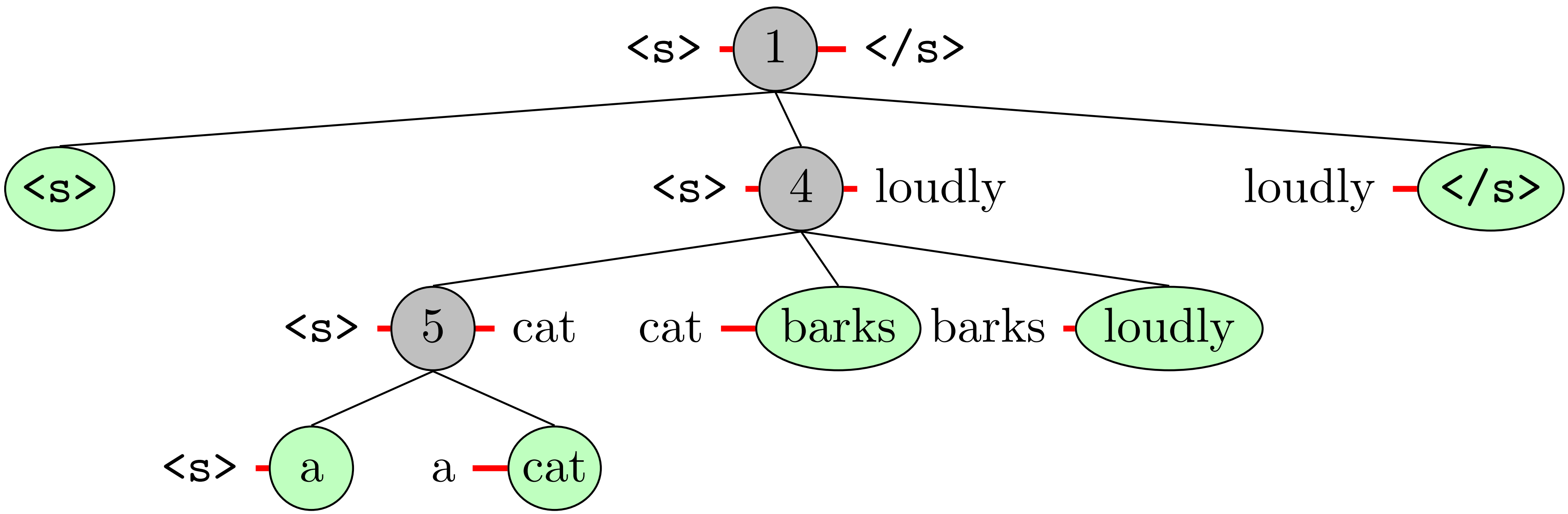
Exact Dynamic Programming

To maximize combined model, need to ensure that bigrams are consistent with parse tree.



Exact Dynamic Programming

To maximize combined model, need to ensure that bigrams are consistent with parse tree.



Original Rules
$5 \rightarrow$ the dog
$5 \rightarrow$ a cat

New Rules
${}_{<s>}5_{cat} \rightarrow {}_{<s>}the_{the}\ {}_{the}dog_{dog}$
${}_{barks}5_{cat} \rightarrow {}_{barks}the_{the}\ {}_{the}dog_{dog}$
${}_{<s>}5_{cat} \rightarrow {}_{<s>}a_{a}\ {}_{a}cat_{cat}$
${}_{barks}5_{cat} \rightarrow {}_{barks}a_{a}\ {}_{a}cat_{cat}$

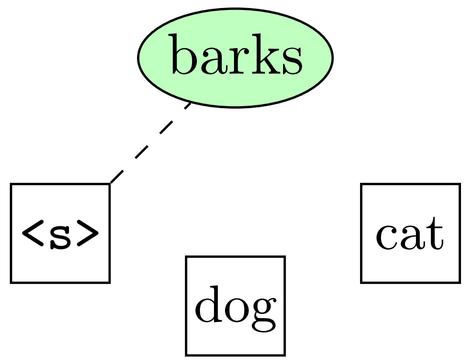
Lagrangian Relaxation Algorithm for Syntactic Translation

Outline:

- Algorithm for simplified version of translation
- Full algorithm with certificate of exactness
- Experimental results

Thought experiment: Greedy language model

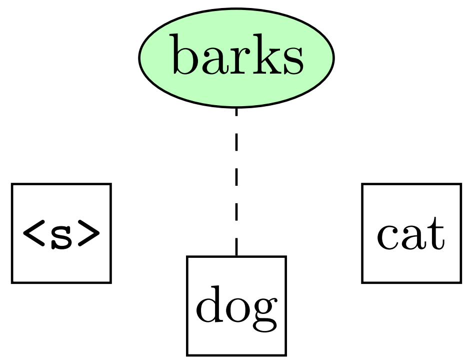
Choose best bigram for a given word



- *score*(<s>, barks)

Thought experiment: Greedy language model

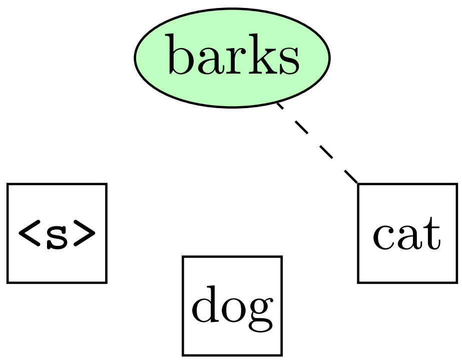
Choose best bigram for a given word



- *score*(<s>, barks)
- *score*(dog, barks)

Thought experiment: Greedy language model

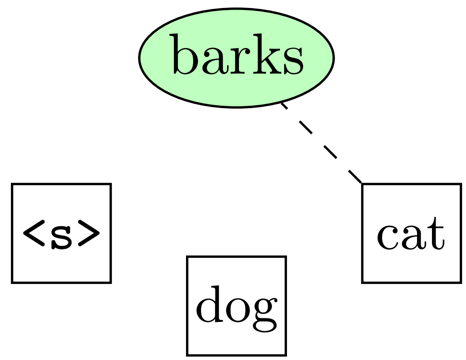
Choose best bigram for a given word



- *score*(<s>, barks)
- *score*(dog, barks)
- *score*(cat, barks)

Thought experiment: Greedy language model

Choose best bigram for a given word



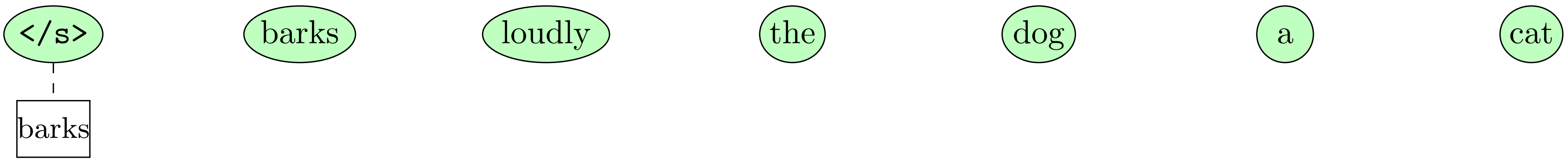
- *score*(<s>, barks)
- *score*(dog, barks)
- *score*(cat, barks)

Can compute with a simple maximization

$$\arg\max_{w:\langle w, \text{barks}\rangle \in \mathcal{B}} score(w, \text{barks})$$

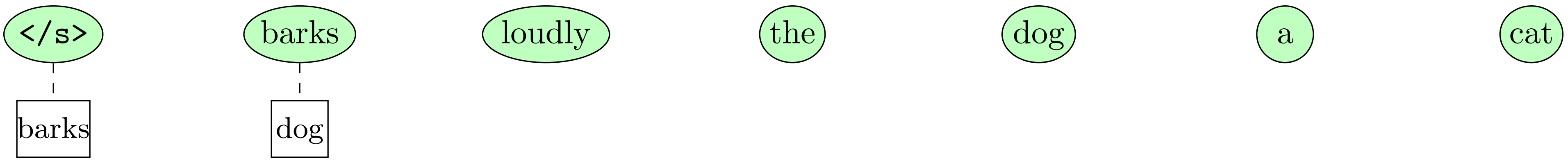
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



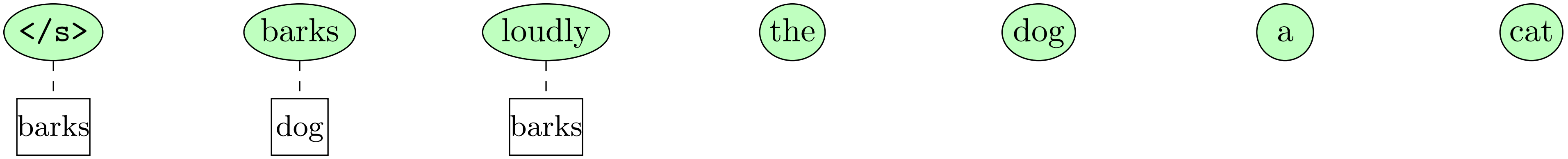
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



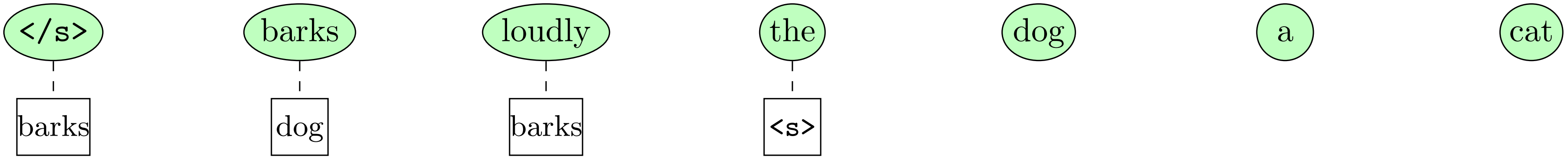
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



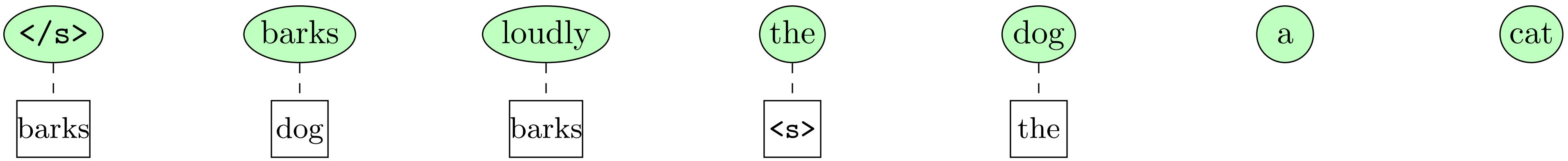
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



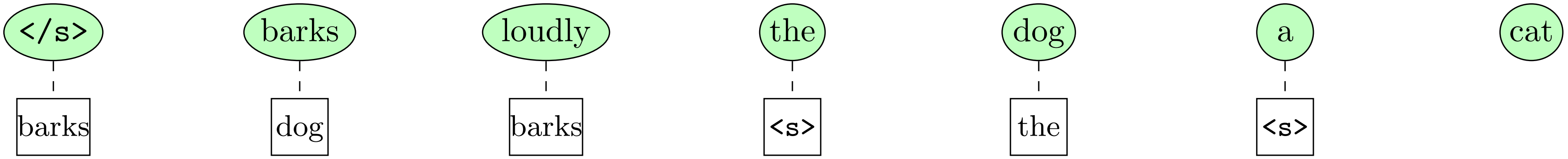
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



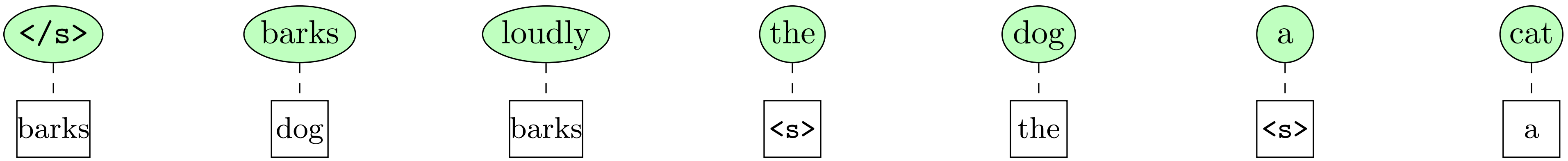
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



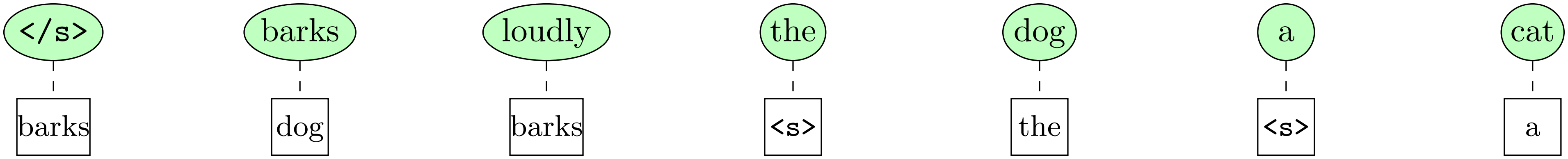
Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word



Thought experiment: Full decoding

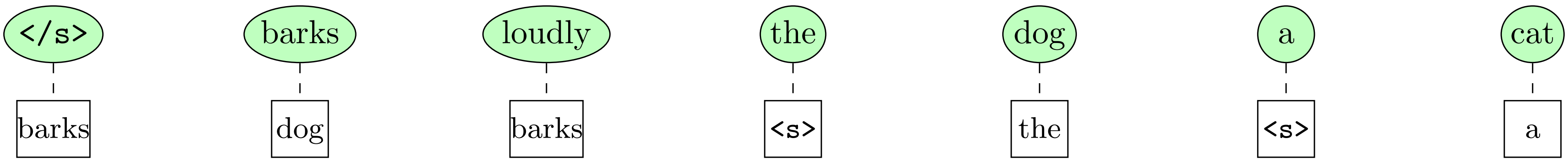
Step 1. Greedily choose best bigram for each word



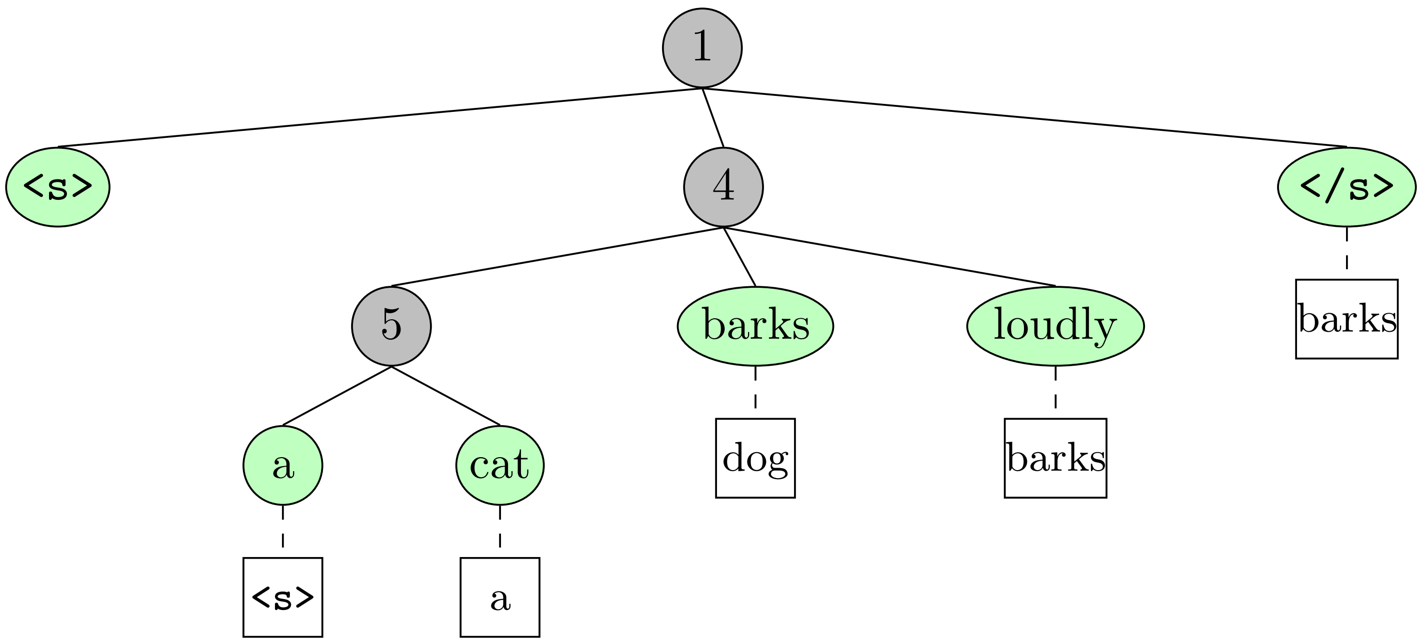
Step 2. Find the best derivation with fixed bigrams

Thought experiment: Full decoding

Step 1. Greedily choose best bigram for each word

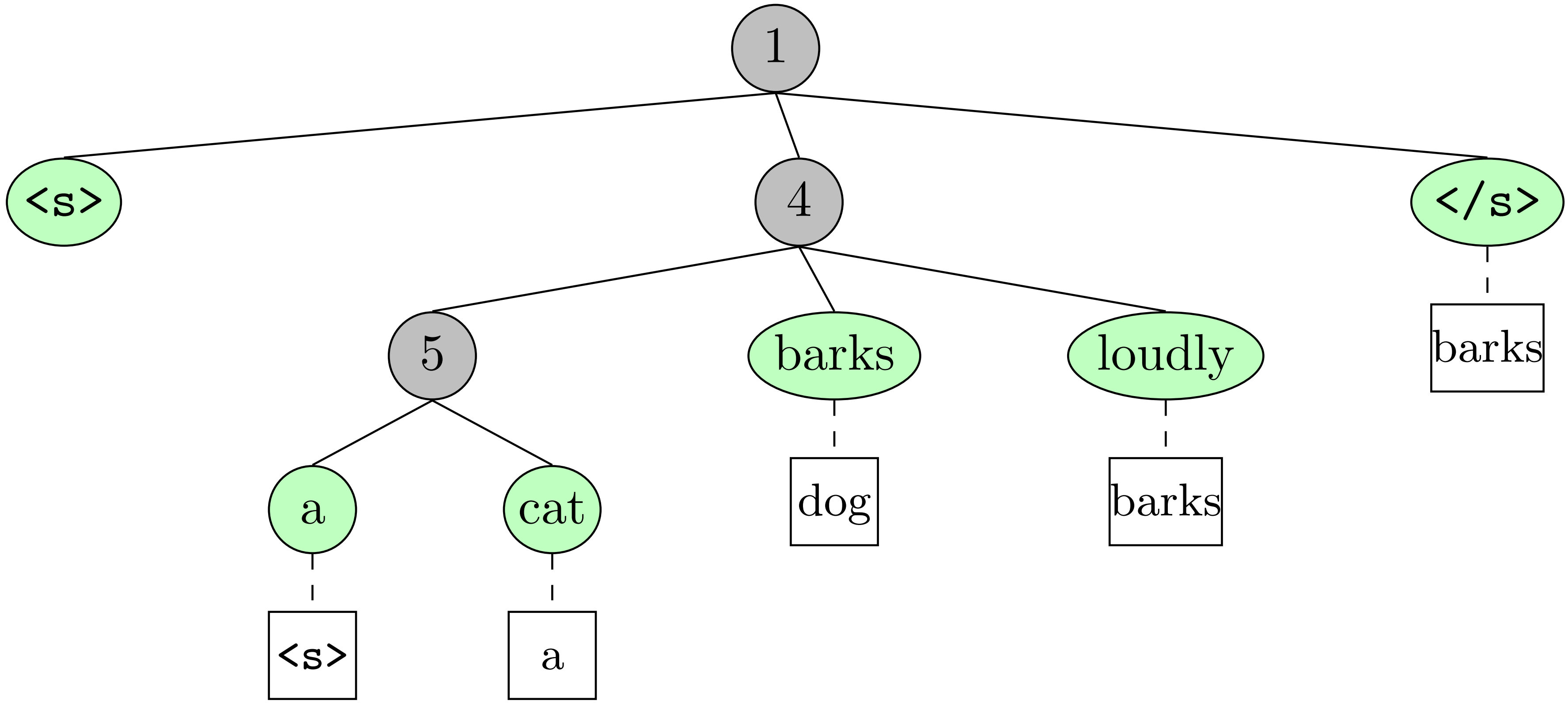


Step 2. Find the best derivation with fixed bigrams



Thought Experiment Problem

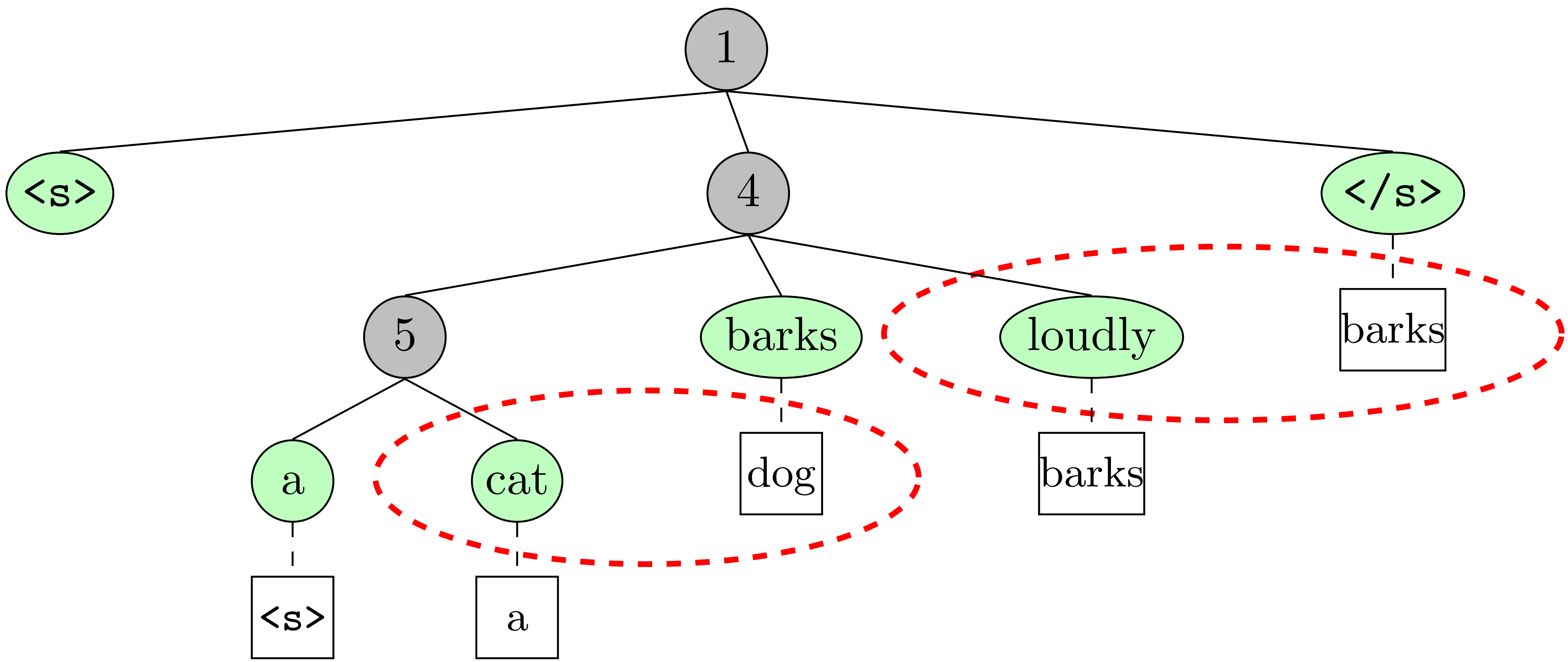
May produce invalid parse and bigram relationship



Greedy bigram selection may conflict with the parse derivation

Thought Experiment Problem

May produce invalid parse and bigram relationship



Greedy bigram selection may conflict with the parse derivation

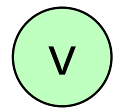
Formal objective

Notation: $y(w, v) = 1$ if the bigram $\langle w, v \rangle \in \mathcal{B}$ is in y

Goal:

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

such that for all words nodes y_v



$$(1)$$

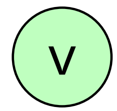
Formal objective

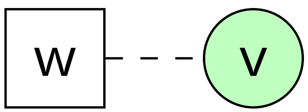
Notation: $y(w, v) = 1$ if the bigram $\langle w, v \rangle \in \mathcal{B}$ is in y

Goal:

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

such that for all words nodes y_v





$$y_v = \sum_{w : \langle w, v \rangle \in \mathcal{B}} y(w, v) \quad (1)$$

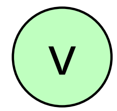
Formal objective

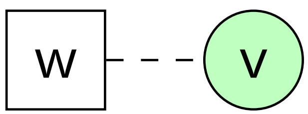
Notation: $y(w, v) = 1$ if the bigram $\langle w, v \rangle \in \mathcal{B}$ is in y

Goal:

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

such that for all words nodes y_v





$$y_v = \sum_{w : \langle w, v \rangle \in \mathcal{B}} y(w, v) \qquad (1)$$

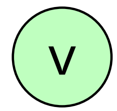
$$y_v = \sum_{w : \langle v, w \rangle \in \mathcal{B}} y(v, w) \qquad (2)$$

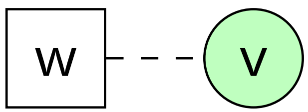
Formal objective

Notation: $y(w, v) = 1$ if the bigram $\langle w, v \rangle \in \mathcal{B}$ is in y

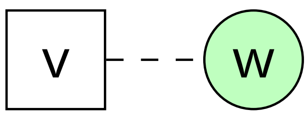
Goal:

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

such that for all words nodes y_v 



$$y_v = \sum_{w : \langle w, v \rangle \in \mathcal{B}} y(w, v) \quad (1)$$



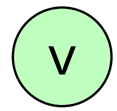
$$y_v = \sum_{w : \langle v, w \rangle \in \mathcal{B}} y(v, w) \quad (2)$$

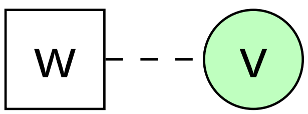
Formal objective

Notation: $y(w,v) = 1$ if the bigram $\langle w, v\rangle \in \mathcal{B}$ is in y

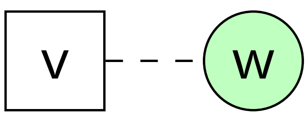
Goal:

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

such that for all words nodes y_v 



$$y_v = \sum_{w : \langle w, v\rangle \in \mathcal{B}} y(w,v) \quad (1)$$



$$y_v = \sum_{w : \langle v, w\rangle \in \mathcal{B}} y(v,w) \quad (2)$$

Lagrangian: Relax constraint (2), leave constraint (1)

$$L(u,y) = \max_{y \in \mathcal{Y}} f(y) + \sum_{w,v} u(v) \left(y_v - \sum_{w : \langle v, w\rangle \in \mathcal{B}} y(v,w) \right)$$

For a given u, $L(u,y)$ can be solved by our greedy LM algorithm

Algorithm

Set $u^{(1)}(v) = 0$ for all $v \in V_L$

For $k = 1$ **to** K

$$y^{(k)} \leftarrow \arg\max_{y \in \mathcal{Y}} L^{(k)}(u, y)$$

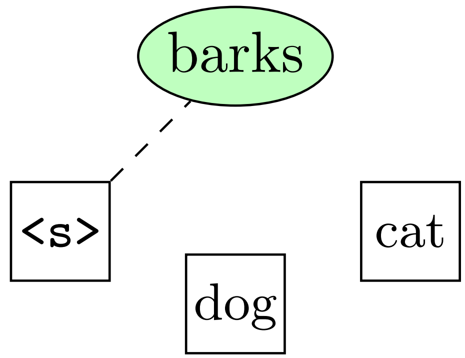
If $y_v^{(k)} = \sum_{w:\langle v,w\rangle \in \mathcal{B}} y^{(k)}(v, w)$ for all v **Return** $(y^{(k)})$

Else

$$u^{(k+1)}(v) \leftarrow u^{(k)}(v) - \alpha_k \left(y_v^{(k)} - \sum_{w:\langle v,w\rangle \in \mathcal{B}} y^{(k)}(v, w) \right)$$

Thought experiment: Greedy with penalties

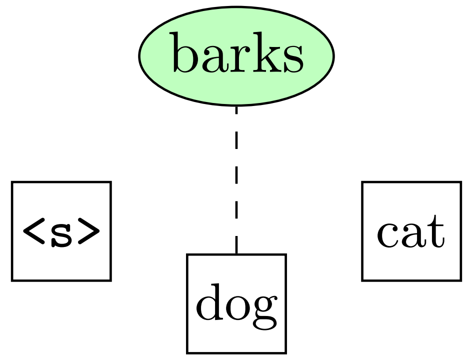
Choose best bigram with penalty for a given word



- $\mathit{score}(\texttt{<s>}, \text{barks}) - u(\texttt{<s>}) + u(\text{barks})$

Thought experiment: Greedy with penalties

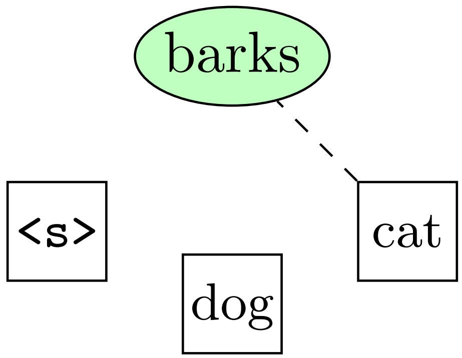
Choose best bigram with penalty for a given word



- $\mathit{score}(\texttt{<s>}, \text{barks}) - u(\texttt{<s>}) + u(\text{barks})$
- $\mathit{score}(\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks})$

Thought experiment: Greedy with penalties

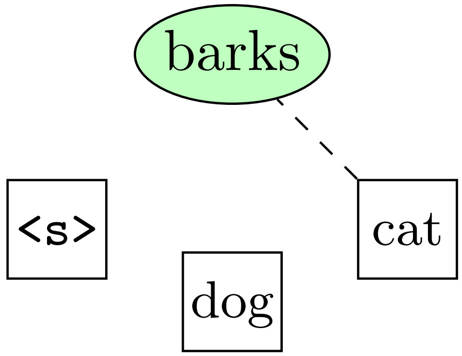
Choose best bigram with penalty for a given word



- $\textit{score}(\texttt{<s>}, \text{barks}) - u(\texttt{<s>}) + u(\text{barks})$
- $\textit{score}(\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks})$
- $\textit{score}(\text{dog}, \text{barks}) - u(\text{dog}) + u(\text{barks})$

Thought experiment: Greedy with penalties

Choose best bigram with penalty for a given word



- $\mathit{score}(\texttt{<s>}, \text{barks}) - u(\texttt{<s>}) + u(\text{barks})$
- $\mathit{score}(\text{cat}, \text{barks}) - u(\text{cat}) + u(\text{barks})$
- $\mathit{score}(\text{dog}, \text{barks}) - u(\text{dog}) + u(\text{barks})$

Can still compute with a simple maximization over

$$\arg \max_{w : \langle w, \text{barks} \rangle \in \mathcal{B}} \mathit{score}(w, \text{barks}) - u(w) + u(\text{barks})$$

Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	0	0	0	0	0	0

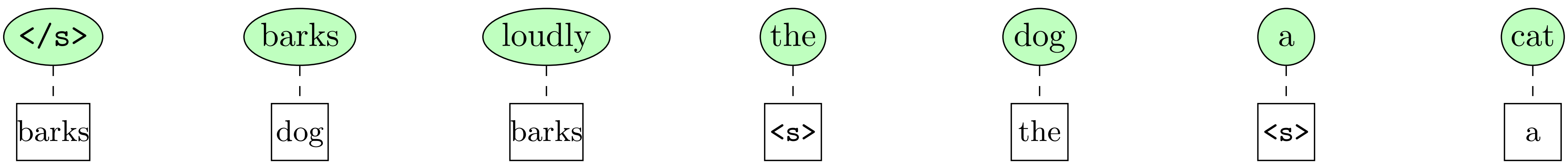
Greedy decoding

Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	0	0	0	0	0	0

Greedy decoding

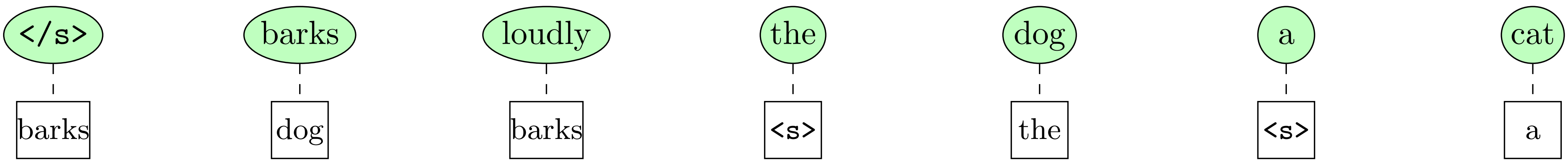


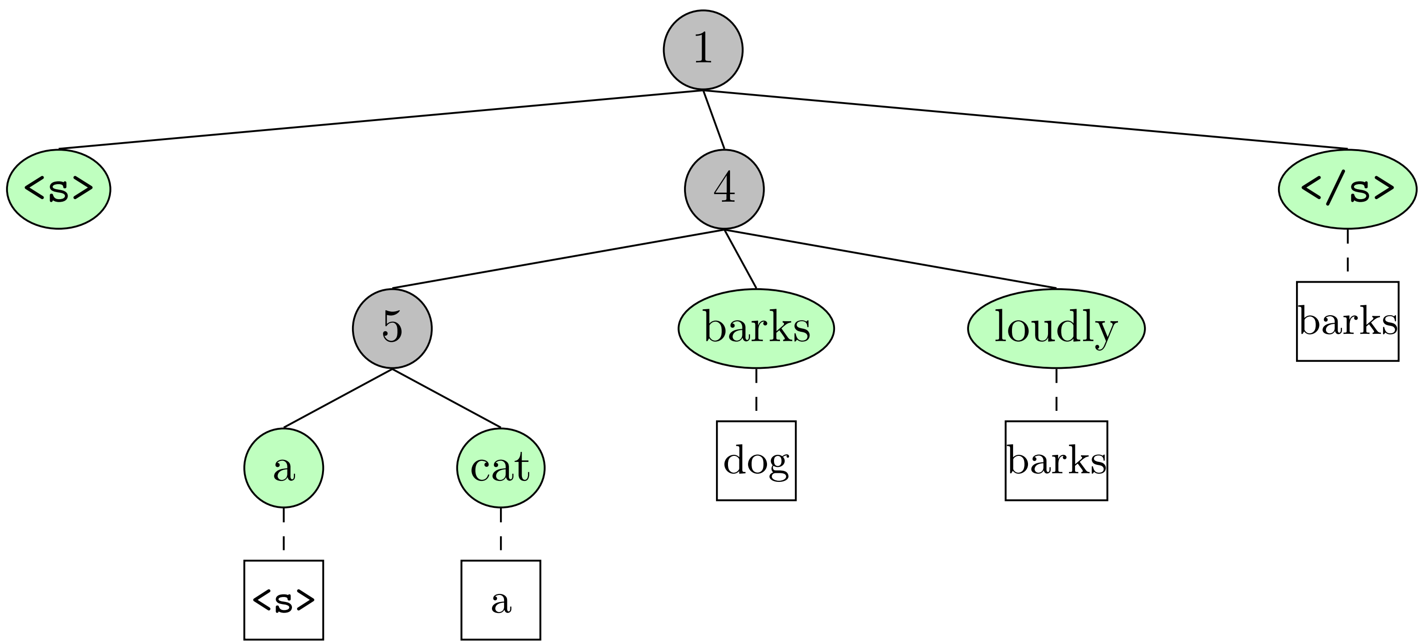
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	0	0	0	0	0	0

Greedy decoding



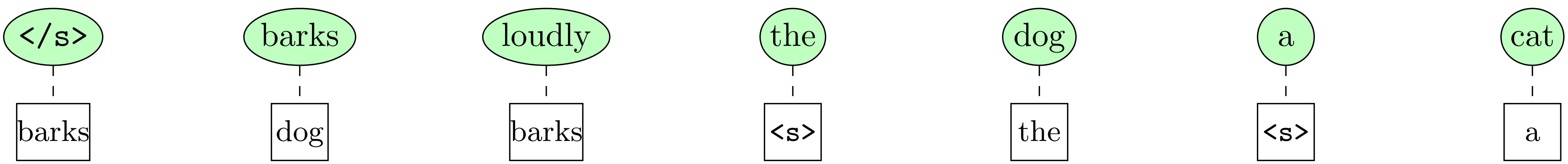


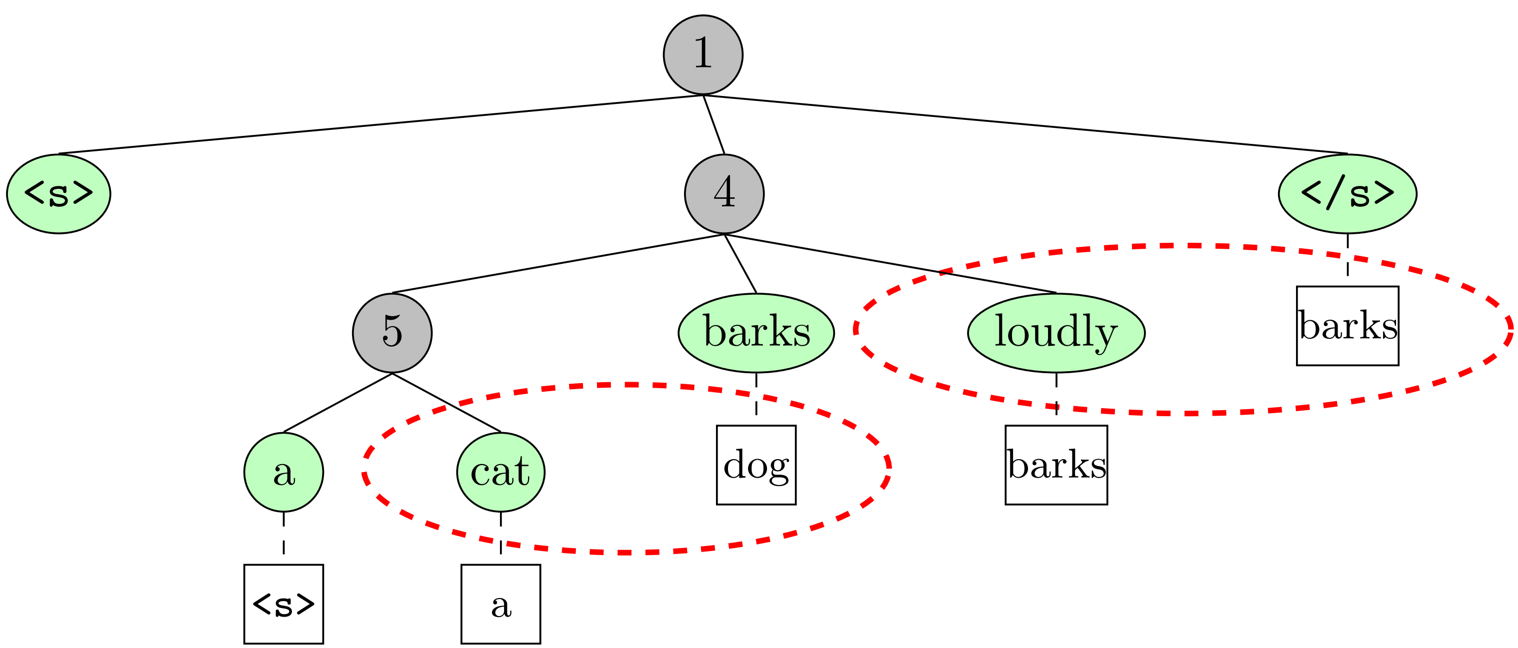
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	0	0	0	0	0	0

Greedy decoding



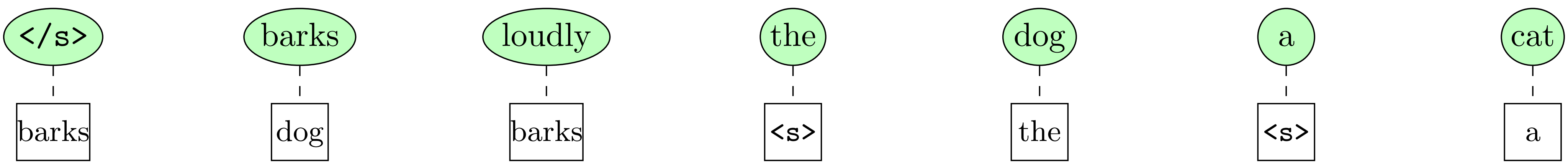


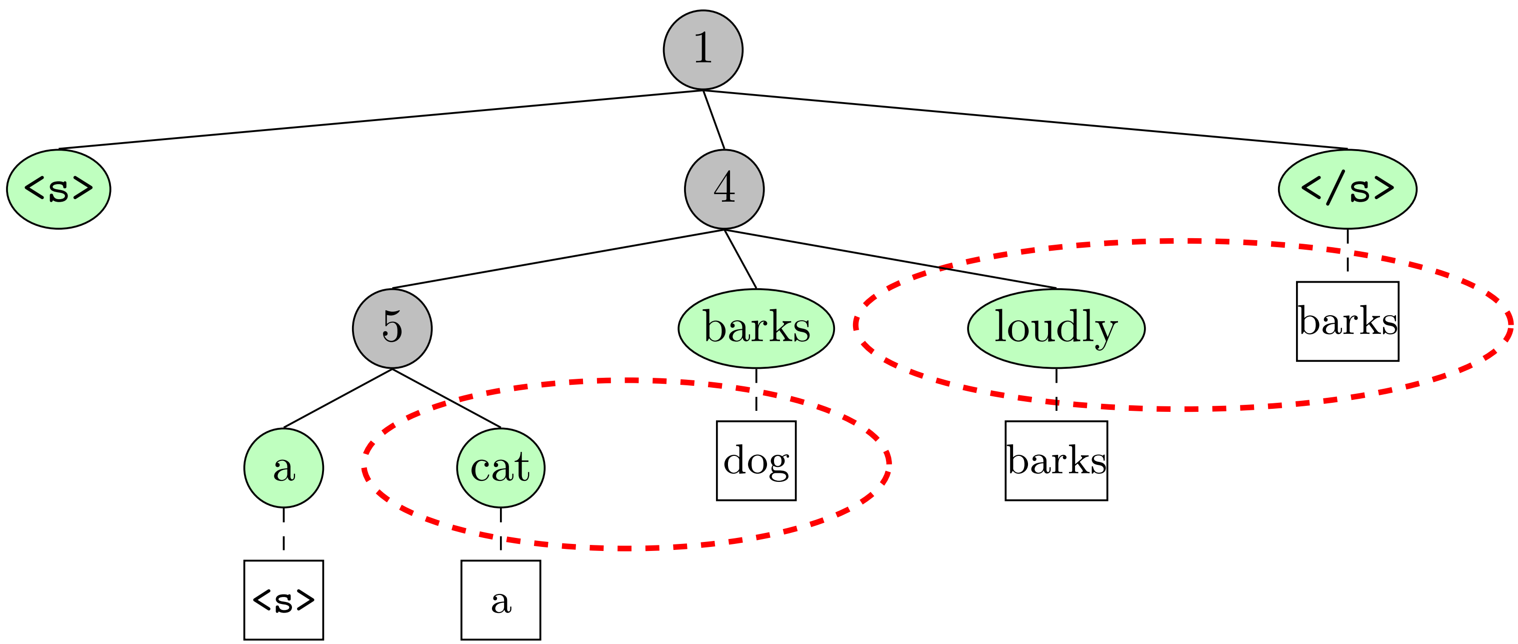
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-1	0	1

Greedy decoding





Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-1	0	1

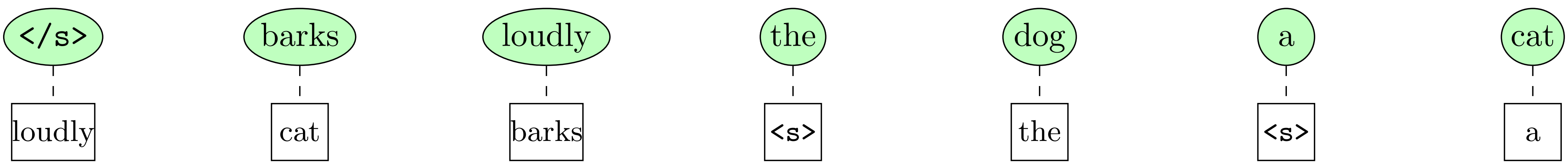
Greedy decoding

Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-1	0	1

Greedy decoding

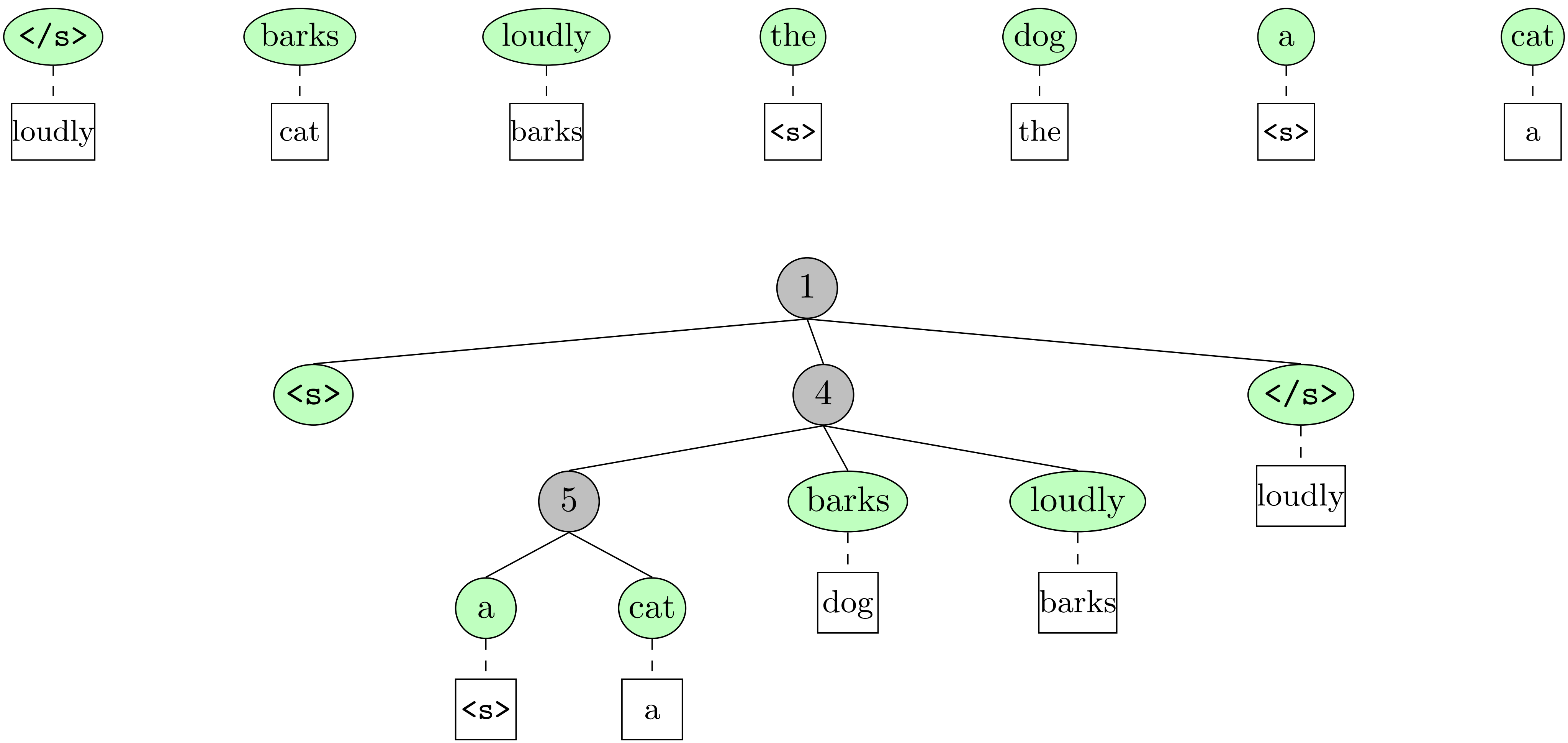


Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-1	0	1

Greedy decoding

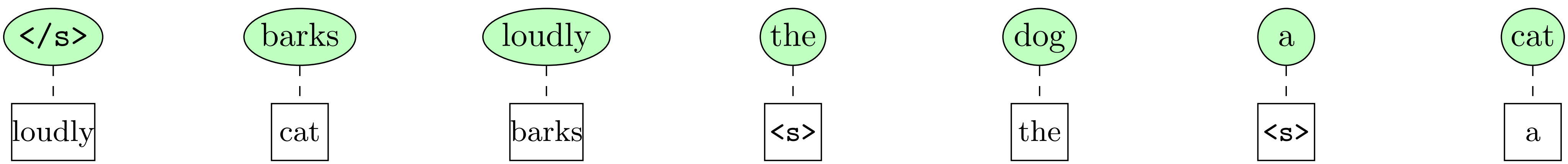


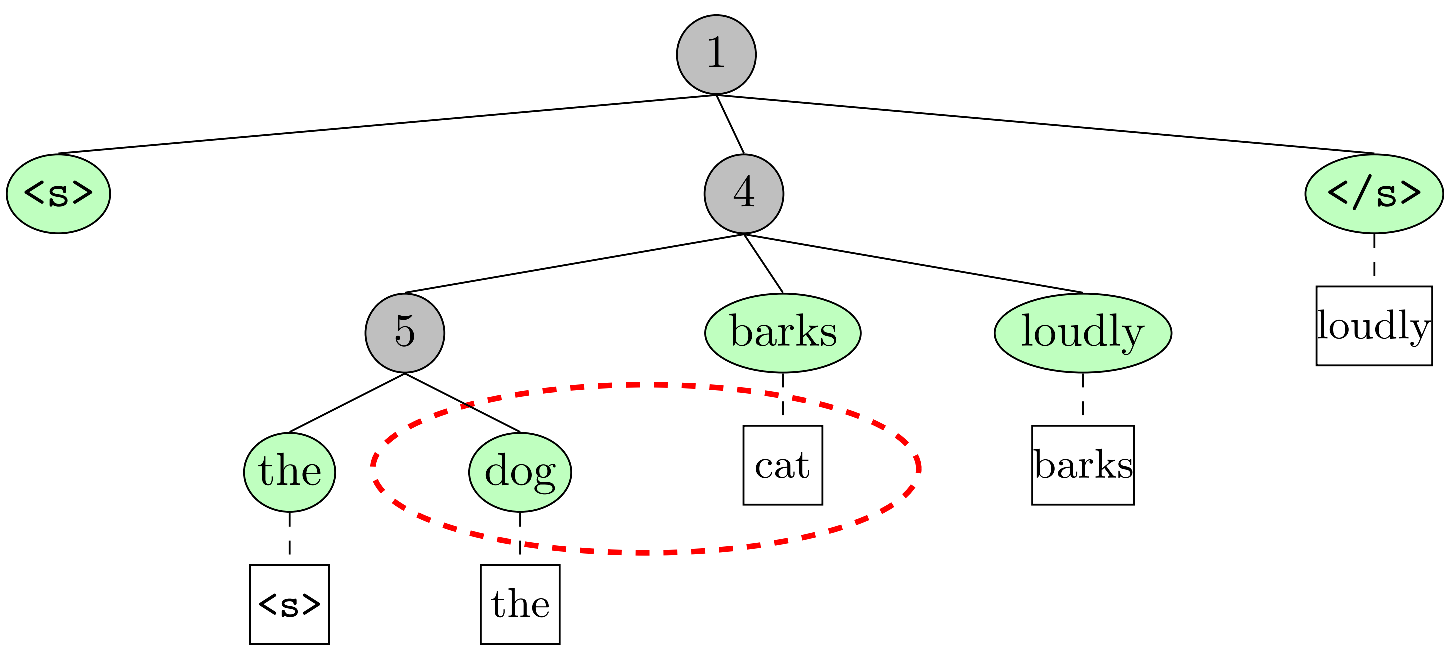
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-1	0	1

Greedy decoding



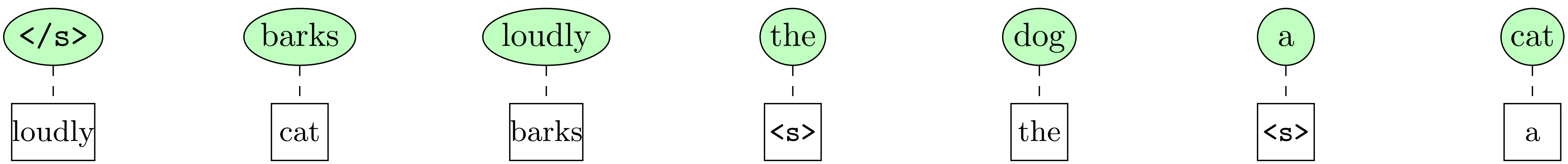


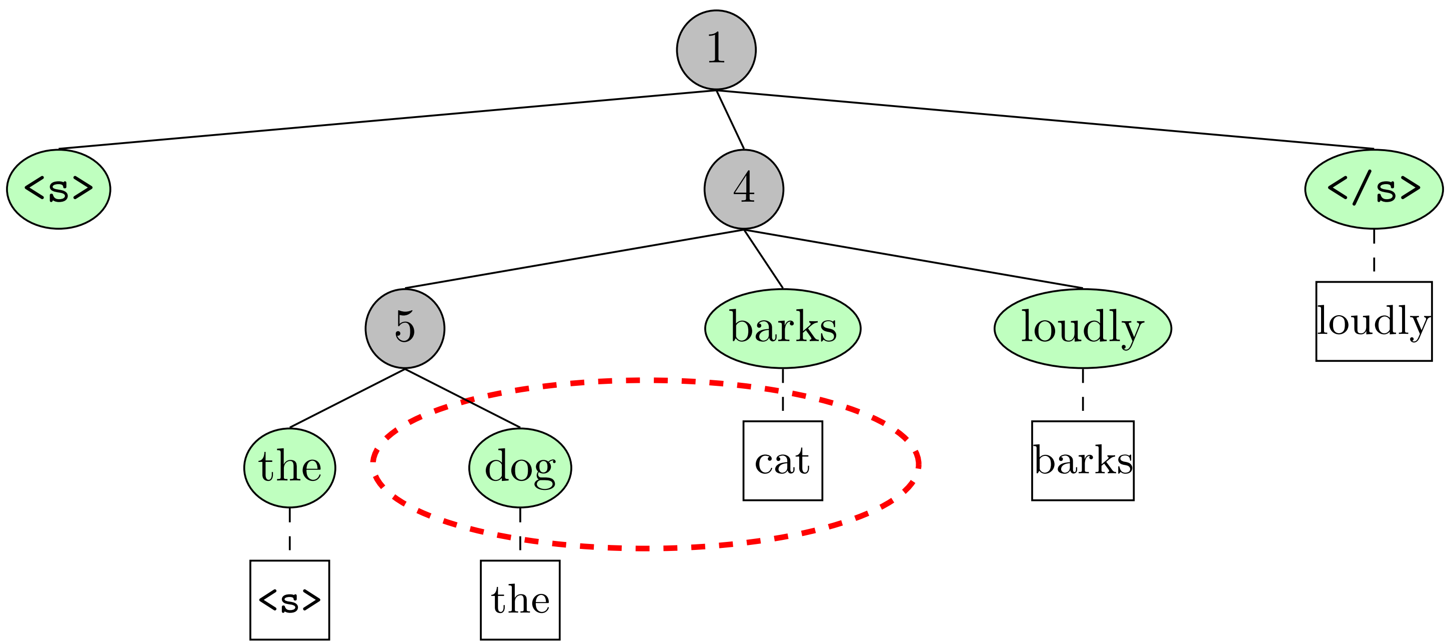
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-0.5	0	0.5

Greedy decoding





Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-0.5	0	0.5

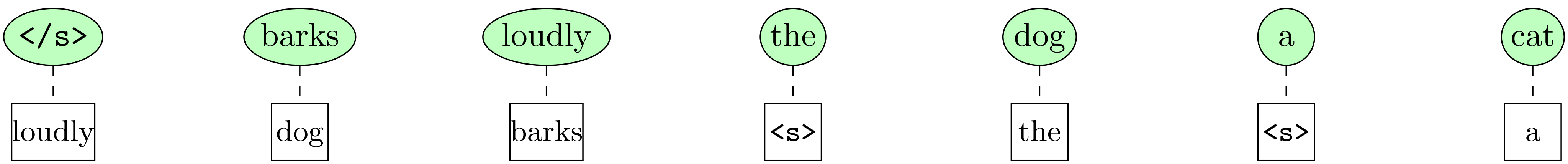
Greedy decoding

Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-0.5	0	0.5

Greedy decoding

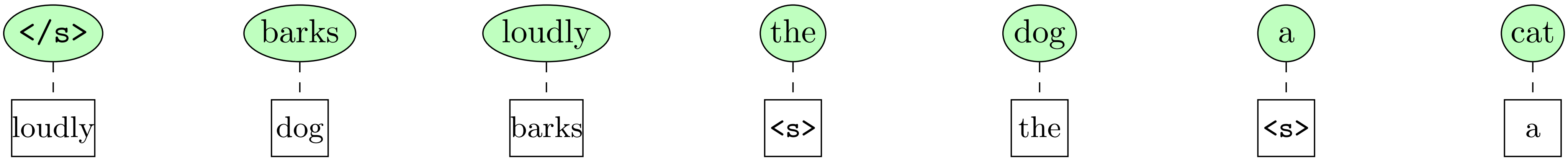


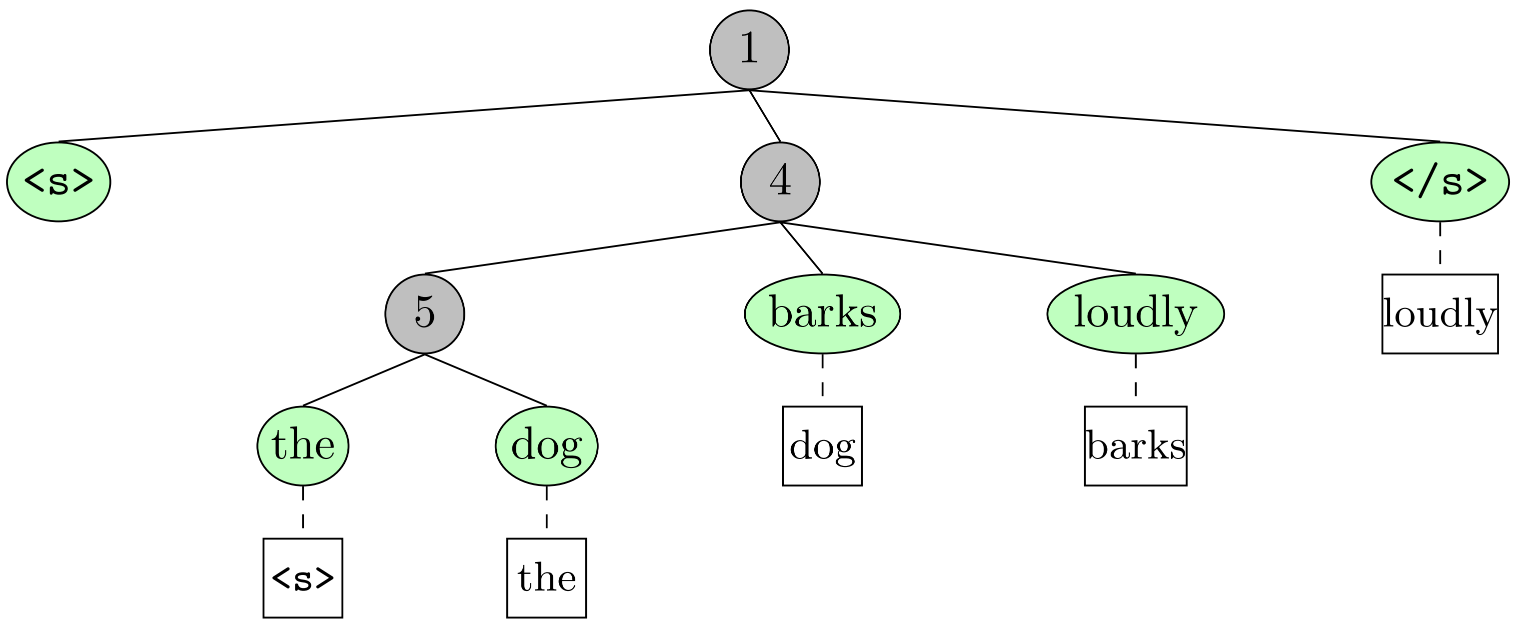
Algorithm example

Penalties

v	</s>	*barks*	*loudly*	*the*	*dog*	*a*	*cat*
u(v)	0	-1	1	0	-0.5	0	0.5

Greedy decoding





Constraint Issue

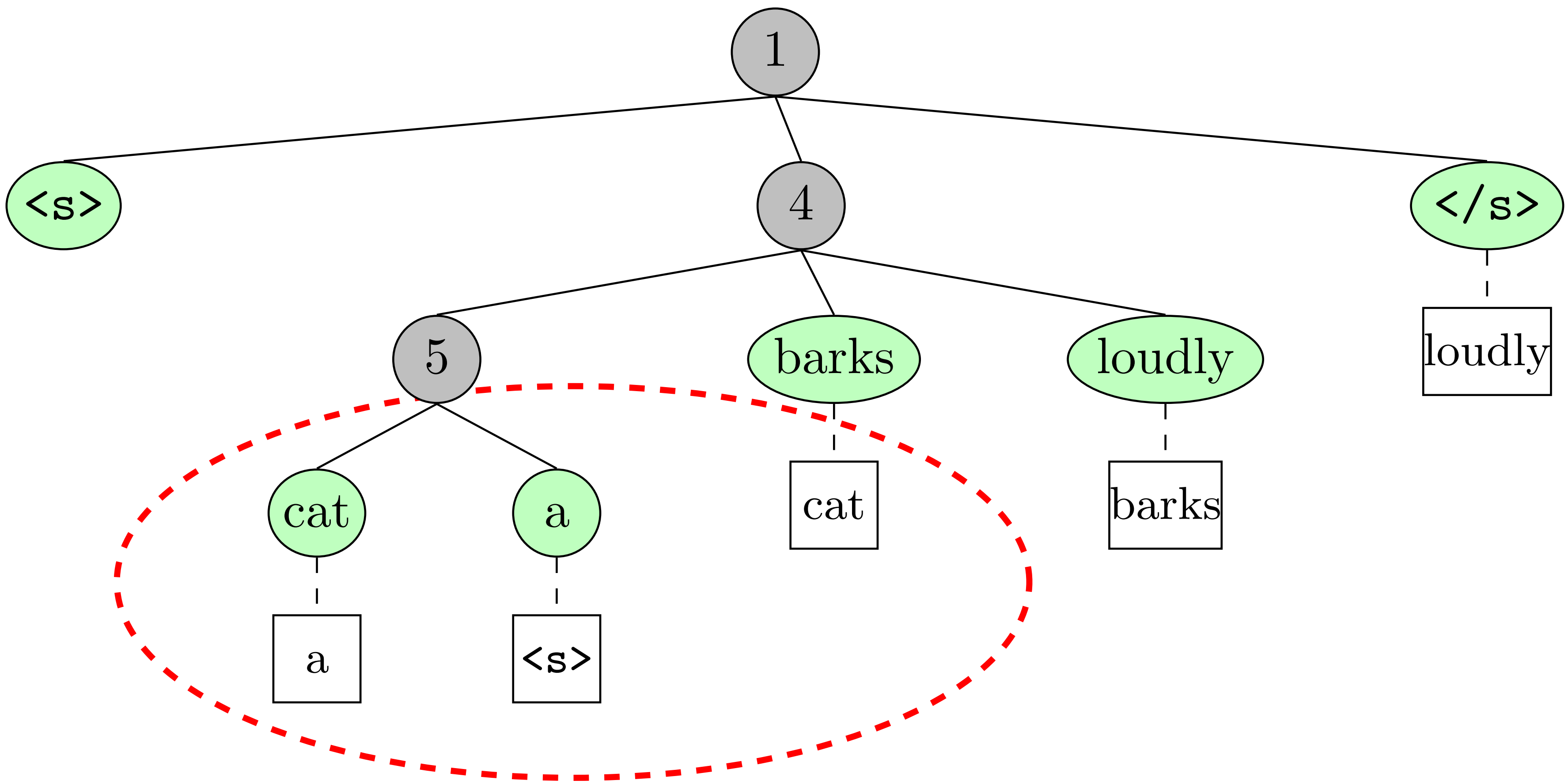
Constraints do not capture all possible reorderings

Example: Add rule $\langle 5 \rightarrow$ cat a$\rangle$ to forest. New derivation

Constraint Issue

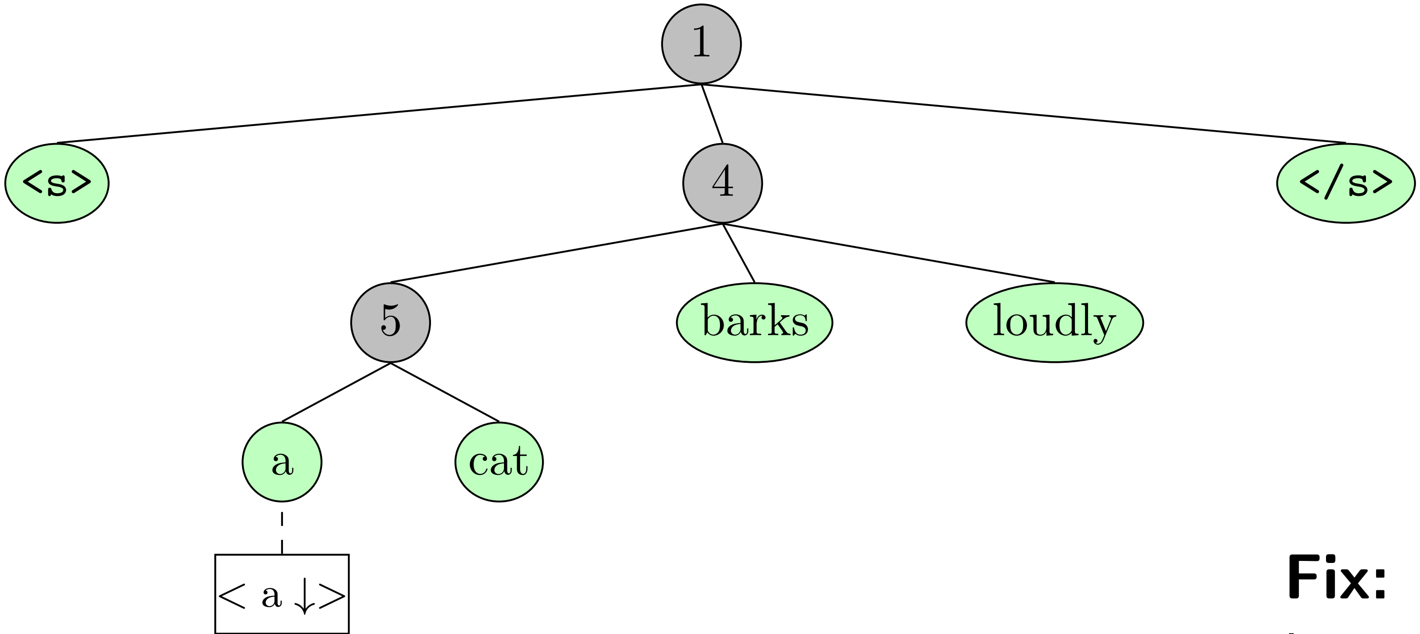
Constraints do not capture all possible reorderings

Example: Add rule $\langle 5 \rightarrow$ cat a$\rangle$ to forest. New derivation



Satisfies both constraints (1) and (2), but is not self-consistent.

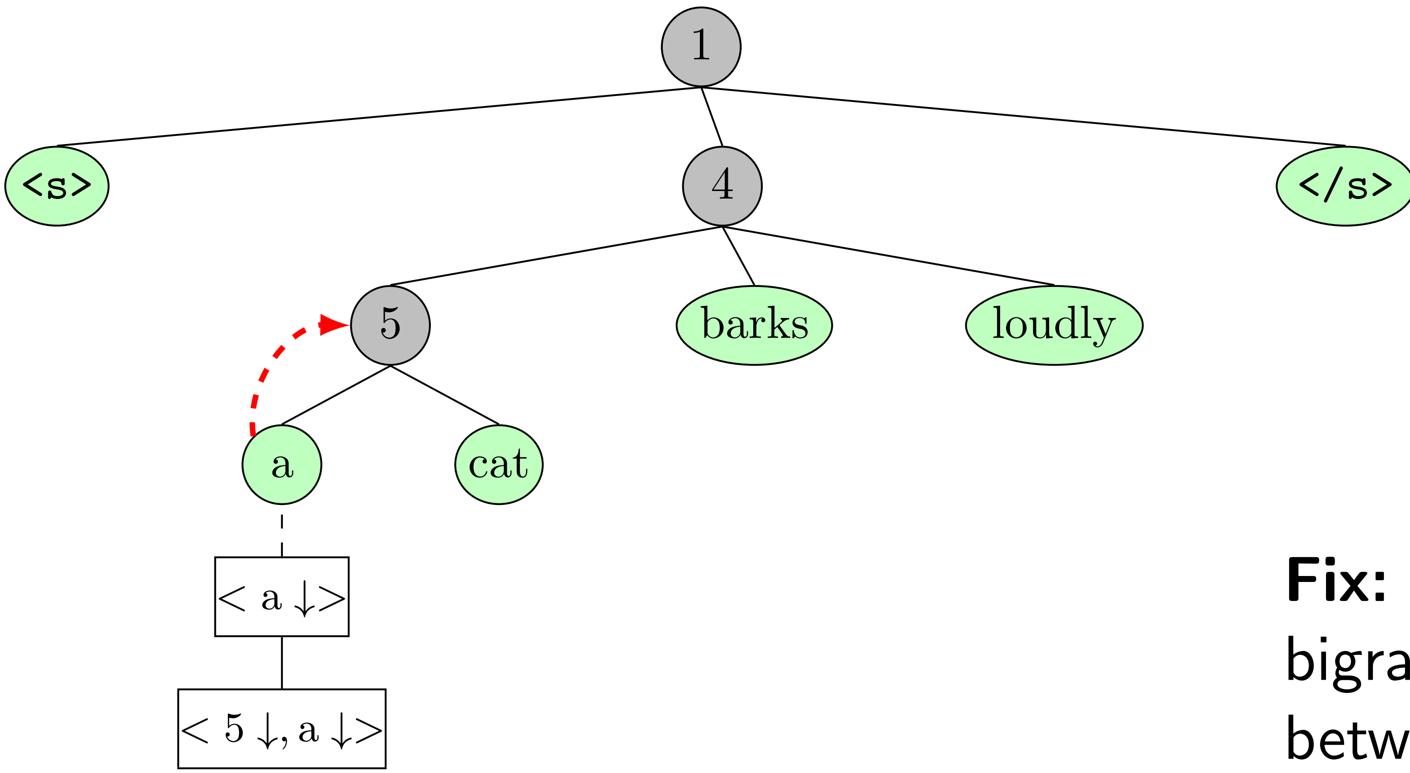
New Constraints: Paths



Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker $\langle 5 \downarrow, 10 \downarrow \rangle$ implies that between two word nodes, we move down from node 5 to node 10

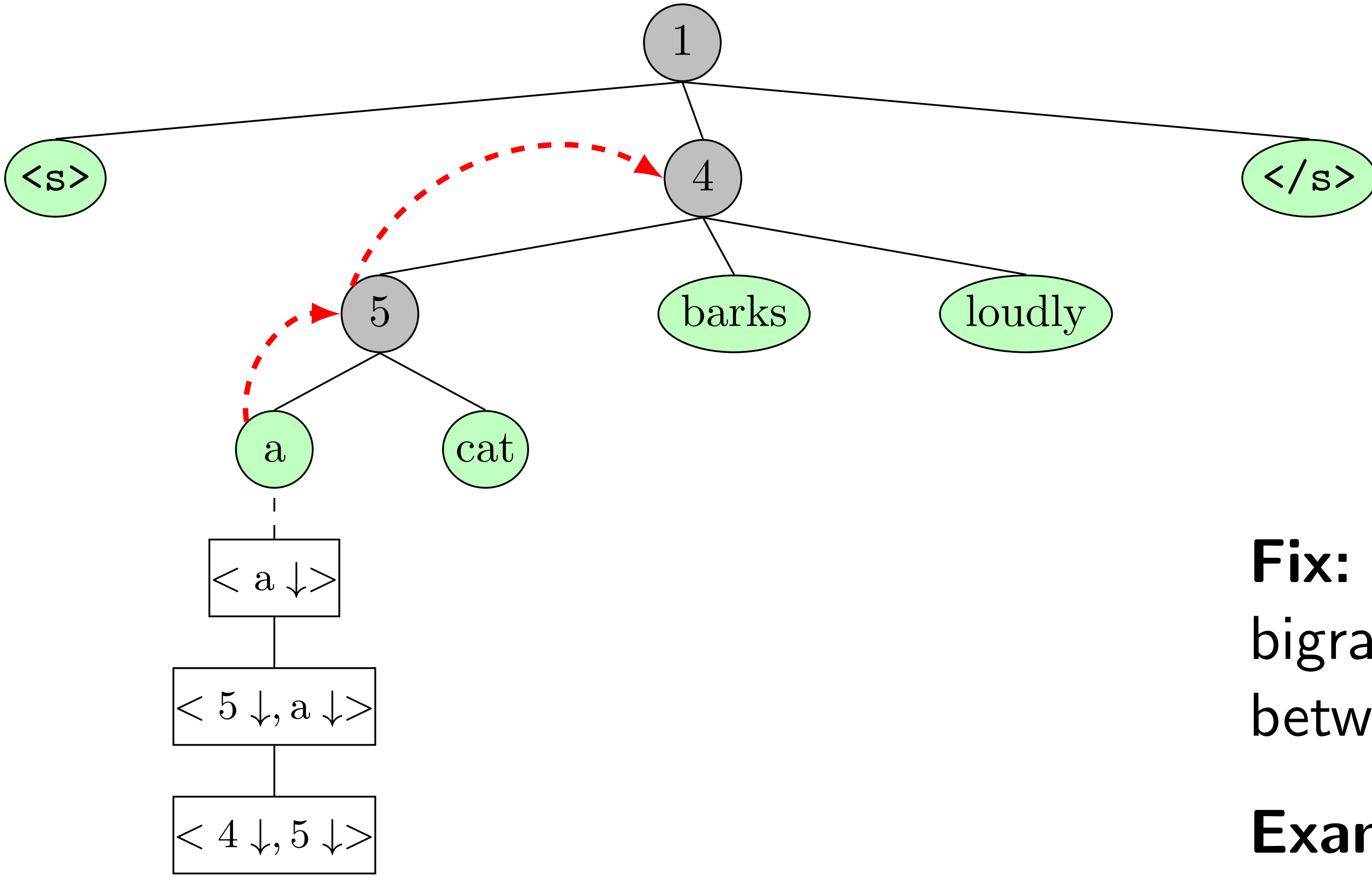
New Constraints: Paths



Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker $\langle 5 \downarrow, 10 \downarrow\rangle$ implies that between two word nodes, we move down from node 5 to node 10

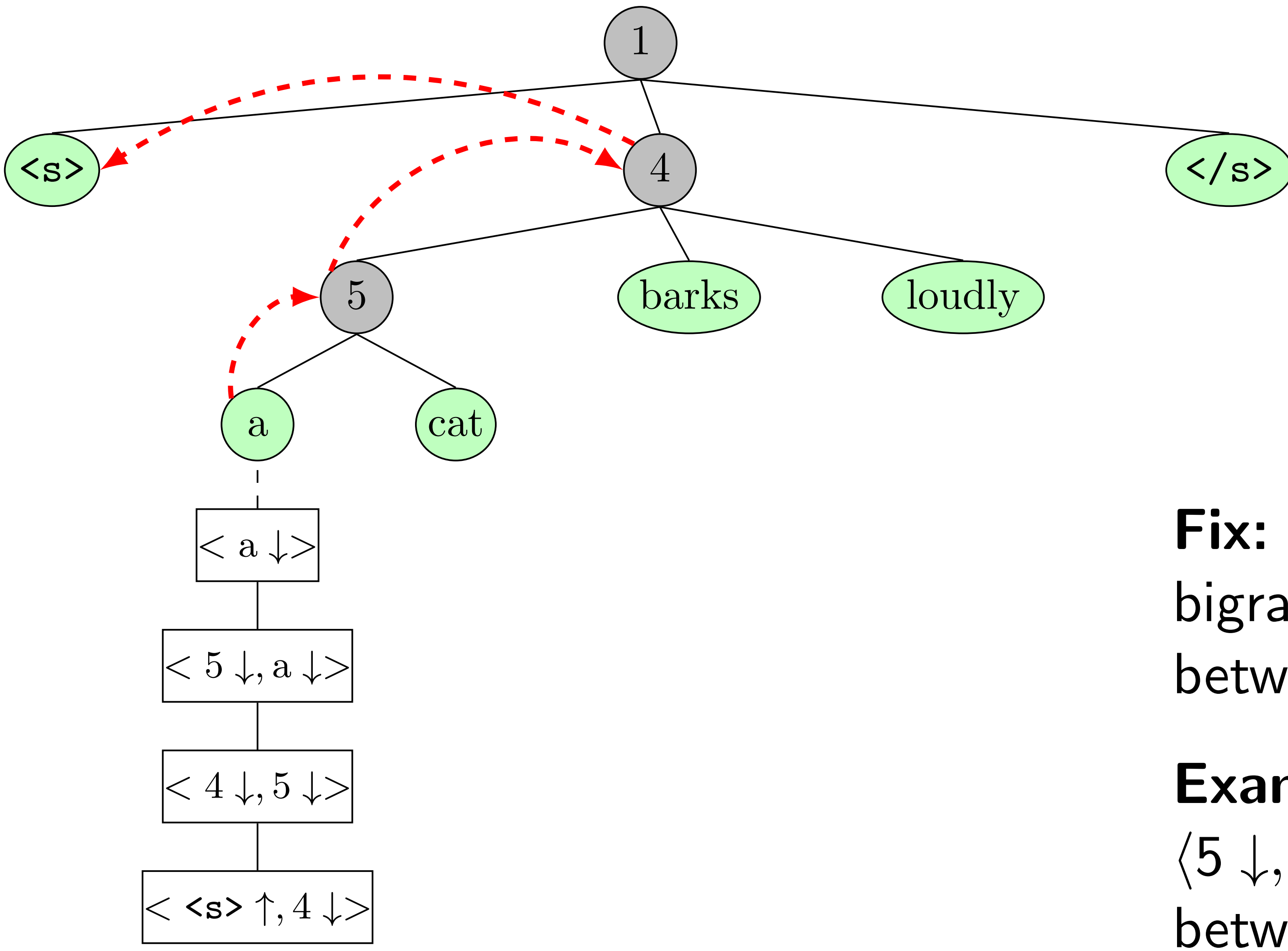
New Constraints: Paths



Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker $\langle 5 \downarrow, 10 \downarrow \rangle$ implies that between two word nodes, we move down from node 5 to node 10

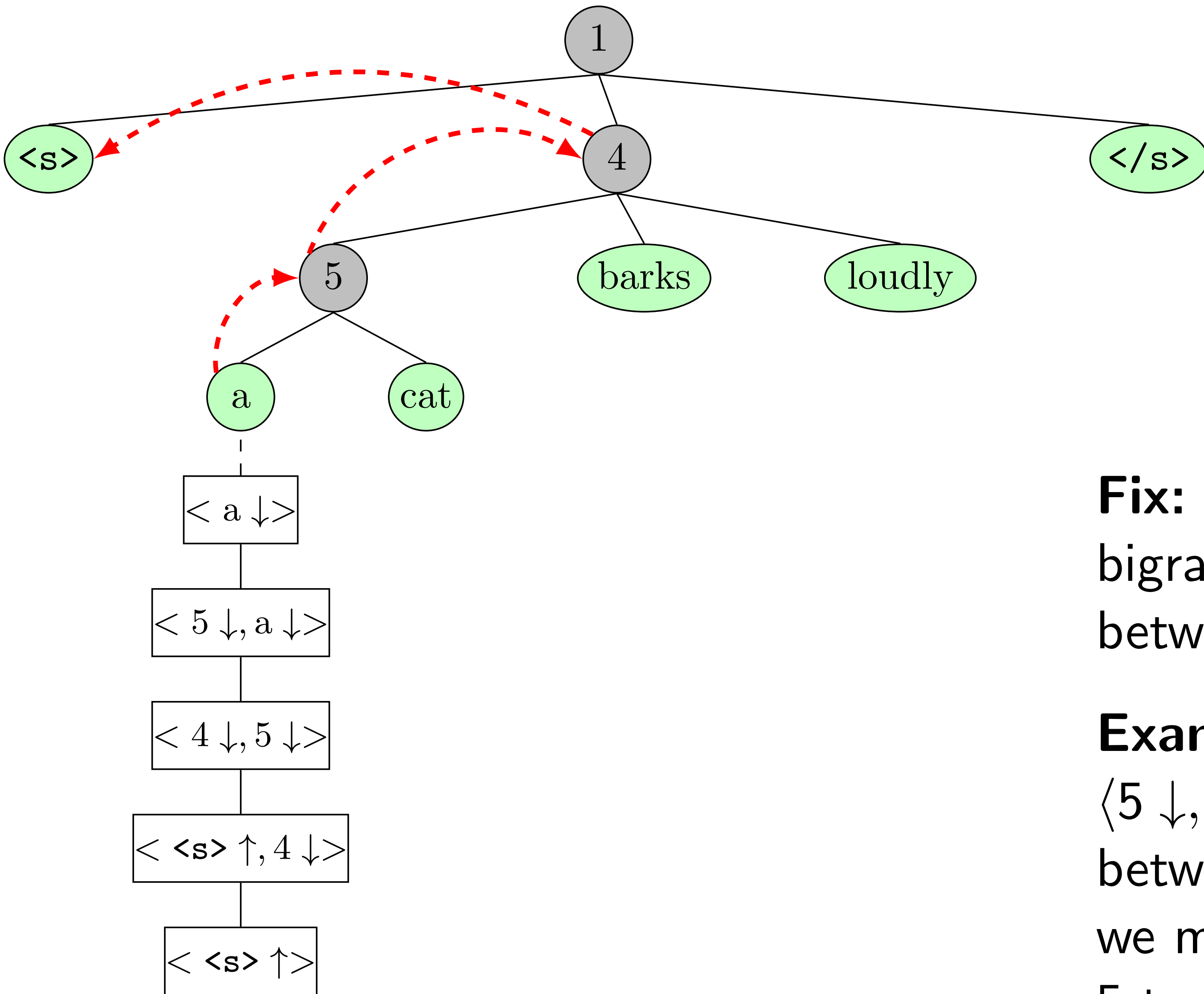
New Constraints: Paths



Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker $\langle 5 \downarrow, 10 \downarrow \rangle$ implies that between two word nodes, we move down from node 5 to node 10

New Constraints: Paths

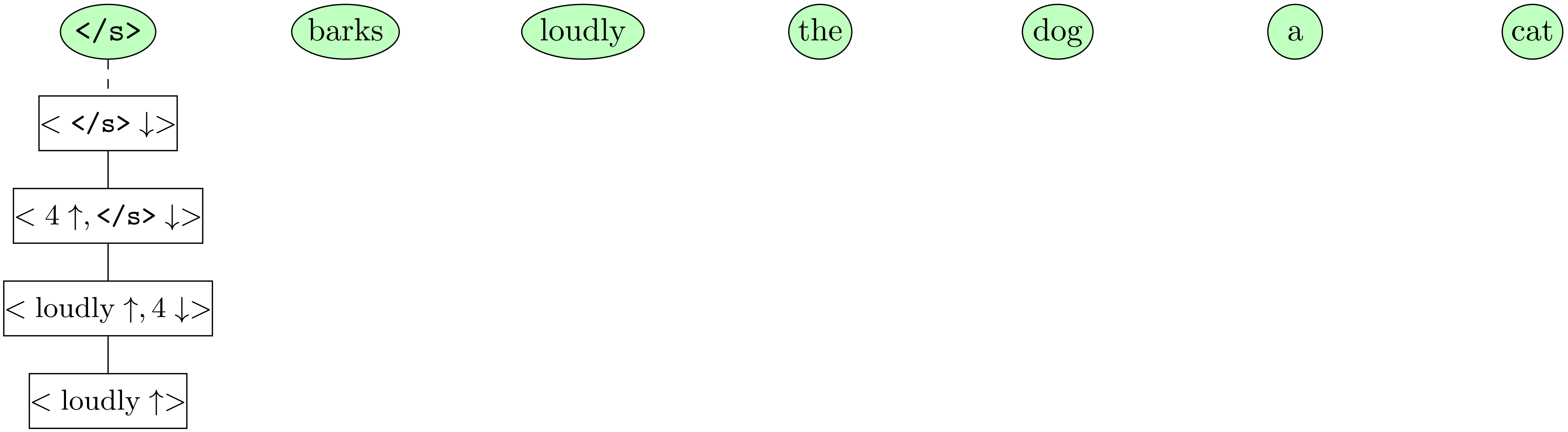


Fix: In addition to bigrams, consider paths between terminal nodes

Example: Path marker $\langle 5 \downarrow, 10 \downarrow\rangle$ implies that between two word nodes, we move down from node 5 to node 10

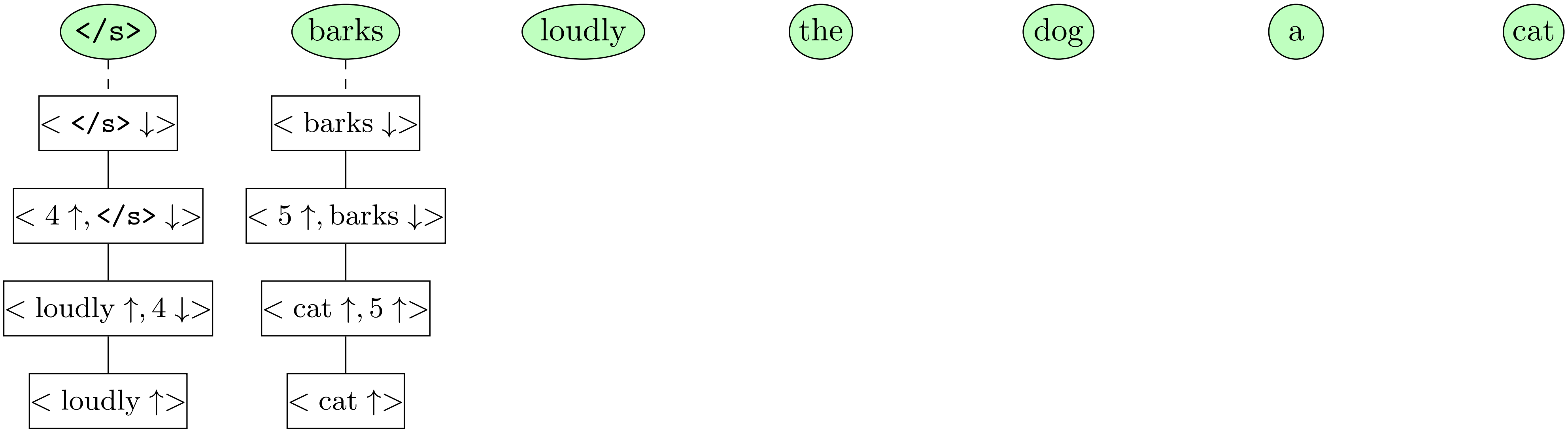
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



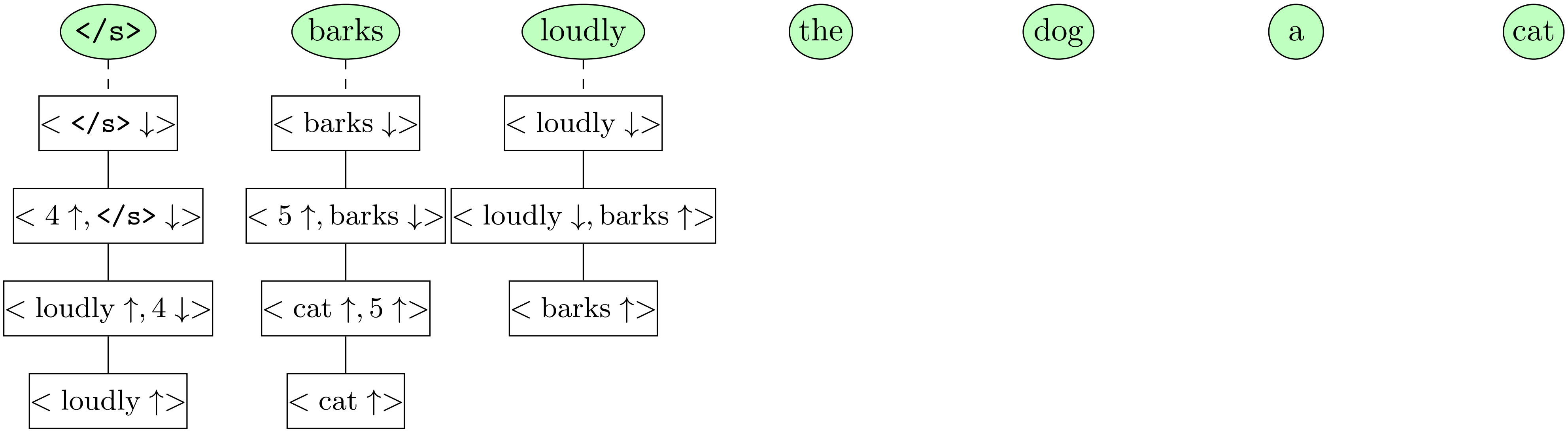
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



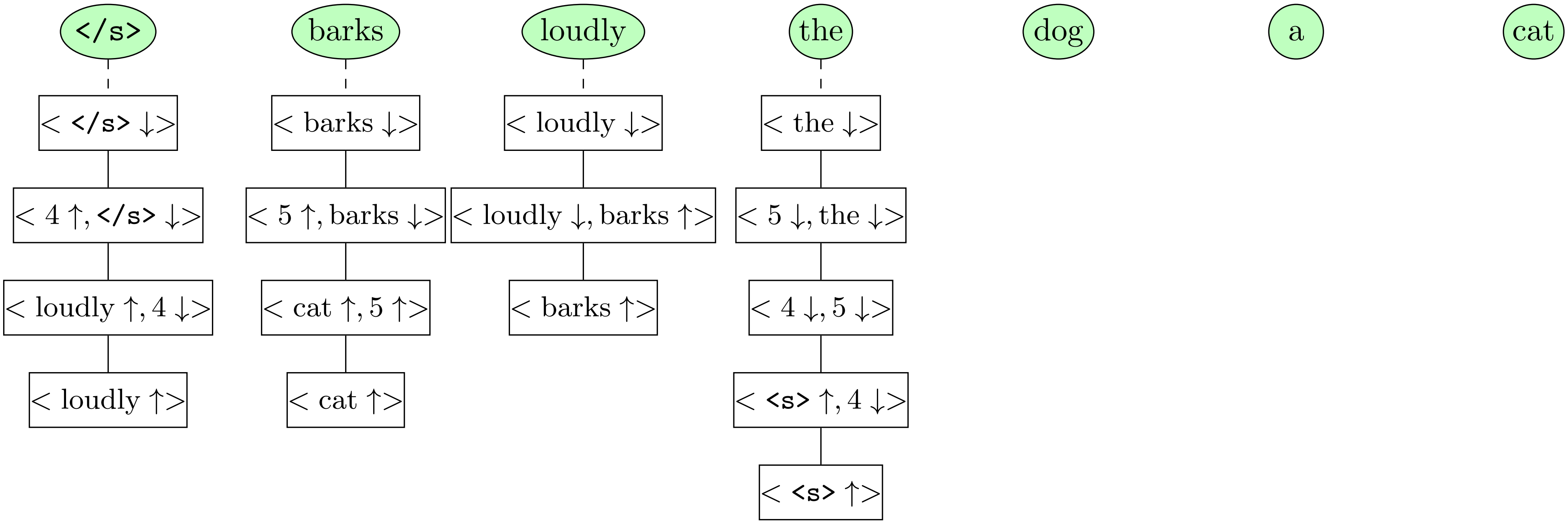
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



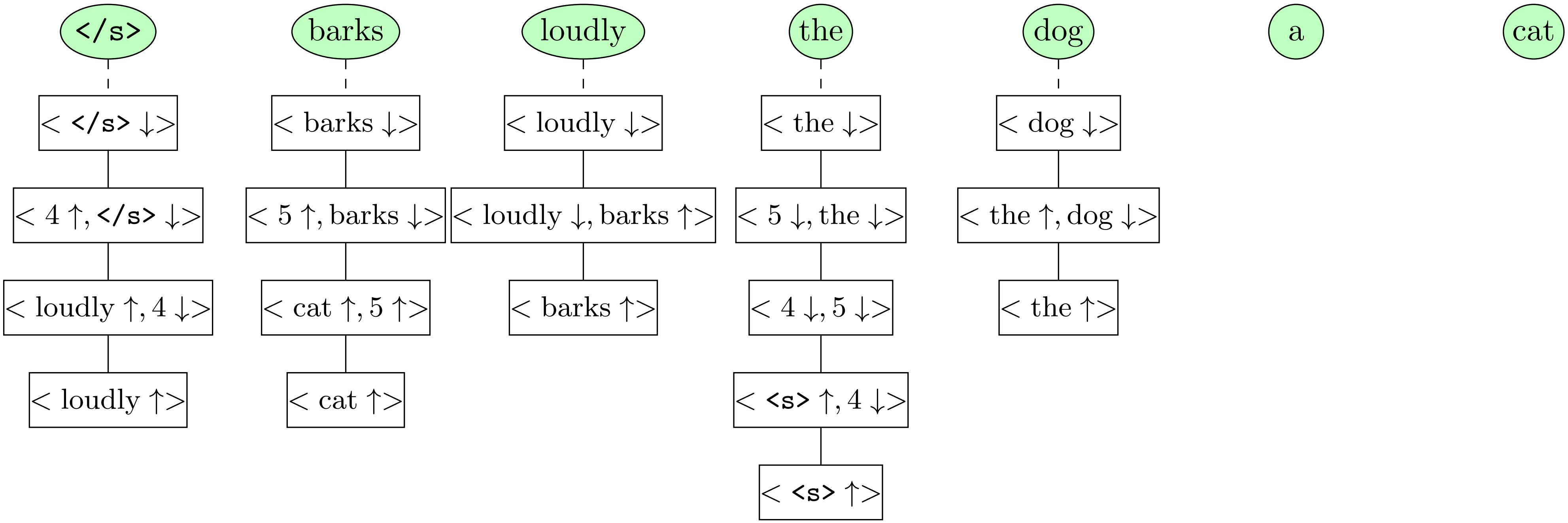
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



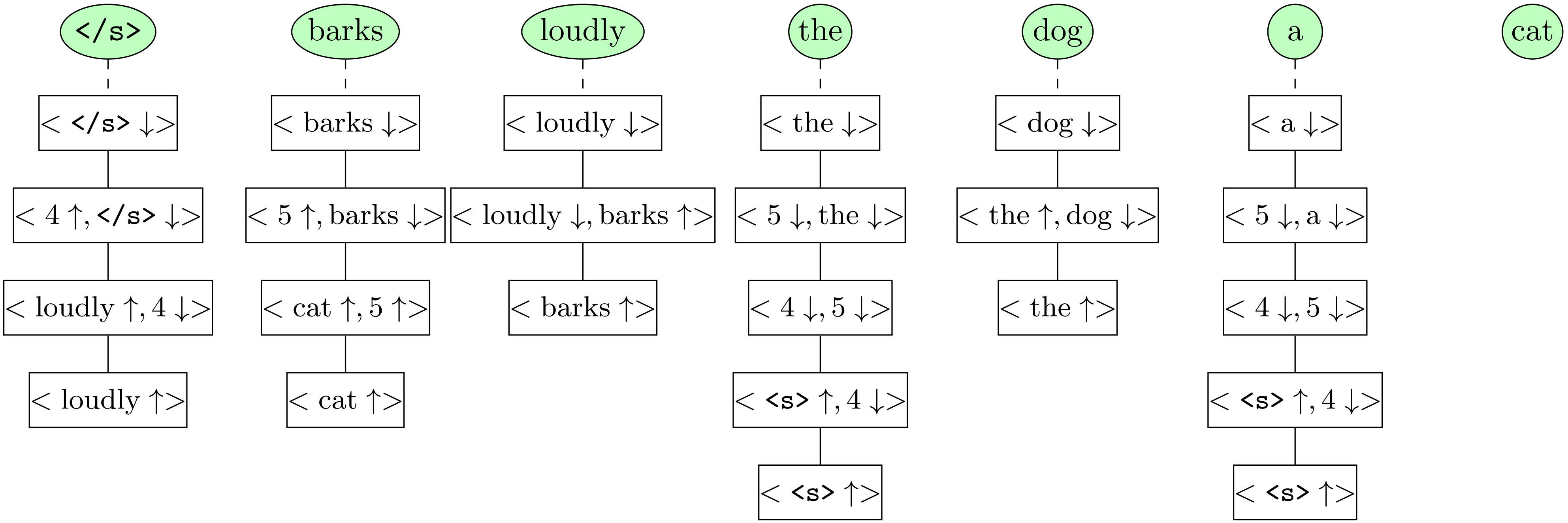
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



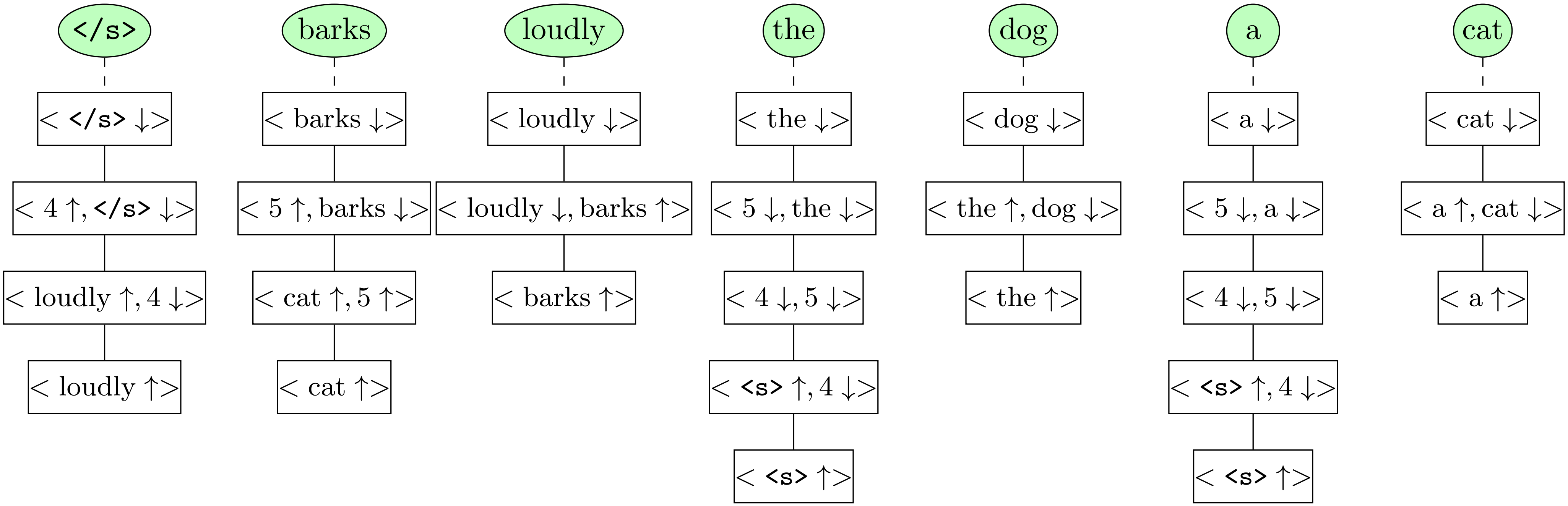
Greedy Language Model with Paths

Step 1. Greedily choose best path each word



Greedy Language Model with Paths

Step 1. Greedily choose best path each word

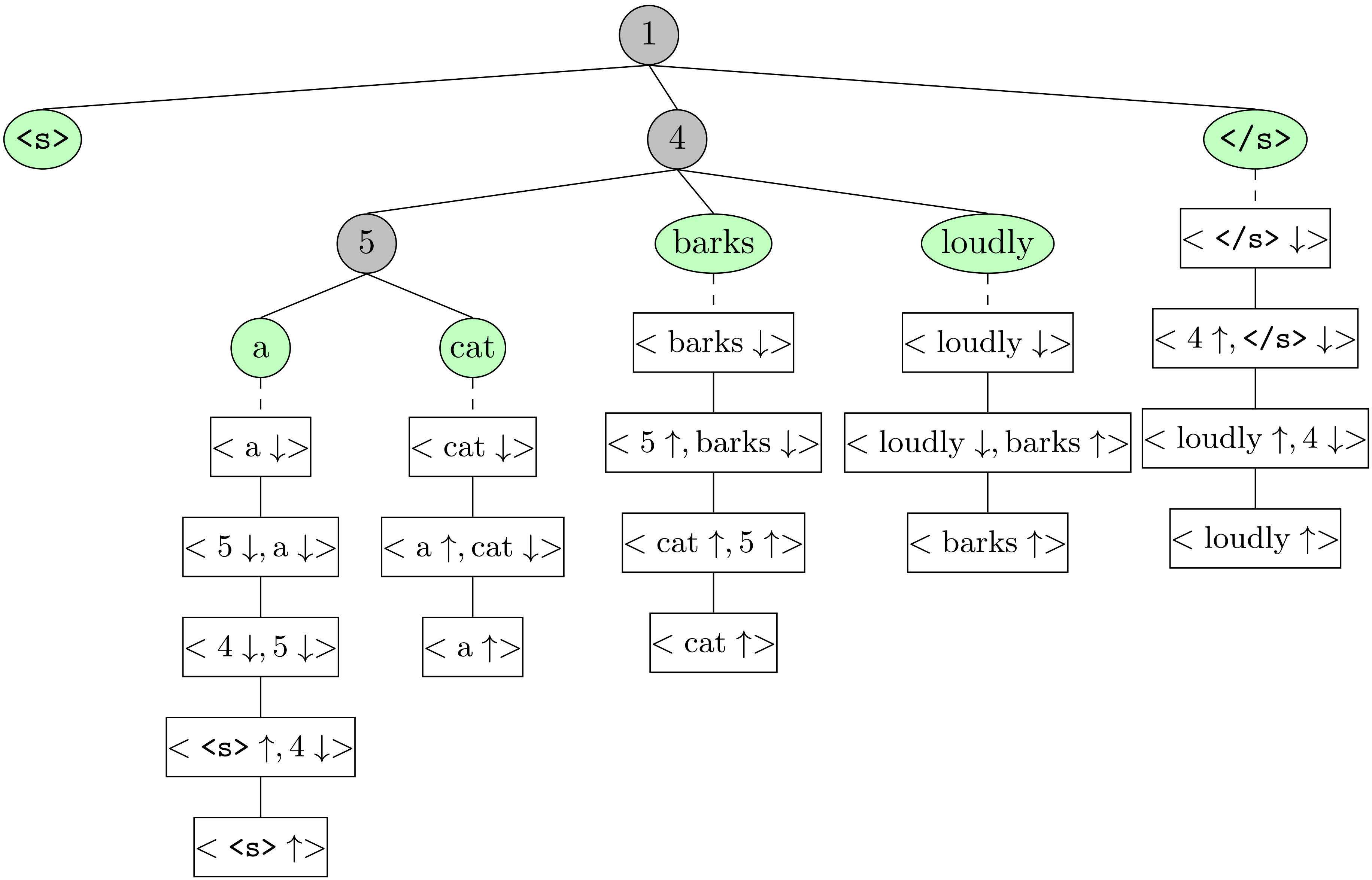


Greedy Language Model with Paths (continued)

Step 2. Find the best derivation over these elements

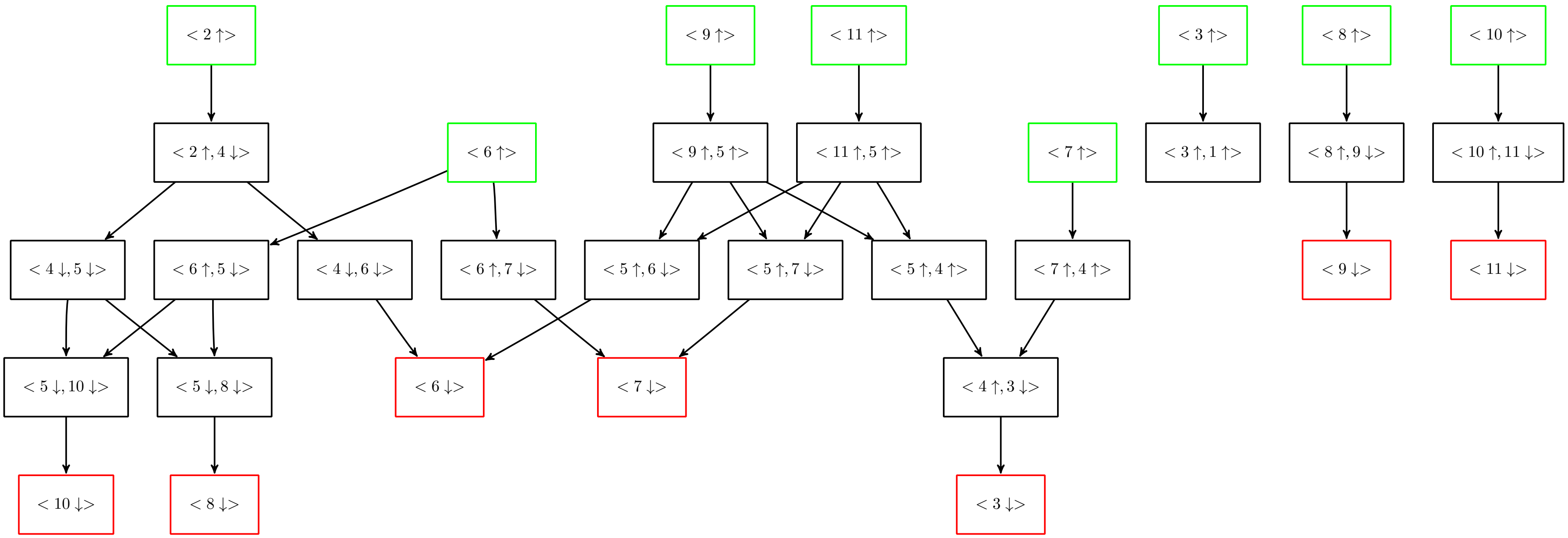
Greedy Language Model with Paths (continued)

Step 2. Find the best derivation over these elements



Efficiently Calculating Best Paths

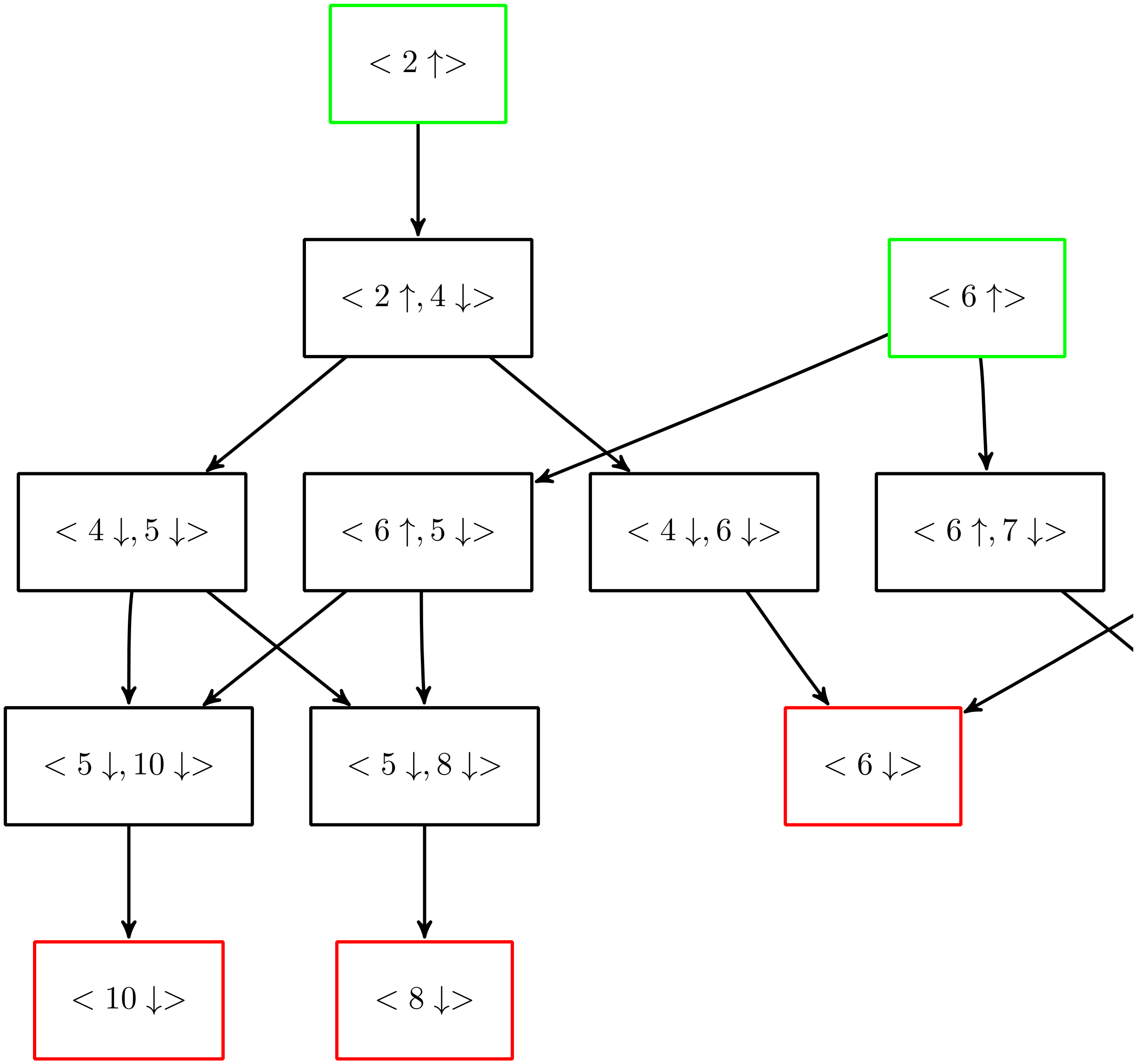
There are too many paths to compute argmax directly, but we can compactly represent all paths as a graph



Graph is linear in the size of the grammar

- Green nodes represent leaving a word
- Red nodes represent entering a word
- Black nodes are intermediate paths

Best Paths



Goal: Find the best path between all word nodes (green and red)

Method: Run all-pairs shortest path to find best paths

Full Algorithm

Algorithm is very similar to simple bigram case. Penalty weights are associated with nodes in the graph instead of just bigram words

Theorem

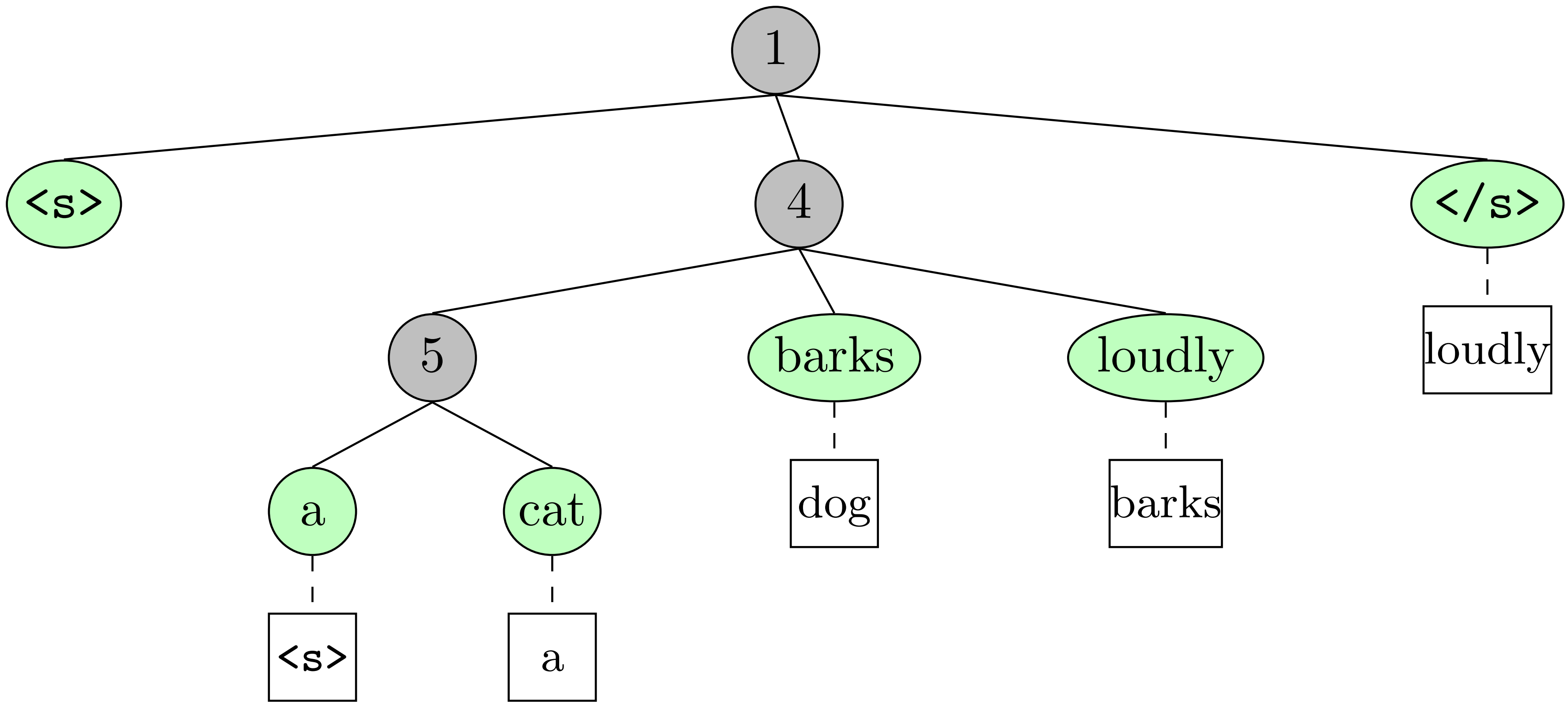
If at any iteration the greedy paths agree with the derivation, then $(y^{(k)})$ is the global optimum.

But what if it does not find the global optimum?

Convergence

The algorithm is not guaranteed to converge

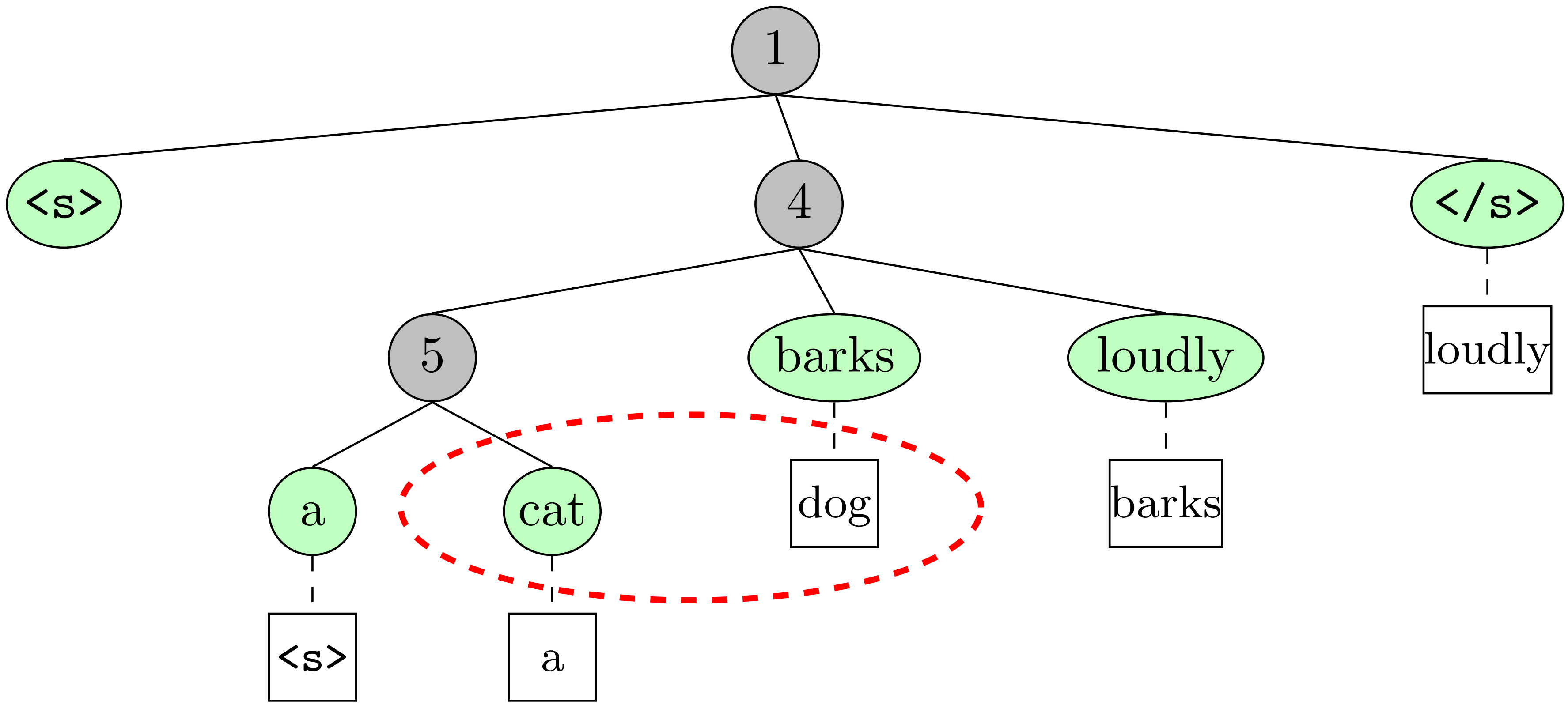
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

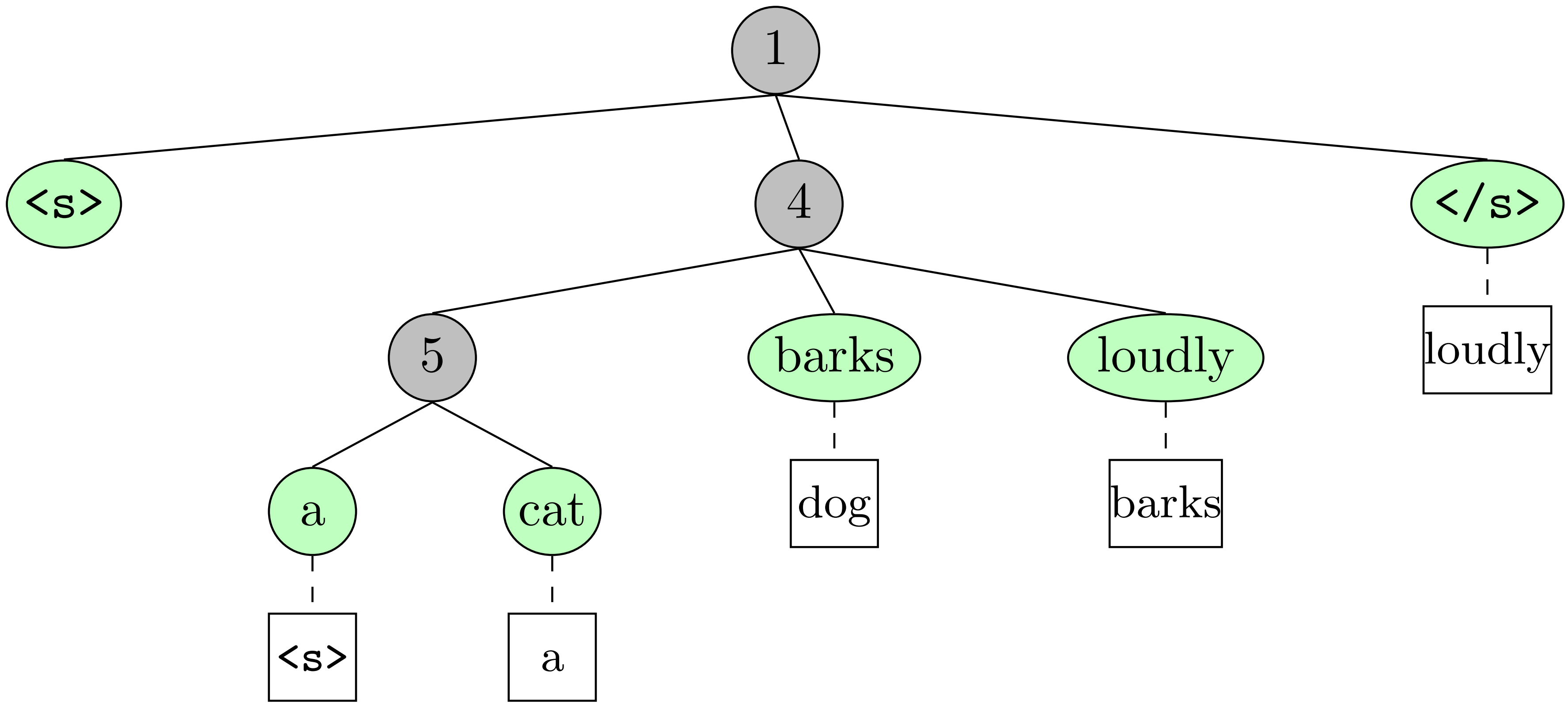
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

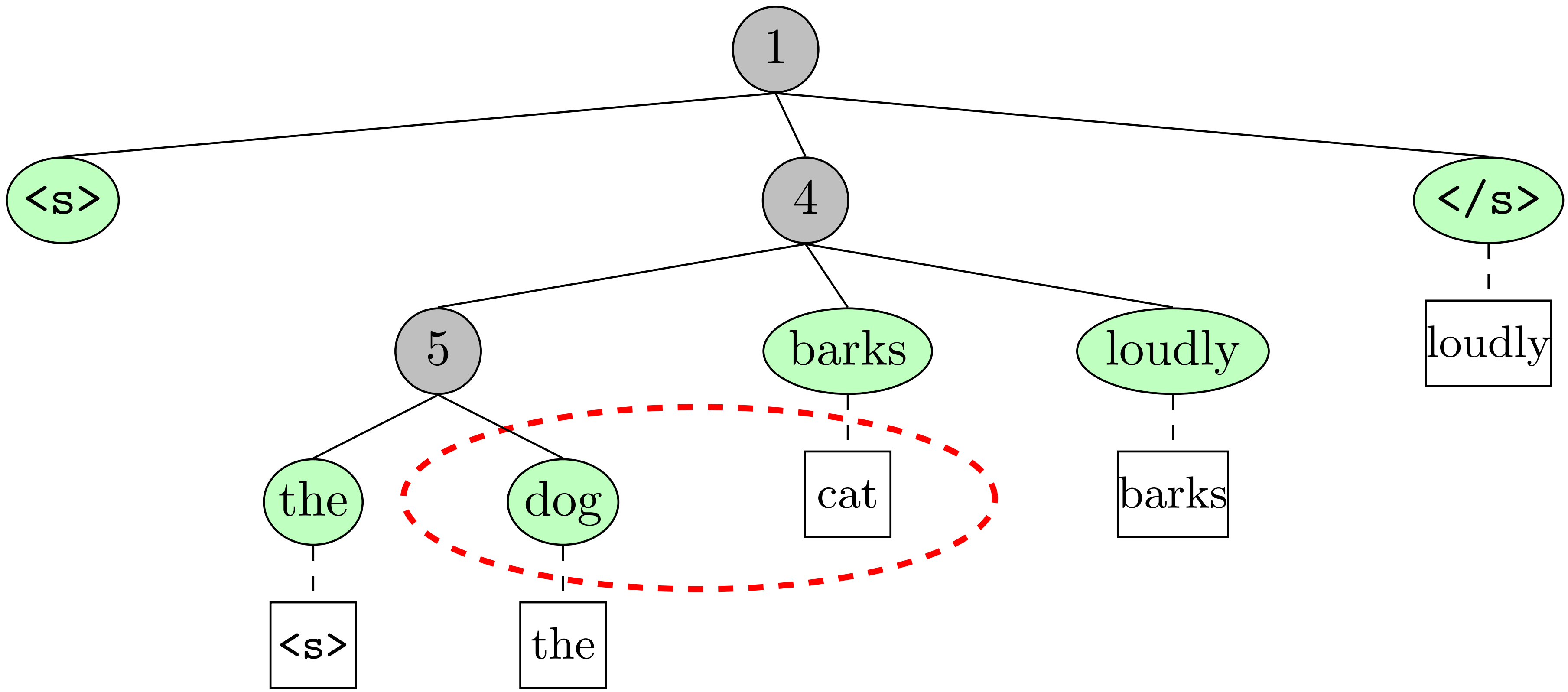
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

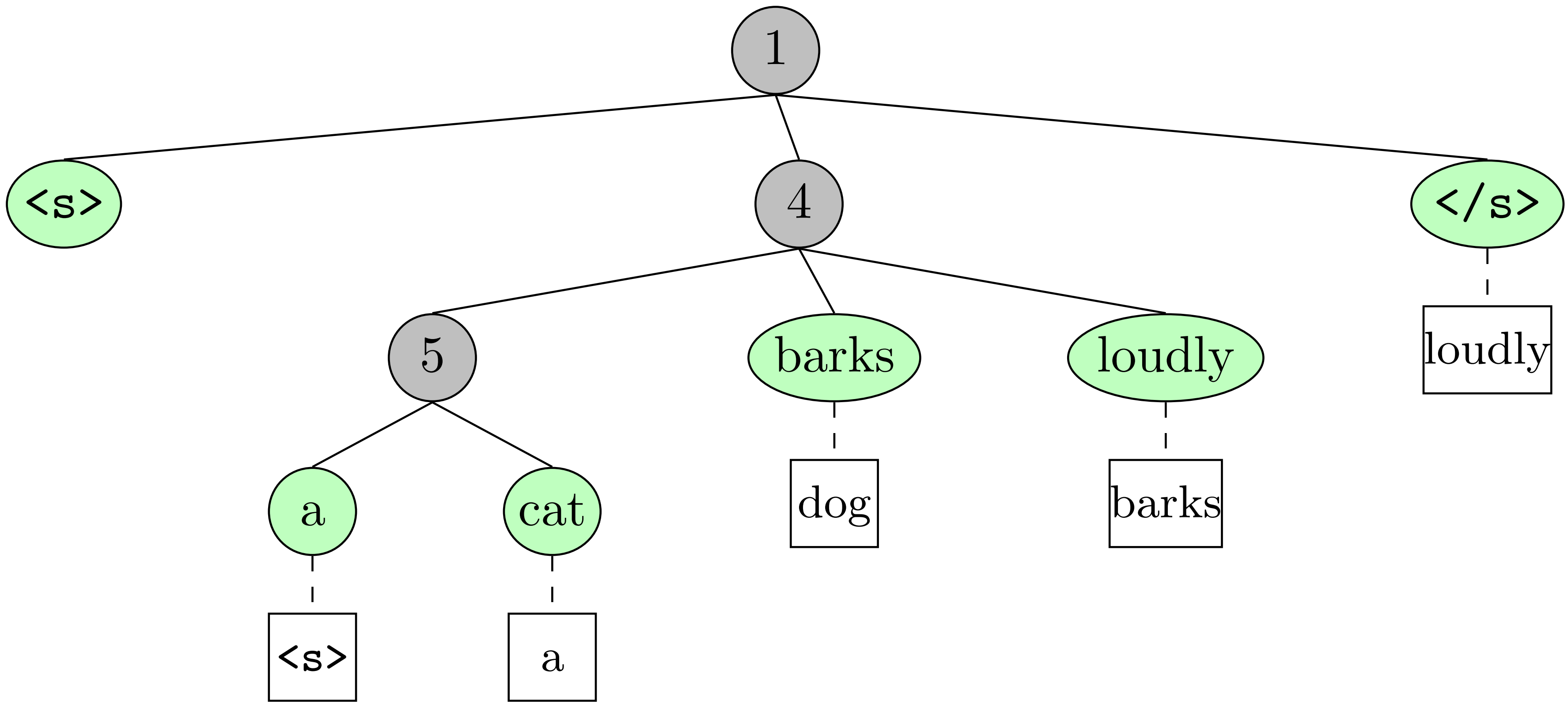
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

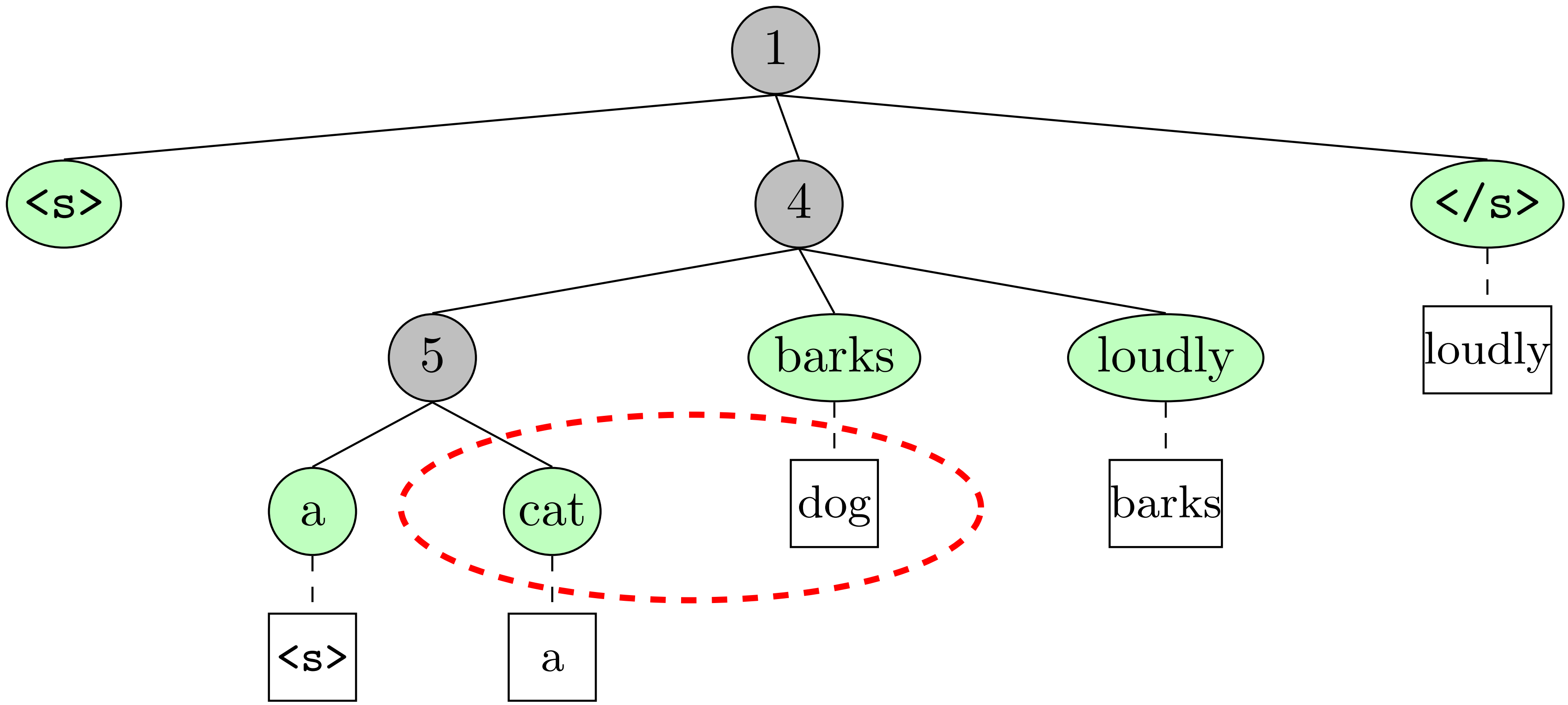
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

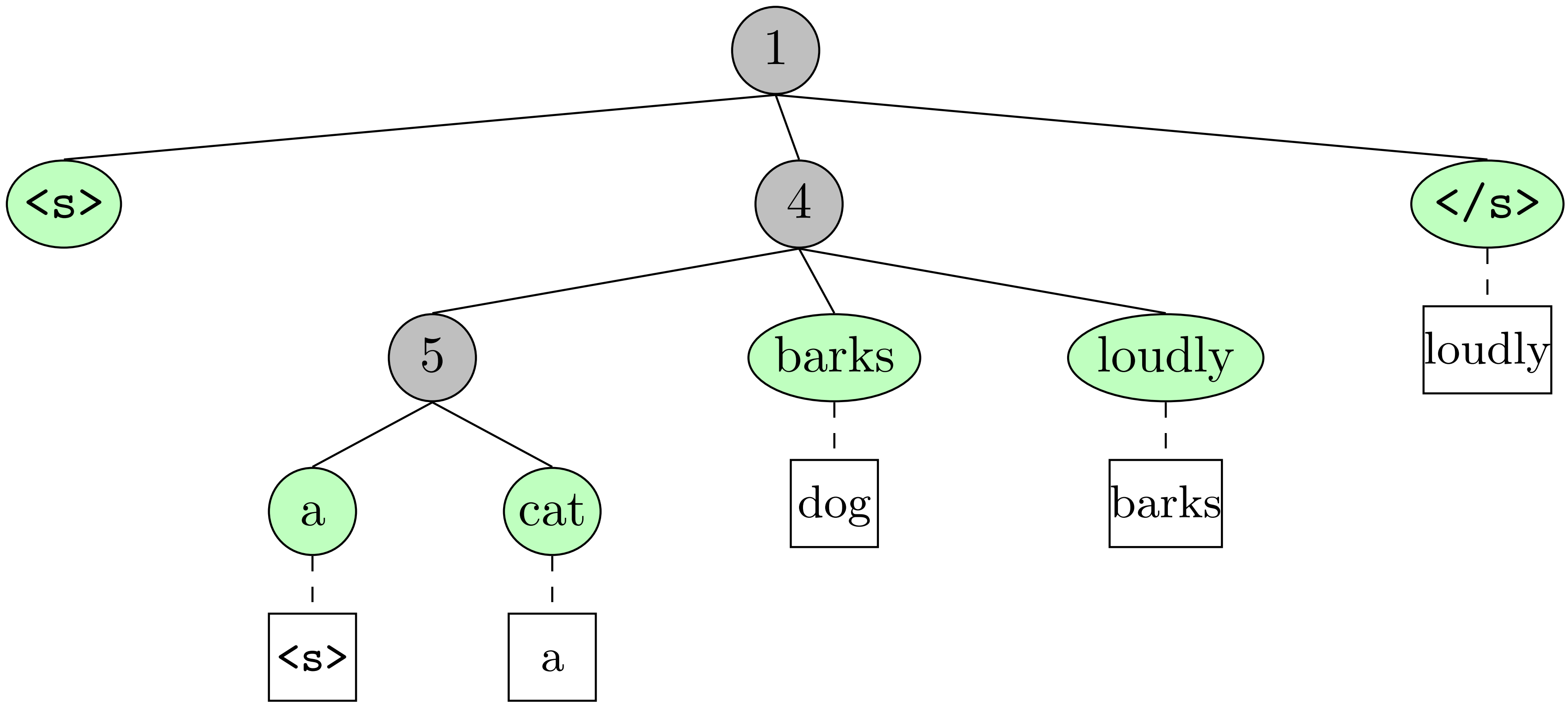
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

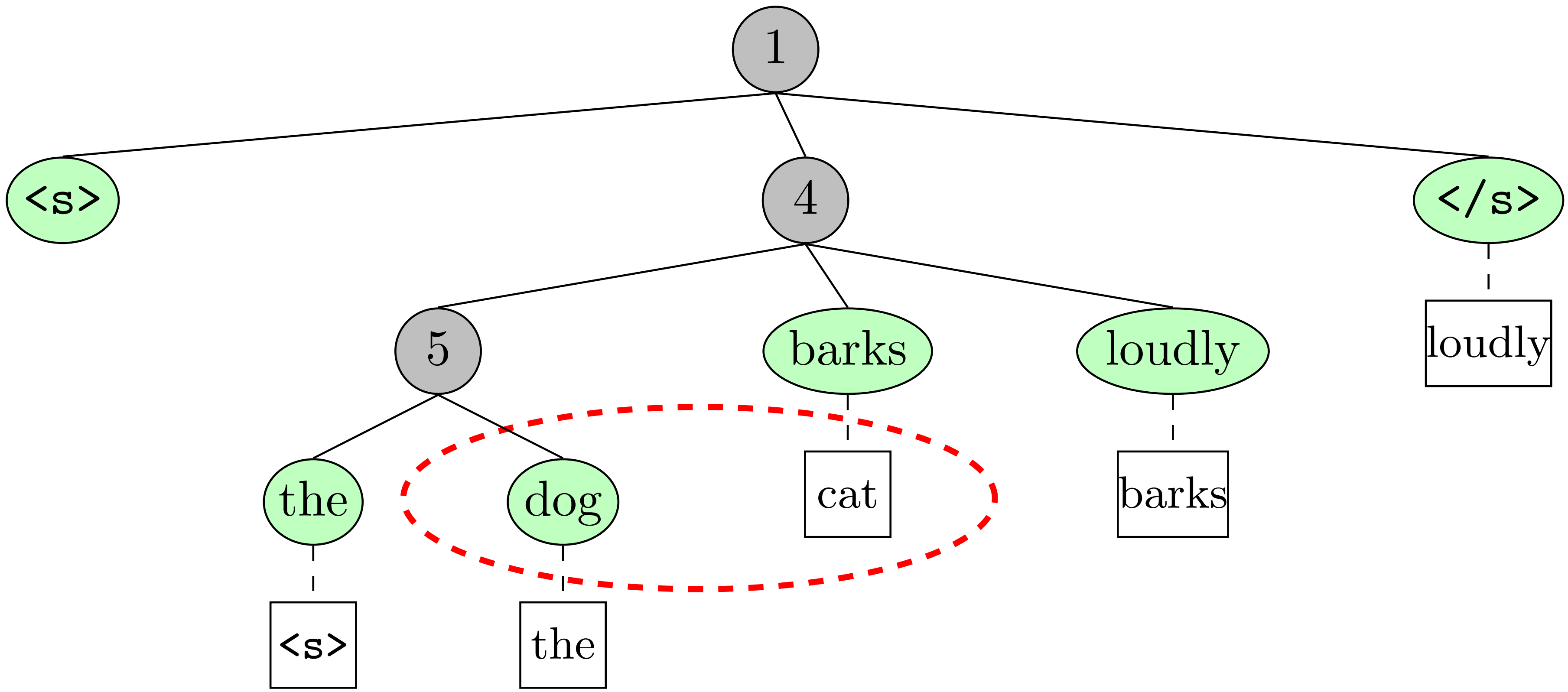
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

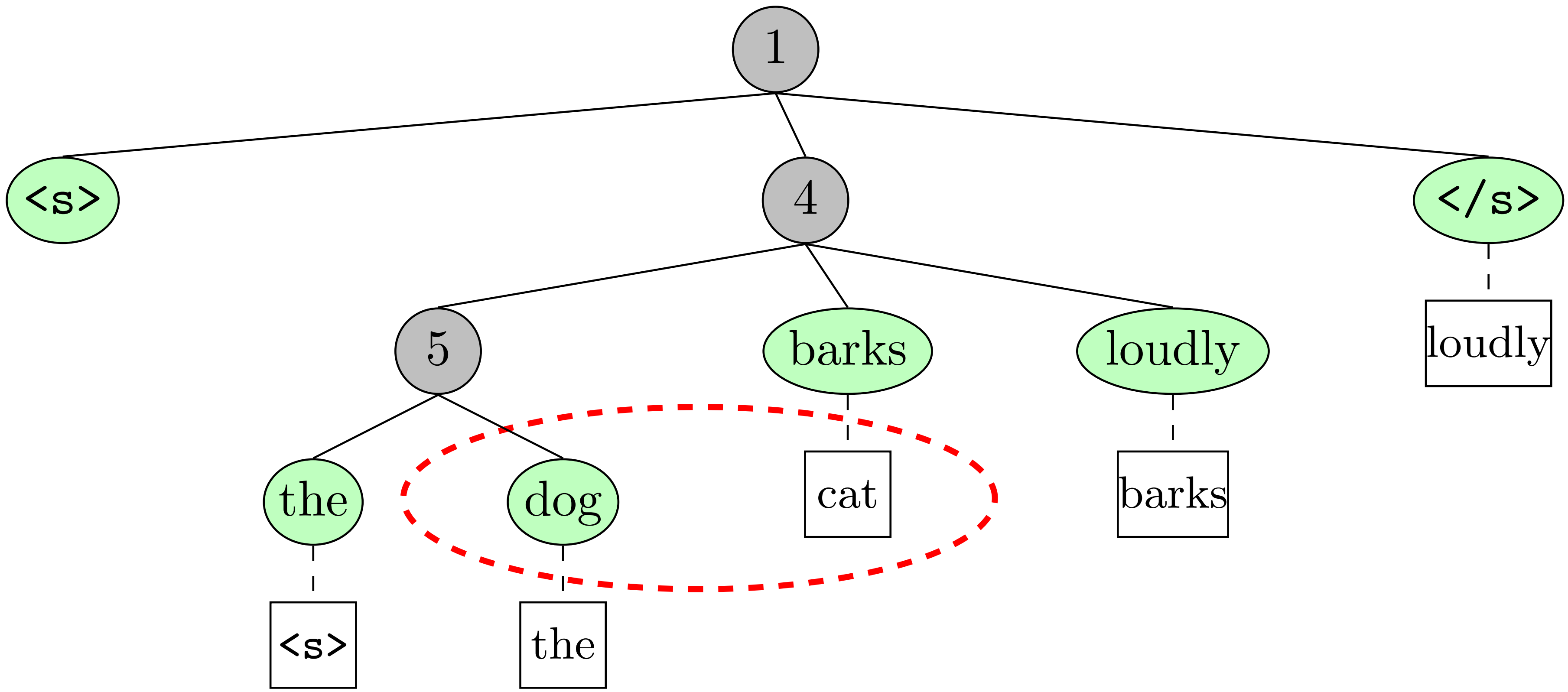
May get stuck between solutions.



Convergence

The algorithm is not guaranteed to converge

May get stuck between solutions.



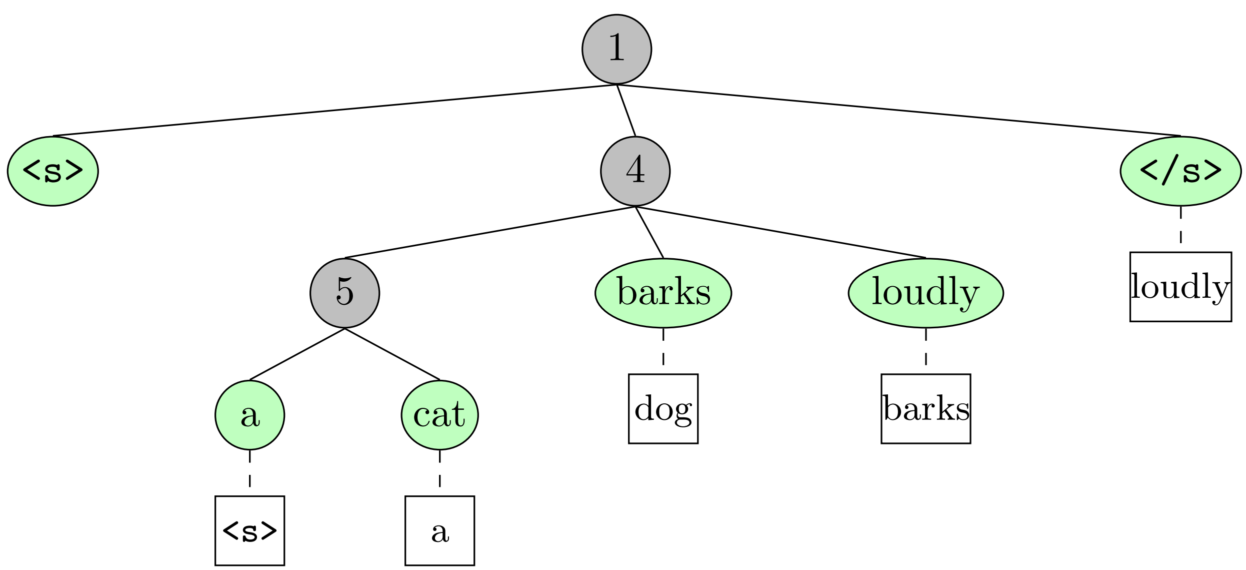
Can fix this by incrementally adding constraints to the problem

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

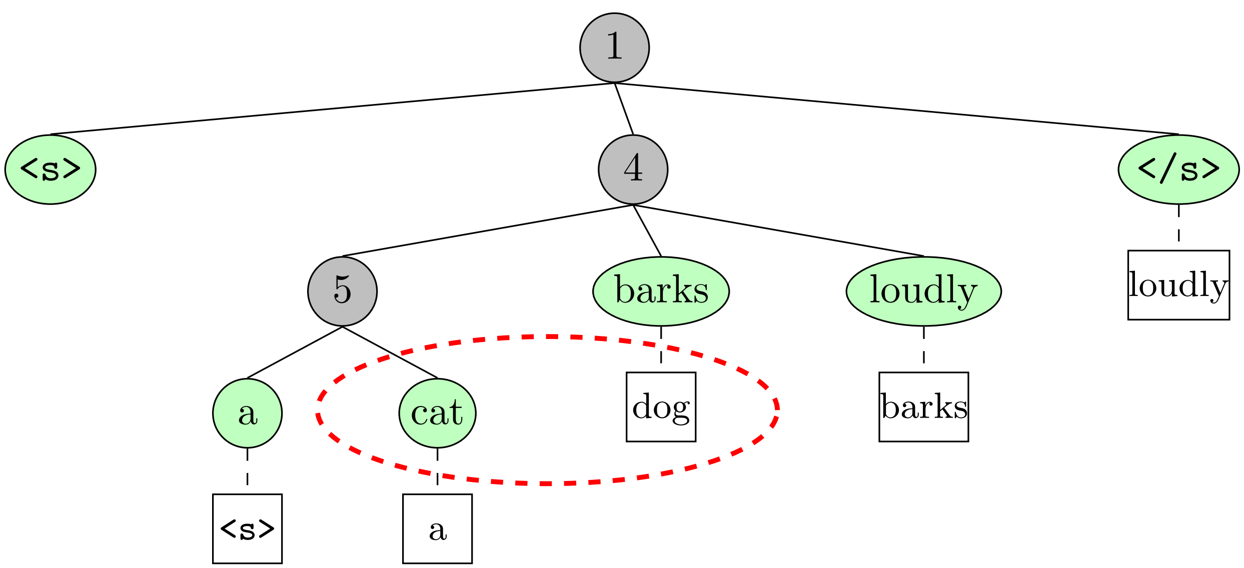
A = {2,6,7,8,9,10,11}
B = {}

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

$A = \{2,6,7,8,9,10,11\}$

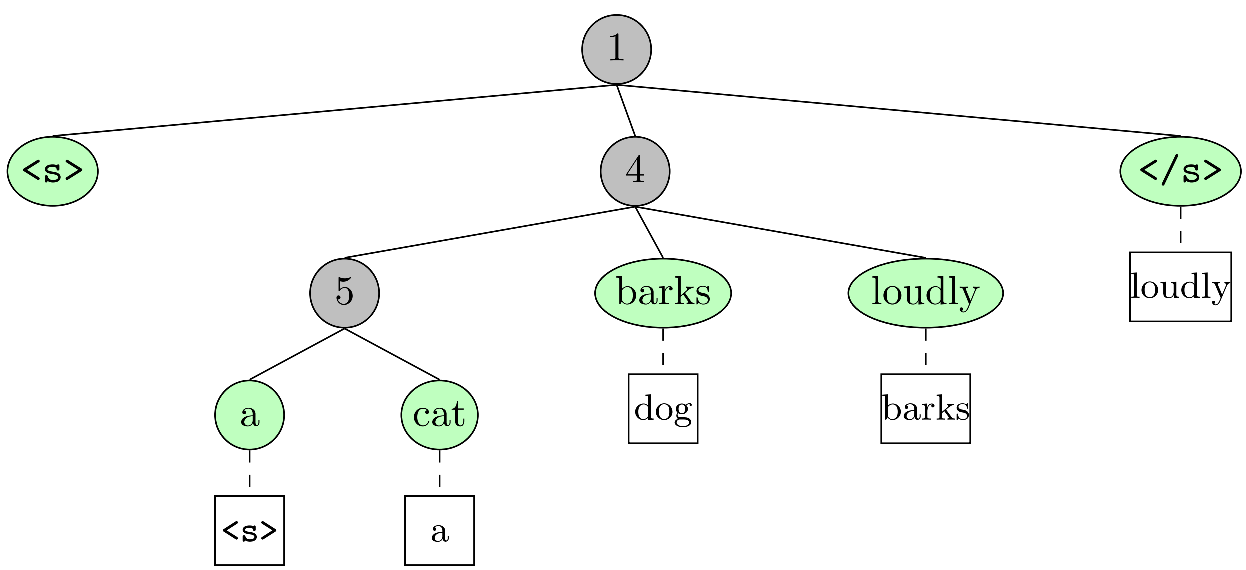
$B = \{\}$

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

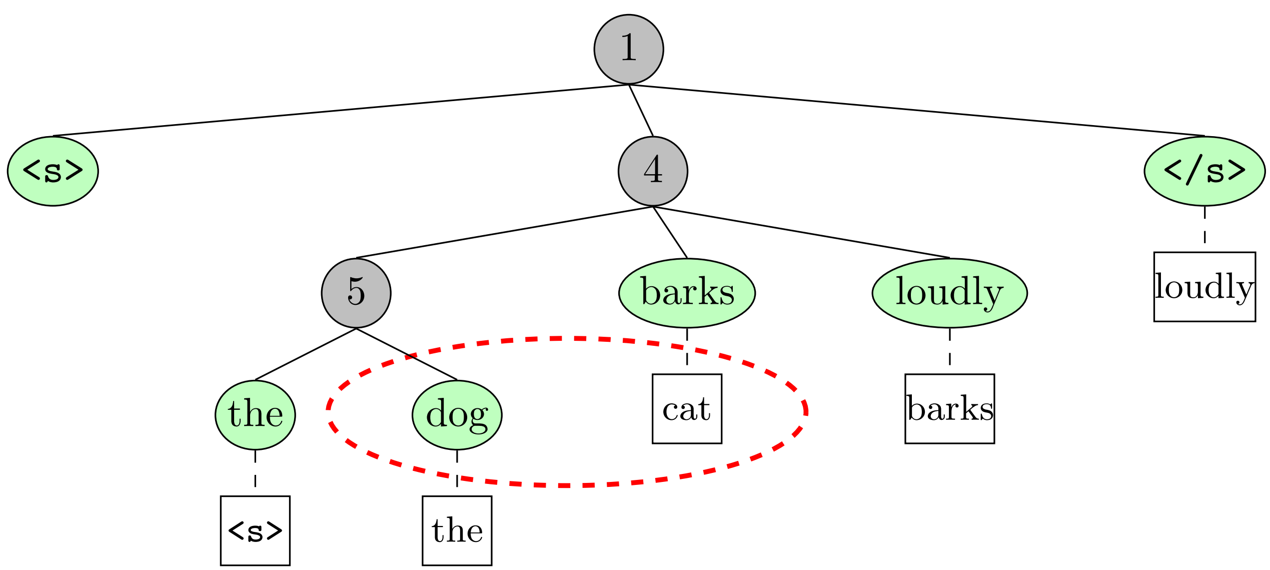
A = {2,6,7,8,9,10,11}
B = {}

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

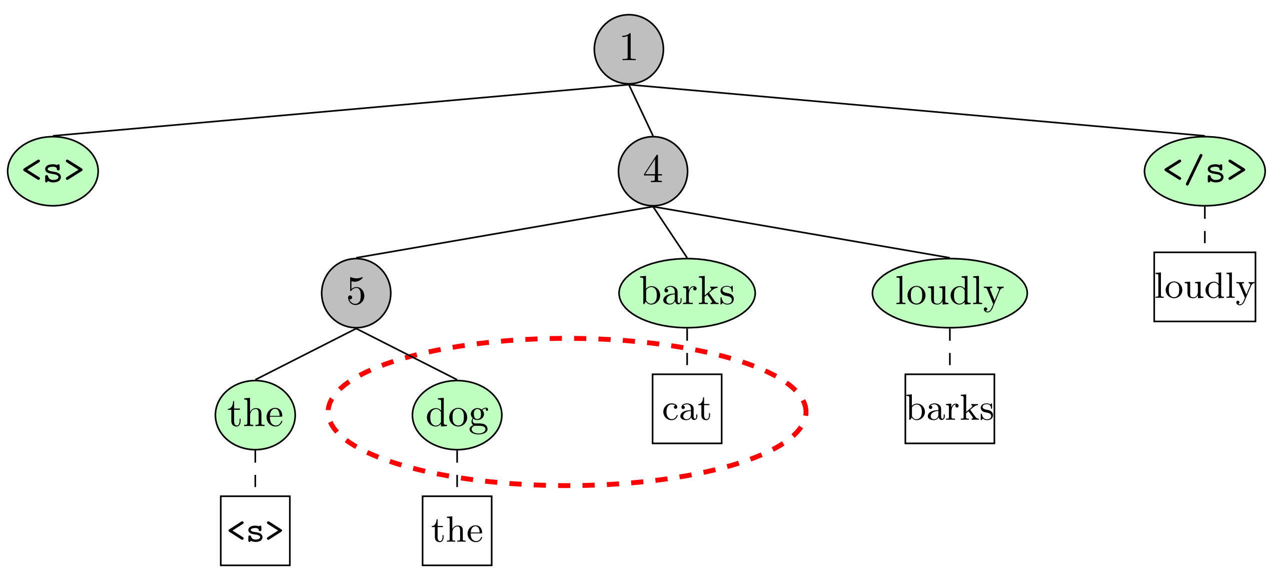
A = {2,6,7,8,9,10,11}
B = {}

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

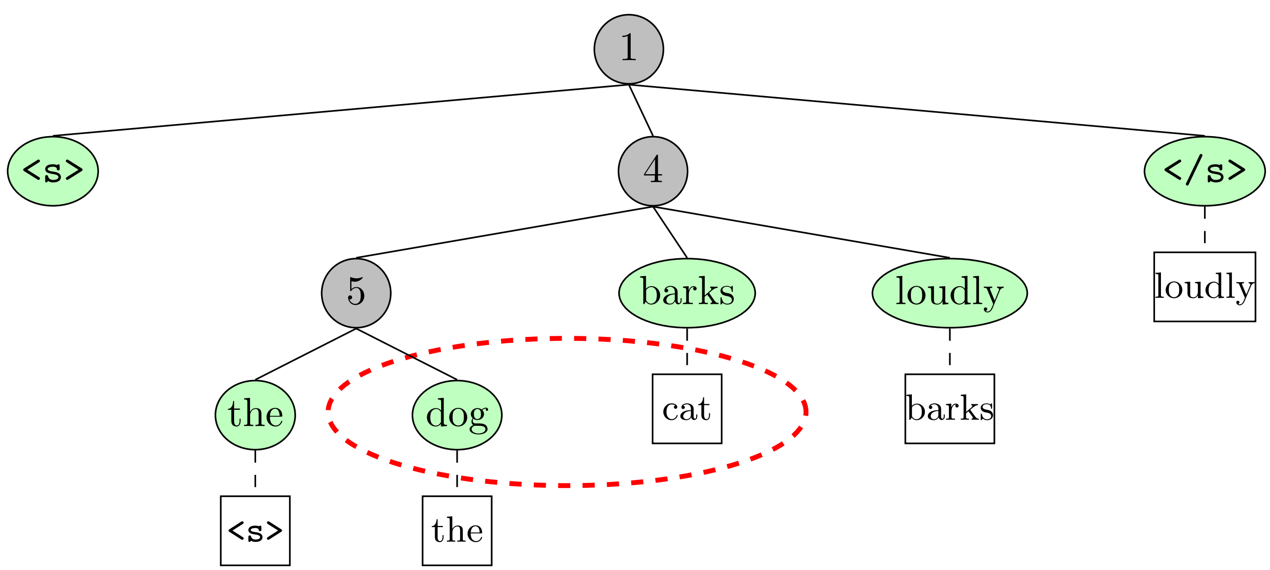
A = {2,6,7,8,9,10}
B = {11}

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

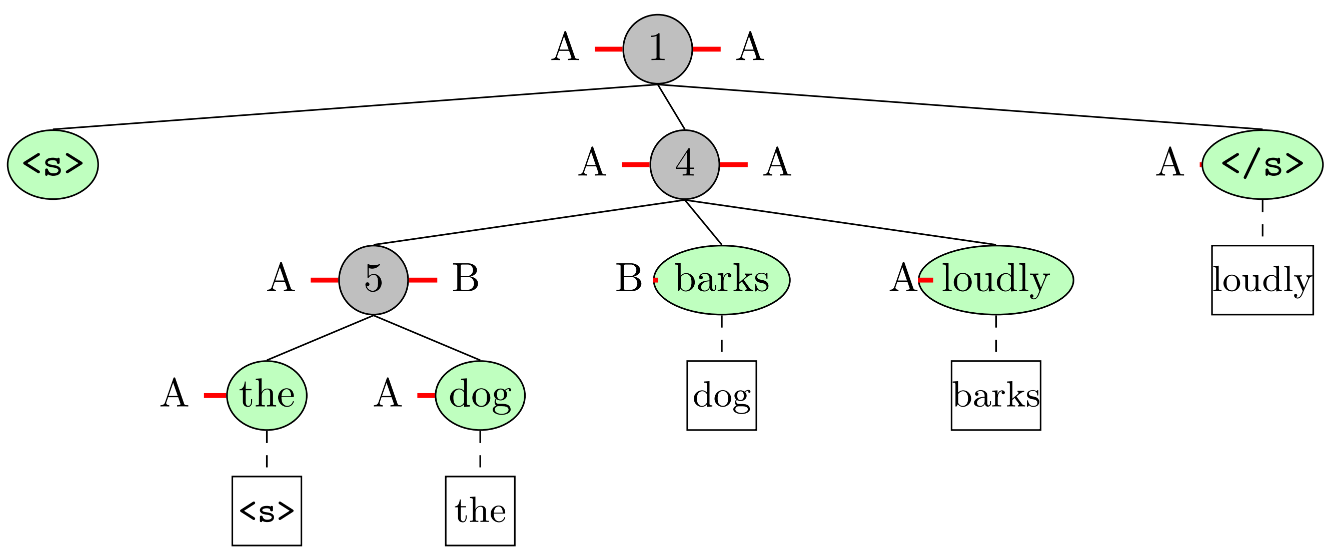
A = {2,6,7,8,9,10}
B = {11}

Tightening

Main idea: Keep partition sets (A and B). The parser treats all words in a partition as the same word.

- Initially place all words in the same partition.
- If the algorithm gets stuck, separate words that conflict
- Run the exact algorithm but only distinguish between partitions (much faster than running full exact algorithm)

Example:



Partitions

$A = \{2,6,7,8,9,10\}$

$B = \{11\}$

Experiments

Properties:

- Exactness
- Translation Speed
- Comparison to Cube Pruning

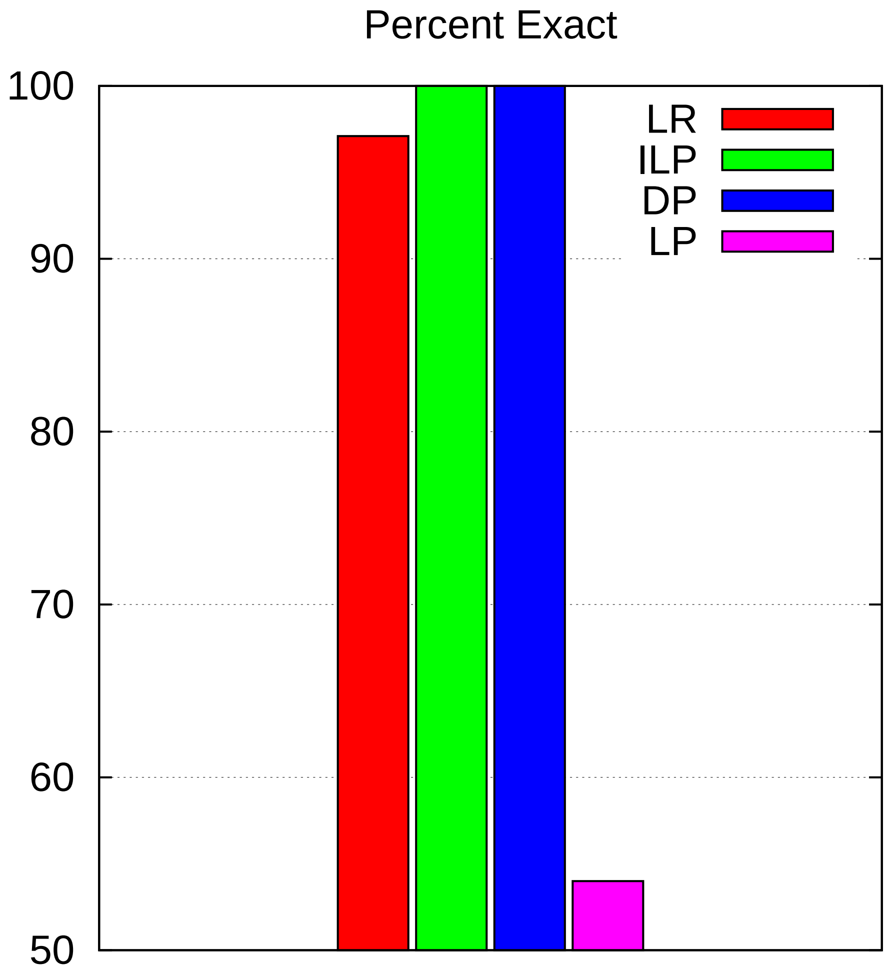
Model:

- Tree-to-String translation model (Huang and Mi, 2010)
- Trained with MERT

Experiments:

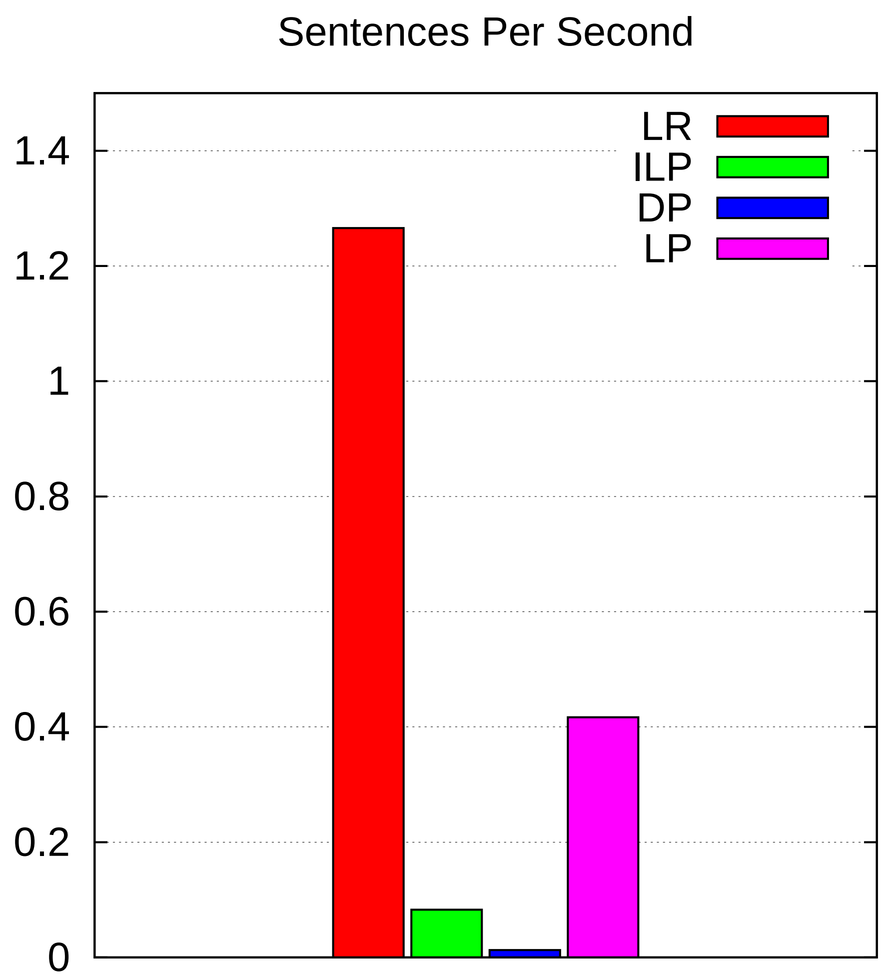
- NIST MT Evaluation Set (2008)

Exactness



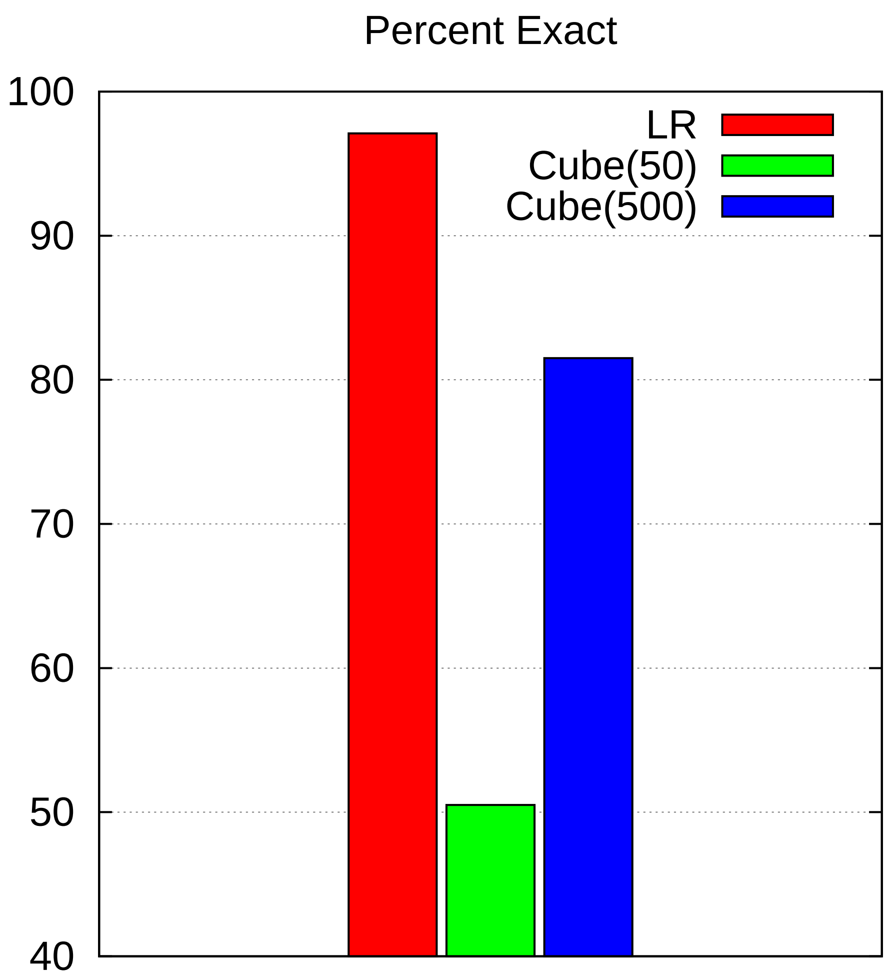
LR	Lagrangian Relaxation
ILP	Integer Linear Programming
DP	Exact Dynanic Programming
LP	Linear Programming

Median Speed



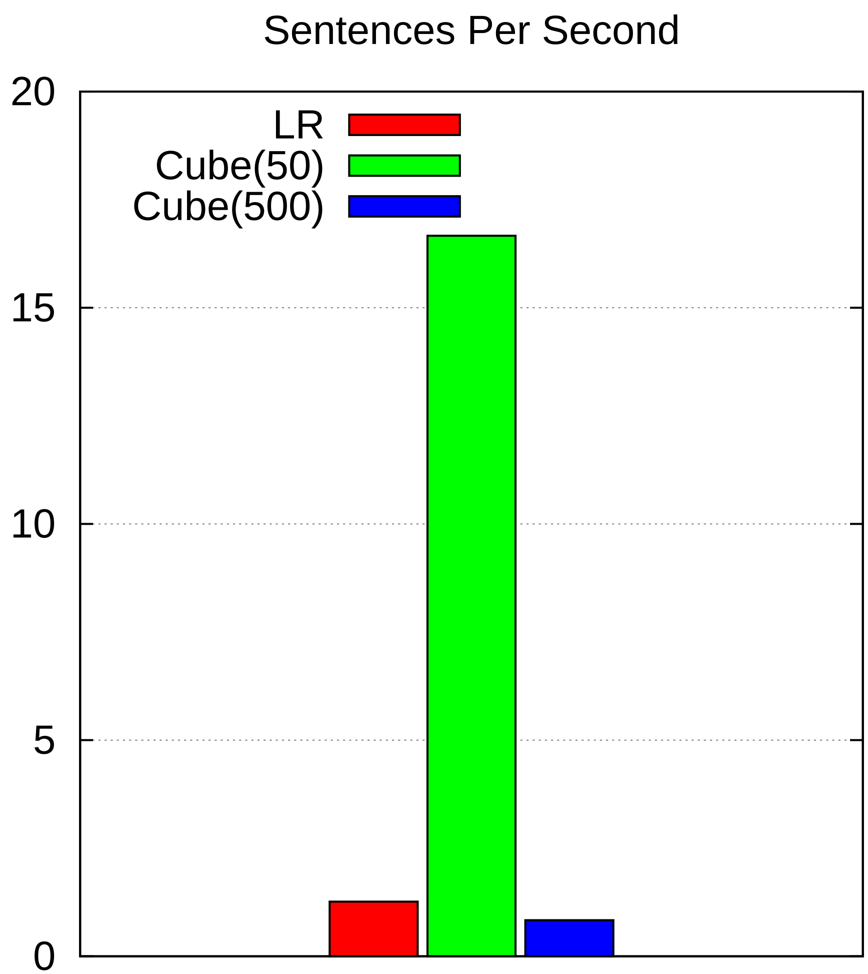
LR	Lagrangian Relaxation
ILP	Integer Linear Programming
DP	Exact Dynanic Programming
LP	Linear Programming

Comparison to Cube Pruning: Exactness



LR	Lagrangian Relaxation
Cube(50)	Cube Pruning (Beam=50)
Cube(500)	Cube Pruning (Beam=500)

Comparison to Cube Pruning: Median Speed



LR	Lagrangian Relaxation
Cube(50)	Cube Pruning (Beam=50)
Cube(500)	Cube Pruning (Beam=500)