Dependency Graph Parsing as Sequence Labeling

Ana Ezquerro

David Vilares

Carlos Gómez-Rodríguez

UNIVERSIDADE DA CORUÑA

citic

Funded by the European Union

European Research Council

MINISTERIO DE CIENCIA E INNOVACIÓN

UNIÓN EUROPEA

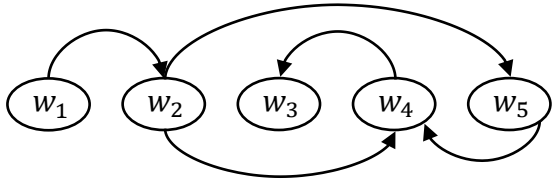
FONDO EUROPEO DE DESARROLLO REGIONAL

AGENCIA ESTATAL DE INVESTIGACIÓN

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

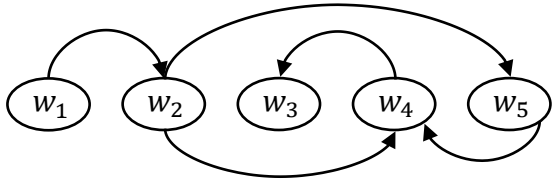
- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.



Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.



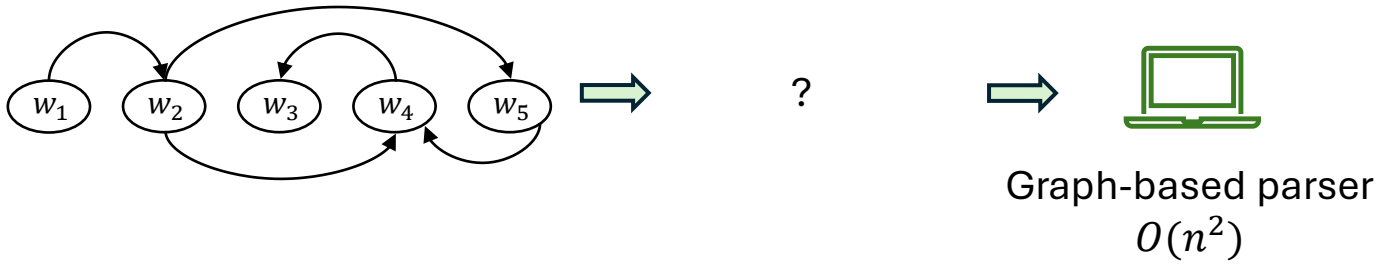
It's **not** Dependency *Tree* Parsing.

- Multiple roots and heads per node.
- Optionally non-connected.
- Cycles are allowed.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

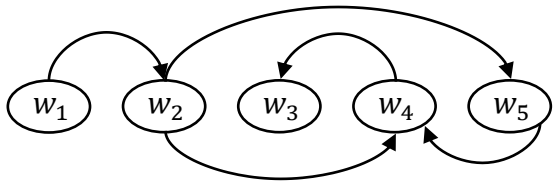


Graph-based parser
$O(n^2)$

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.



Adjacency matrix

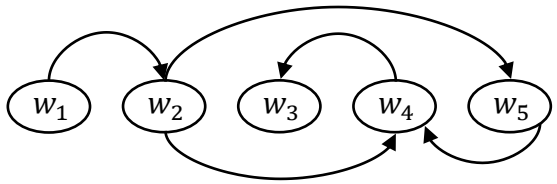
0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	1
0	1	0	0	0

Graph-based parser
$O(n^2)$

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.



Adjacency matrix

0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	1
0	1	0	0	0

Graph-based parser
$O(n^2)$

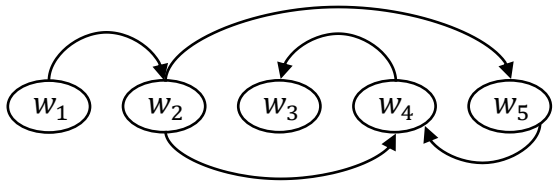
Predicted scores

0	0.2	0.4	0.3	0.1
0.7	0	0	0.5	0.1
0.2	0	0	0.6	0
0.1	0.5	0.1	0	0.8
0.1	0.1	0.3	0.4	0

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.



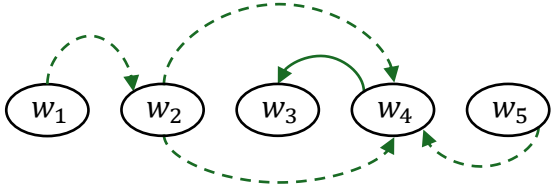
Adjacency matrix

0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	1
0	1	0	0	0

Graph-based parser
$O(n^2)$

Predicted scores

0	0.2	0.4	0.3	0.1
0.7	0	0	0.5	0.1
0.2	0	0	0.6	0
0.1	0.5	0.1	0	0.8
0.1	0.1	0.3	0.4	0

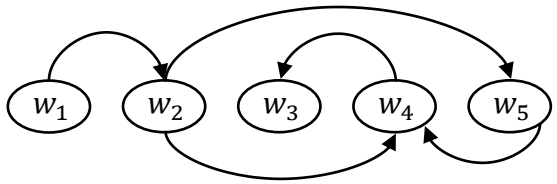


Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



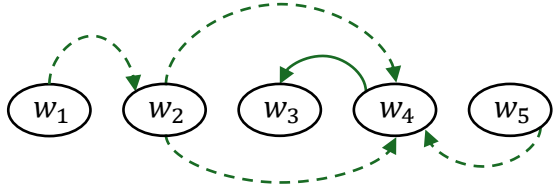
Adjacency matrix

0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	1
0	1	0	0	0

Graph-based parser
$O(n^2)$

Predicted scores

0	0.2	0.4	0.3	0.1
0.7	0	0	0.5	0.1
0.2	0	0	0.6	0
0.1	0.5	0.1	0	0.8
0.1	0.1	0.3	0.4	0

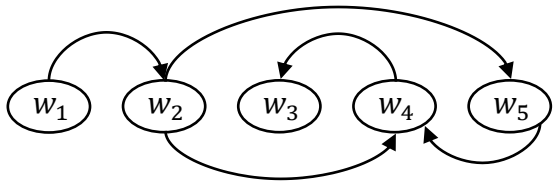


Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



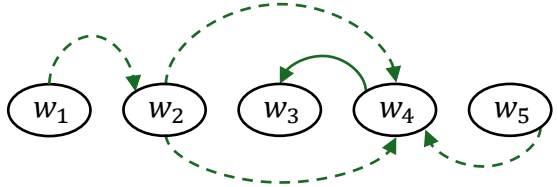
Adjacency matrix

0	0	0	0	0
1	0	0	0	0
0	0	0	1	0
0	1	0	0	1
0	1	0	0	0

Graph-based parser
$O(n^2)$

Predicted scores

0	0.2	0.4	0.3	0.1
0.7	0	0	0.5	0.1
0.2	0	0	0.6	0
0.1	0.5	0.1	0	0.8
0.1	0.1	0.3	0.4	0



Encoding

$\ell_1 \quad \ell_2 \quad \ell_3 \quad \ell_4 \quad \ell_5$

Encoded labels

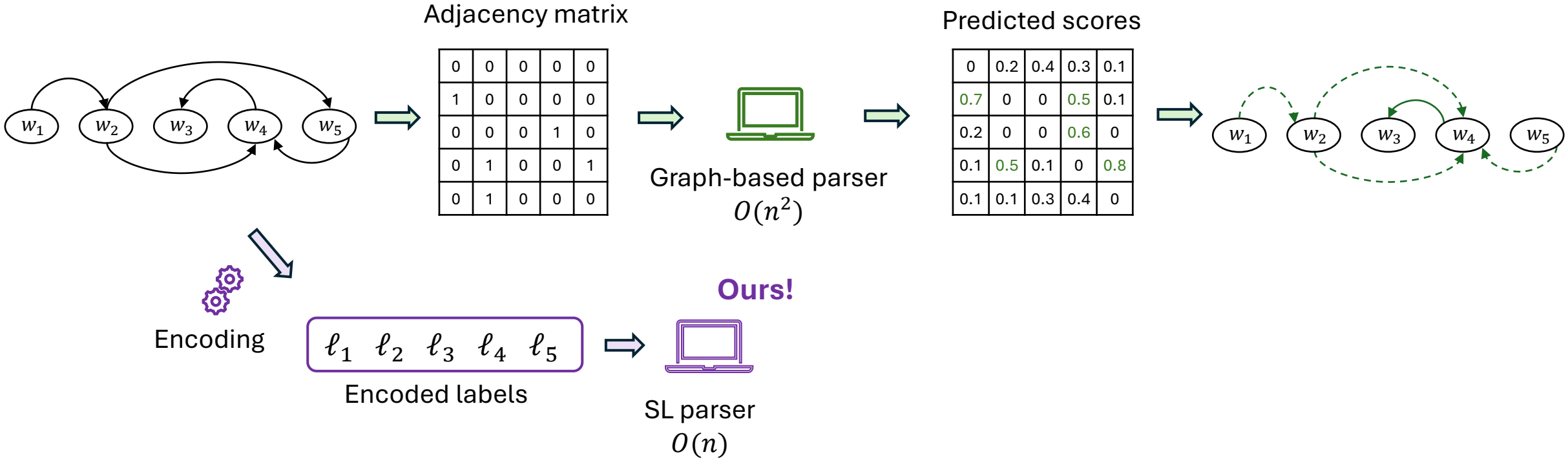
… as **Sequence Labeling**.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



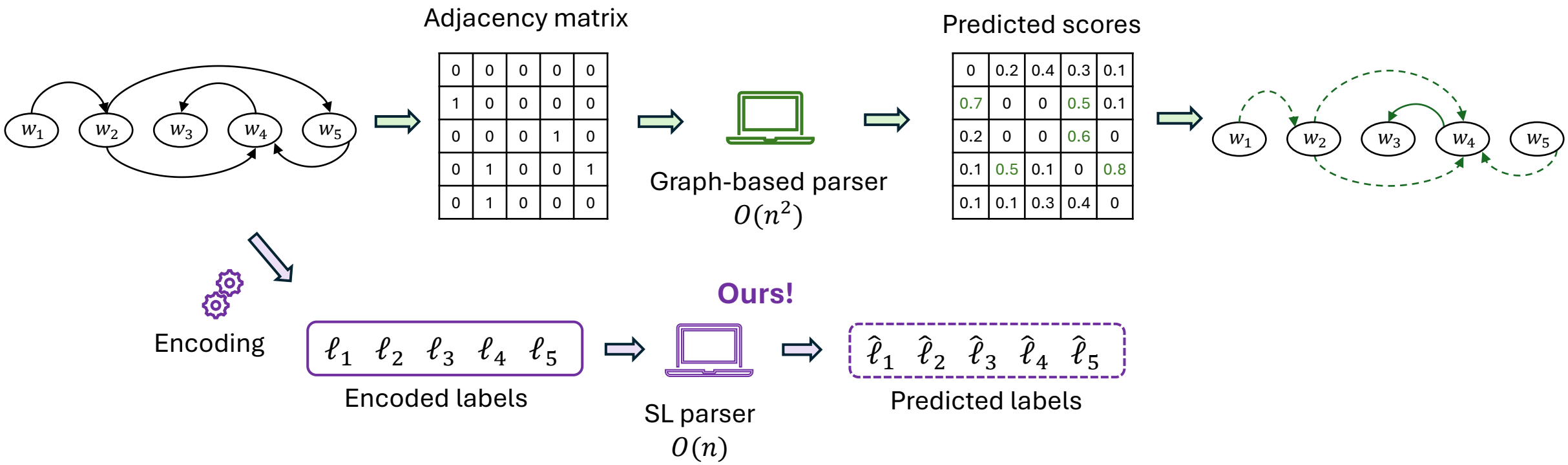
… as **Sequence Labeling**.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



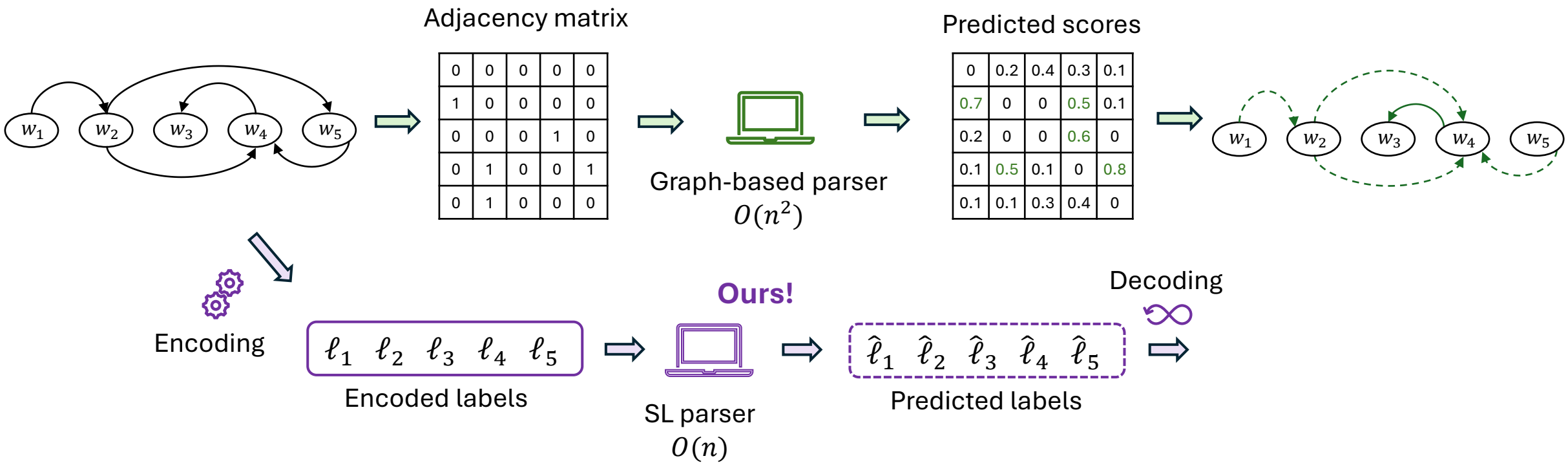
… as **Sequence Labeling**.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



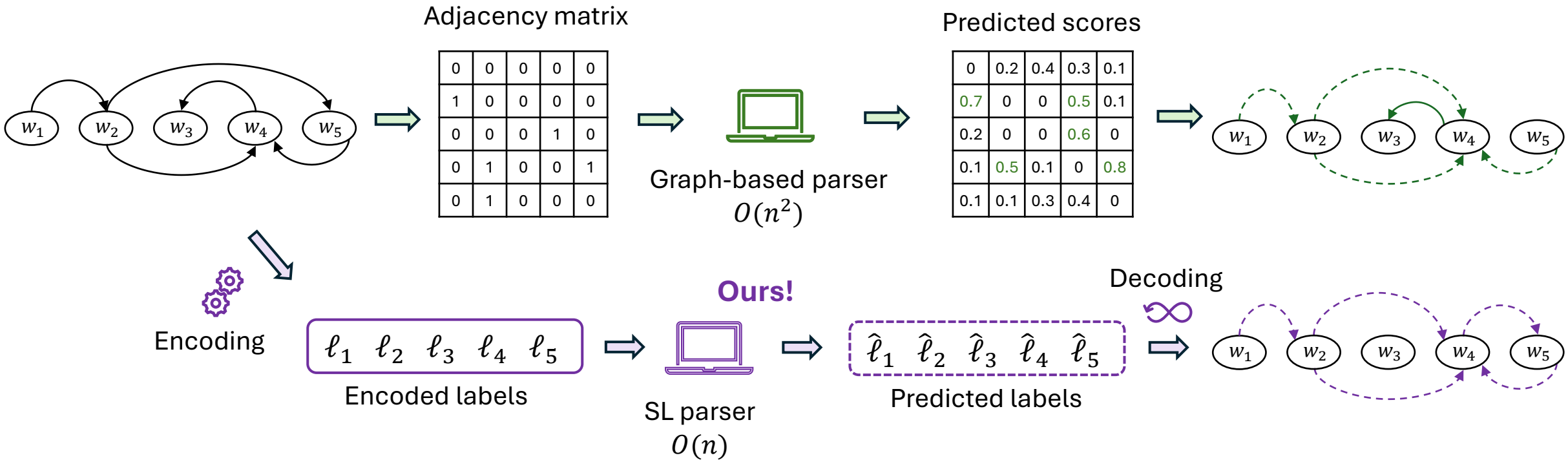
… as **Sequence Labeling**.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing…

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



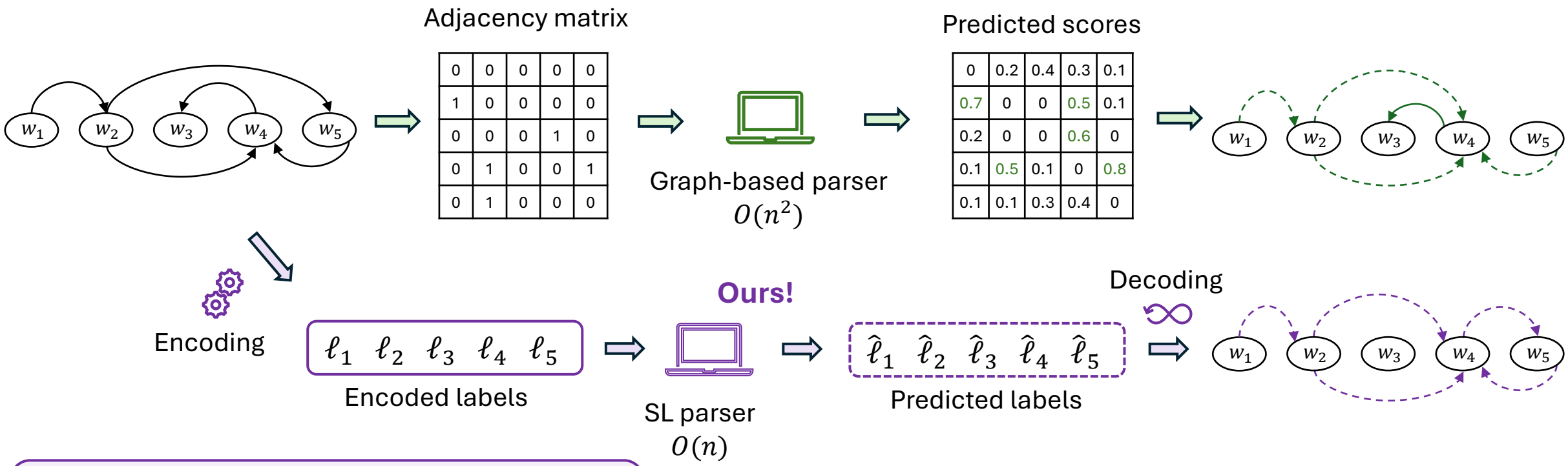
… as **Sequence Labeling**.

Motivation: Approaches in Dependency Graph Parsing

Dependency **Graph** Parsing...

- Sentence: $W = (w_1, \dots, w_n)$.
- Arcs: $A = \{(h, d): w_h \rightarrow w_d, h \neq d\}$.

- Dozat & Manning (2018): ~ 90 UF
- Ji et al. (2019): ~95 UF



1. Define the encoding and decoding functions.
2. Then tackle as a classical tagging task

... as **Sequence Labeling**.

We propose various encoding ( ) and decoding ( ) functions

Unbounded Encodings

Positional Encodings

Absolute
Relative

k-planar Bracketing Encoding

</\>

Non-fixed number of unique labels.

Bounded Encodings

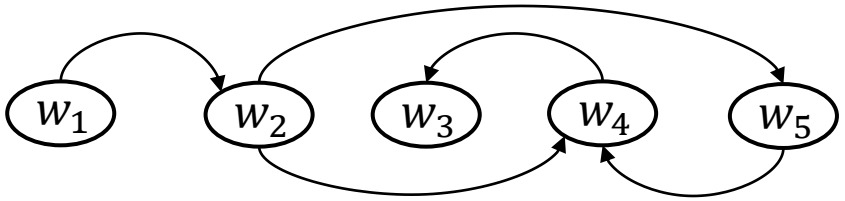
$4k$-bit Encoding

$6k$-bit Encoding

Fixed number of unique labels.

Positional encodings: absolute (A) and relative (R)

- **Absolute**: Sorted sequence of head positions.

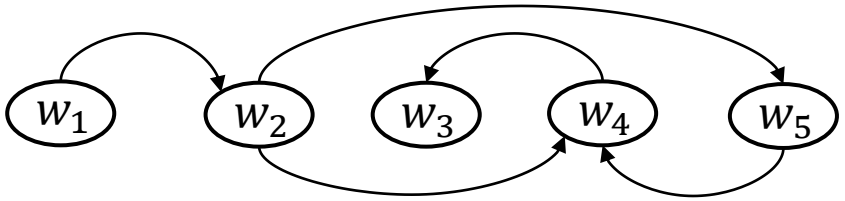


	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5
Absolute:	(,)	(1)	(4)	(2,5)	(2)

Adapted from Strzyz et al. (2019)

Positional encodings: absolute (A) and relative (R)

- **Absolute**: Sorted sequence of head positions.
- **Relative**: Sorted sequence of head relative positions.



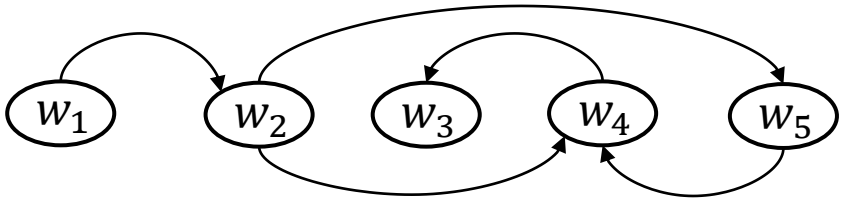
	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5
Absolute:	(,)	(1)	(4)	(2,5)	(2)
Relative:	(,)	(-1)	(1)	(-2, 1)	(-3)

Adapted from Strzyz et al. (2019)

Positional encodings: absolute (A) and relative (R)

- **Absolute**: Sorted sequence of head positions.
- **Relative**: Sorted sequence of head relative positions.

Fast and simple decoding (easy to parallelize).



	ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5
Absolute:	(,)	(1)	(4)	(2,5)	(2)
Relative:	(,)	(-1)	(1)	(-2, 1)	(-3)

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$ < / \ >

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$ <

/

\

>

$\ell_i =$ <</

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

< : There is a head at the right.

/

\

>

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\`
- `>`

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

- < : There is a head at the right.
- / : There is a dependent at the right.
- \ : There is a dependent at the left.
- >

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

- $<$: There is a head at the right.
- $/$: There is a dependent at the right.
- $\backslash$: There is a dependent at the left.
- $>$: There is a head is at the left.

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Adapted from Strzyz et al. (2019)

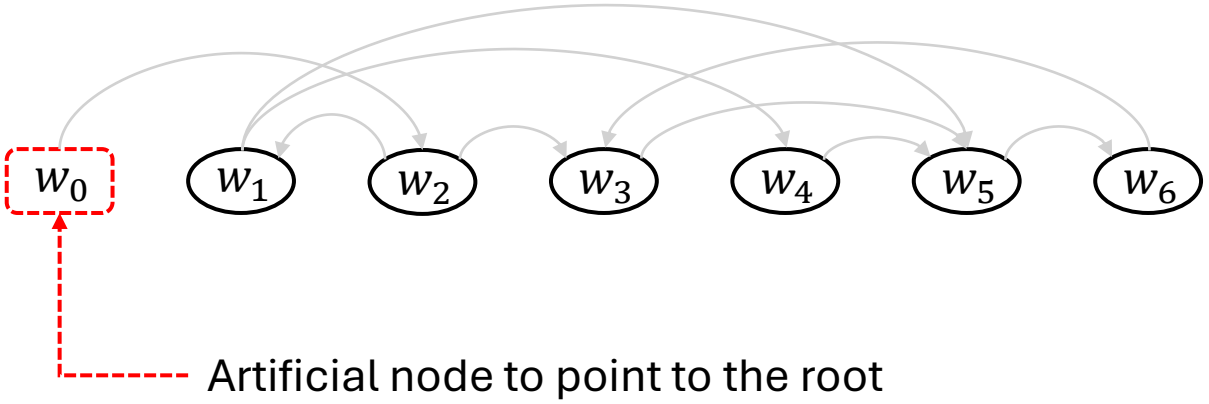
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

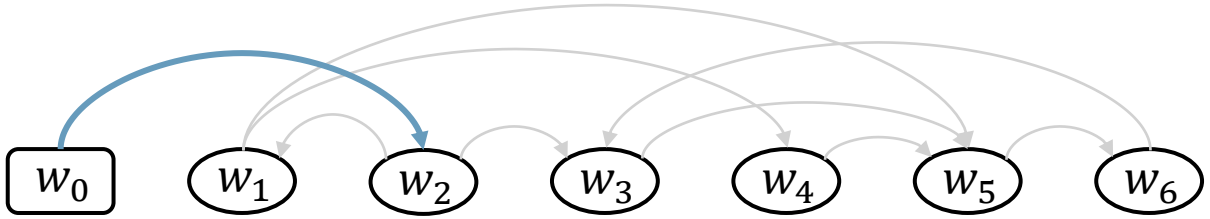
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

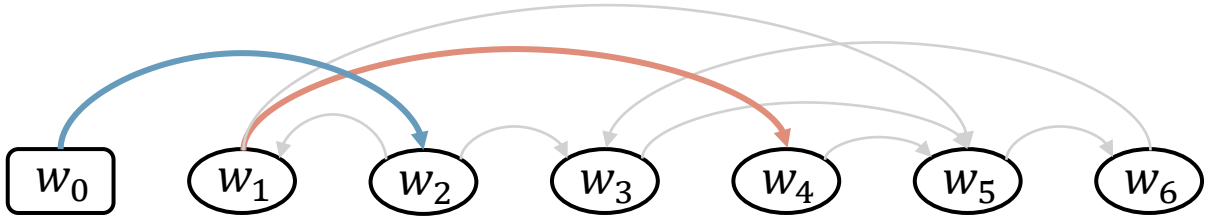
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

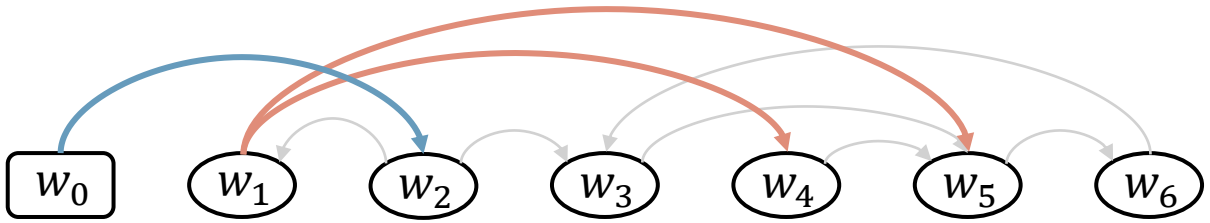
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

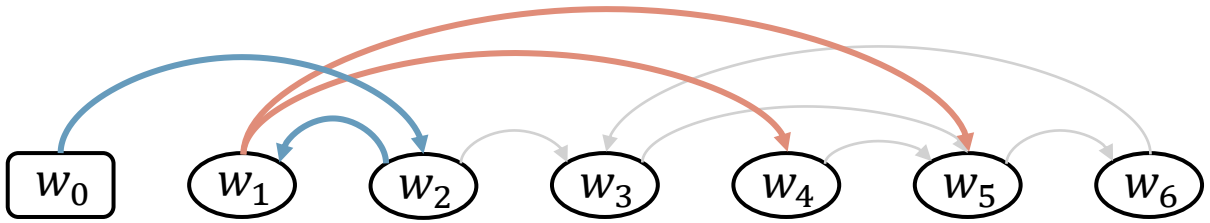
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

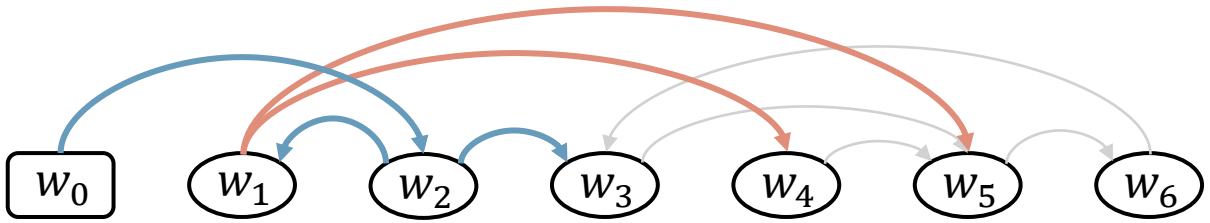
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

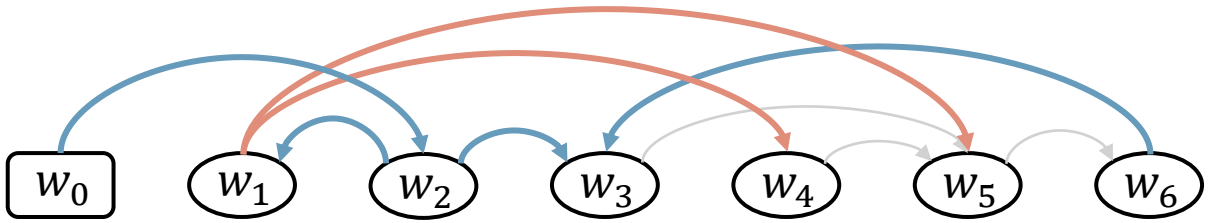
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

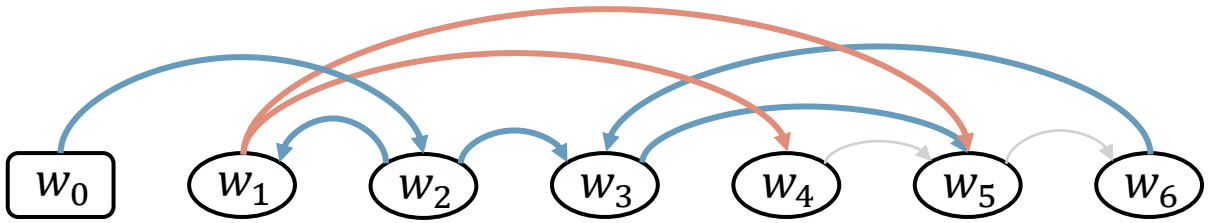
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

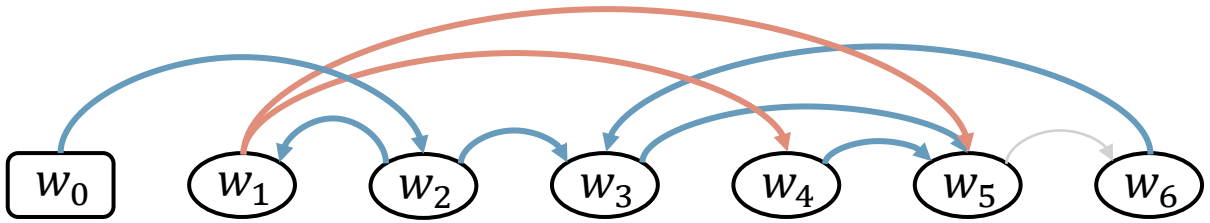
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

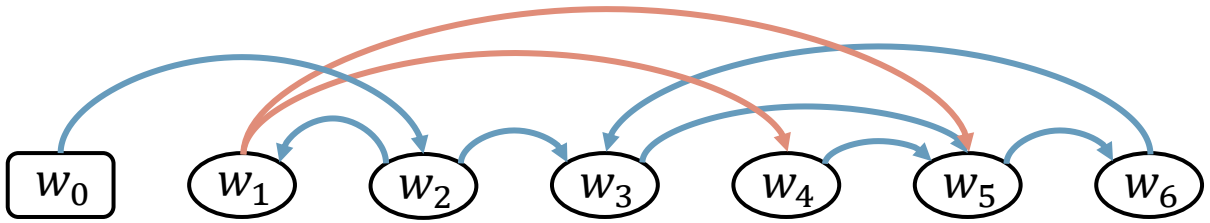
k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 1: Distribute in planes.



Adapted from Strzyz et al. (2019)

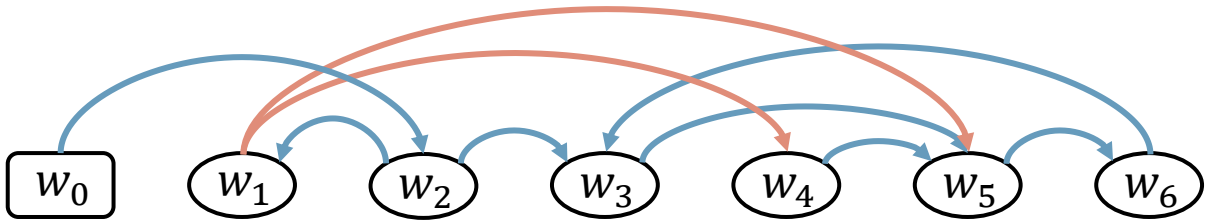
k-planar Bracketing Encoding

$\ell_i \sim$

- < : There is a head at the right.
- / : There is a dependent at the right.
- \ : There is a dependent at the left.
- > : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol * for brackets of other planes.
- First plane: </\>.
- Second plane: <*/**>*.
- Third plane: <**/****>**.

Step 2: Assign labels to each plane.



Adapted from Strzyz et al. (2019)

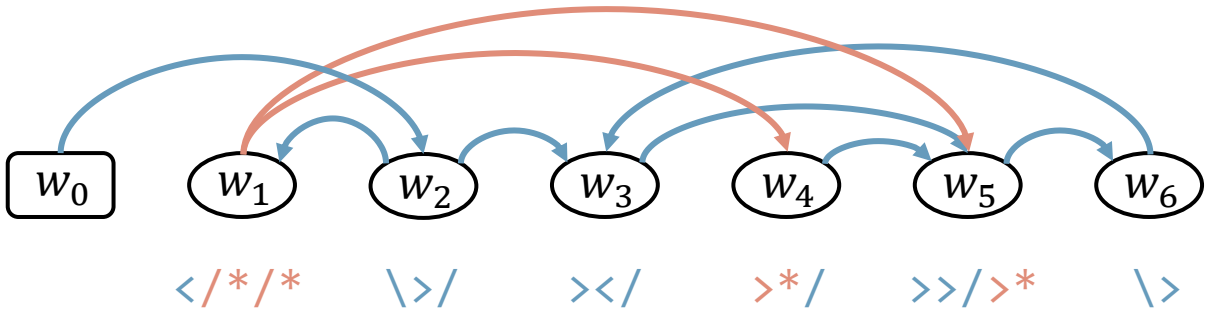
k-planar Bracketing Encoding

$\ell_i \sim$

- < : There is a head at the right.
- / : There is a dependent at the right.
- \ : There is a dependent at the left.
- > : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol * for brackets of other planes.
- First plane: </\>.
- Second plane: <*/**>*.
- Third plane: <**/****>**.

Step 2: Assign labels to each plane.



Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

$\ell_i \sim$

- `<` : There is a head at the right.
- `/` : There is a dependent at the right.
- `\` : There is a dependent at the left.
- `>` : There is a head is at the left.

- Distribute the arcs in planes (non-crossing arcs).
- Add symbol `*` for brackets of other planes.
- First plane: `</\>`.
- Second plane: `<*/*\*>*`.
- Third plane: `<**/**\**>**`.

Step 2: Assign labels to each plane.

w_0	w_1	w_2	w_3	w_4	w_5	w_6
	`</*/*`	`\>/`	`></`	`>*/`	`>>/>*`	`\>`

We predict this!

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

How do we recover the arcs from the labels $(\ell_1, \dots, \ell_n)$?

Adapted from Strzyz et al. (2019)

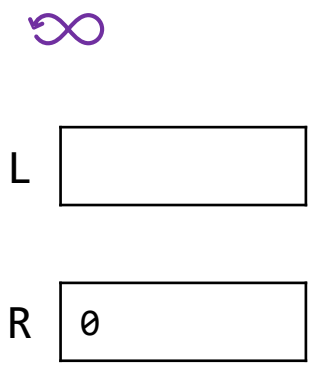
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

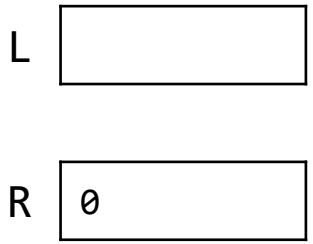


Adapted from Strzyz et al. (2019)

k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.

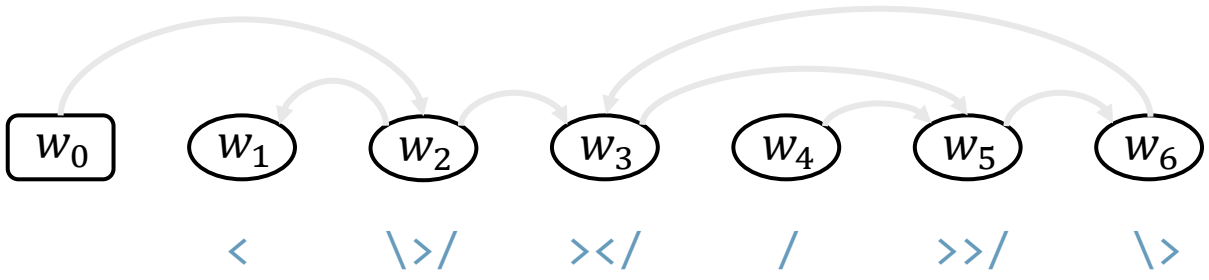


Adapted from Strzyz et al. (2019)

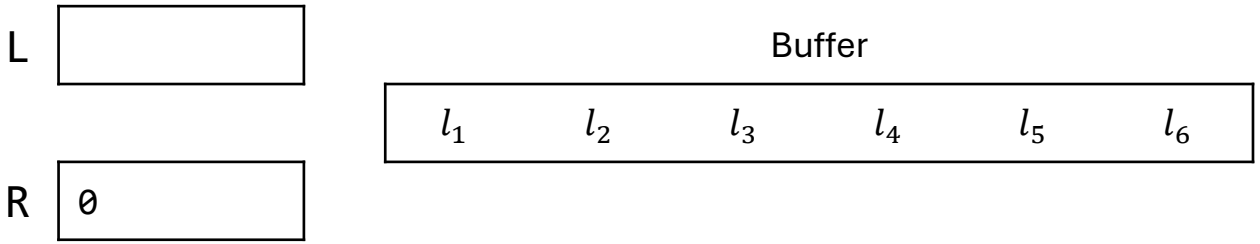
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Initial state: Labels to buffer and **R** with the root dependency.

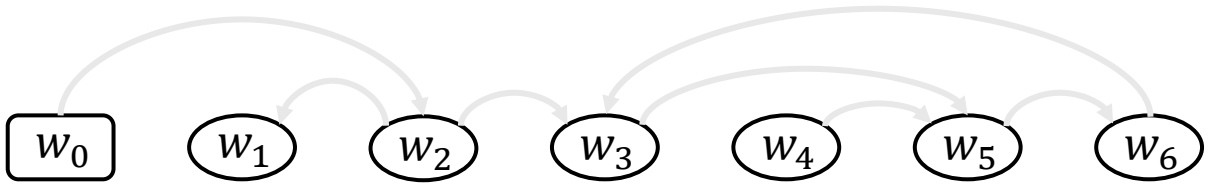


Adapted from Strzyz et al. (2019)

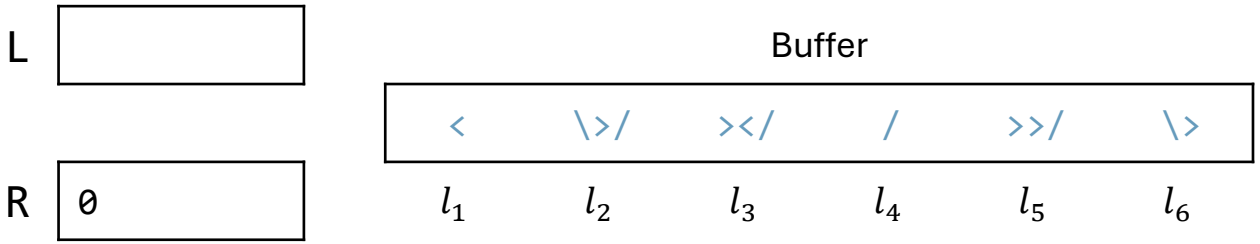
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Initial state: Labels to buffer and **R** with the root dependency.

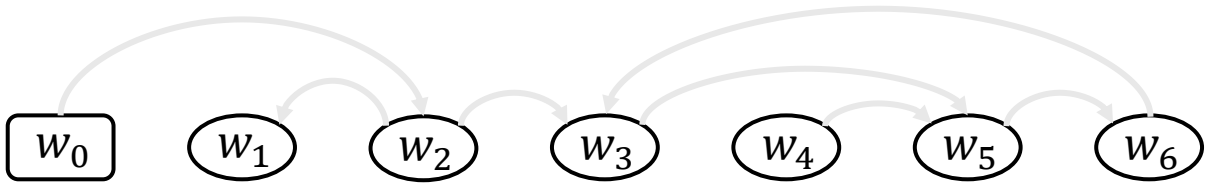


Adapted from Strzyz et al. (2019)

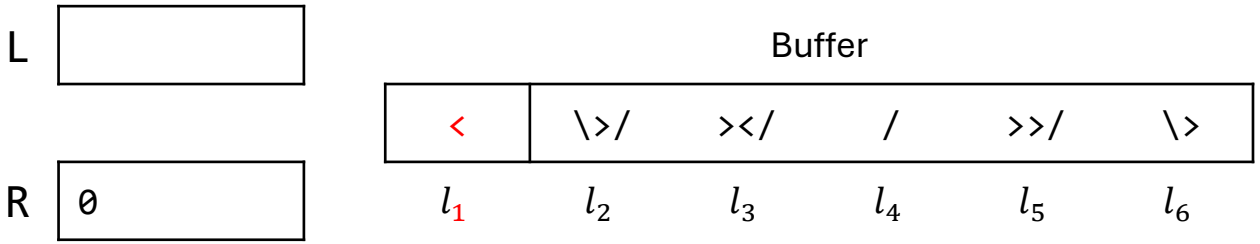
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 1.

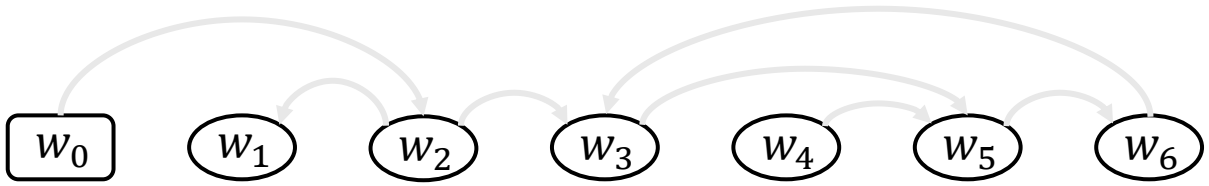


Adapted from Strzyz et al. (2019)

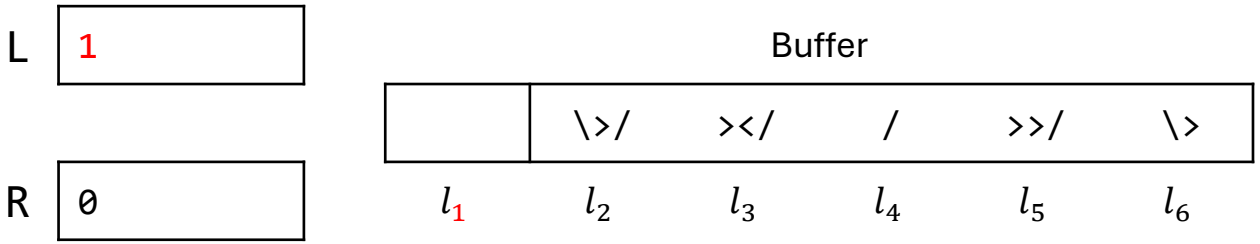
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 1.

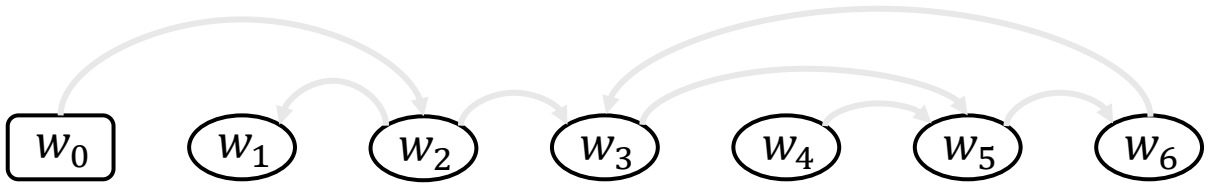


Adapted from Strzyz et al. (2019)

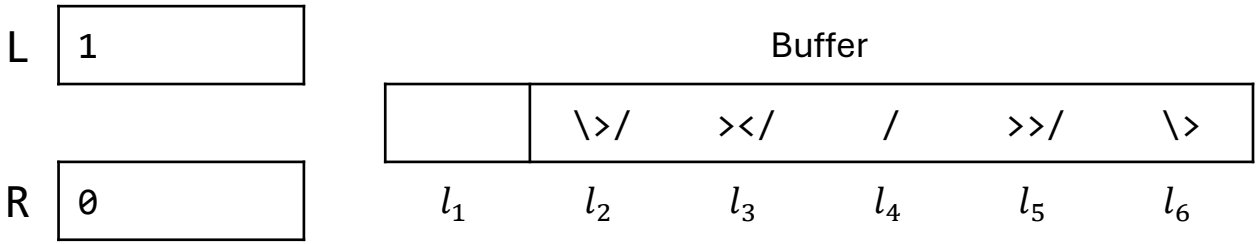
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 1.

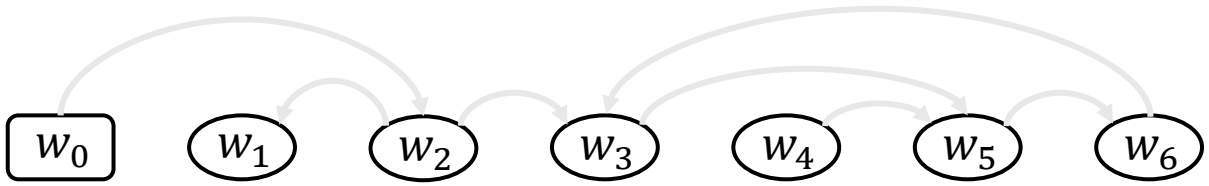


Adapted from Strzyz et al. (2019)

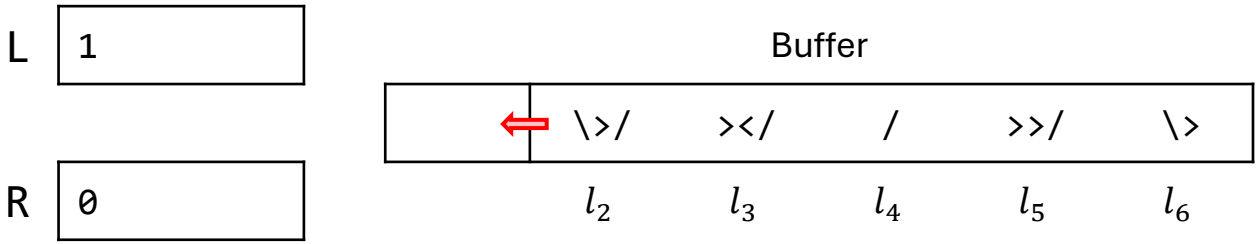
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 1.

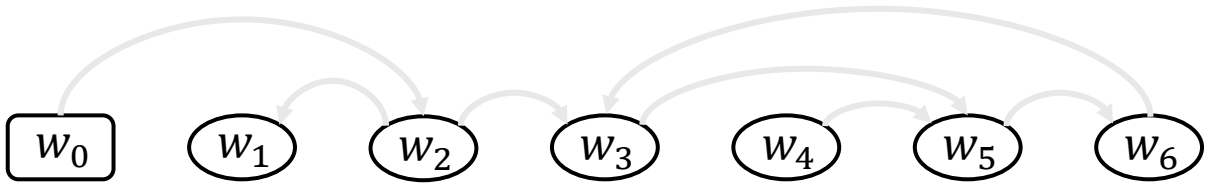


Adapted from Strzyz et al. (2019)

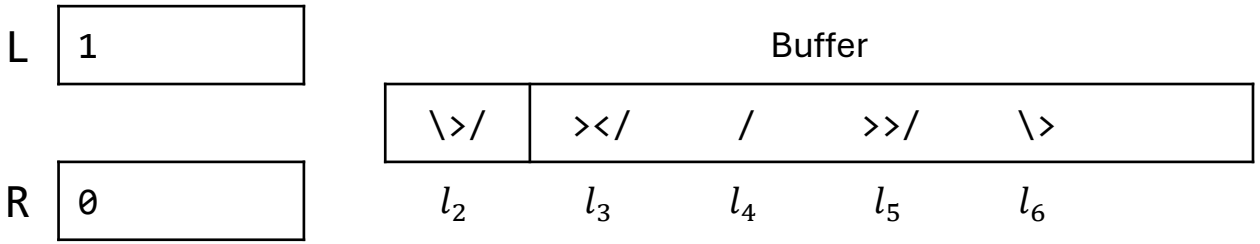
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

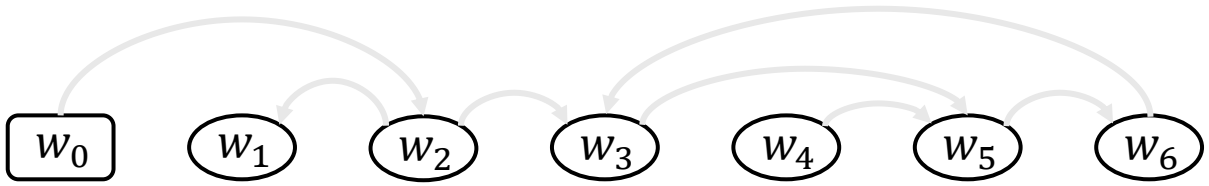


Adapted from Strzyz et al. (2019)

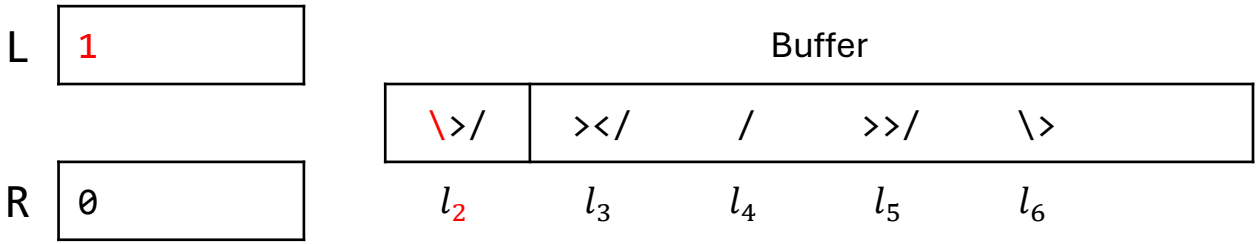
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

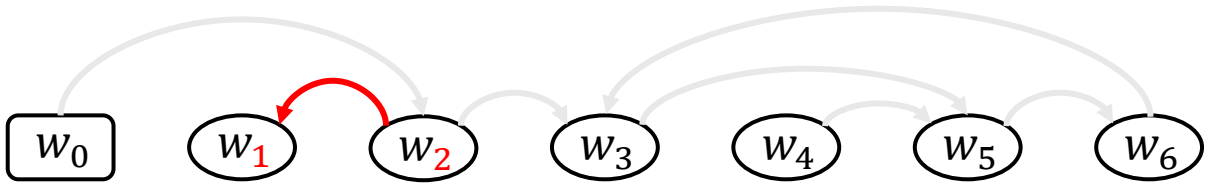


Adapted from Strzyz et al. (2019)

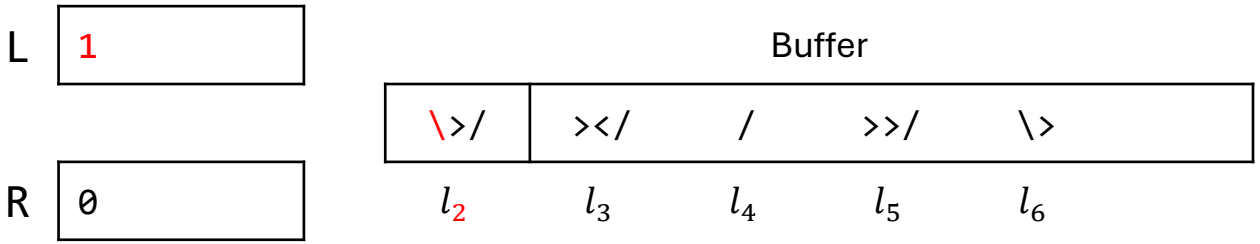
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

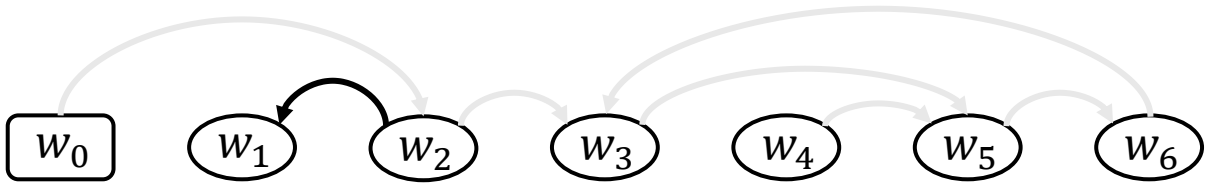


Adapted from Strzyz et al. (2019)

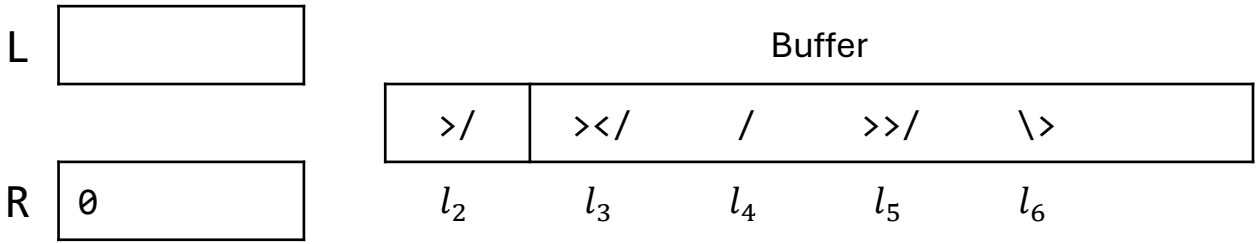
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

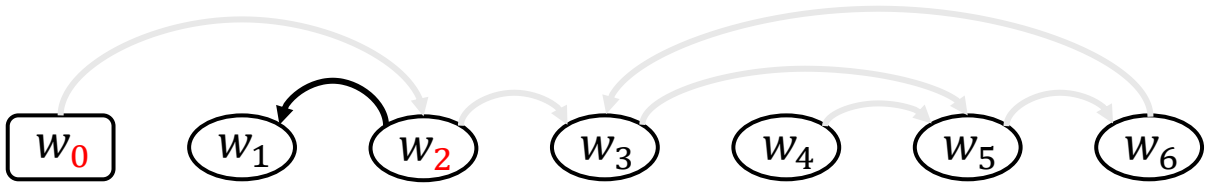


Adapted from Strzyz et al. (2019)

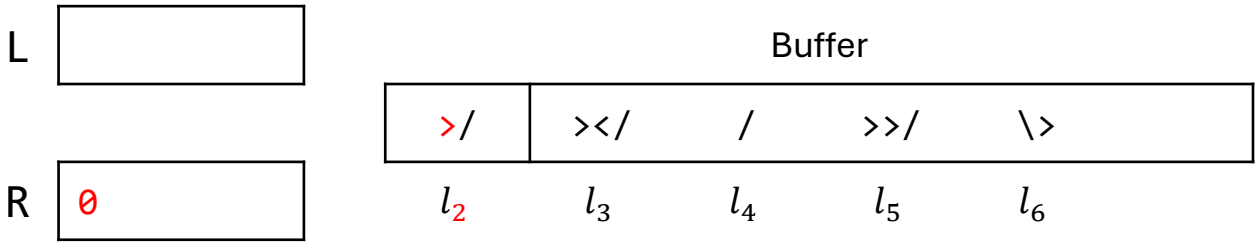
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

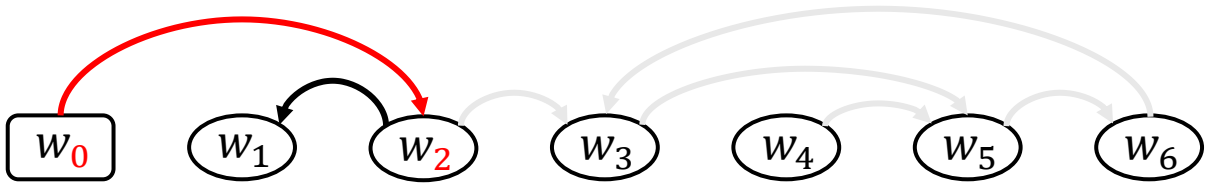


Adapted from Strzyz et al. (2019)

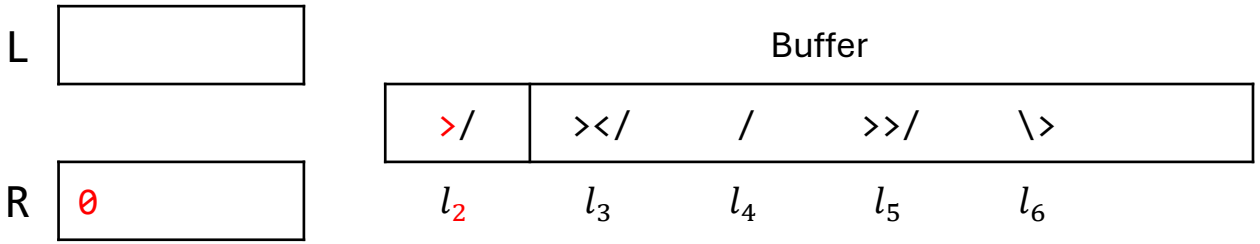
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

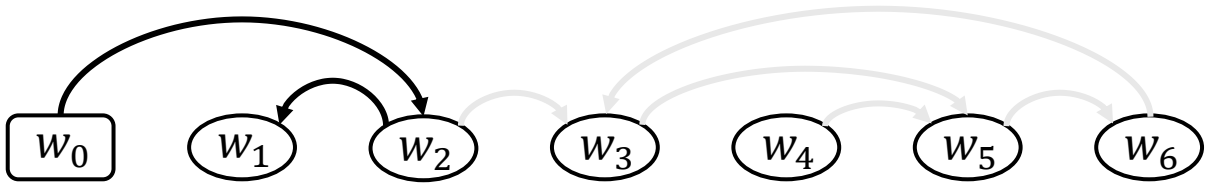


Adapted from Strzyz et al. (2019)

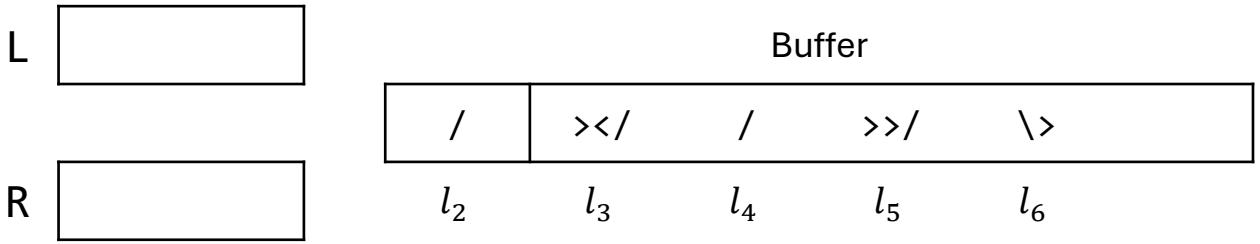
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

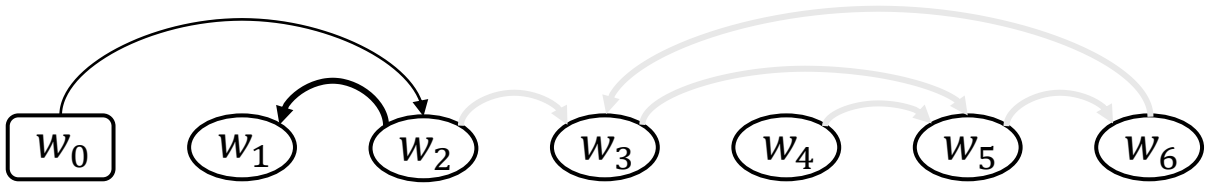


Adapted from Strzyz et al. (2019)

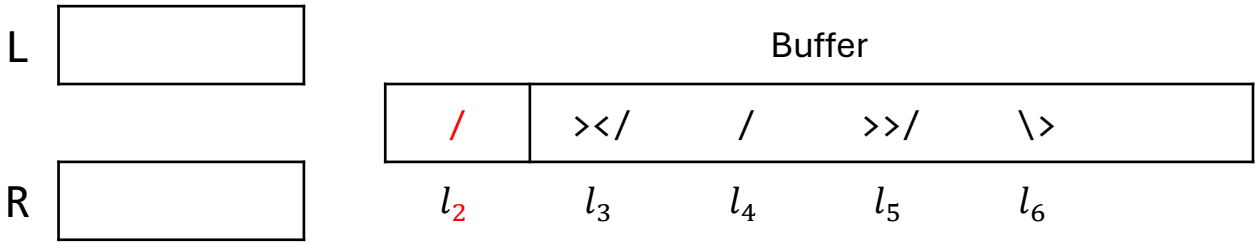
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

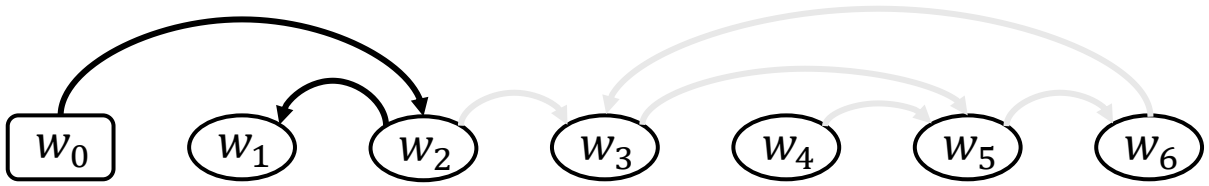


Adapted from Strzyz et al. (2019)

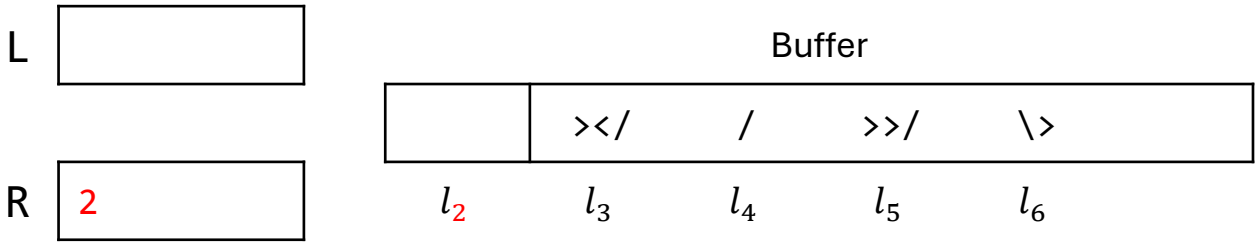
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

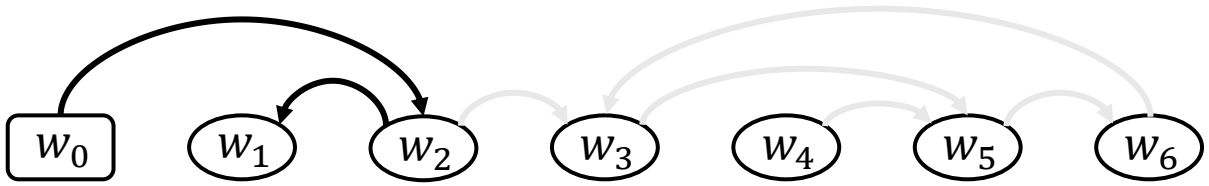


Adapted from Strzyz et al. (2019)

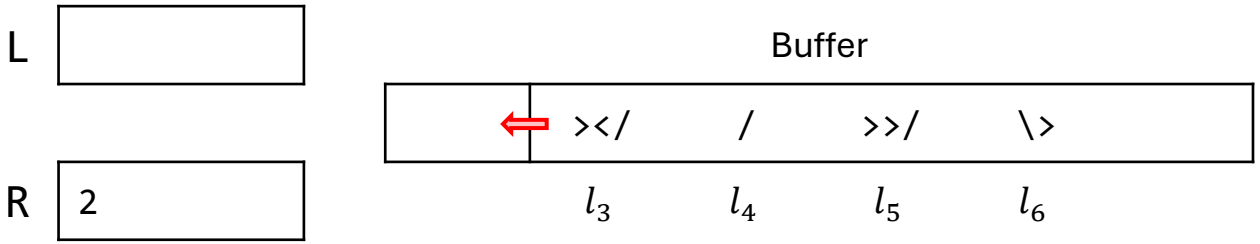
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 2.

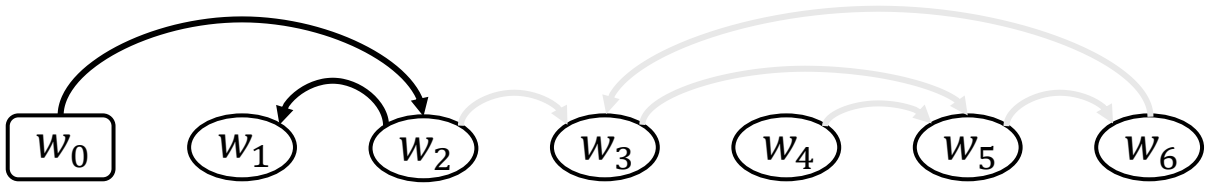


Adapted from Strzyz et al. (2019)

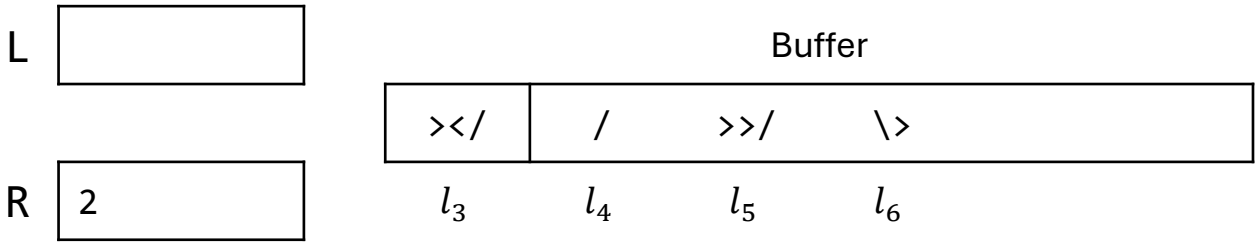
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

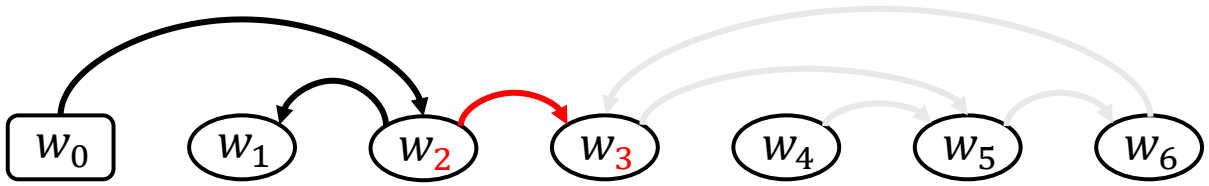


Adapted from Strzyz et al. (2019)

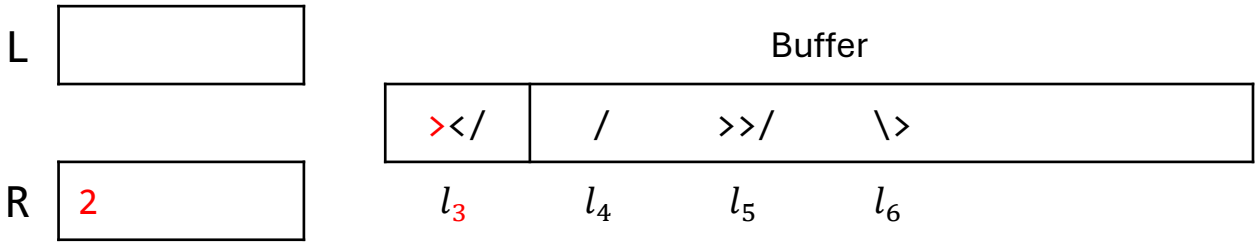
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

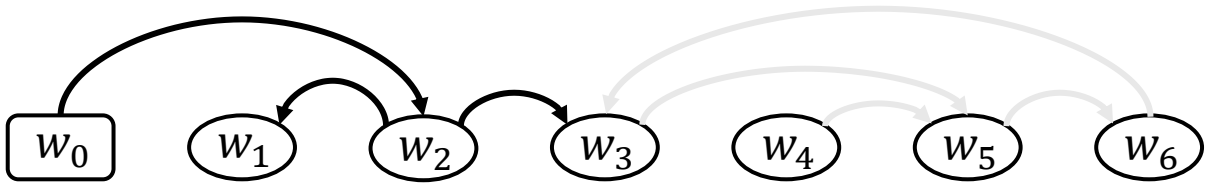


Adapted from Strzyz et al. (2019)

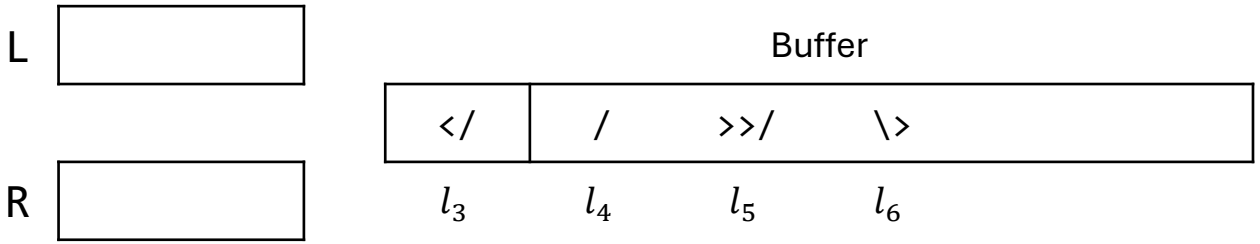
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

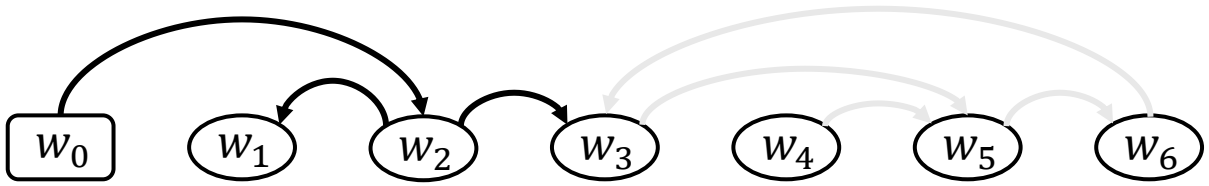


Adapted from Strzyz et al. (2019)

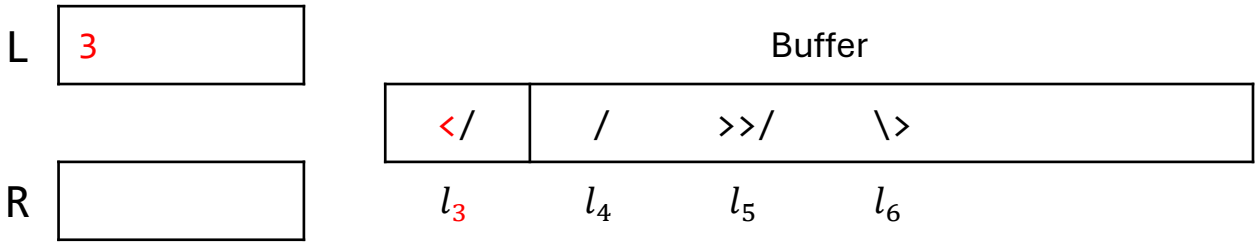
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

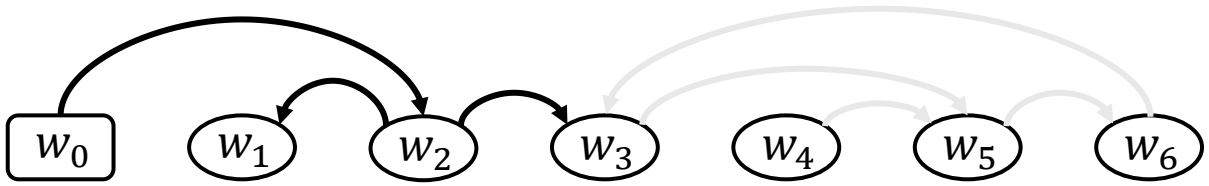


Adapted from Strzyz et al. (2019)

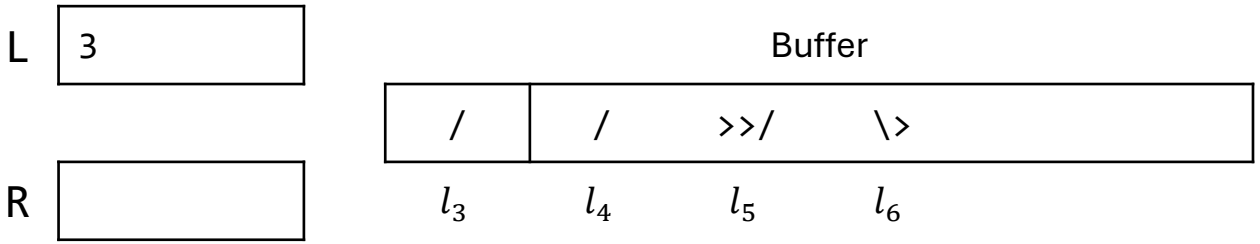
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

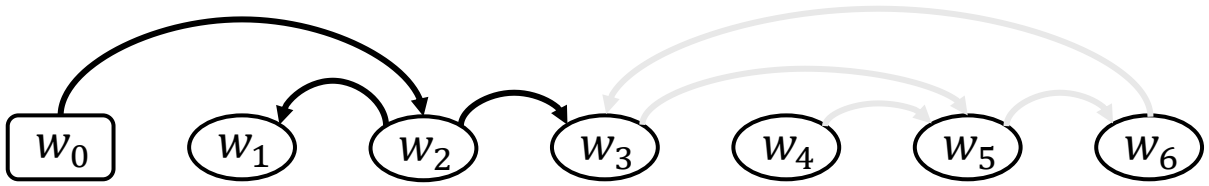


Adapted from Strzyz et al. (2019)

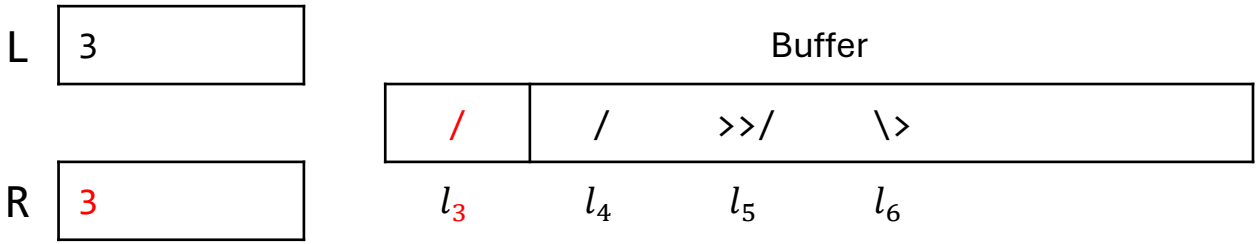
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

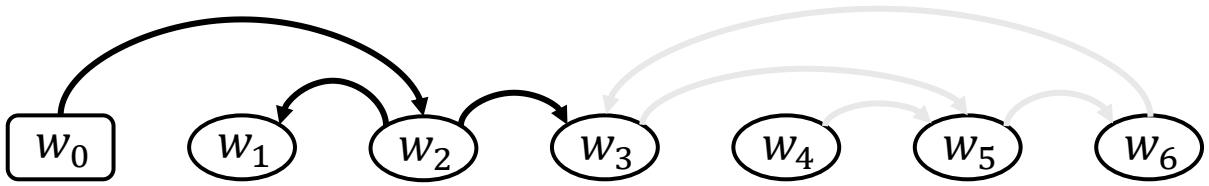


Adapted from Strzyz et al. (2019)

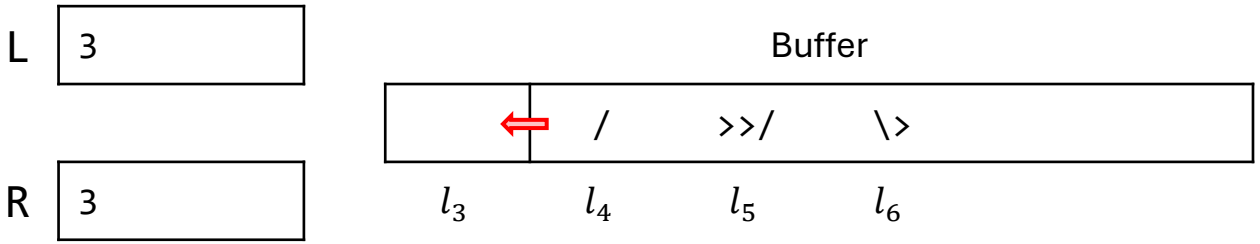
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 3.

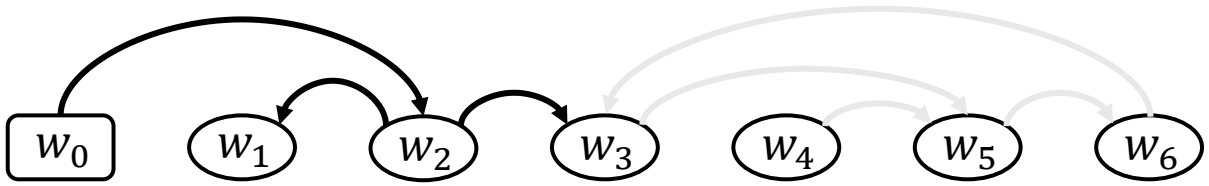


Adapted from Strzyz et al. (2019)

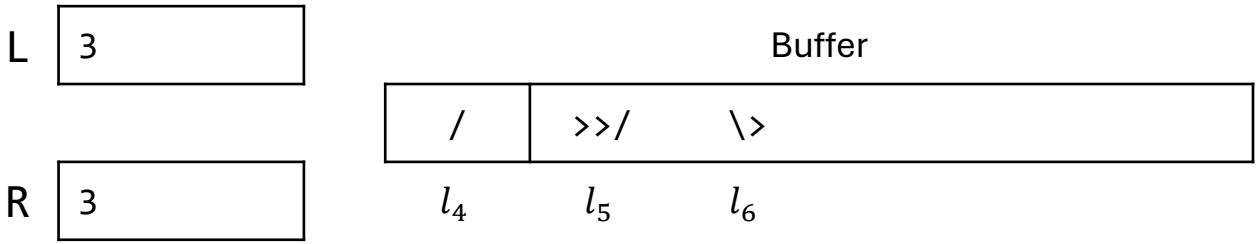
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 4.

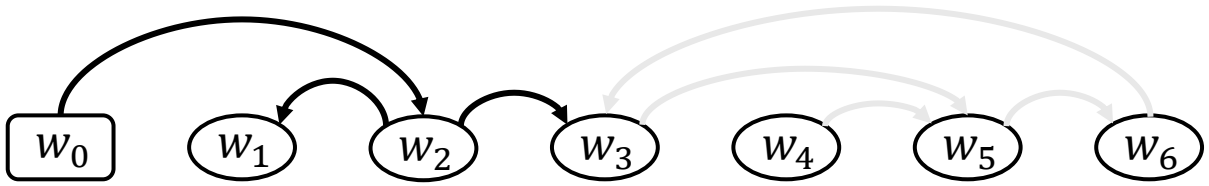


Adapted from Strzyz et al. (2019)

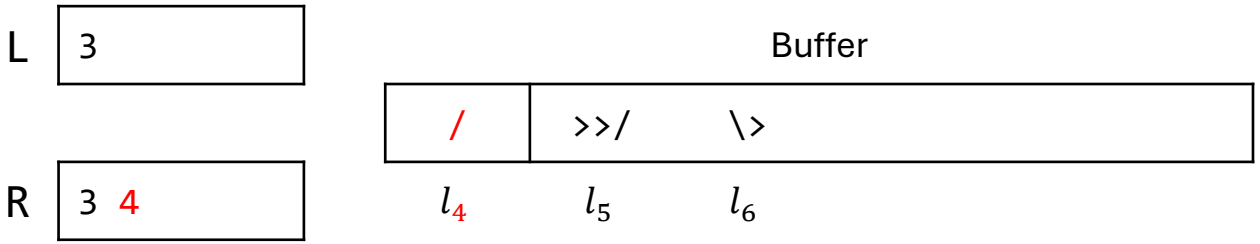
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 4.

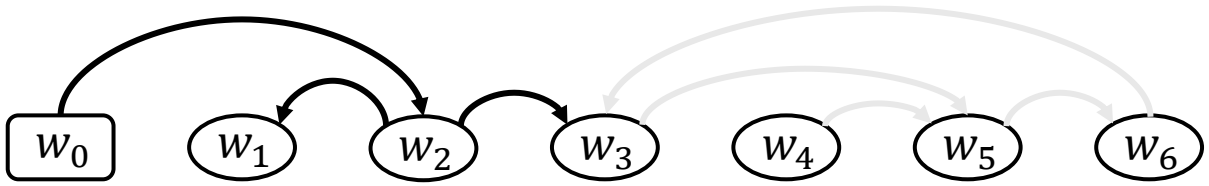


Adapted from Strzyz et al. (2019)

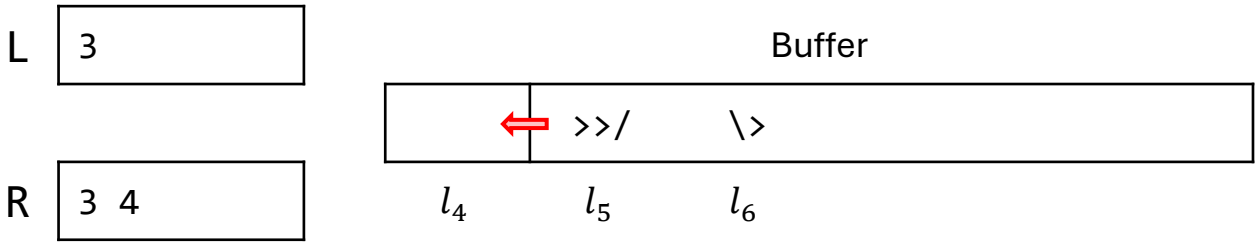
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 4.

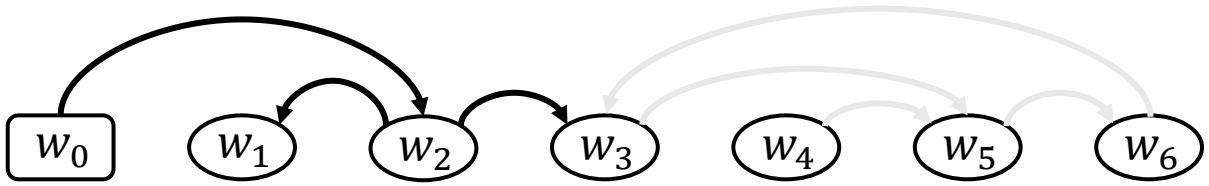


Adapted from Strzyz et al. (2019)

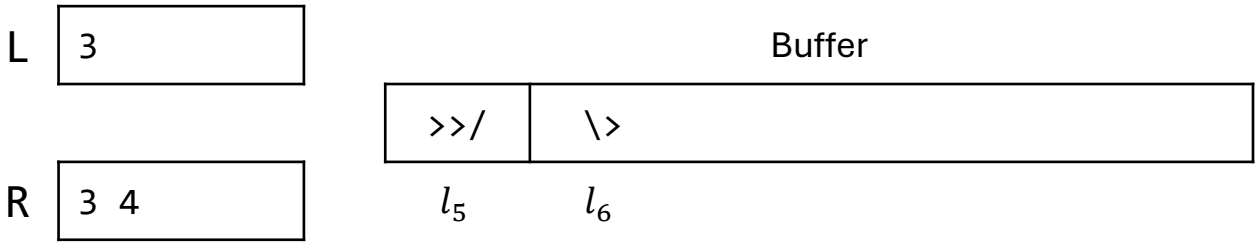
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

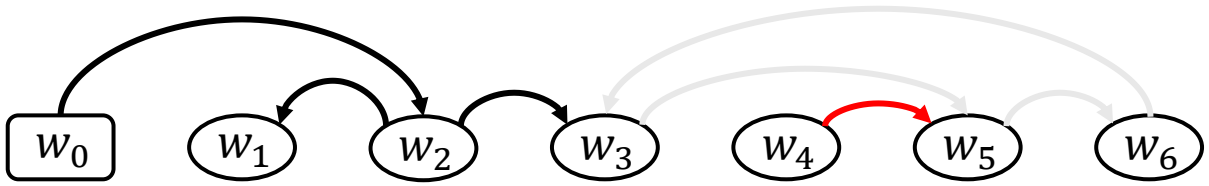


Adapted from Strzyz et al. (2019)

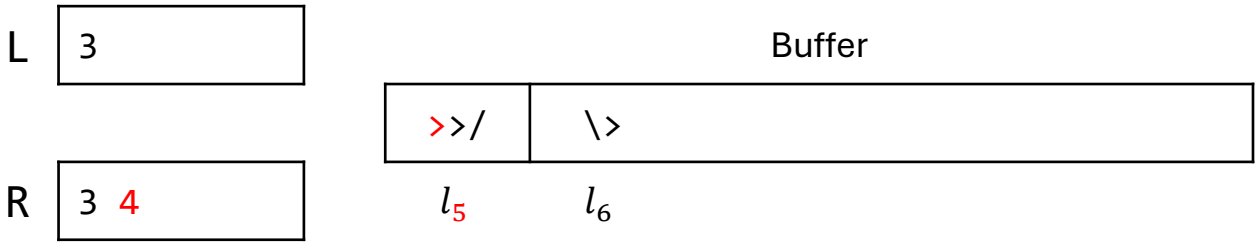
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

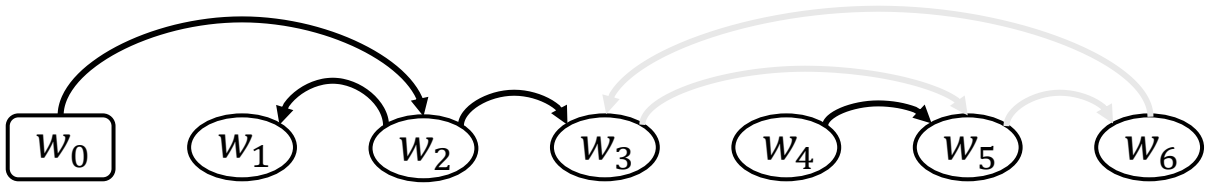


Adapted from Strzyz et al. (2019)

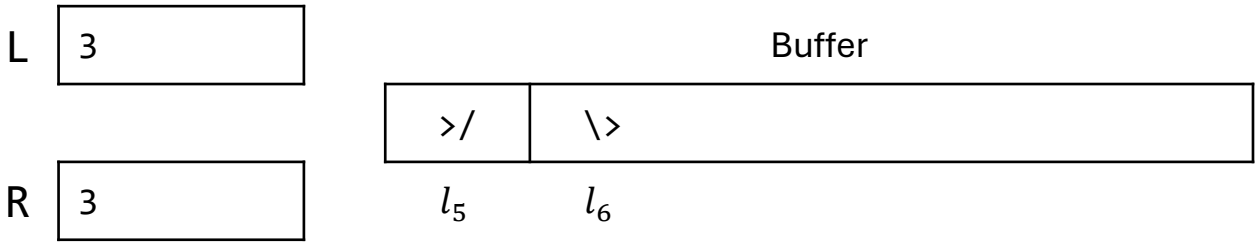
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

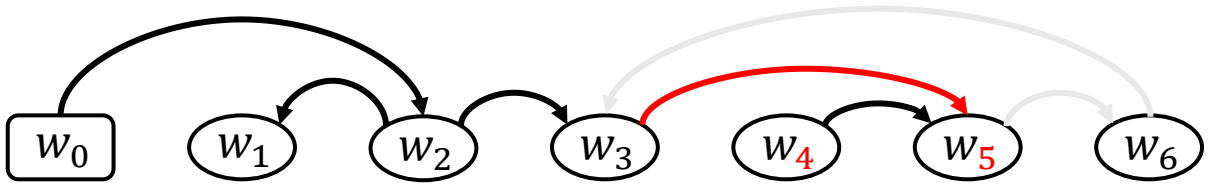


Adapted from Strzyz et al. (2019)

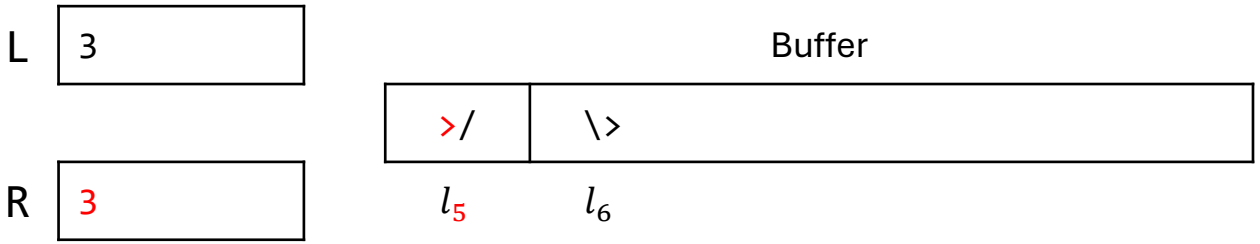
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

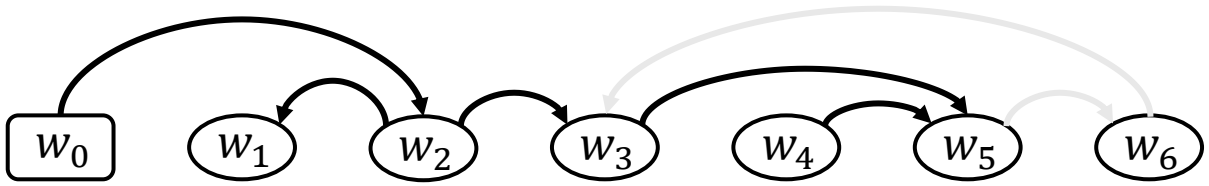


Adapted from Strzyz et al. (2019)

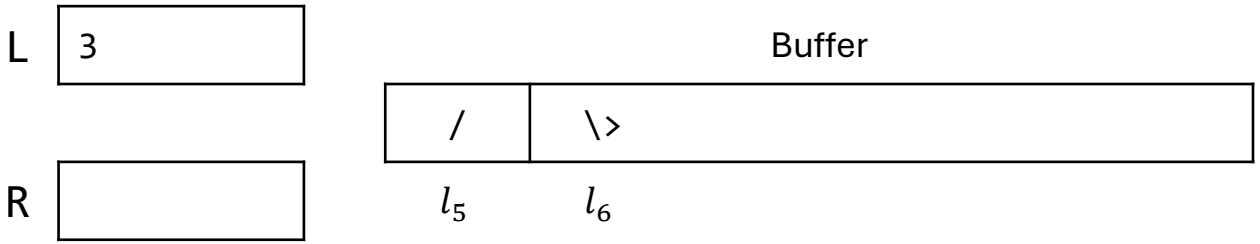
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

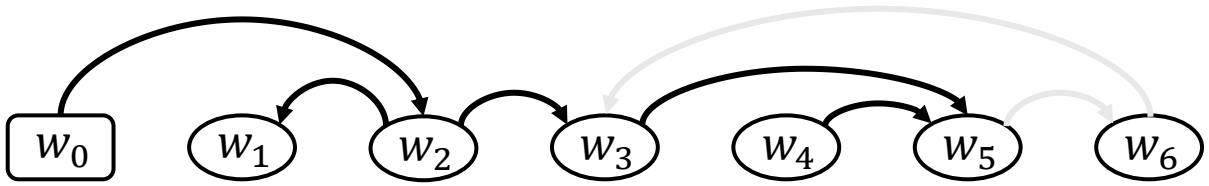


Adapted from Strzyz et al. (2019)

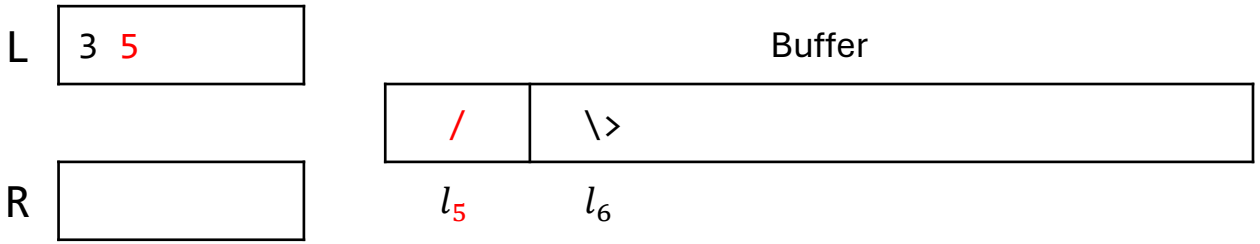
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

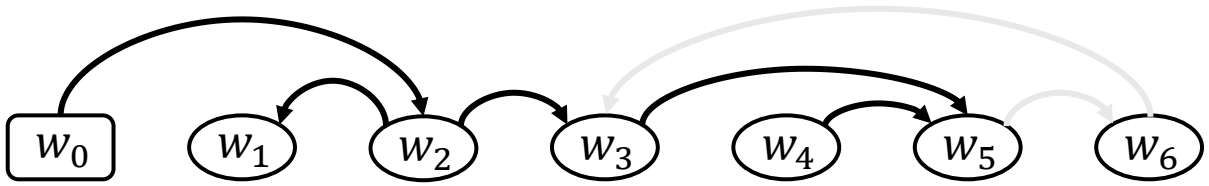


Adapted from Strzyz et al. (2019)

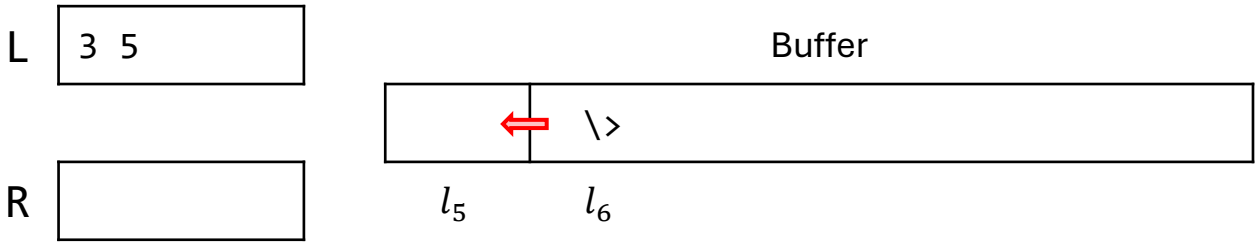
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 5.

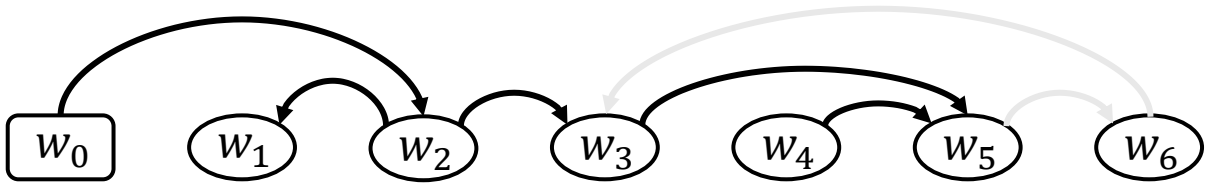


Adapted from Strzyz et al. (2019)

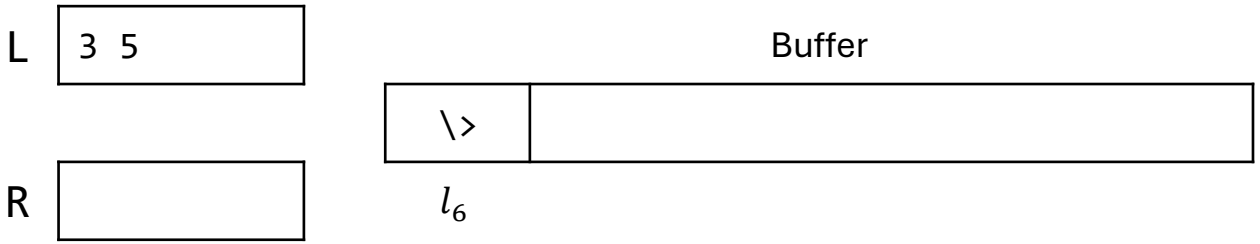
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 6.

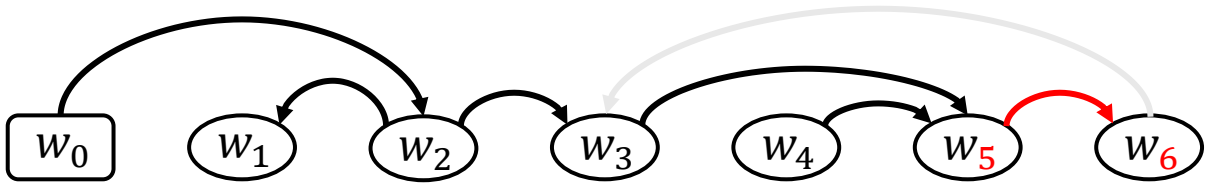


Adapted from Strzyz et al. (2019)

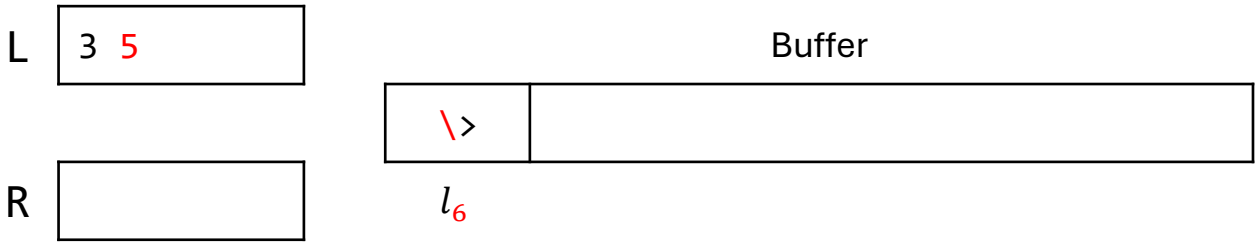
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 6.

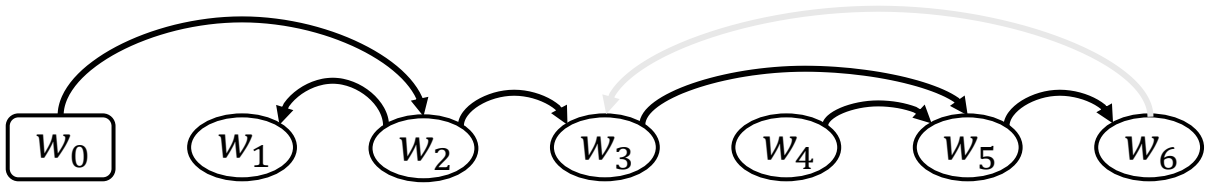


Adapted from Strzyz et al. (2019)

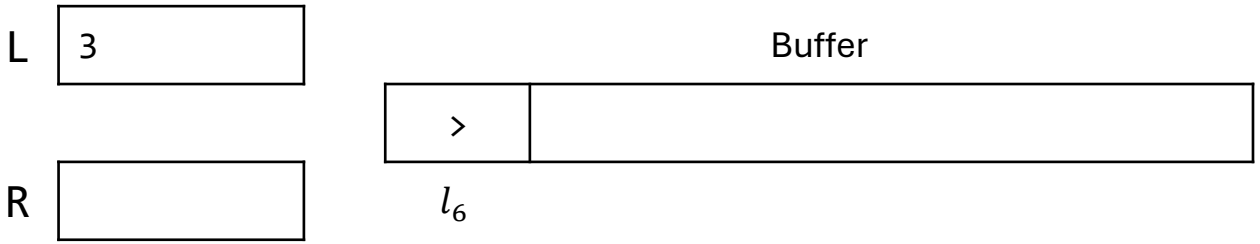
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 6.

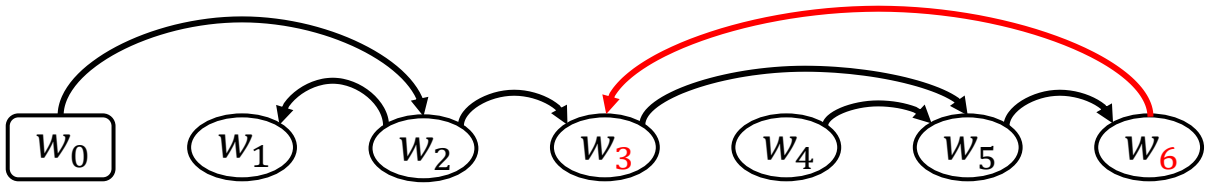


Adapted from Strzyz et al. (2019)

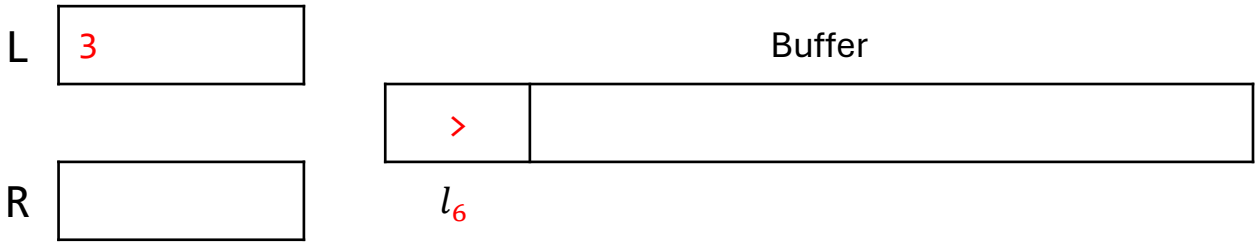
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Step 6.

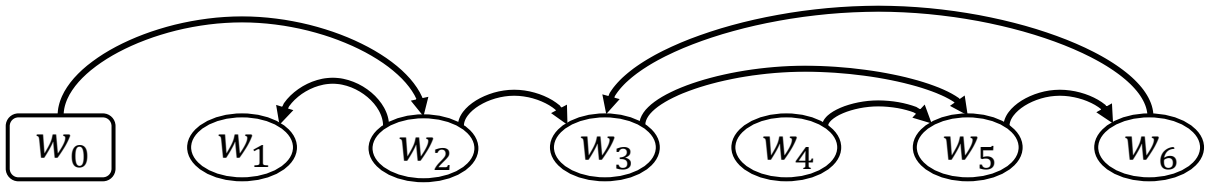


Adapted from Strzyz et al. (2019)

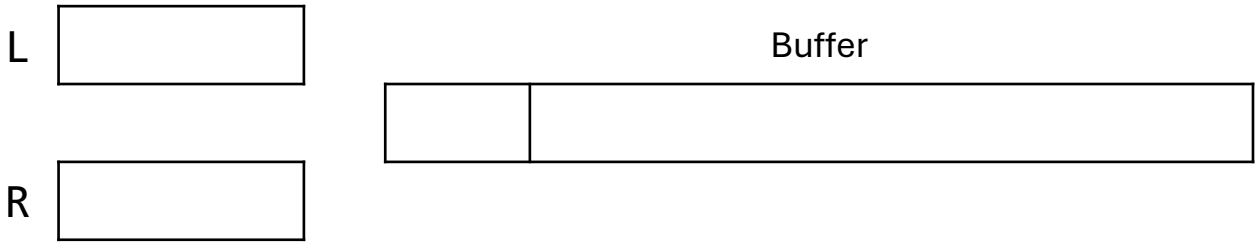
k-planar Bracketing Encoding

- Two stacks (**L** and **R**) per plane.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).

< : Add i to **L**.
/ : Add i to **R**.
**** : Resolve **L** $\leftarrow w_i$.
> : Resolve **R** $\rightarrow w_i$.



Finished!



Adapted from Strzyz et al. (2019)

Some terminology...

1-planar graph

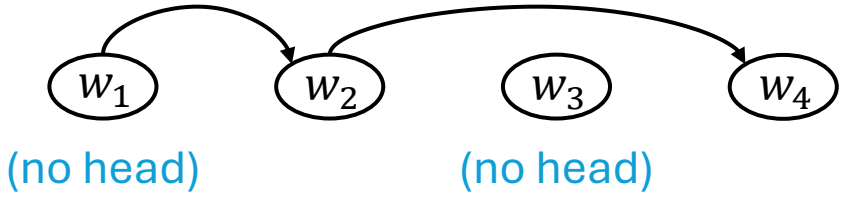
- No crossing arcs.

Relaxed 1-planar *graph*

- No crossing arcs *in the same direction*.

Relaxed 1-planar *tree*

- No crossing arcs *in the same direction*.
- *Only one* head per node.



✓ 1-planar

✓ Relaxed 1-planar graph

✗ Relaxed 1-planar tree

Some terminology...

1-planar graph

- No crossing arcs.

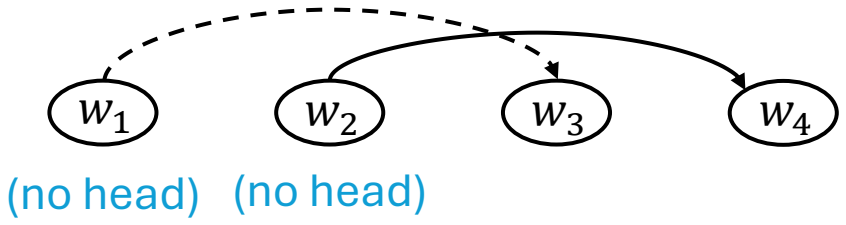
Relaxed 1-planar *graph*

- No crossing arcs *in the same direction*.

Relaxed 1-planar *tree*

- No crossing arcs *in the same direction*.
- *Only one* head per node.

(crossing arcs in the same direction)



✗ 1-planar

✗ Relaxed 1-planar graph

✗ Relaxed 1-planar tree

Some terminology...

1-planar graph

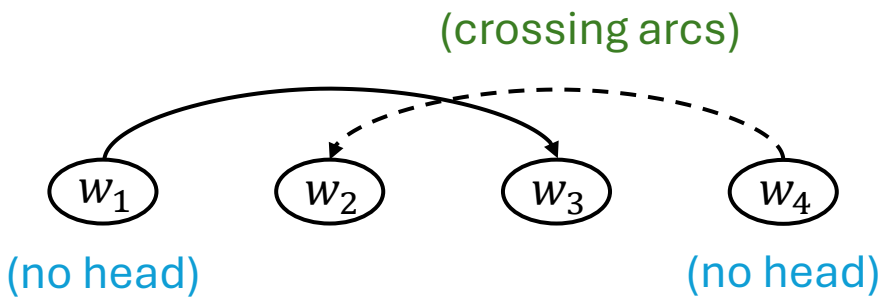
- No crossing arcs.

Relaxed 1-planar *graph*

- No crossing arcs *in the same direction*.

Relaxed 1-planar *tree*

- No crossing arcs *in the same direction*.
- *Only one* head per node.



✗ 1-planar

✓ Relaxed 1-planar graph

✗ Relaxed 1-planar tree

Some terminology...

1-planar graph

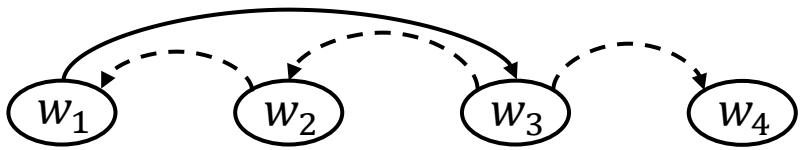
- No crossing arcs.

Relaxed 1-planar *graph*

- No crossing arcs *in the same direction*.

Relaxed 1-planar *tree*

- No crossing arcs *in the same direction*.
- *Only one* head per node.



✓ 1-planar

✓ Relaxed 1-planar graph

✓ Relaxed 1-planar tree

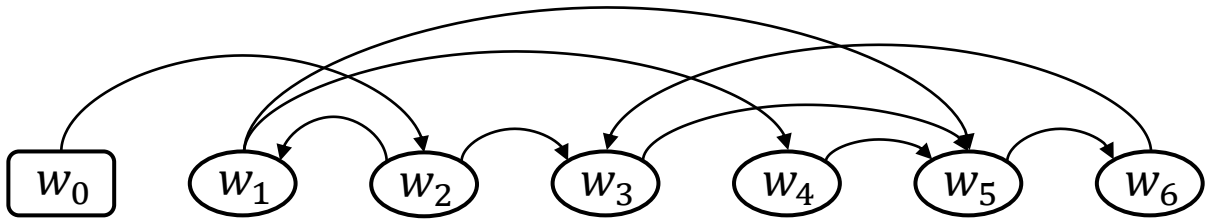
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

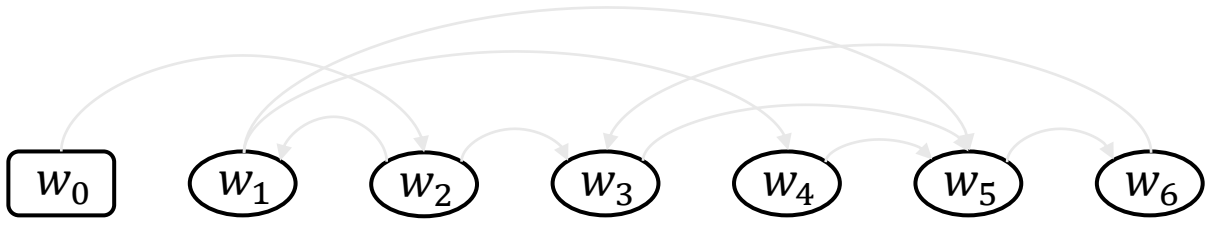
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

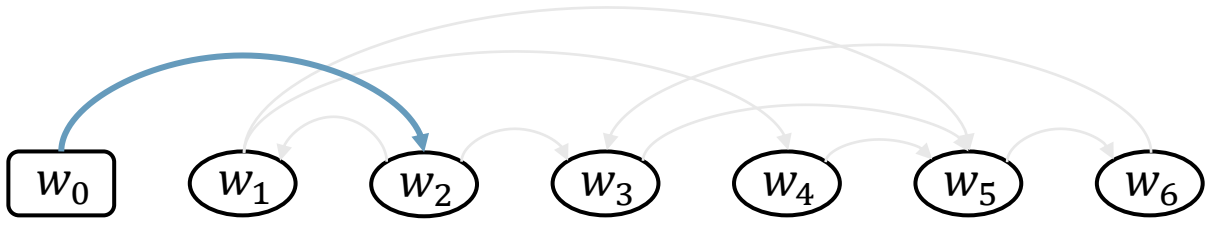
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

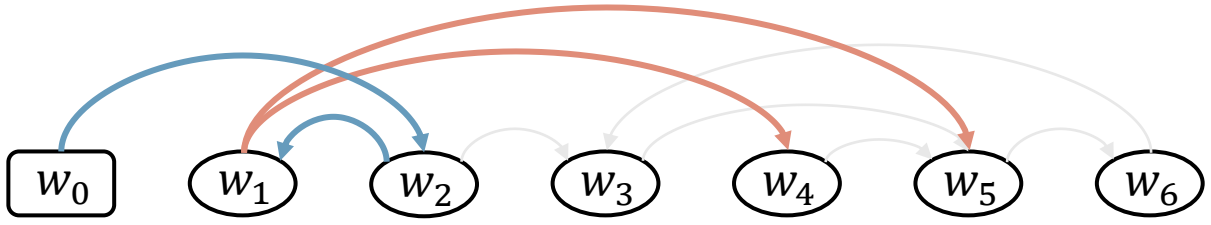
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

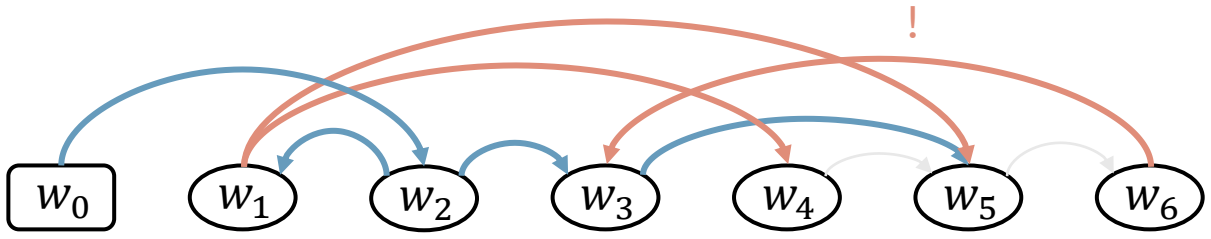
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

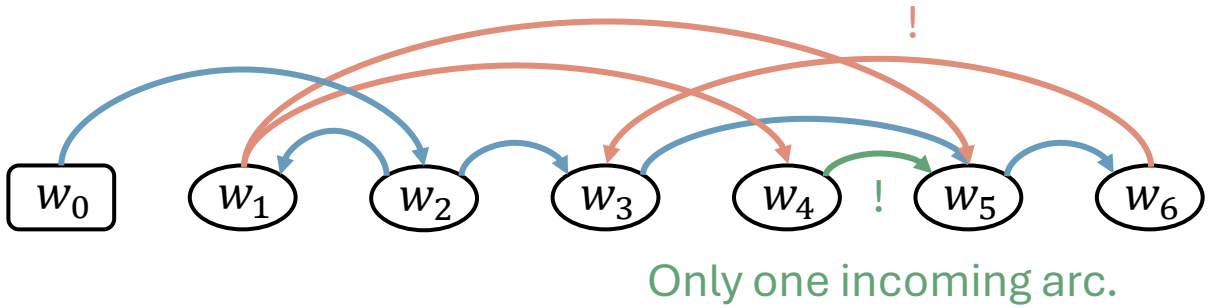
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 1: Distribute the arcs in **relaxed 1-planar trees.**



Sort and incrementally assign!

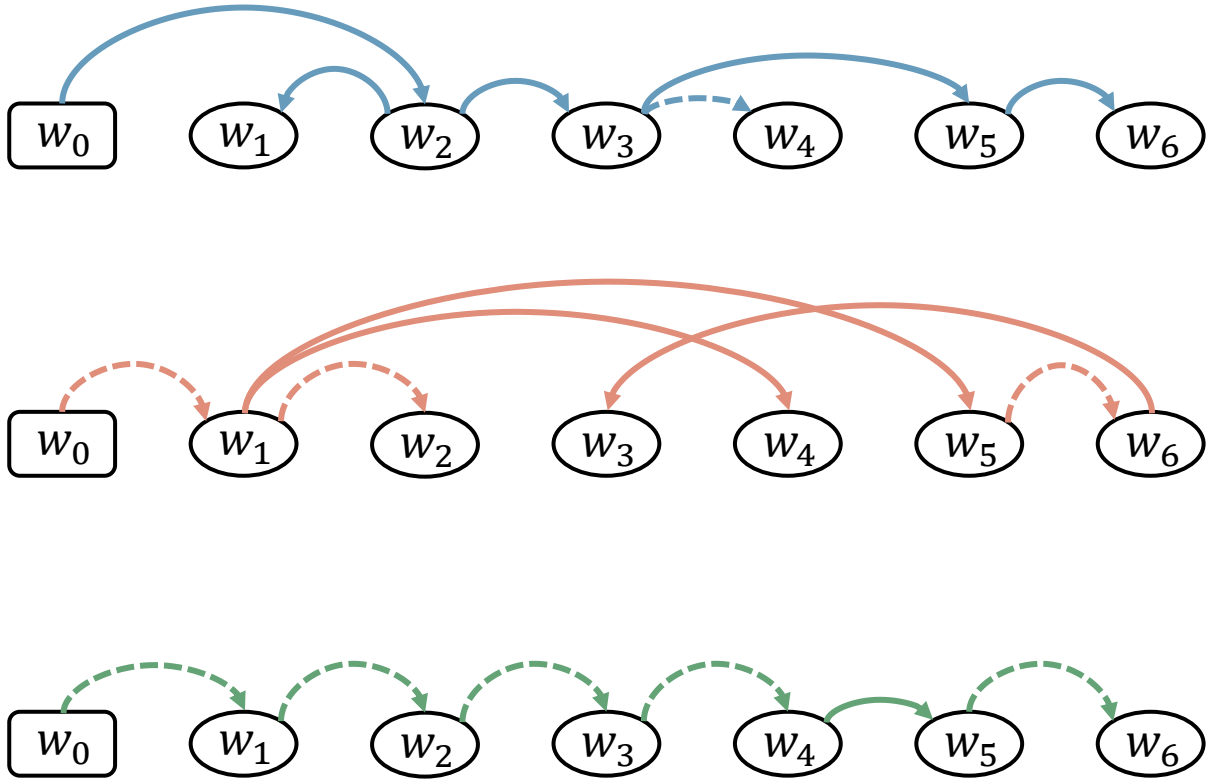
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 2: Each plane is a tree. Create artificial heads to those nodes without incoming arcs.



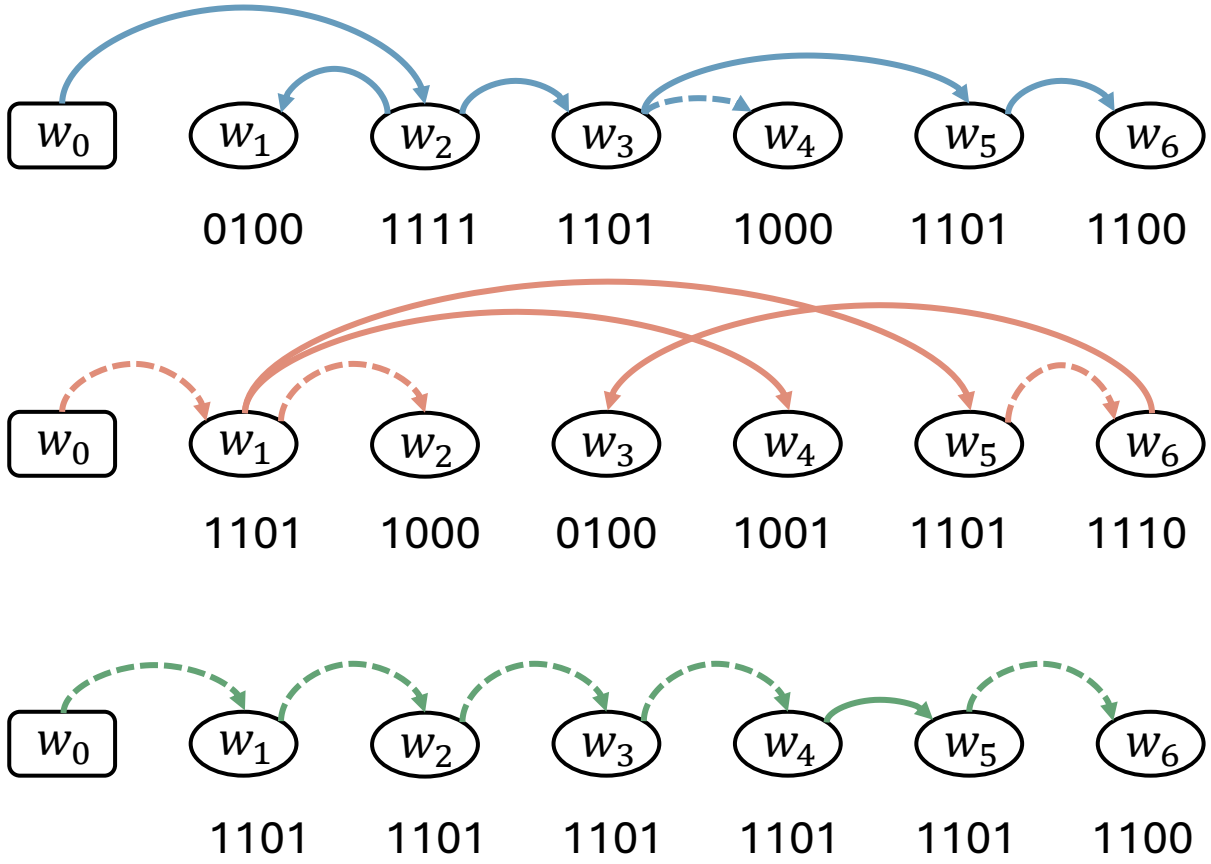
$4k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar trees.**
- Encode each label ℓ_i with 4 bits: $\mathbf{b_0 b_1 b_2 b_3}$.

$\mathbf{b_0}$	**0**	w_i has a right head.
	1	w_i has a left head.
$\mathbf{b_1}$		w_i is the outermost dependent.
$\mathbf{b_2}$		w_i has left dependents.
$\mathbf{b_3}$		w_i has right dependents.

Step 3. Assign labels.



$4k$-bit Encoding

What about decoding?

$4k$-bit Encoding

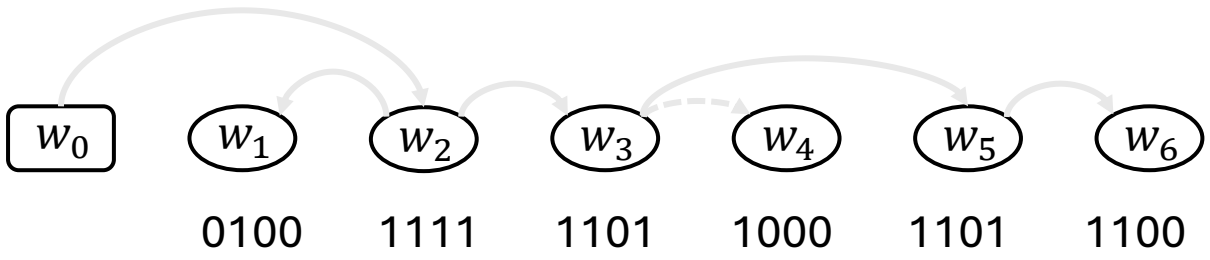
- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.

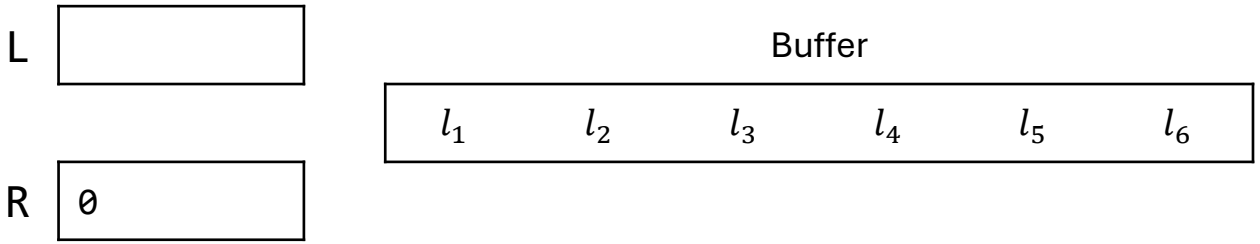
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



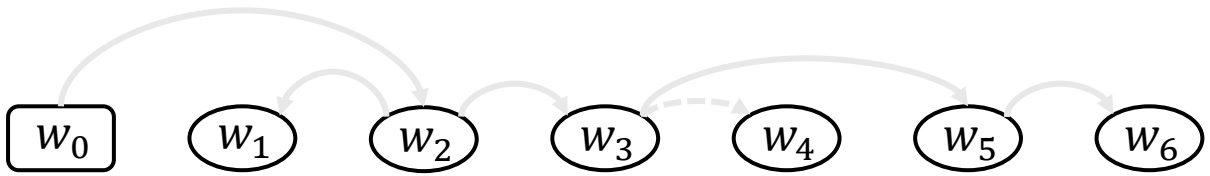
Initial state: Labels to buffer and **R** with the root dependency.



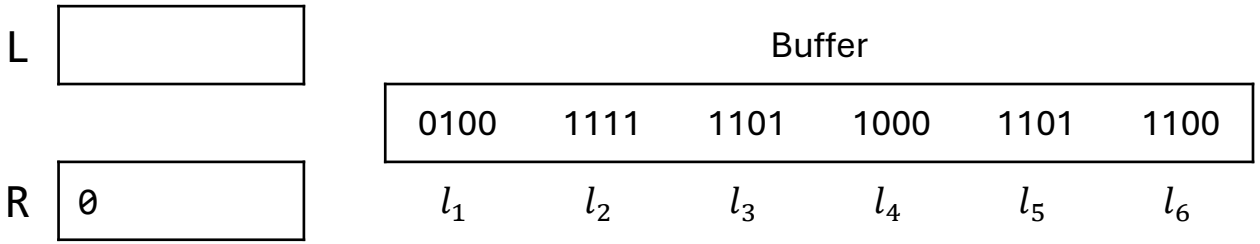
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



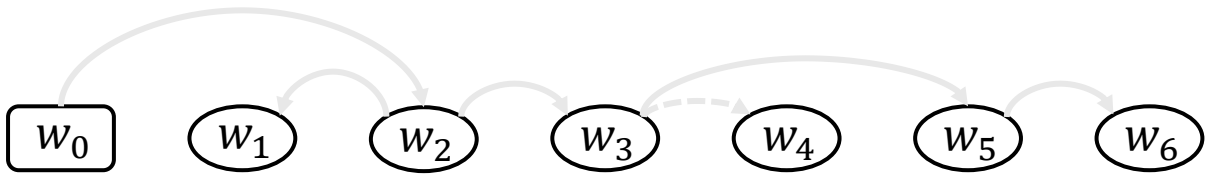
Initial state: Labels to buffer and **R** with the root dependency.



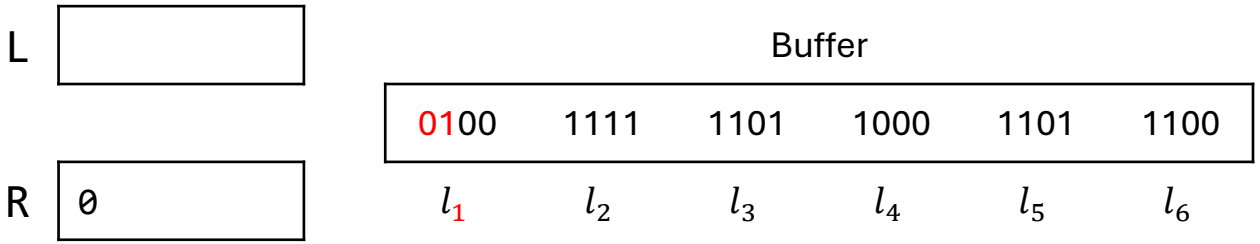
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



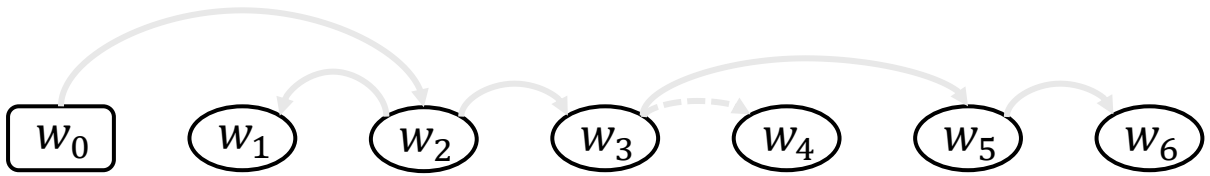
Initial state: Labels to buffer and **R** with the root dependency.



$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



Initial state: Labels to buffer and **R** with the root dependency.

L 1_1

R 0

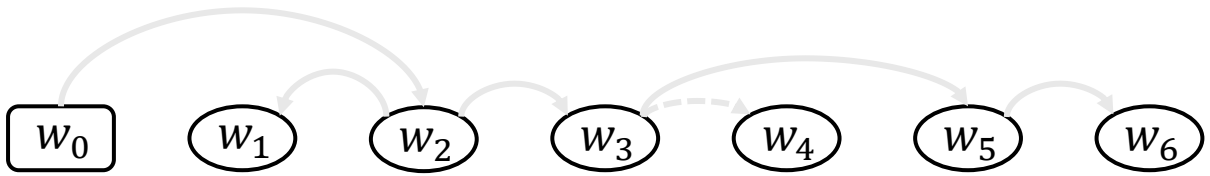
Buffer

0100	1111	1101	1000	1101	1100
l_1	l_2	l_3	l_4	l_5	l_6

$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



Initial state: Labels to buffer and **R** with the root dependency.

L 1_1

R 0

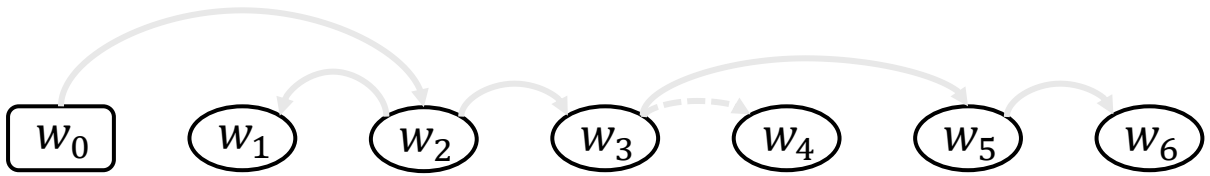
Buffer

0100	1111	1101	1000	1101	1100
l_1	l_2	l_3	l_4	l_5	l_6

$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



Initial state: Labels to buffer and **R** with the root dependency.

L 1_1

R 0

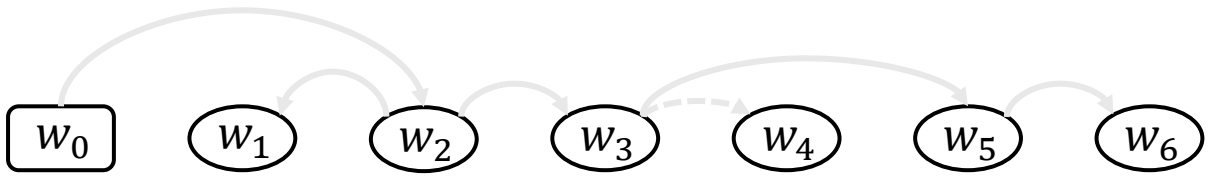
Buffer

	1111	1101	1000	1101	1100
l_1	l_2	l_3	l_4	l_5	l_6

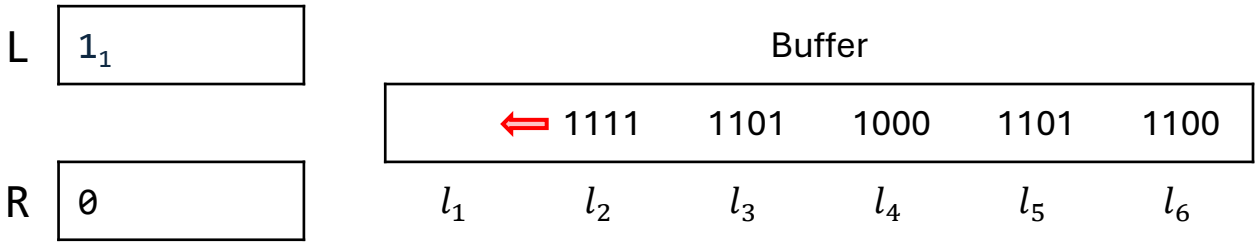
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



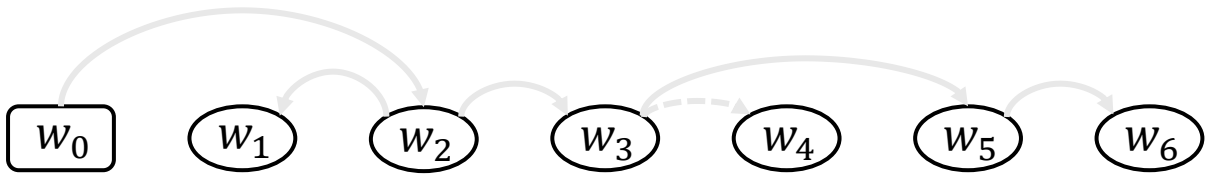
Initial state: Labels to buffer and **R** with the root dependency.



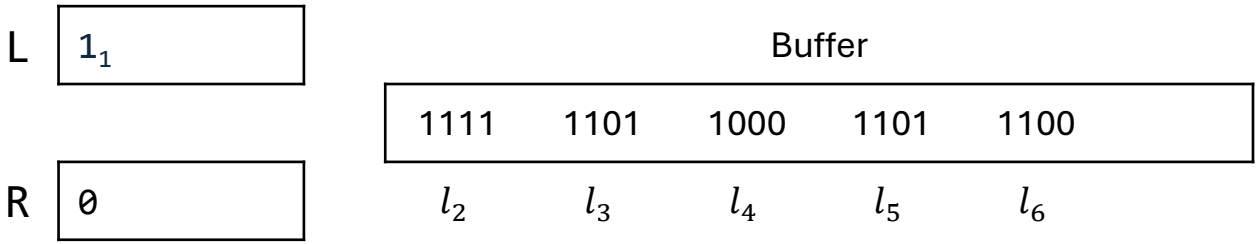
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



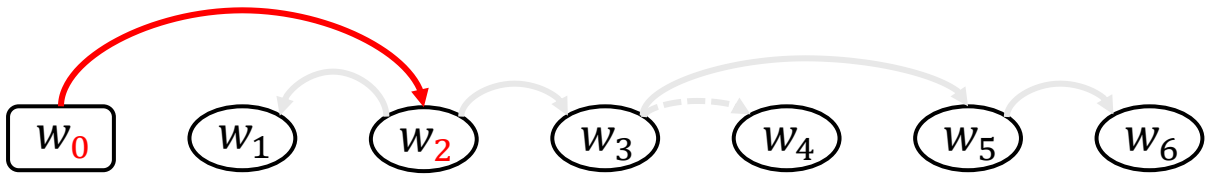
Initial state: Labels to buffer and **R** with the root dependency.



$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



Initial state: Labels to buffer and **R** with the root dependency.

L 1_1

R 0

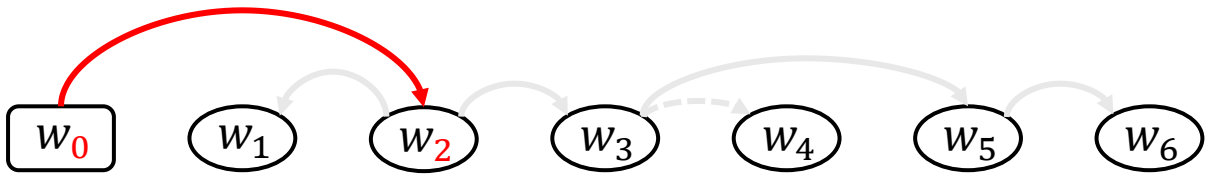
Buffer

1111	1101	1000	1101	1100
l_2	l_3	l_4	l_5	l_6

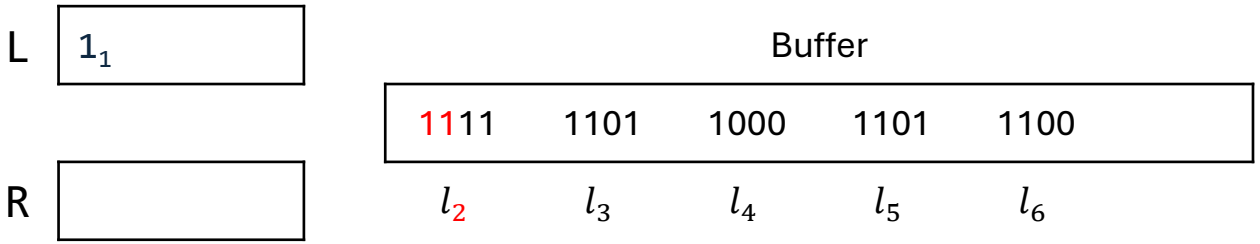
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



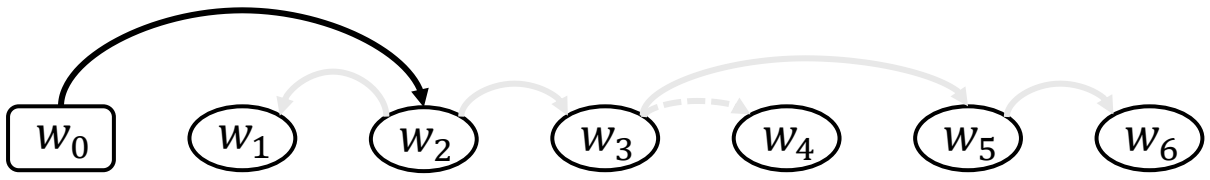
Initial state: Labels to buffer and **R** with the root dependency.



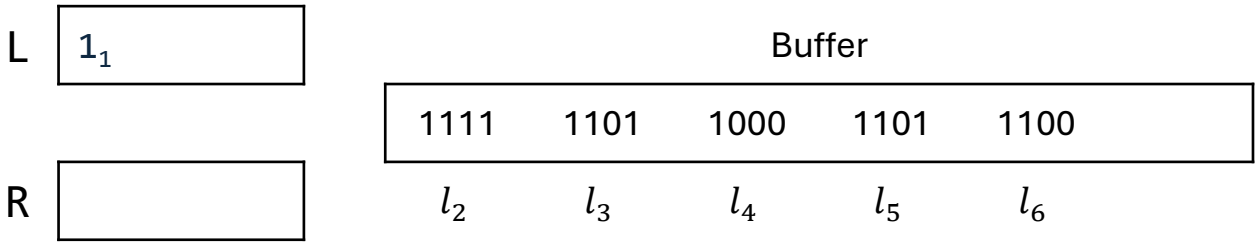
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



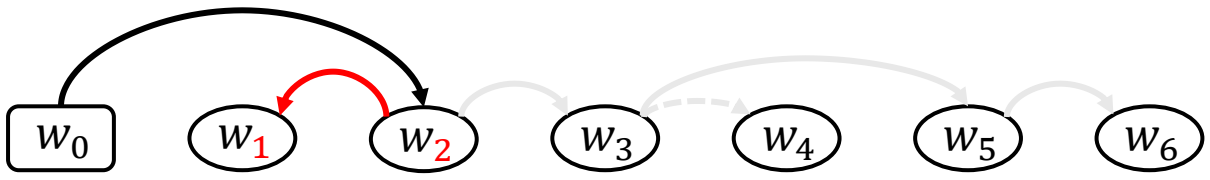
Initial state: Labels to buffer and **R** with the root dependency.



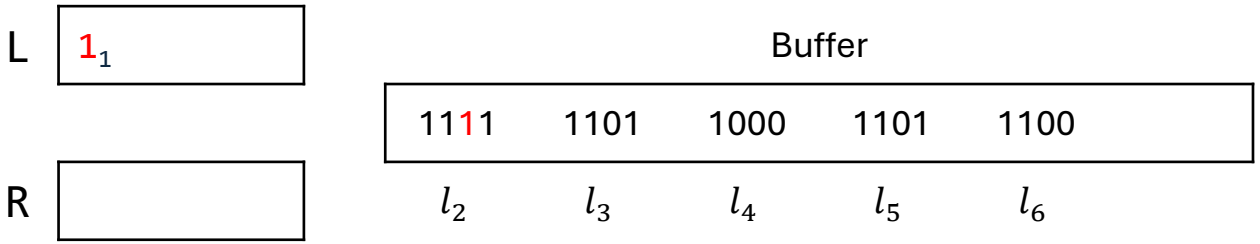
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



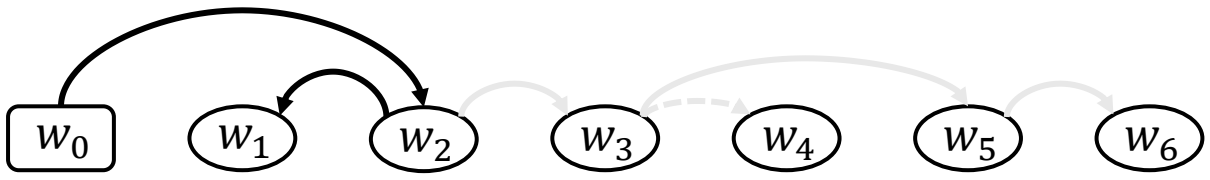
Initial state: Labels to buffer and **R** with the root dependency.



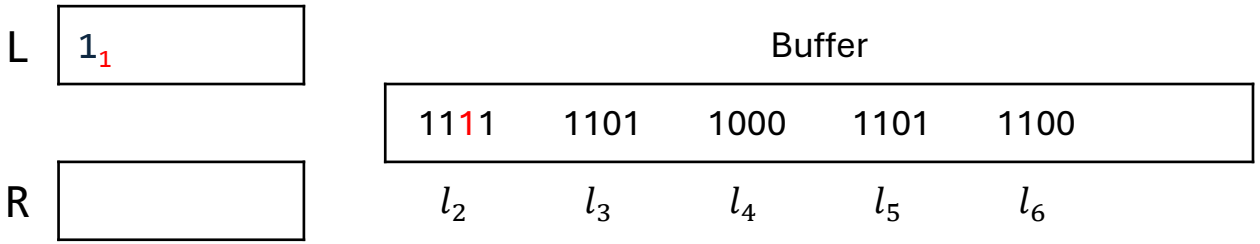
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



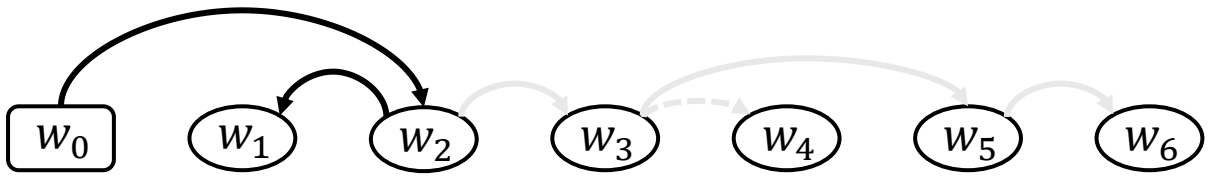
Initial state: Labels to buffer and **R** with the root dependency.



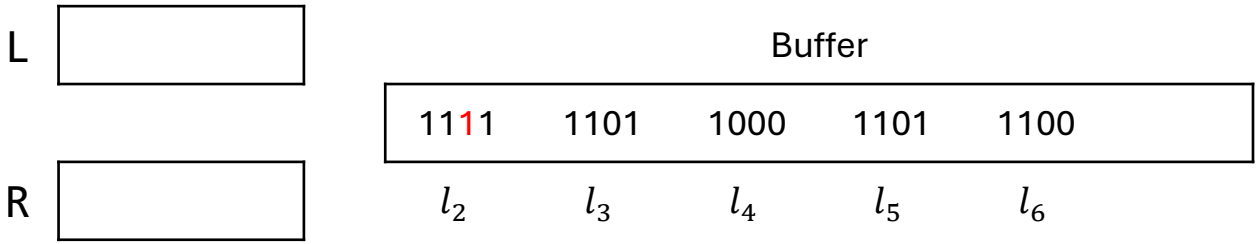
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



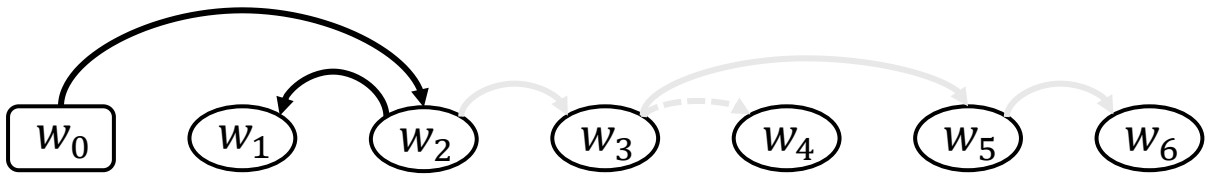
Initial state: Labels to buffer and **R** with the root dependency.



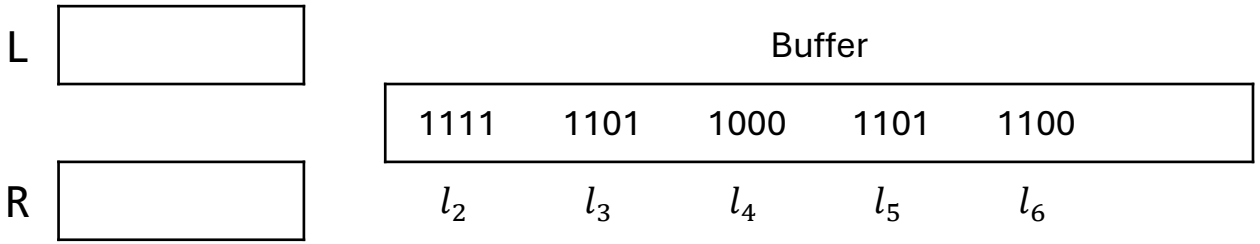
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



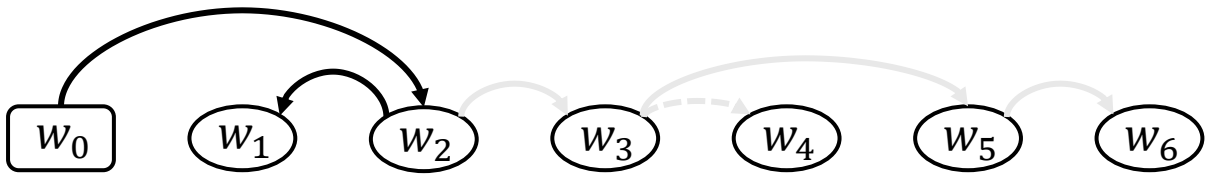
Initial state: Labels to buffer and **R** with the root dependency.



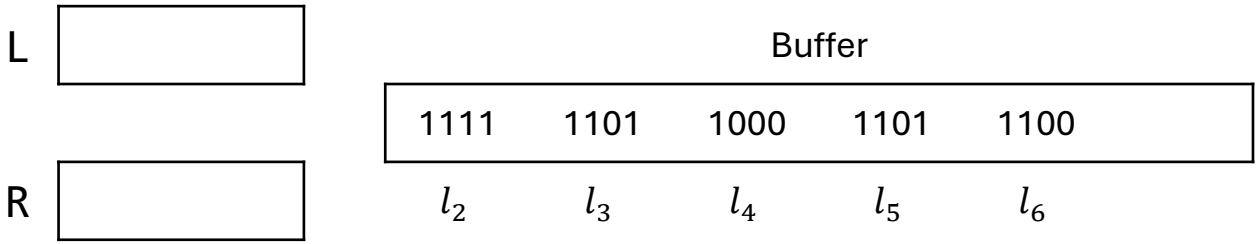
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



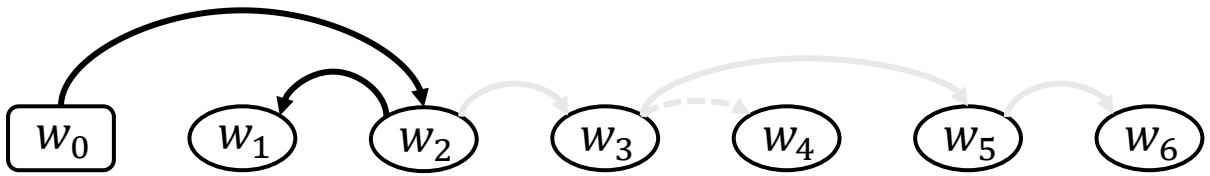
Initial state: Labels to buffer and **R** with the root dependency.



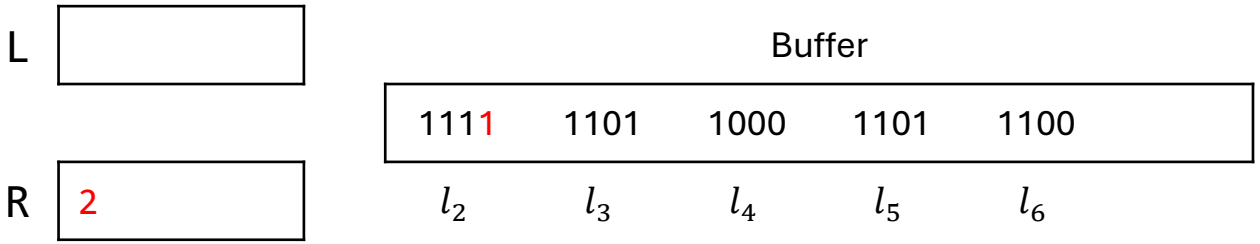
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



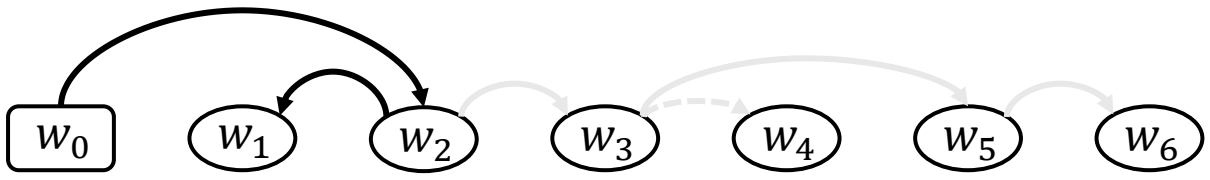
Initial state: Labels to buffer and **R** with the root dependency.



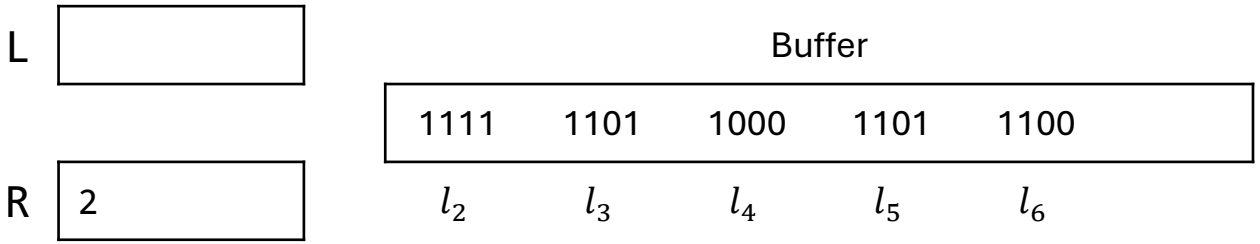
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



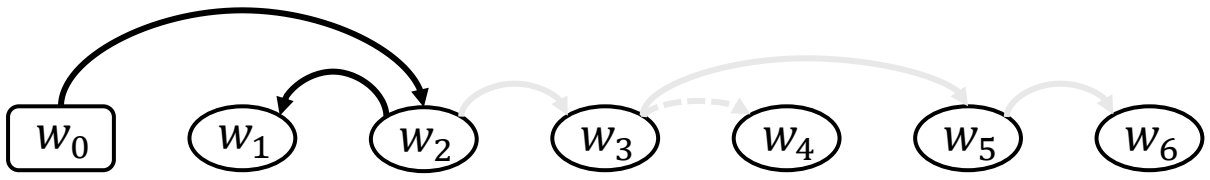
Initial state: Labels to buffer and **R** with the root dependency.



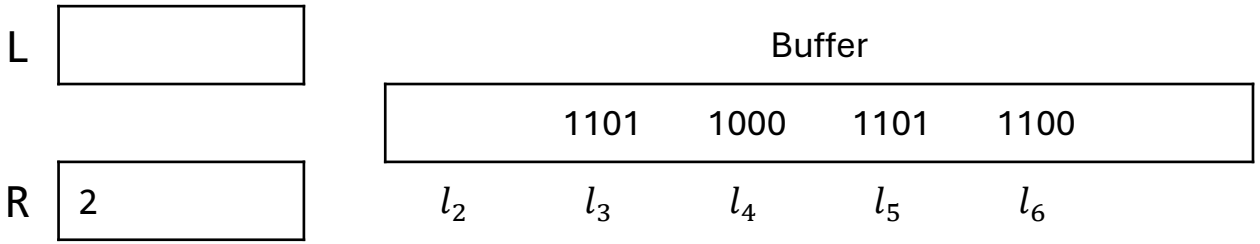
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



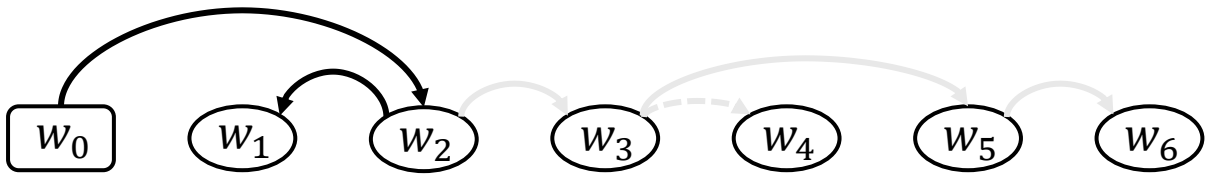
Initial state: Labels to buffer and **R** with the root dependency.



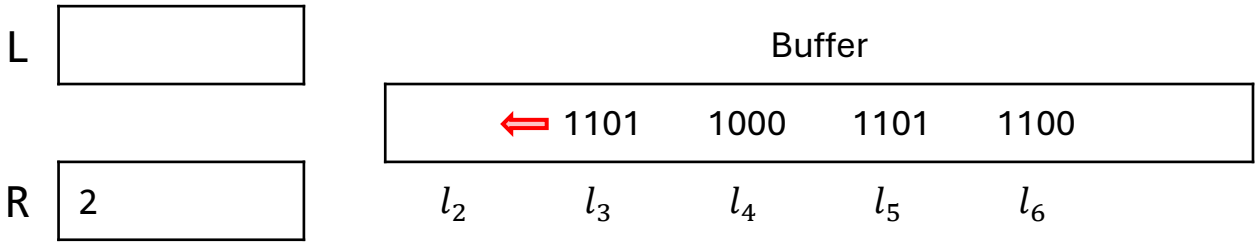
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



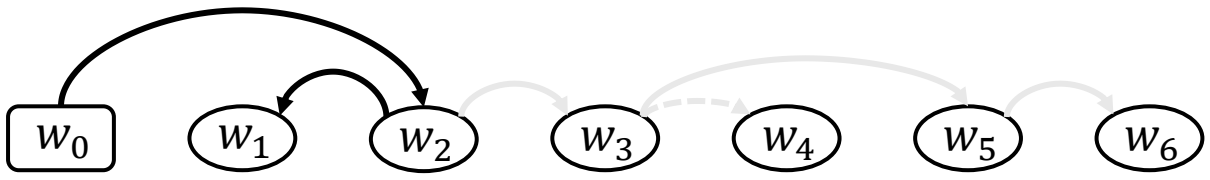
Initial state: Labels to buffer and **R** with the root dependency.



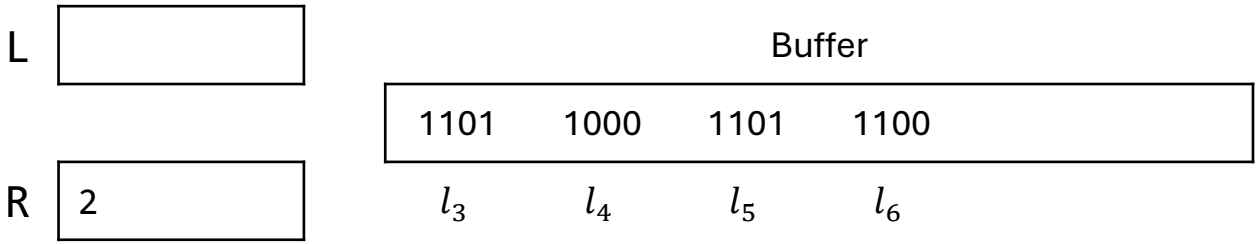
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



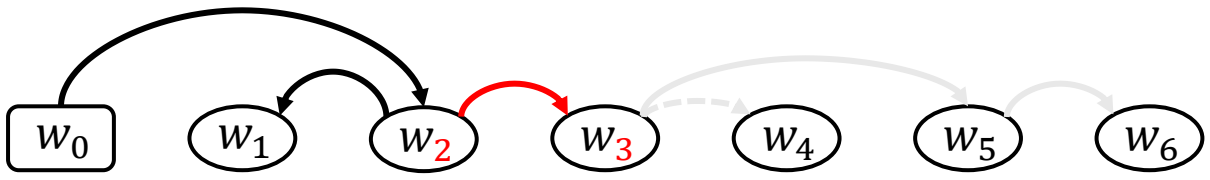
Initial state: Labels to buffer and **R** with the root dependency.



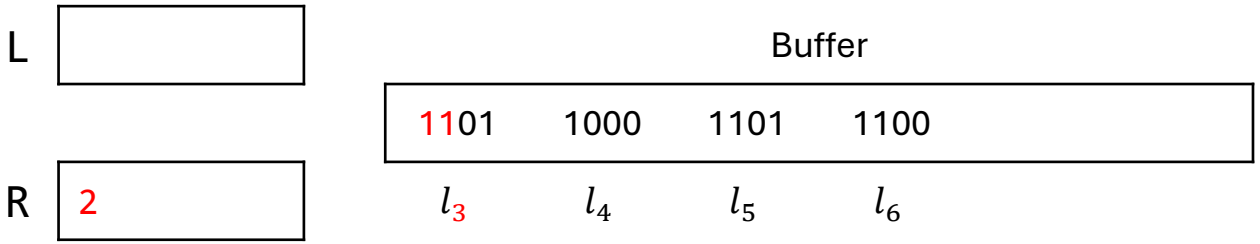
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



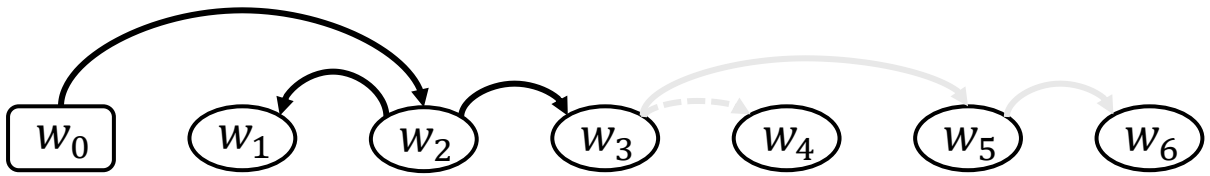
Initial state: Labels to buffer and **R** with the root dependency.



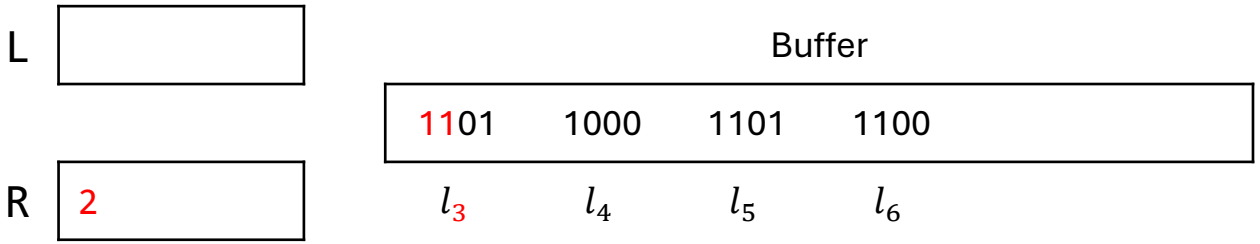
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



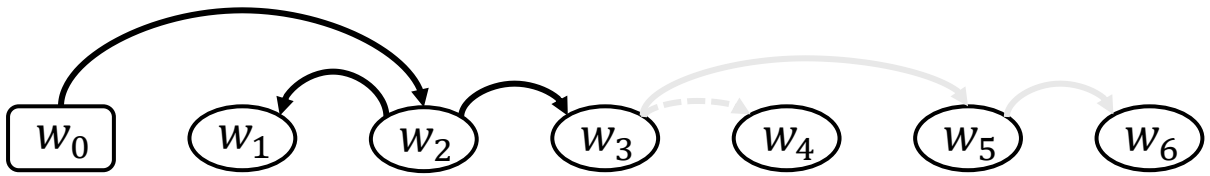
Initial state: Labels to buffer and **R** with the root dependency.



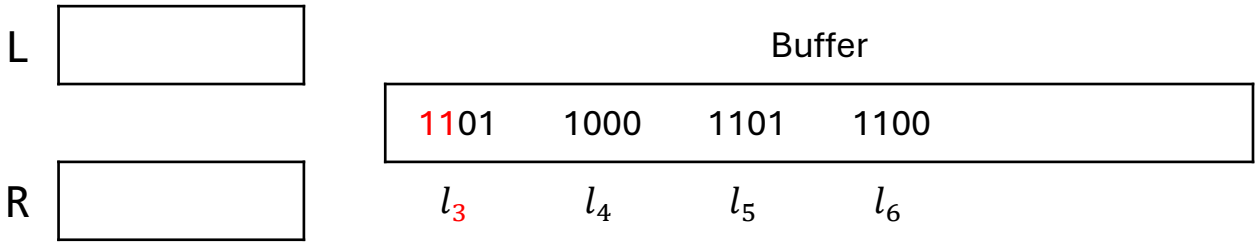
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



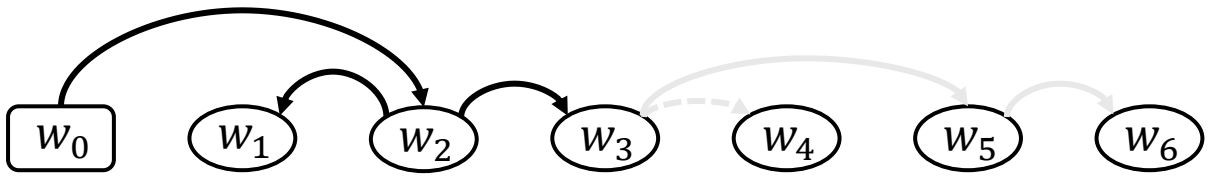
Initial state: Labels to buffer and **R** with the root dependency.



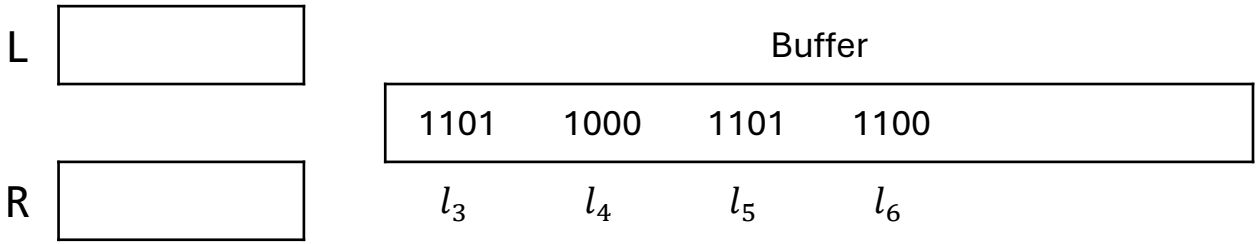
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



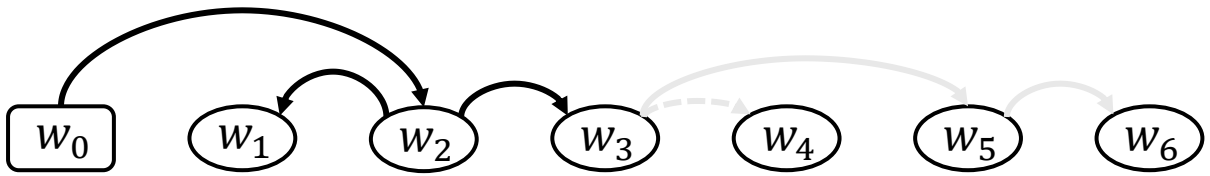
Initial state: Labels to buffer and **R** with the root dependency.



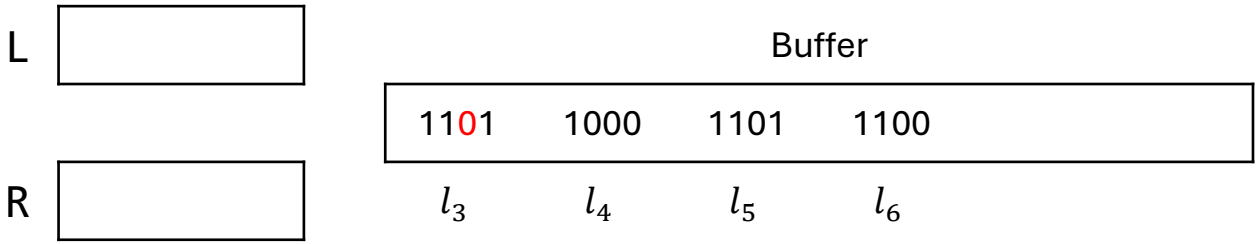
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



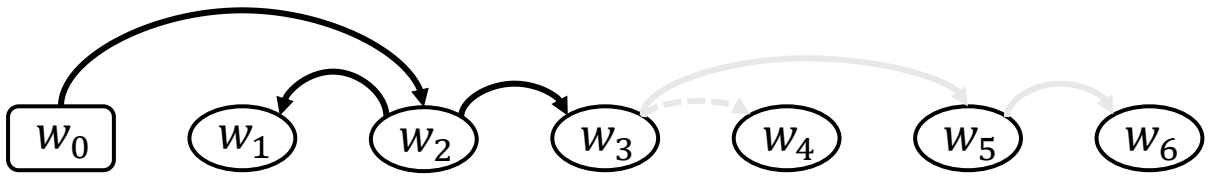
Initial state: Labels to buffer and **R** with the root dependency.



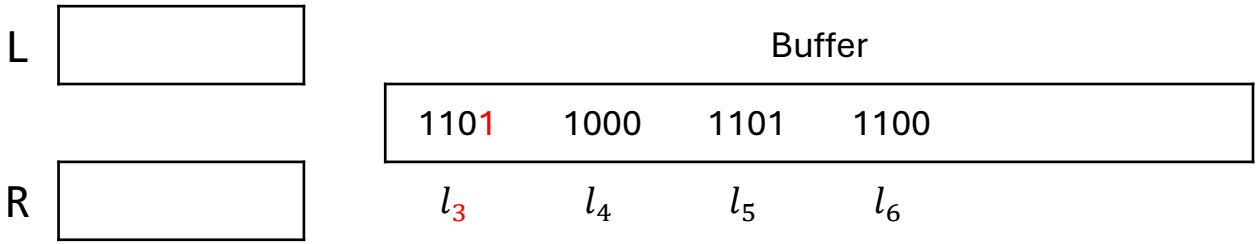
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



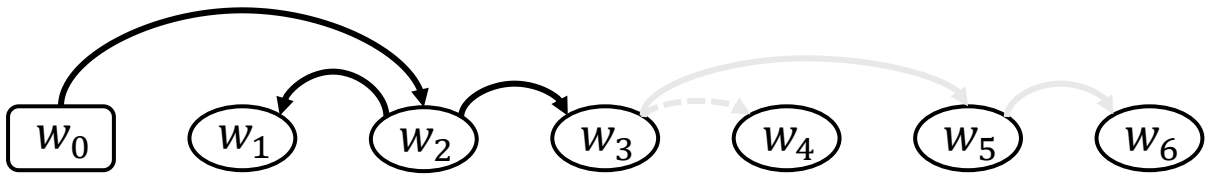
Initial state: Labels to buffer and **R** with the root dependency.



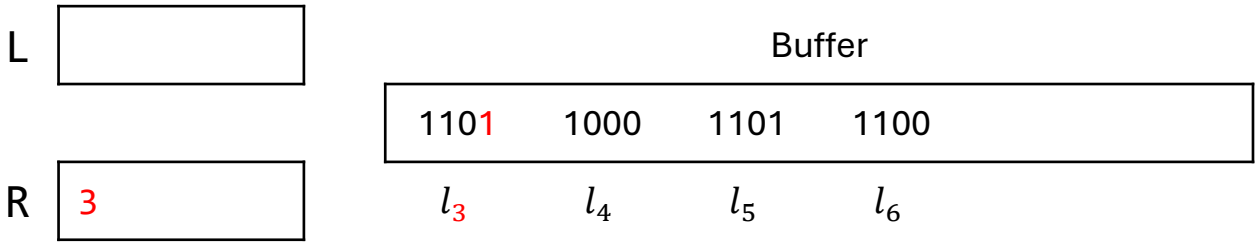
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



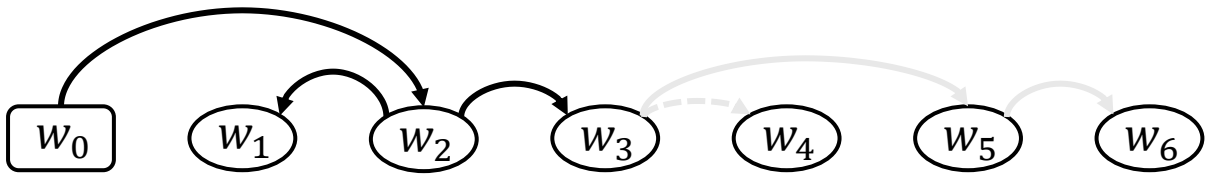
Initial state: Labels to buffer and **R** with the root dependency.



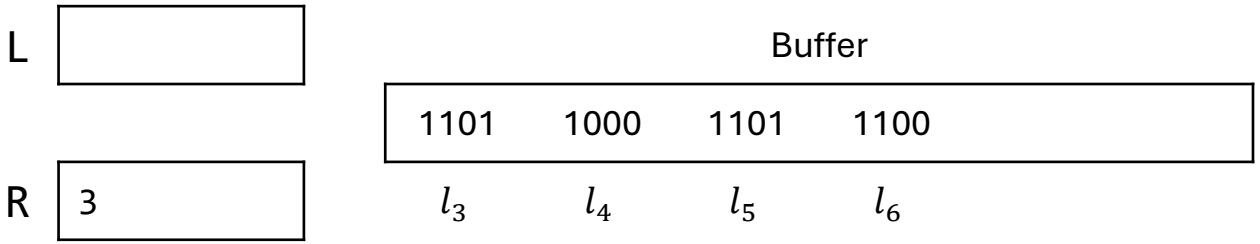
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



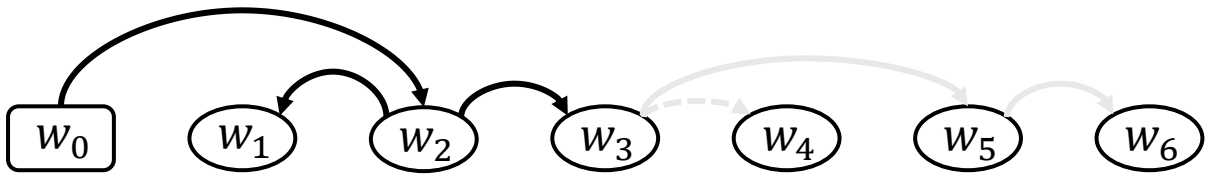
Initial state: Labels to buffer and **R** with the root dependency.



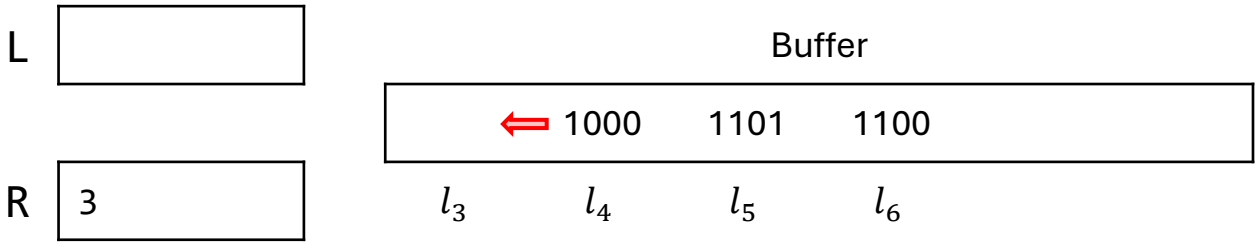
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



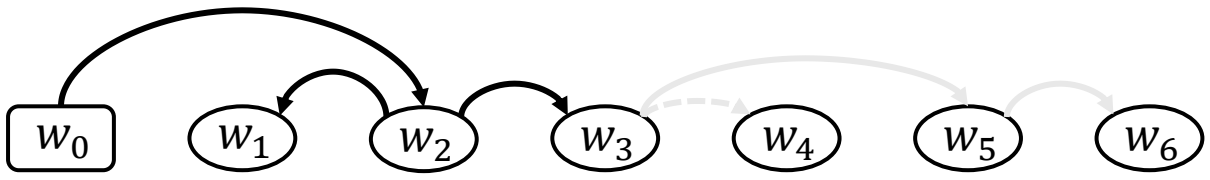
Initial state: Labels to buffer and **R** with the root dependency.



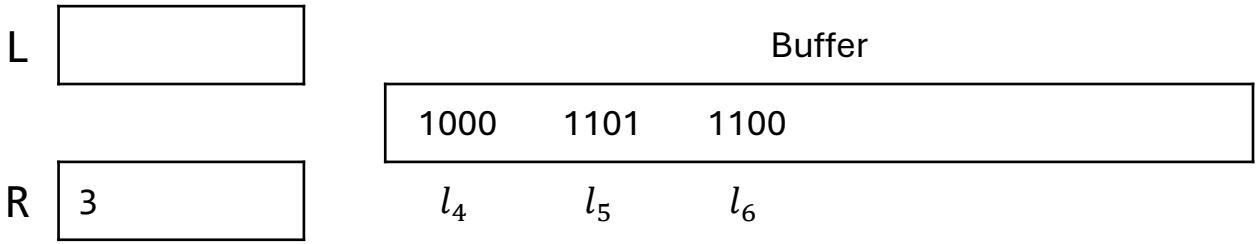
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



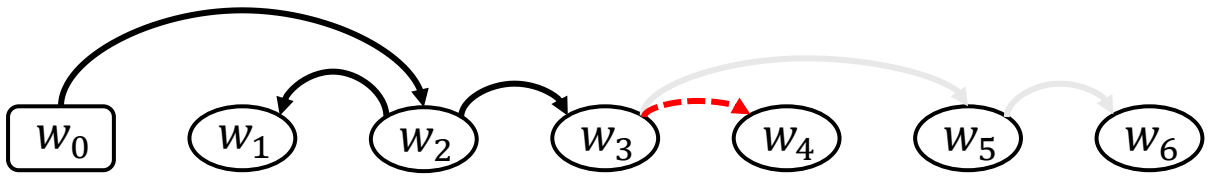
Initial state: Labels to buffer and **R** with the root dependency.



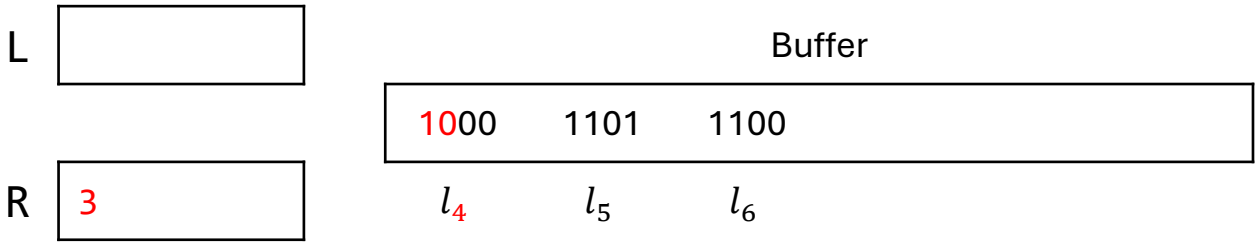
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** → w_i.
	11	Resolve **R** → w_i and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p}$ ← w_i and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



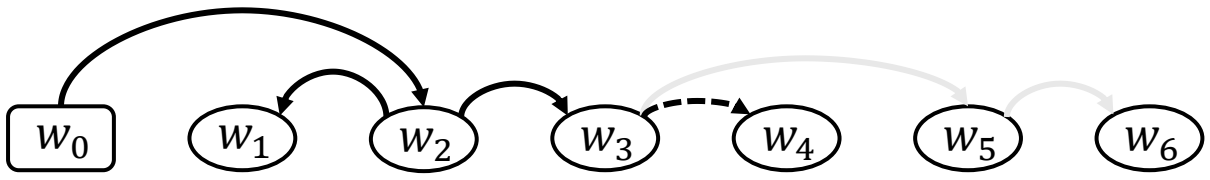
Initial state: Labels to buffer and **R** with the root dependency.



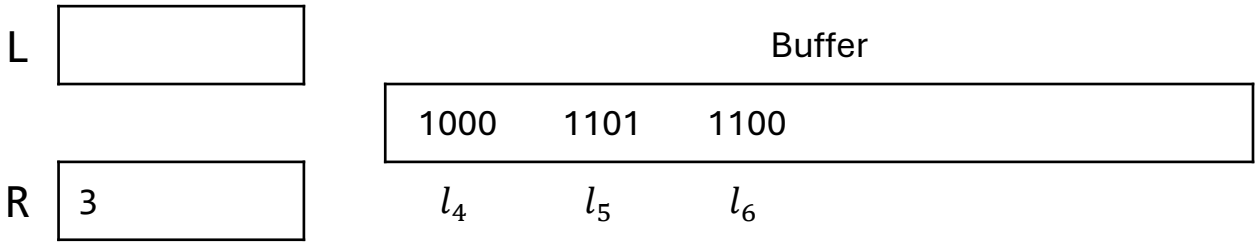
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



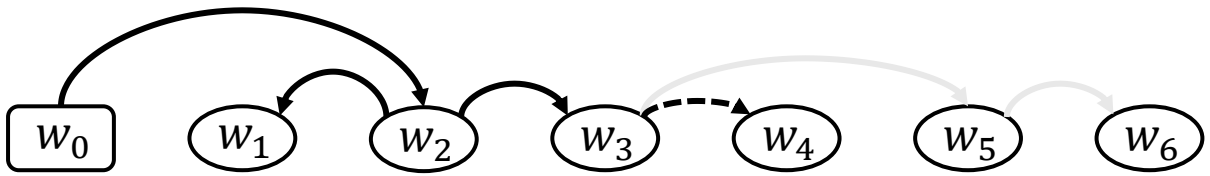
Initial state: Labels to buffer and **R** with the root dependency.



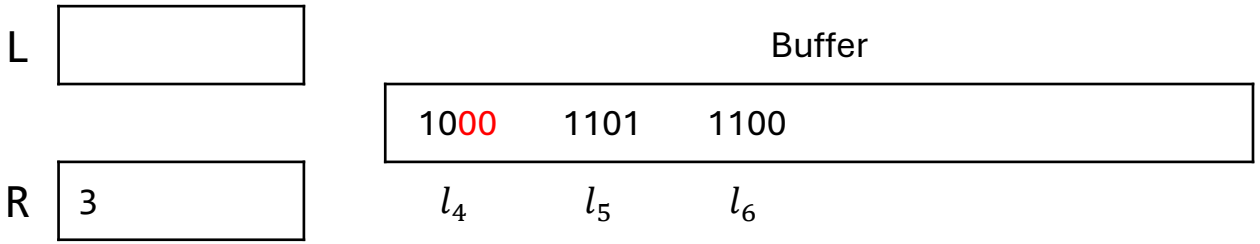
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



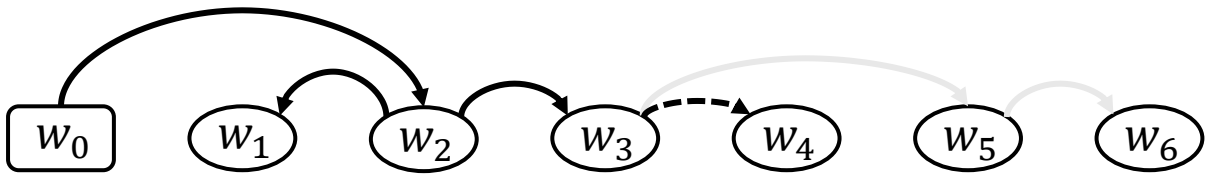
Initial state: Labels to buffer and **R** with the root dependency.



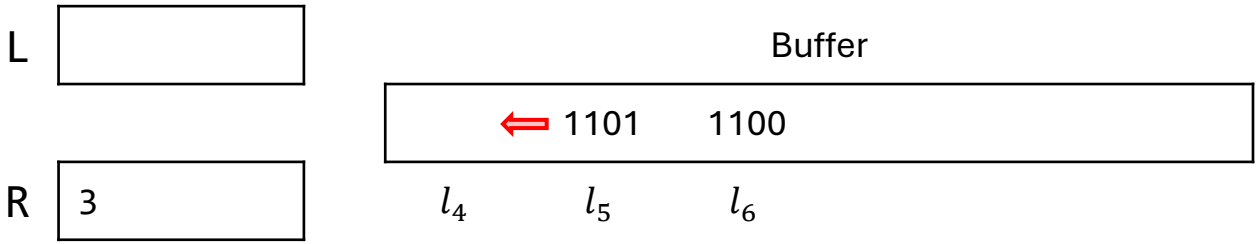
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



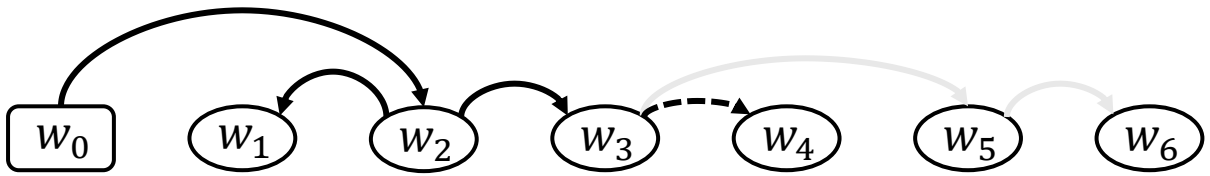
Initial state: Labels to buffer and **R** with the root dependency.



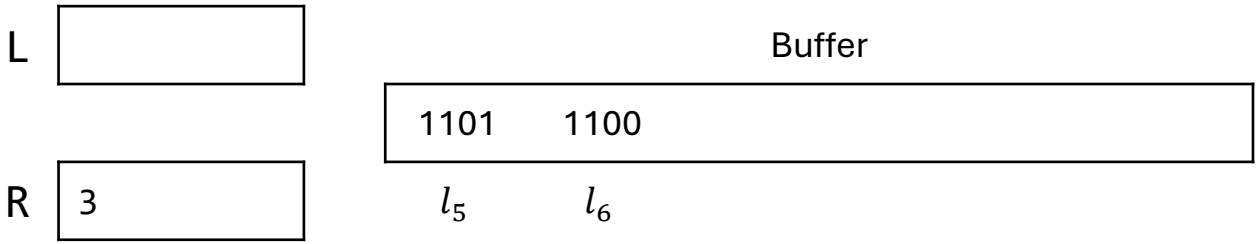
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



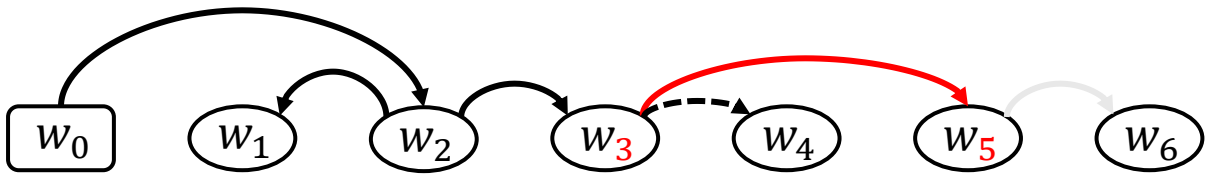
Initial state: Labels to buffer and **R** with the root dependency.



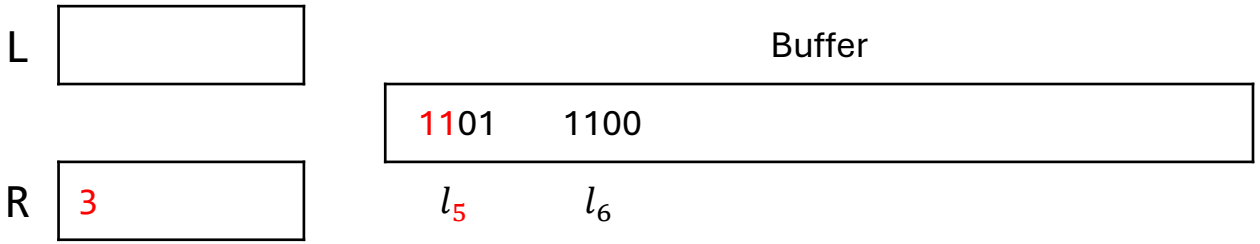
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



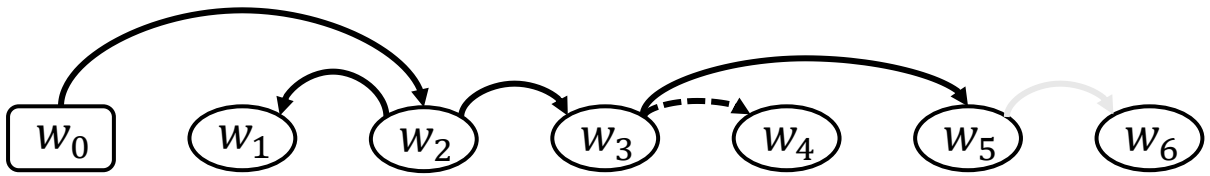
Initial state: Labels to buffer and **R** with the root dependency.



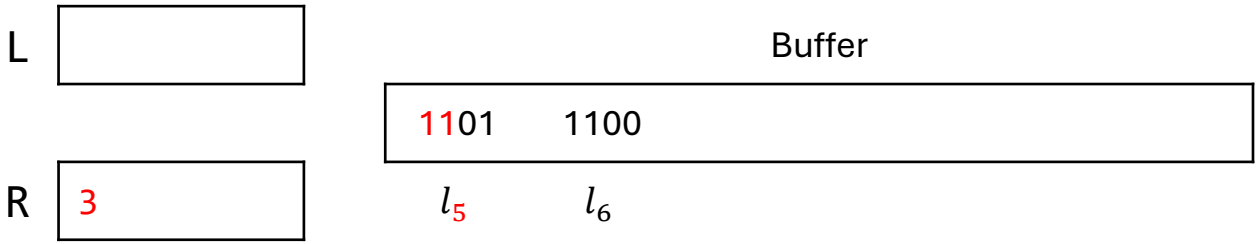
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



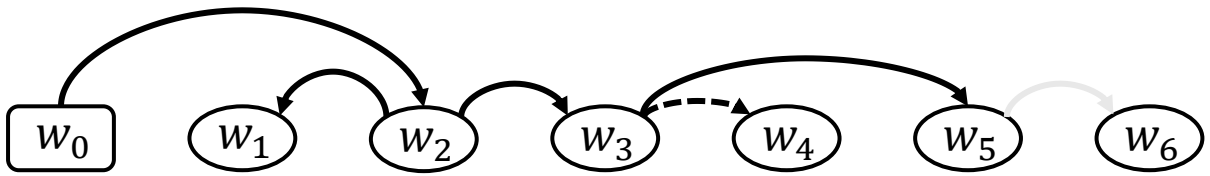
Initial state: Labels to buffer and **R** with the root dependency.



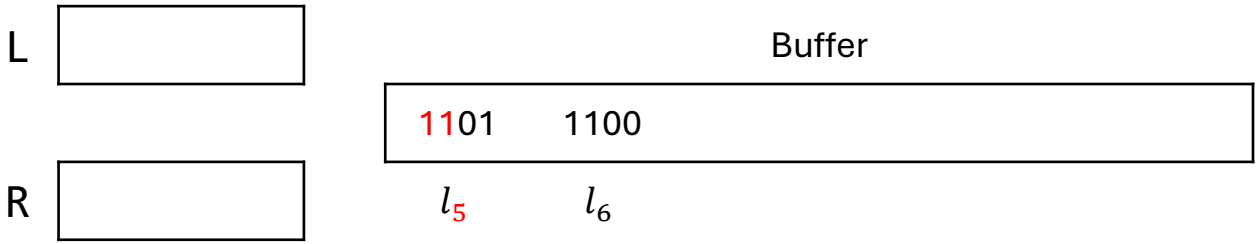
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



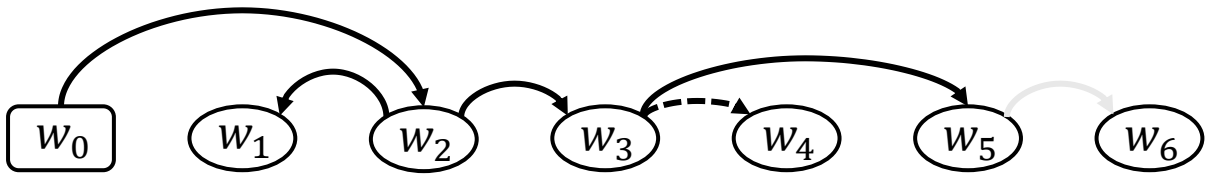
Initial state: Labels to buffer and **R** with the root dependency.



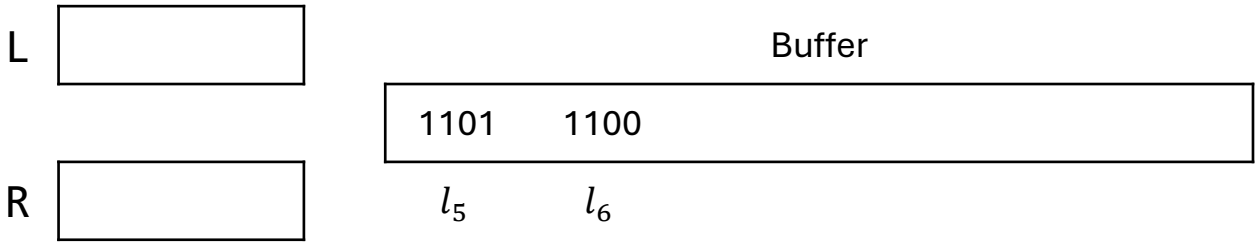
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



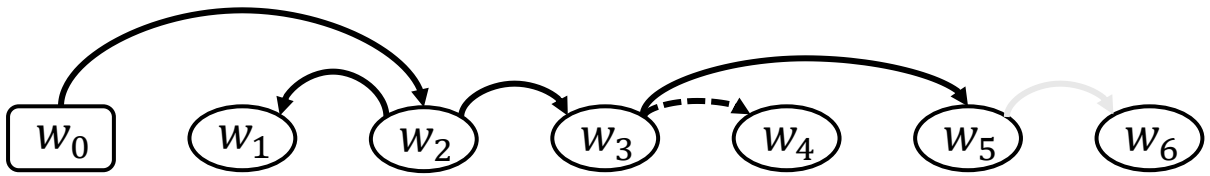
Initial state: Labels to buffer and **R** with the root dependency.



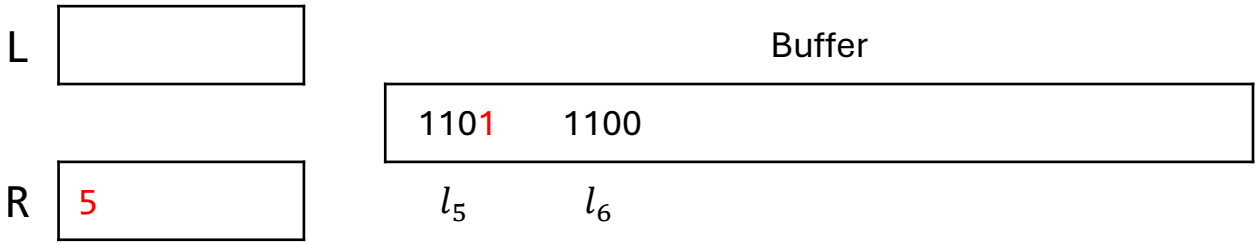
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



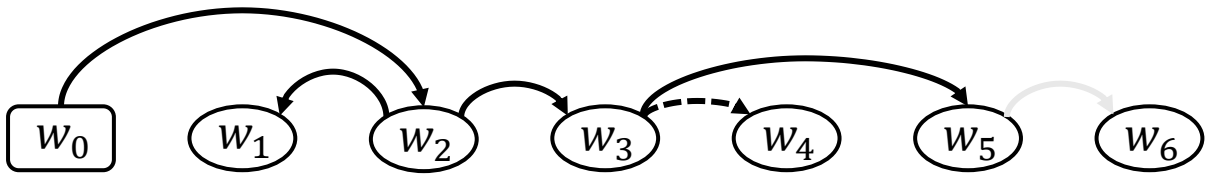
Initial state: Labels to buffer and **R** with the root dependency.



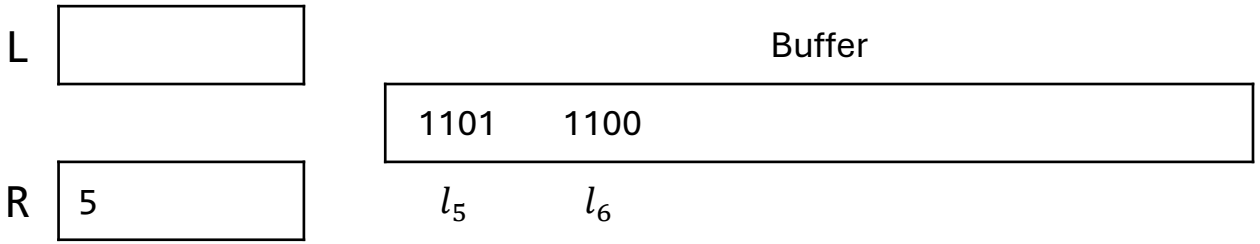
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



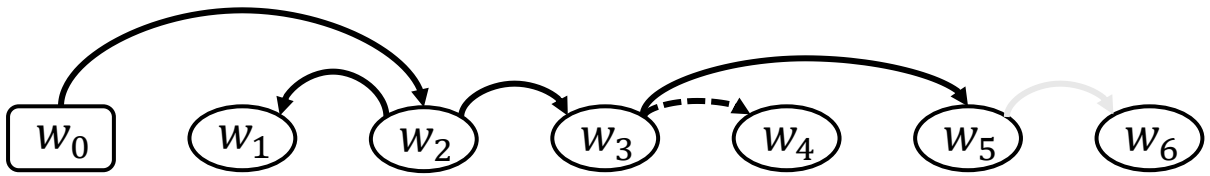
Initial state: Labels to buffer and **R** with the root dependency.



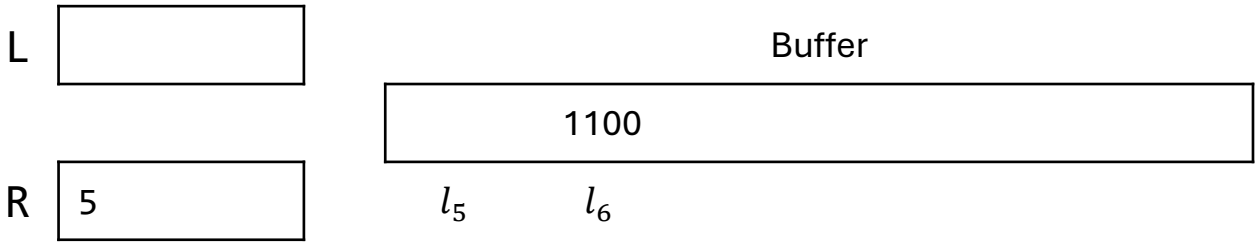
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



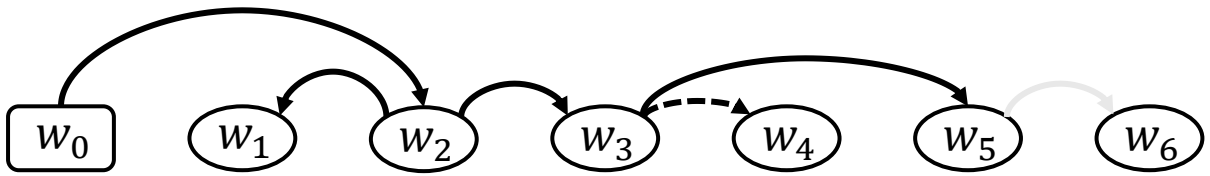
Initial state: Labels to buffer and **R** with the root dependency.



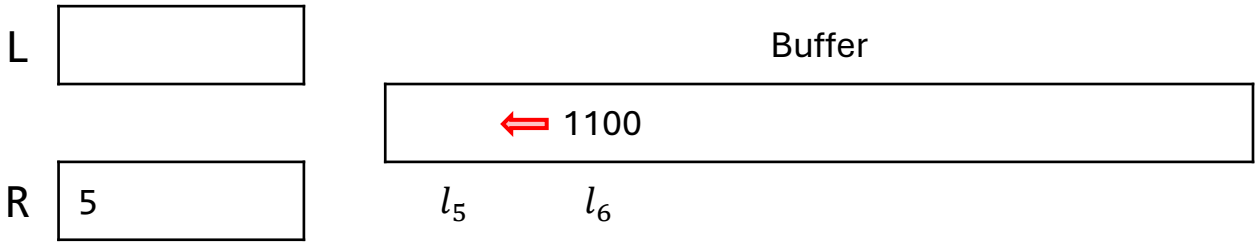
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



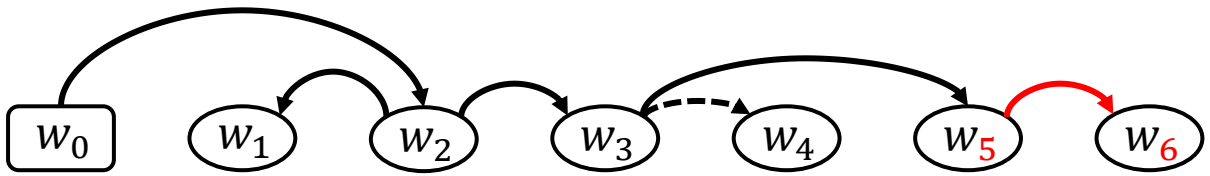
Initial state: Labels to buffer and **R** with the root dependency.



$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.

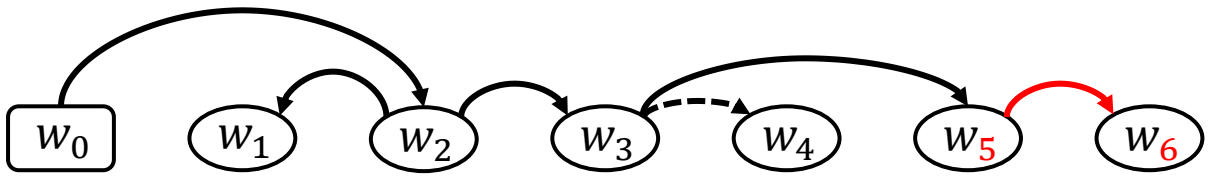


Initial state: Labels to buffer and **R** with the root dependency.

$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.

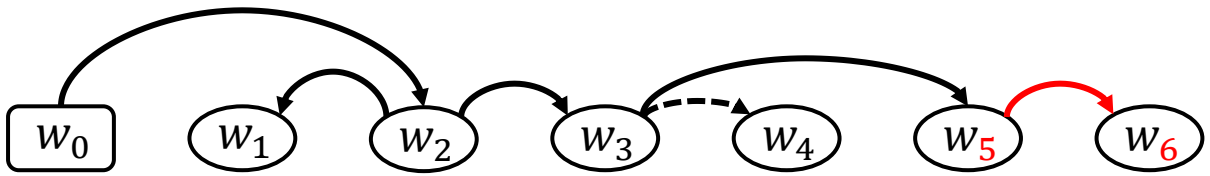


Initial state: Labels to buffer and **R** with the root dependency.

$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.

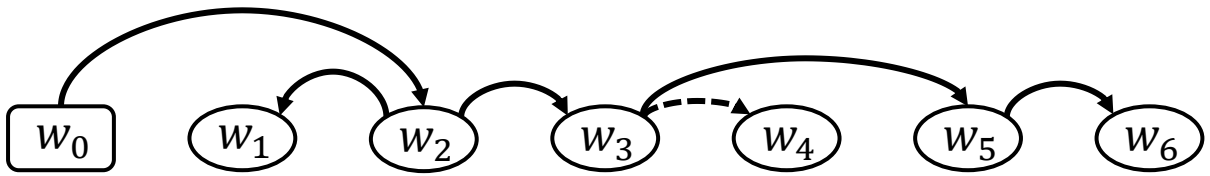


Initial state: Labels to buffer and **R** with the root dependency.

$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.

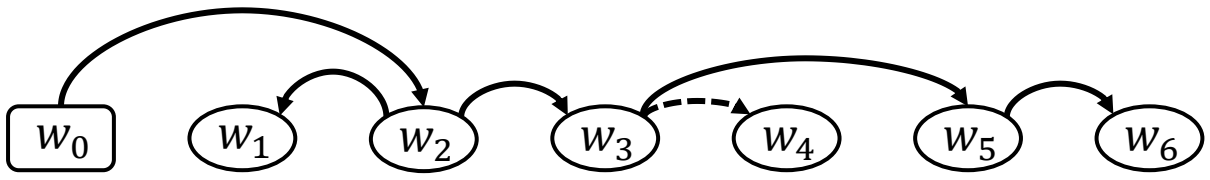


Initial state: Labels to buffer and **R** with the root dependency.

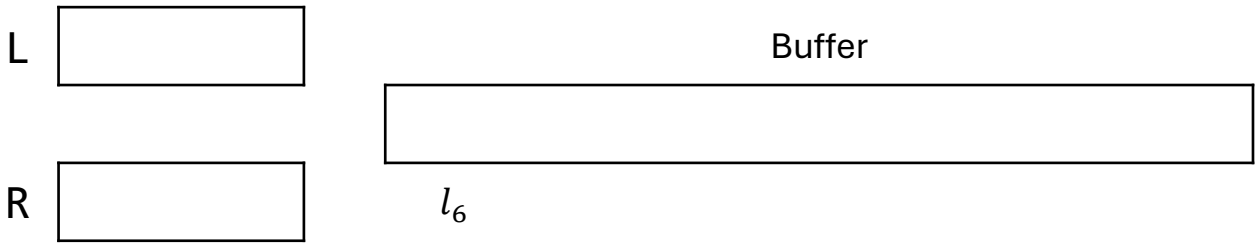
$4k$-bit Encoding

- Same as bracketing-encoding.
- **L** for left arcs (**<**)and **R** right arcs (**/>**).
- Store i and $\mathbf{b_1}$.

$\mathbf{b_0 b_1}$	**00**	Add i_0 to **L**.
	01	Add i_1 to **L**.
	10	Resolve **R** $\rightarrow w_i$.
	11	Resolve **R** $\rightarrow w_i$ and pop **R**.
$\mathbf{b_2}$	**1**	Resolve $\mathbf{L_p} \leftarrow w_i$ and pop if **p**.
$\mathbf{b_3}$	**1**	Add i to **R**.



Finished!



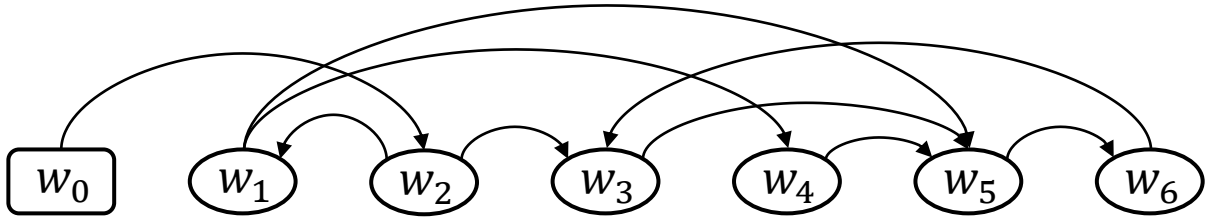
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



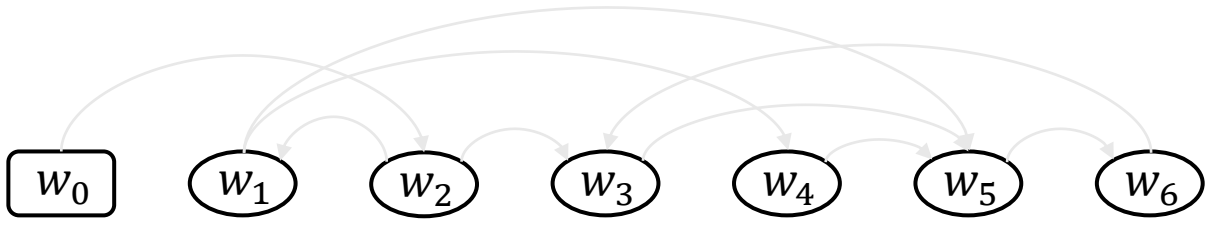
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



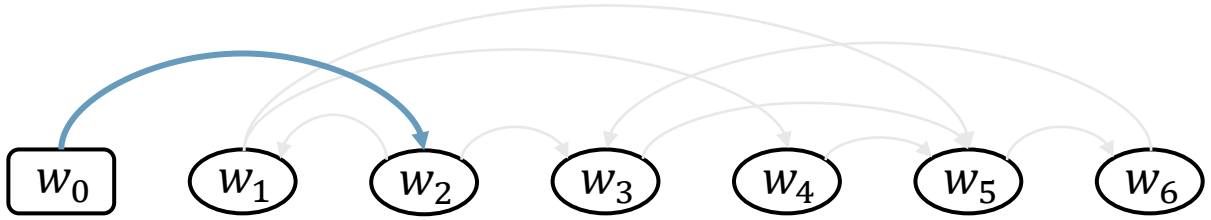
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



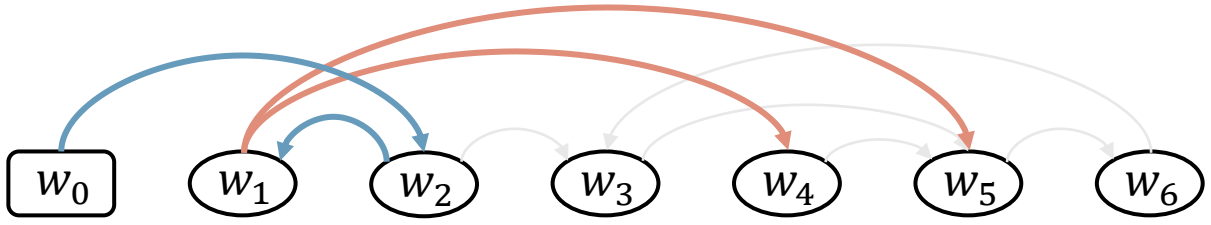
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



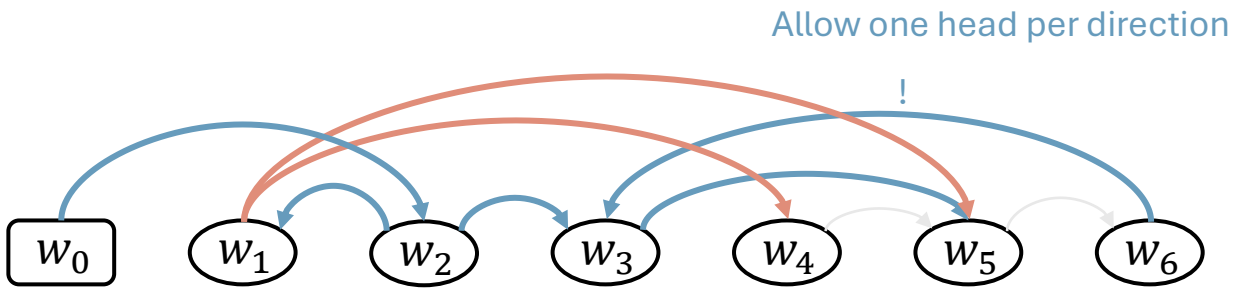
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



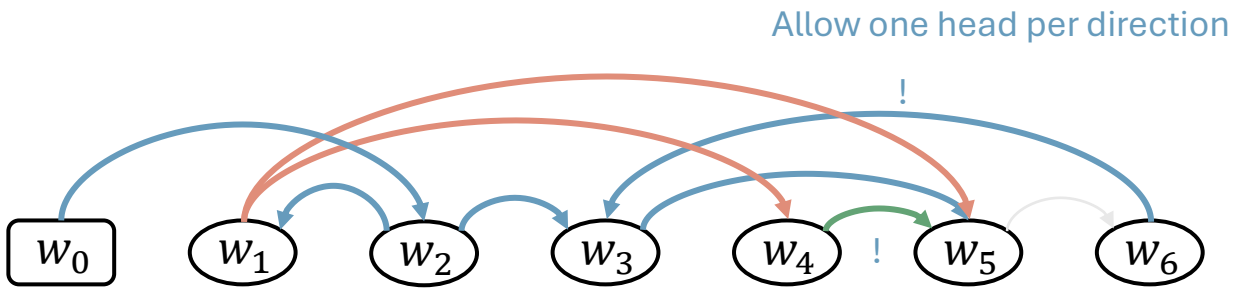
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



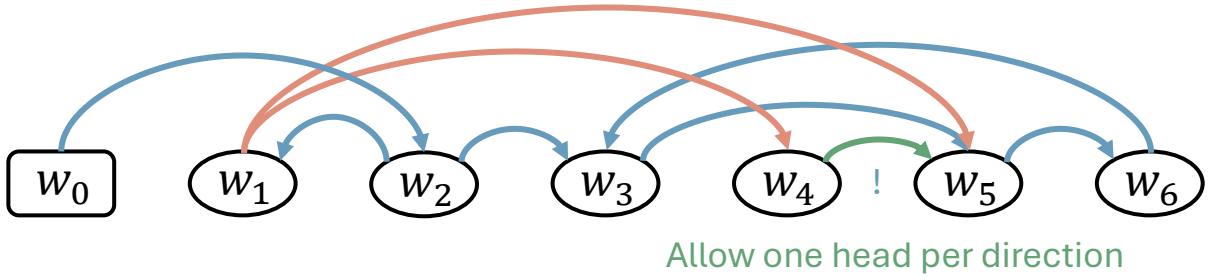
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 1: Distribute the arcs in **relaxed 1-planar graphs.**



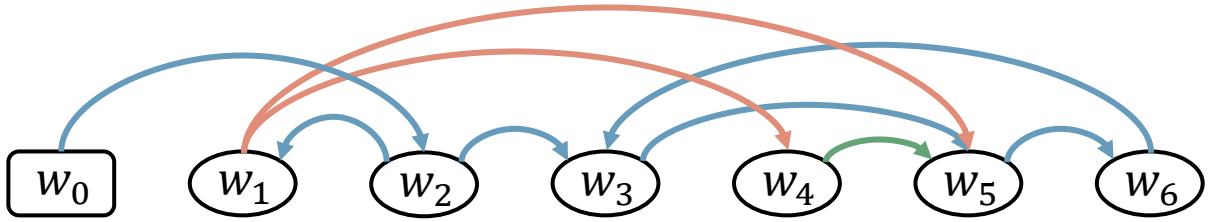
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

Step 2: Assign labels.



P1	000110	001001	111110	000000	111000	110001
P2	001000	000000	000000	100000	110000	000000
P3	000000	000000	000000	001000	110000	000000

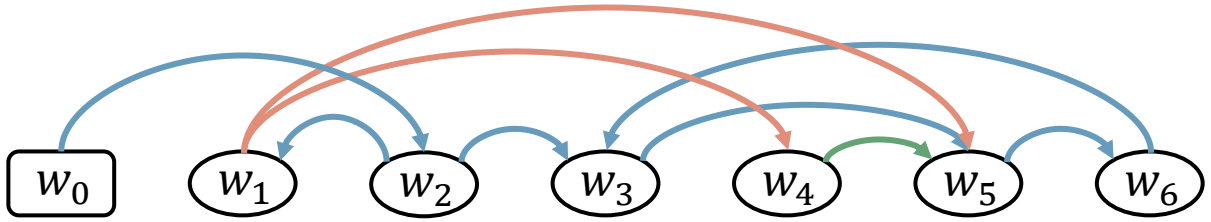
$6k$-bit Encoding

Assumptions:

- Distribute the arcs in **relaxed 1-planar graphs**.
- For each node limit one head per direction.
- Encode each label ℓ_i with 6 bits: $\mathbf{b_0 b_1 b_2 b_3 b_4 b_5}$.

$\mathbf{b_0}$	w_i has a left head.
$\mathbf{b_1}$	w_i is the outermost right dependent.
$\mathbf{b_2}$	w_i has right dependents.
$\mathbf{b_3}$	w_i has a right head.
$\mathbf{b_4}$	w_i is the outermost left dependent.
$\mathbf{b_5}$	w_i has left dependents.

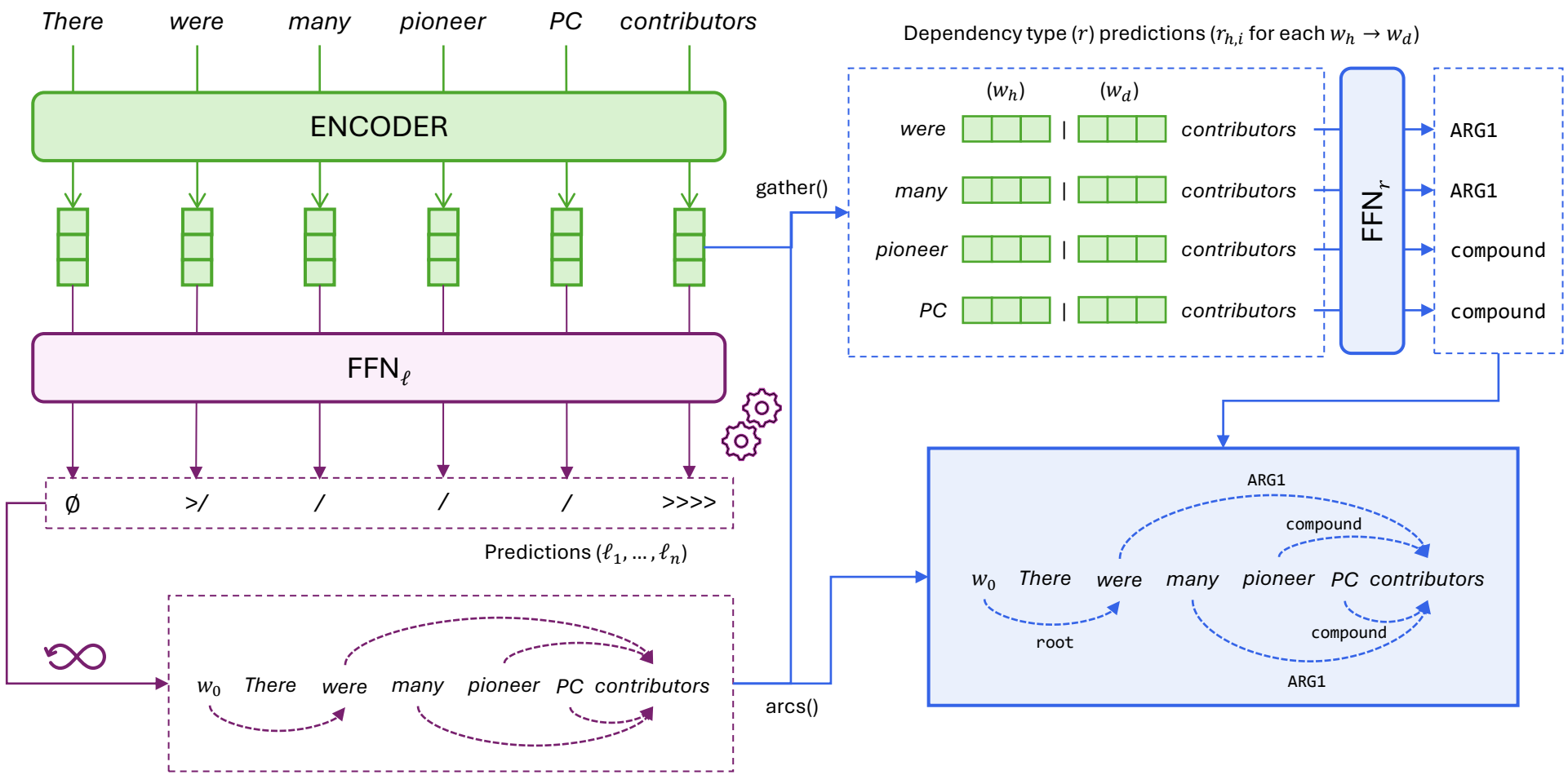
Step 2: Assign labels.



P1	000110	001001	111110	000000	111000	110001
P2	001000	000000	000000	100000	110000	000000
P3	000000	000000	000000	001000	110000	000000

Same as $4k$-bit encoding!

Encoder-Decoder Architecture



Experiments

Encoders:

- 4-layered BiLSTM.
- XLM-RoBERTa (Conneau et al., 2020).
- XLNet (Yang et al., 2019).

Decoders:

- Absolute (A) and relative (R).
- Bracketing (B) with $k \in \{2,3\}$.
- $4k$-bit (B4) with $k \in \{2,3,4\}$.
- $6k$-bit (B6) with $k \in \{2,3,4\}$.

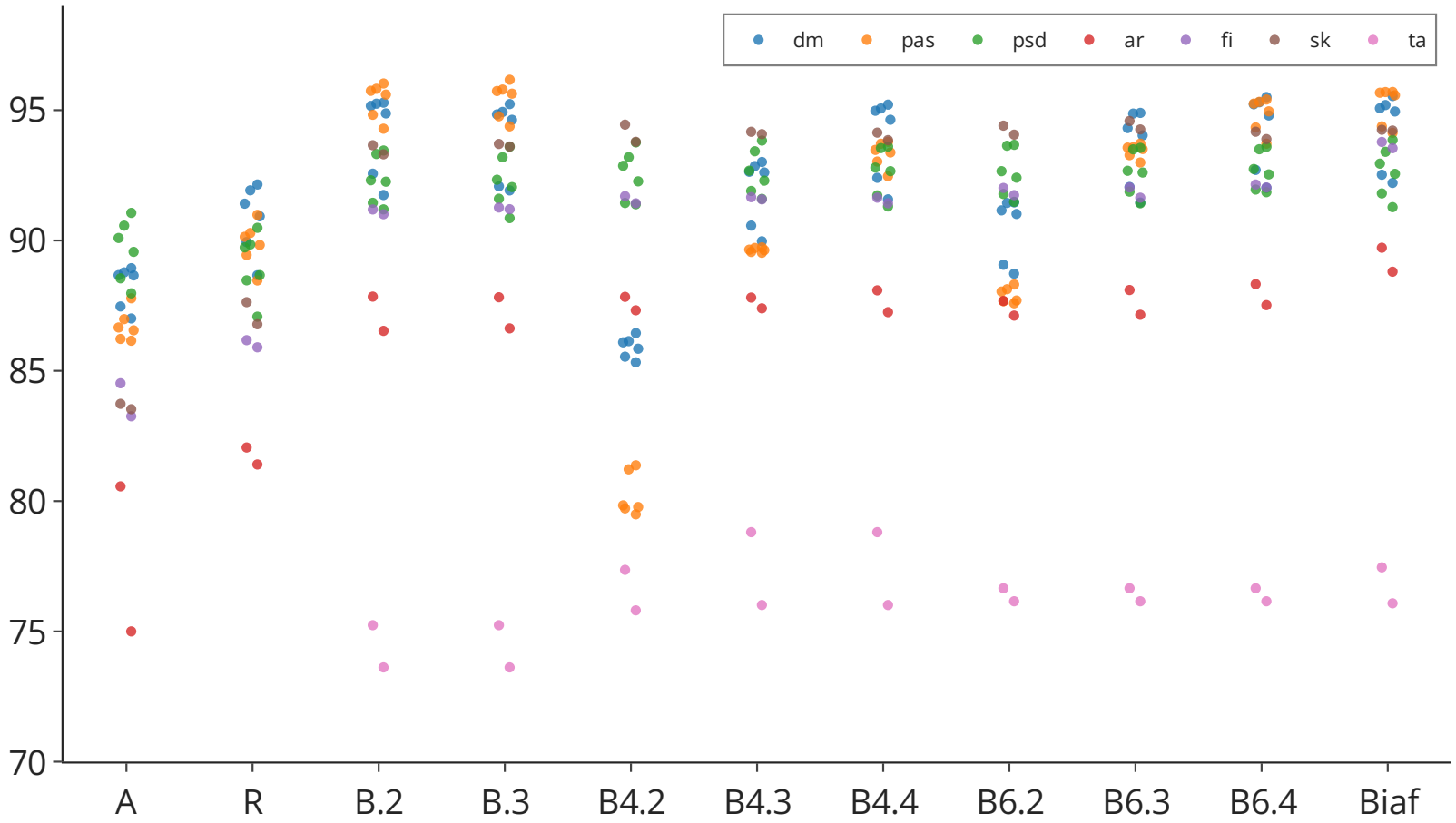
Evaluation

- SemEval 2015 Task 18 and IWPT-2021 Shared Task.
- UF and LF score.
- **Biaffine** baseline (Dozat & Manning, 2018).
- **Coverage** (score with gold labels).

Results

Performance

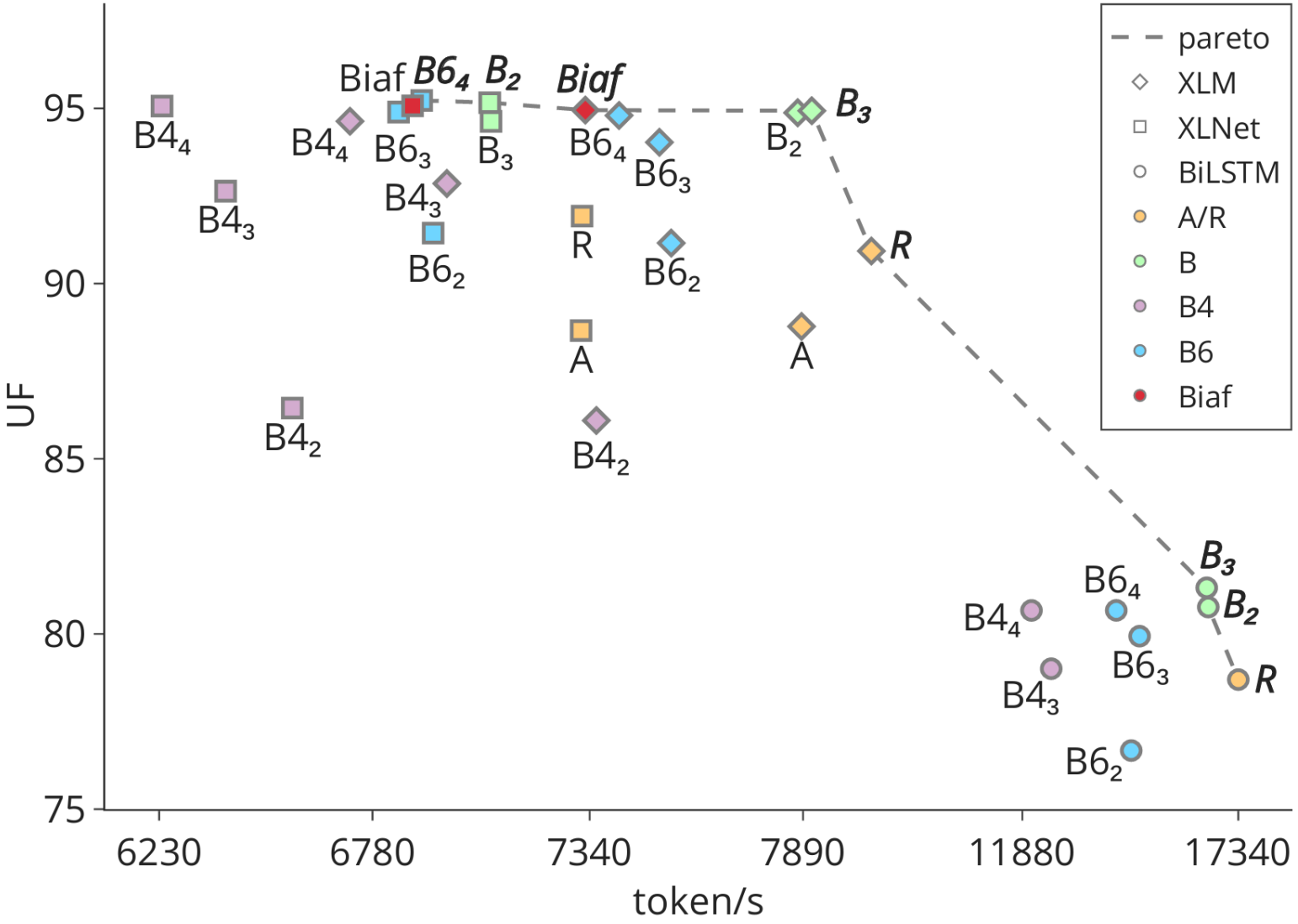
- Worst performance with **A** and **B**.
- **B2** and **B3** reach a similar performance to **Biaf**.
- Clear improvement increasing k.



Results

Speed

- Faster decoding for **A** and **R**.
- **B4.*k*** is slower than **Biaf** (likely due to the extra arcs generated).
- **B6.*k*** and **B.*k*** offer a trade-off between accuracy and speed.



Conclusions

- Several bounded and unbounded linearizations are proposed for graph parsing.
- Multilingual benchmark.
- SL algorithms and graph-based parsers have similar performance.
- SL is a faster and more efficient alternative to graph-based models.
- Hyperparameter k must be configured for bracket and bit encodings.

Thanks for listening!

Ana Ezquerro

David Vilares

Carlos Gómez-Rodríguez

We acknowledge the European Research Council (ERC), which has funded this research under the Horizon Europe research and innovation programme (SALSA, grant agreement No 101100615). We also acknowledge grants SCANNER-UDC (PID2020-113230RB-C21) funded by MICIU/AEI/10.13039/501100011033; GAP (PID2022-139308OA-I00) funded by MICIU/AEI/10.13039/501100011033/ and ERDF, EU; LATCHING (PID2023-147129OB-C21) funded by MICIU/AEI/10.13039/501100011033 and ERDF, EU; and TSI-100925-2023-1 funded by Ministry for Digital Transformation and Civil Service and "NextGenerationEU" PRTR; as well as funding by Xunta de Galicia (ED431C 2024/02), and Centro de Investigación de Galicia "CITIC", funded by the Xunta de Galicia through the collaboration agreement between the Consellería de Cultura, Educación, Formación Profesional e Universidades and the Galician universities for the reinforcement of the research centres of the Galician University System (CIGUS).