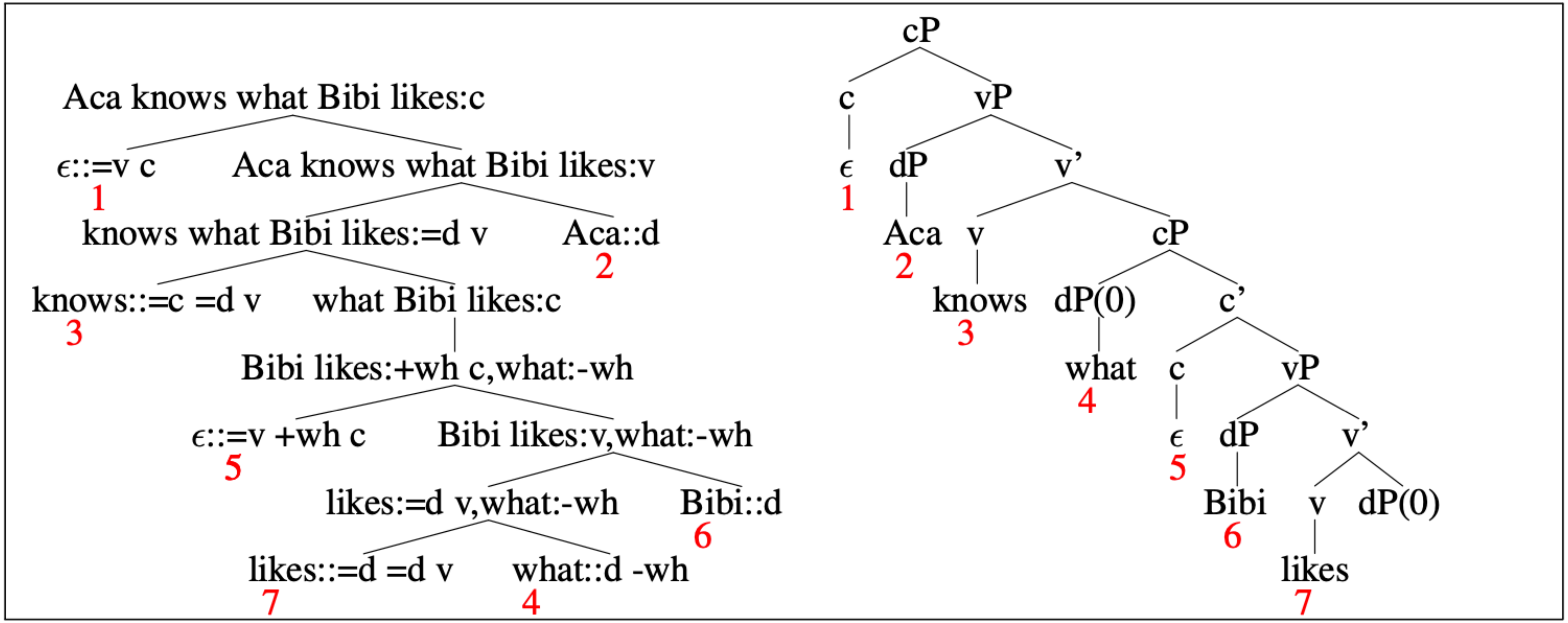
Left-Corner Parsing


Aca knows what Bibi likes:c
ϵ::=v c
1
Aca knows what Bibi likes:v
knows what Bibi likes:=d v
Aca::d
2
knows::=c =d v
3
what Bibi likes:c
Bibi likes:+wh c,what:-wh
ϵ::=v +wh c
5
Bibi likes:v,what:-wh
likes:=d v,what:-wh
Bibi::d
6
likes::=d =d v
7
what::d -wh
4
cP
c
ϵ
1
vP
dP
Aca
2
v'
v
knows
3
cP
dP(0)
what
4
c'
c
ϵ
5
vP
dP
Bibi
6
v'
v
likes
7
dP(0)

ϵ :: =v c

ϵ :: =v +wh c

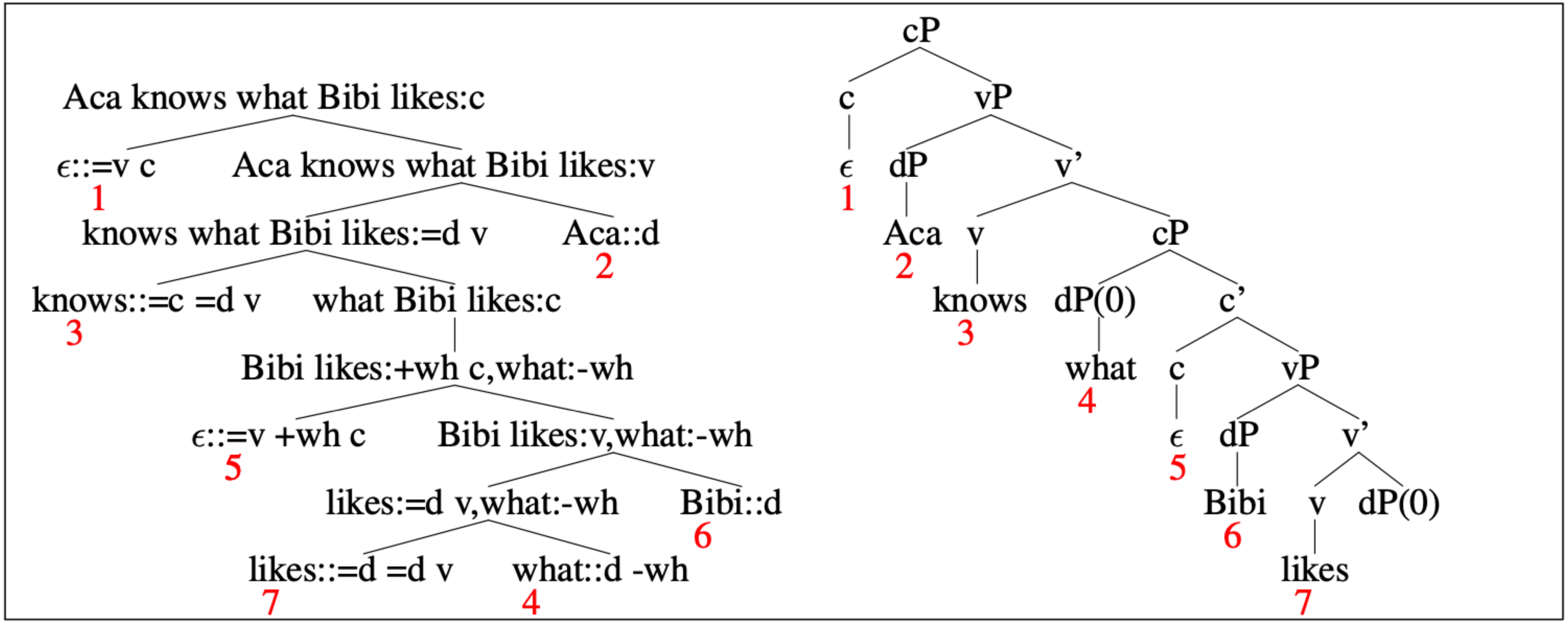
Aca :: d

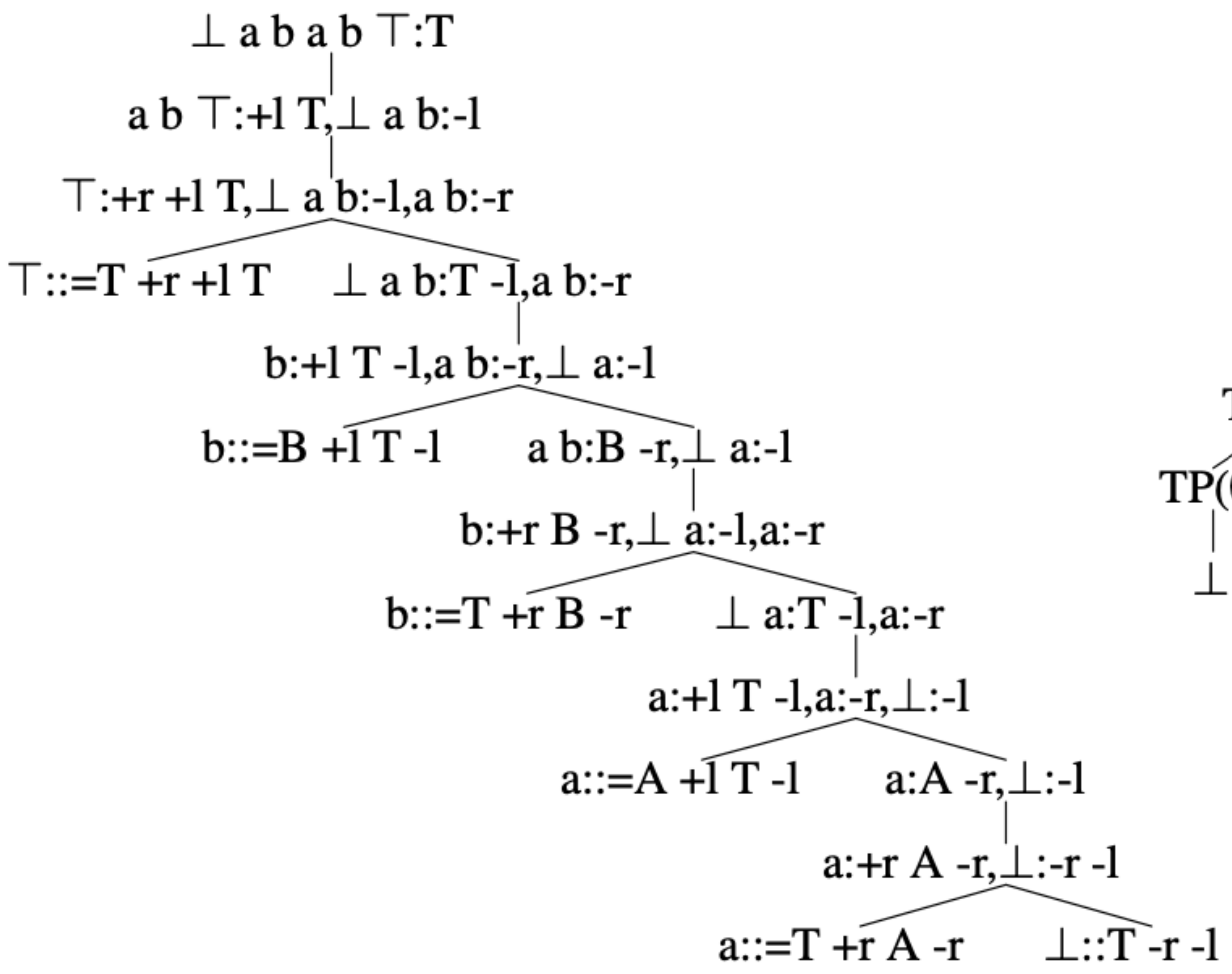
Bibi :: d

knows :: =c =d v

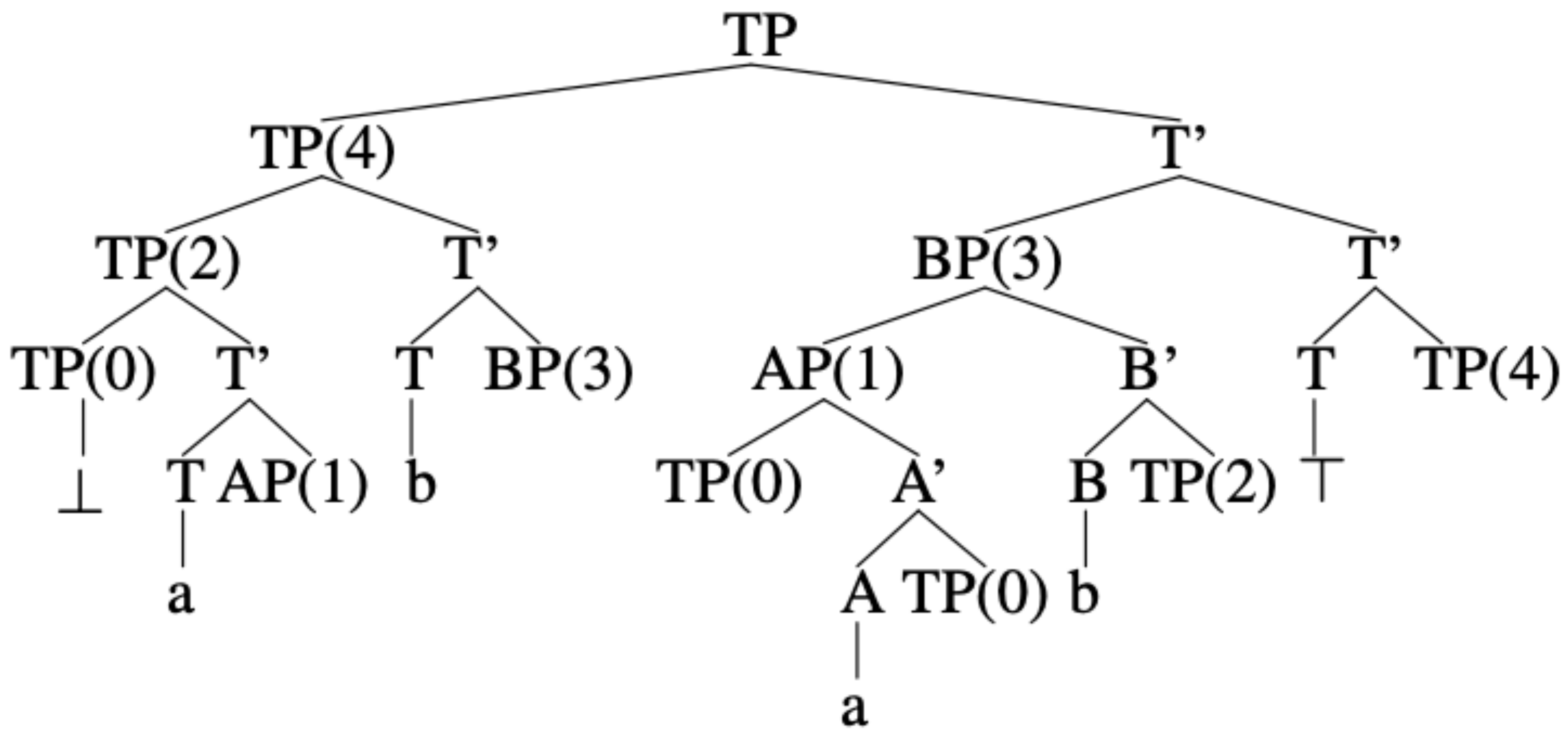
likes :: =d =d v

what :: d -wh

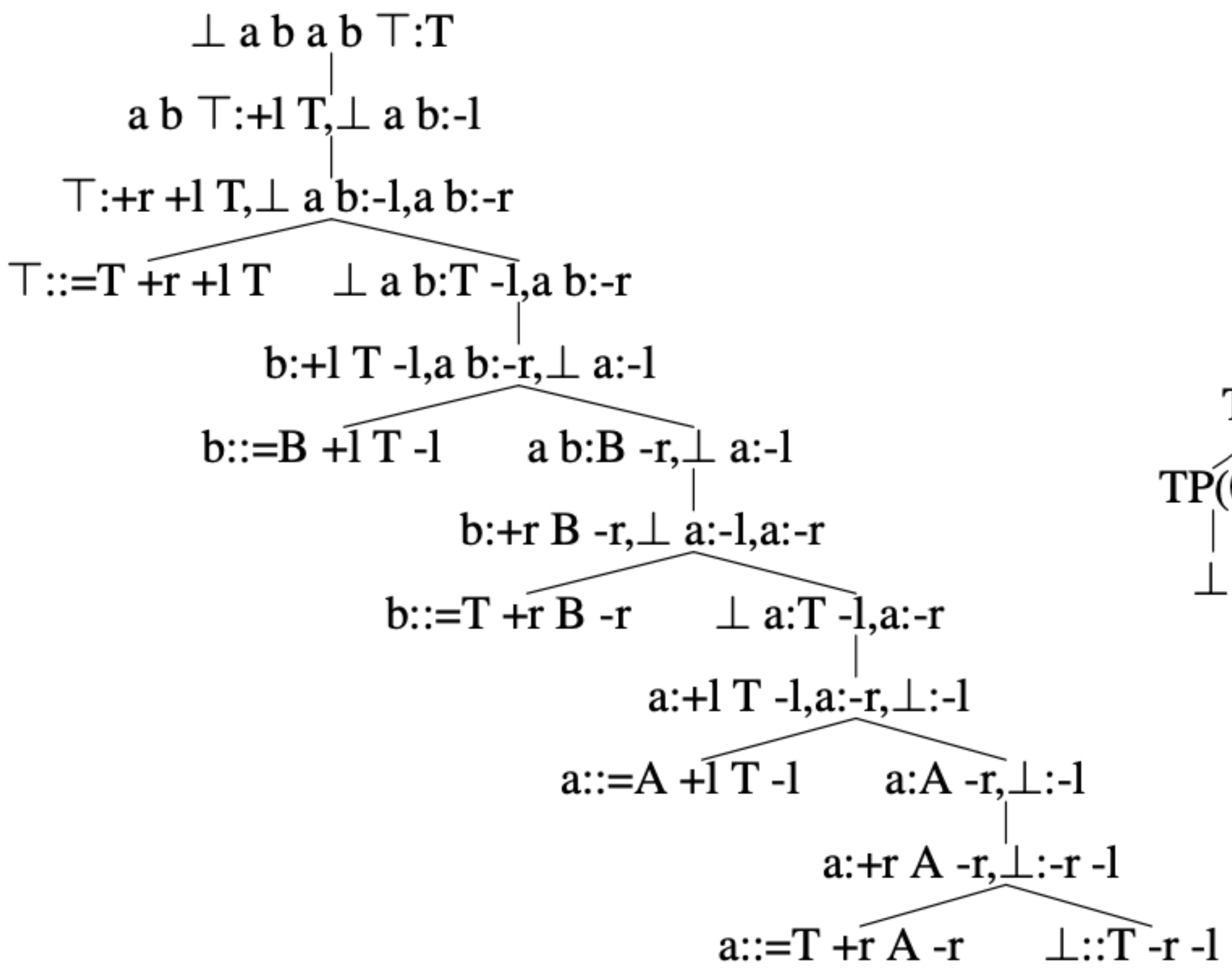


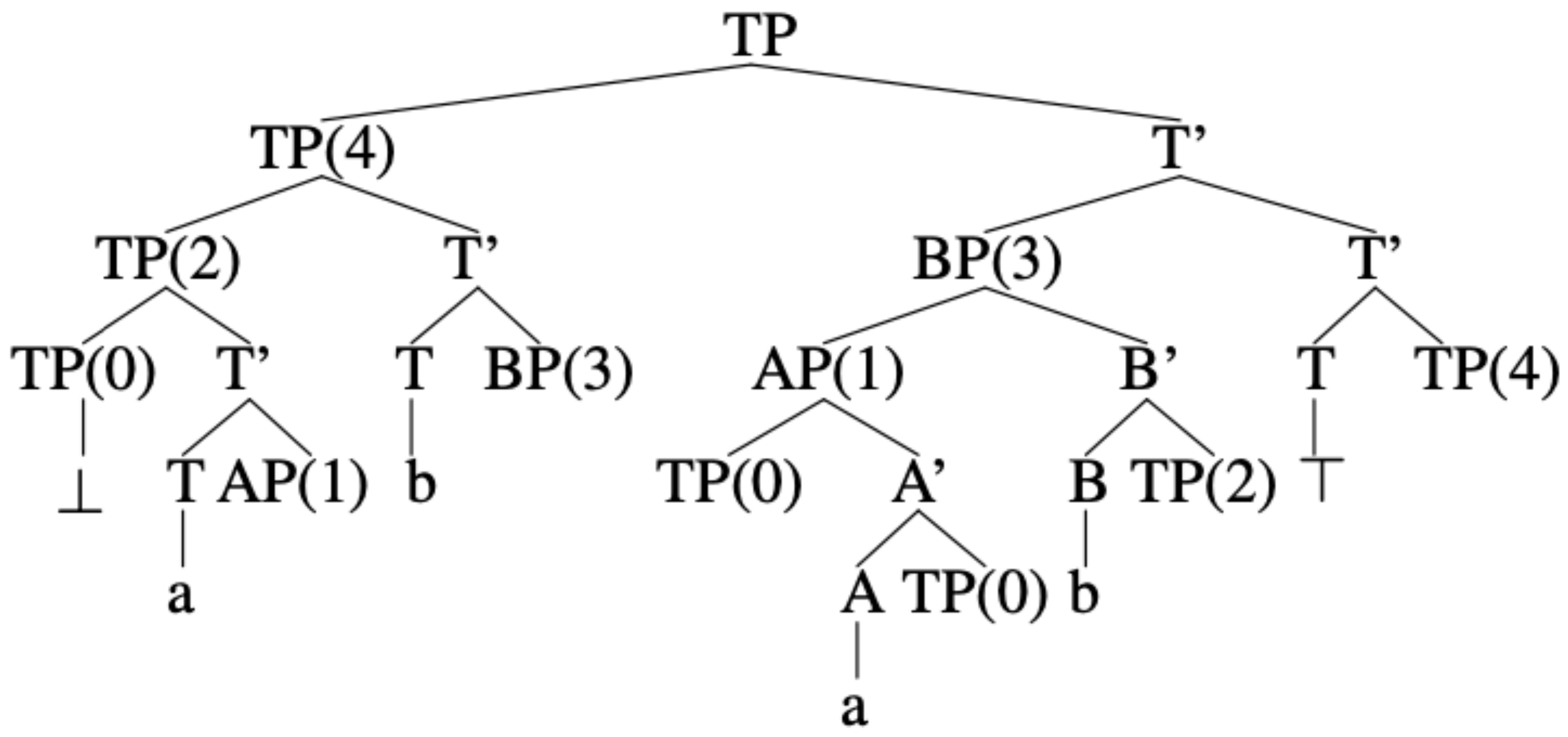


⊥ a b a b ⊤:T
a b ⊤:+l T,⊥ a b:-l
⊤:+r +l T,⊥ a b:-l,a b:-r
⊤::=T +r +l T
⊥ a b:T -l,a b:-r
b:+l T -l,a b:-r,⊥ a:-l
b::=B +l T -l
a b:B -r,⊥ a:-l
b:+r B -r,⊥ a:-l,a:-r
b::=T +r B -r
⊥ a:T -l,a:-r
a:+l T -l,a:-r,⊥:-l
a::=A +l T -l
a:A -r,⊥:-l
a:+r A -r,⊥:-r -l
a::=T +r A -r
⊥::T -r -l



TP
TP(4)
T’
TP(2)
T’
BP(3)
T’
TP(0)
T’
T
BP(3)
AP(1)
B’
T
TP(4)
⊥
T
AP(1)
b
TP(0)
A’
B
TP(2)
⊤
a
A
TP(0)
b
a





⊥ :: T -r -l

a :: =A +l T -l

b :: =B +l T -l

⊤ :: =T +r +l T

a :: =T +r A -r

b :: =T +r B -r

merge is the union of the following 3 rules, each with 2 elements on the right,
for strings $s, t \in \Sigma^*$, for types $\cdot \in \{:, ::\}$ (lexical and derived, respectively),
for feature sequences $\gamma \in F^*$, $\delta \in F^+$, and for chains $\alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l$ $(0 \leq k, l)$

(MERGE1) lexical item s selects non-mover t to produce the merged st

$$st : \gamma, \alpha_1, \ldots, \alpha_k \quad \leftarrow \quad s :: {=}f\gamma \qquad t \cdot f, \alpha_1, \ldots, \alpha_k$$

(MERGE2) derived item s selects a non-mover t to produce the merged ts

$$ts : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l \quad \leftarrow \quad s : {=}f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f, \iota_1, \ldots, \iota_l$$

(MERGE3) any item s selects a mover t to produce the merged s with chain t

$$s : \gamma, \alpha_1, \ldots, \alpha_k, t : \delta, \iota_1, \ldots, \iota_l \quad \leftarrow \quad s \cdot {=}f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f\delta, \iota_1, \ldots, \iota_l$$

move is the union of the following 2 rules, each with 1 element on the right,
for $\delta \in F^+$, such that none of the chains $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k$ has -f as its first feature:

(MOVE1) final move of t, so its -f chain is eliminated on the left

$$ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k \quad \leftarrow \quad s : {+}f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : {-}f, \alpha_{i+1}, \ldots, \alpha_k$$

(MOVE2) nonfinal move of t, so its chain continues with features δ

$$s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k \quad \leftarrow \quad s : {+}f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : {-}f\delta, \alpha_{i+1}, \ldots, \alpha_k$$


A
B
B
A
B
B
C
A
B
B
C
A
B
B
C
C
D

So the parser state is given by

$$\text{(remaining input, current position, queue)},$$

and we begin with

$$(\text{input}, 0, \epsilon).$$

For any input of length n, we then attempt to apply the LC rules to get

$$(\epsilon, n, 0\text{-}n{\cdot}\text{c}),$$

(0) The SHIFT rule takes an initial (possibly empty) element w with span x-y from the beginning of the remaining input, where the lexicon has $w :: \gamma$, and puts x-y::γ onto the queue.

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
```

(1) For an MG rule R of the form $A \leftarrow B\ C$ with left corner B, if an instance of B is on top of the queue, lc1(R) removes B from the top of the queue and replaces it with an element $C \Rightarrow A$. Since any merge rule can have the selector as its left corner, we have the LC rules LC1(MERGE1), LC1(MERGE2), and LC1(MERGE3).

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
```

ϵ :: =v c
ϵ :: =v +wh c
Aca :: d
Bibi :: d

knows :: =c =d v
likes :: =d =d v
what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
```

(1) For an MG rule R of the form $A \leftarrow B\ C$ with left corner B, if an instance of B is on top of the queue, lc1(R) removes B from the top of the queue and replaces it with an element $C \Rightarrow A$. Since any merge rule can have the selector as its left corner, we have the LC rules LC1(MERGE1), LC1(MERGE2), and LC1(MERGE3).

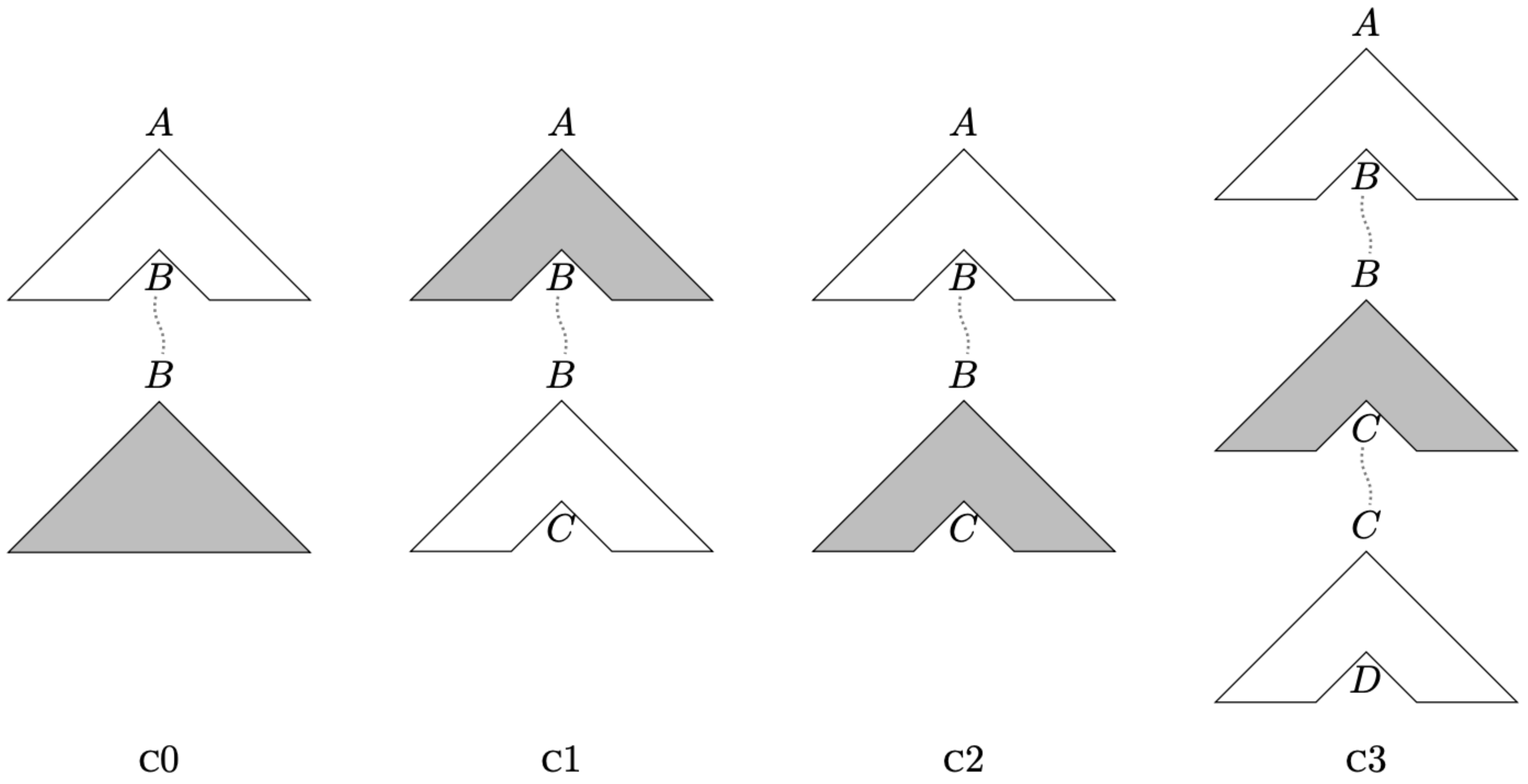
(2) For an MG rule R of the form $A \leftarrow B\ C'$ with completed left corner C and $C\theta = C'\theta$, lc2(R) replaces C on top of the queue by $(B \Rightarrow A)\theta$. For this case, where the second argument on the right side is the left corner, we have the LC rules LC2(MERGE2) and LC2(MERGE3).

c(R) If LC rule R creates a constituent B, and the queue has $B' \Rightarrow A$, where $B\theta = B'\theta$, then c(R) removes $B' \Rightarrow A$ puts $A\theta$ onto the queue.

c1(R) If LC rule R creates $B \Rightarrow A$ and we already have $C \Rightarrow B'$ on the queue, where $B\theta = B'\theta$, then c1(R) removes $C \Rightarrow B'$ and puts $(C \Rightarrow A)\theta$ onto the queue.

c2(R) If LC rule R creates $C \Rightarrow B$ and we already have $B' \Rightarrow A$ on the queue, where $B\theta = B'\theta$, c2(R) removes $B' \Rightarrow A$ and puts $(C \Rightarrow A)\theta$ onto the queue.

c3(R) If LC rule R creates a constituent $C \Rightarrow B$ and we already have $B' \Rightarrow A$ and $D \Rightarrow C'$ on the queue, where $B\theta = B'\theta$ and $C\theta = C'\theta$ c3(R) removes $B' \Rightarrow A$ and $D \Rightarrow C'$ and puts $(D \Rightarrow A)\theta$ onto the queue.



ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
4. c1(lc2(merge2)) [knows,what,Bibi,likes]
    (1-_:=d v _M  =>  0-_:c _M)
```

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
4. c1(lc2(merge2)) [knows,what,Bibi,likes]
    (1-_:=d v _M  =>  0-_:c _M)
5. shift [what,Bibi,likes]
    1-2::=c =d v
    (1-_:=d v _M  =>  0-_:c _M)
```

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
4. c1(lc2(merge2)) [knows,what,Bibi,likes]
    (1-_:=d v _M  =>  0-_:c _M)
5. shift [what,Bibi,likes]
    1-2::=c =d v
    (1-_:=d v _M  =>  0-_:c _M)
6. c1(lc1(merge1)) [what,Bibi,likes]
    (2-_.c _M  =>  0-_:c _M)
```

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
4. c1(lc2(merge2)) [knows,what,Bibi,likes]
    (1-_:=d v _M  =>  0-_:c _M)
5. shift [what,Bibi,likes]
    1-2::=c =d v
    (1-_:=d v _M  =>  0-_:c _M)
6. c1(lc1(merge1)) [what,Bibi,likes]
    (2-_.c _M  =>  0-_:c _M)
7. shift [Bibi,likes]
    2-3::d -wh
    (2-_.c _M  =>  0-_:c _M)
```

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
7. shift [Bibi,likes]
    2-3::d -wh
    (2-_.c _M  =>  0-_:c _M)
8. lc2(merge3) [Bibi,likes]
    (_-_.=d _Fs_M  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
9. shift [Bibi,likes]
    3-3::=v +wh c
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
10. lc1(merge1) [Bibi,likes]
    (3-_.v _M  =>  3-_:+wh c _M)
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _N  =>  0-_:c _N)
11. shift [likes]
    3-4::d
    (3-_.v _M  =>  3-_:+wh c _M)
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _N  =>  0-_:c _N)
```

(3) Similarly for MG rules $A \leftarrow B$, the only possible leftcorner is a constituent B where $B\theta = B'\theta$, replacing B' by $A\theta$. So we have LC1(MOVE1) and LC1(MOVE2) in this case.

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
11. shift [likes]
    3-4::d
    (3-_.v _M  =>  3-_:+wh c _M)
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _N  =>  0-_:c _N)
12. c3(lc2(merge2)) [likes]
    (4-_.=d =d v   =>  3-_:+wh c ,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
13. c(shift) []
    3-5:+wh c ,2-3:-wh
    (2-_.c _M  =>  0-_:c _M)
14. c(lc1(move1)) []
    0-5:c
```

ϵ :: =v c

ϵ :: =v +wh c

Aca :: d

Bibi :: d

knows :: =c =d v

likes :: =d =d v

what :: d -wh

```
1. shift [Aca,knows,what,Bibi,likes]
    0-0::=v c
2. lc1(merge1) [Aca,knows,what,Bibi,likes]
    (0-_.v _M  =>  0-_:c _M)
3. shift [knows,what,Bibi,likes]
    0-1::d
    (0-_.v _M  =>  0-_:c _M)
4. c1(lc2(merge2)) [knows,what,Bibi,likes]
    (1-_:=d v _M  =>  0-_:c _M)
5. shift [what,Bibi,likes]
    1-2::=c =d v
    (1-_:=d v _M  =>  0-_:c _M)
6. c1(lc1(merge1)) [what,Bibi,likes]
    (2-_.c _M  =>  0-_:c _M)
7. shift [Bibi,likes]
    2-3::d -wh
    (2-_.c _M  =>  0-_:c _M)
8. lc2(merge3) [Bibi,likes]
    (_-_.=d _Fs_M  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
9. shift [Bibi,likes]
    3-3::=v +wh c
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
10. lc1(merge1) [Bibi,likes]
    (3-_.v _M  =>  3-_:+wh c _M)
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _N  =>  0-_:c _N)
11. shift [likes]
    3-4::d
    (3-_.v _M  =>  3-_:+wh c _M)
    (_-_.=d _Fs  =>  _-_:_Fs,2-3:-wh )
    (2-_.c _N  =>  0-_:c _N)
12. c3(lc2(merge2)) [likes]
    (4-_.=d =d v   =>  3-_:+wh c ,2-3:-wh )
    (2-_.c _M  =>  0-_:c _M)
13. c(shift) []
    3-5:+wh c ,2-3:-wh
    (2-_.c _M  =>  0-_:c _M)
14. c(lc1(move1)) []
    0-5:c
```

(0) The SHIFT rule takes an initial (possibly empty) element w with span x-y from the beginning of the remaining input, where the lexicon has $w :: \gamma$, and puts x-y::γ onto the queue.

(1) For an MG rule R of the form $A \leftarrow B\ C$ with left corner B, if an instance of B is on top of the queue, lc1(R) removes B from the top of the queue and replaces it with an element $C \Rightarrow A$. Since any merge rule can have the selector as its left corner, we have the LC rules LC1(MERGE1), LC1(MERGE2), and LC1(MERGE3).

(2) For an MG rule R of the form $A \leftarrow B\ C'$ with completed left corner C and $C\theta = C'\theta$, lc2(R) replaces C on top of the queue by $(B \Rightarrow A)\theta$. For this case, where the second argument on the right side is the left corner, we have the LC rules LC2(MERGE2) and LC2(MERGE3).

(3) Similarly for MG rules $A \leftarrow B$, the only possible leftcorner is a constituent B where $B\theta = B'\theta$, replacing B' by $A\theta$. So we have LC1(MOVE1) and LC1(MOVE2) in this case.

c(R) If LC rule R creates a constituent B, and the queue has $B' \Rightarrow A$, where $B\theta = B'\theta$, then c(R) removes $B' \Rightarrow A$ puts $A\theta$ onto the queue.

c1(R) If LC rule R creates $B \Rightarrow A$ and we already have $C \Rightarrow B'$ on the queue, where $B\theta = B'\theta$, then c1(R) removes $C \Rightarrow B'$ and puts $(C \Rightarrow A)\theta$ onto the queue.

c2(R) If LC rule R creates $C \Rightarrow B$ and we already have $B' \Rightarrow A$ on the queue, where $B\theta = B'\theta$, c2(R) removes $B' \Rightarrow A$ and puts $(C \Rightarrow A)\theta$ onto the queue.

c3(R) If LC rule R creates a constituent $C \Rightarrow B$ and we already have $B' \Rightarrow A$ and $D \Rightarrow C'$ on the queue, where $B\theta = B'\theta$ and $C\theta = C'\theta$ c3(R) removes $B' \Rightarrow A$ and $D \Rightarrow C'$ and puts $(D \Rightarrow A)\theta$ onto the queue.