## 3. Coinductive Predicates

# Example of Coinductive Definition

Given a set A, let LazyList(A) be the set of "lazy lists" (finite or infinite lists) with elements in A – they have the form  $[a_1, a_2, \ldots, a_n]$  or  $[a_1, a_2, \ldots]$ . We write a # as for the lazy list obtained by consing a to as.

We wish to define the sublist relation, *subll*, on lazy lists.

The relation *subl* on (finite) lists is defined inductively by the rules:

$$\frac{\cdot}{subl[] as} \text{ (Nil)} \qquad \frac{subl as as'}{subl as (a\#as')} \text{ (ConsR)}$$
$$\frac{subl as as'}{subl (a\#as) (a\#as')} \text{ (Cons)}$$

The inductive interpretation means: smallest relation closed under the rules (Nil), (ConsR) and (Cons).

Given a set A, let LazyList(A) be the set of "lazy lists" (finite or infinite lists) with elements in A – they have the form  $[a_1, a_2, \ldots, a_n]$  or  $[a_1, a_2, \ldots]$ . We write a # as for the lazy list obtained by consing a to as.

We wish to define the sublist relation, *subll*, on lazy lists.

The relation *subl* on (finite) lists is defined inductively by the rules:

$$\frac{\cdot}{subl[] as} \text{ (Nil)} \qquad \frac{subl as as'}{subl as (a\#as')} \text{ (ConsR)}$$
$$\frac{subl as as'}{subl (a\#as) (a\#as')} \text{ (Cons)}$$

The inductive interpretation means: smallest relation closed under the rules (Nil), (ConsR) and (Cons). We've seen that this does not work for lazy lists: No infinite list would be a sublist of any list.

Given a set A, let LazyList(A) be the set of "lazy lists" (finite or infinite lists) with elements in A – they have the form  $[a_1, a_2, \ldots, a_n]$  or  $[a_1, a_2, \ldots]$ . We write a # as for the lazy list obtained by consing a to as.

We wish to define the sublist relation, *subll*, on lazy lists.

The relation *subl* on (finite) lists is defined inductively by the rules:

$$\frac{\cdot}{subl[] as} \text{ (Nil)} \qquad \frac{subl as as'}{subl as (a\#as')} \text{ (ConsR)}$$
$$\frac{subl as as'}{subl (a\#as) (a\#as')} \text{ (Cons)}$$

The inductive interpretation means: smallest relation closed under the rules (Nil), (ConsR) and (Cons). We've seen that this does not work for lazy lists: No infinite list would be a sublist of any list.

Should we rather go for the greatest relation closed under these rules?

Given a set A, let LazyList(A) be the set of "lazy lists" (finite or infinite lists) with elements in A – they have the form  $[a_1, a_2, \ldots, a_n]$  or  $[a_1, a_2, \ldots]$ . We write a # as for the lazy list obtained by consing a to as.

We wish to define the sublist relation, *subll*, on lazy lists.

The relation *subl* on (finite) lists is defined inductively by the rules:

$$\frac{\cdot}{subl[]as} \text{ (Nil)} \qquad \frac{subl as as'}{subl as (a\#as')} \text{ (ConsR)}$$
$$\frac{subl as as'}{subl (a\#as) (a\#as')} \text{ (Cons)}$$

The inductive interpretation means: smallest relation closed under the rules (Nil), (ConsR) and (Cons). We've seen that this does not work for lazy lists: No infinite list would be a sublist of any list.

Should we rather go for the greatest relation closed under these rules? No! This would give us the total relation  $\lambda as$ , as'. T.

Given a set A, let LazyList(A) be the set of "lazy lists" (finite or infinite lists) with elements in A – they have the form  $[a_1, a_2, \ldots, a_n]$  or  $[a_1, a_2, \ldots]$ . We write a # as for the lazy list obtained by consing a to as.

We wish to define the sublist relation, *subll*, on lazy lists.

The relation *subl* on (finite) lists is defined inductively by the rules:

$$\frac{\cdot}{subl[] as} \text{ (Nil)} \qquad \frac{subl as as'}{subl as (a\#as')} \text{ (ConsR)}$$
$$\frac{subl as as'}{subl (a\#as) (a\#as')} \text{ (Cons)}$$

The inductive interpretation means: smallest relation closed under the rules (Nil), (ConsR) and (Cons). We've seen that this does not work for lazy lists: No infinite list would be a sublist of any list.

Should we rather go for the greatest relation closed under these rules? No! This would give us the total relation  $\lambda as$ , as'. T. Let's take it easy, starting with selecting the properties that we want...

Say  $A = \mathbb{N}$ .

For finite lists, *subll* should behave just like *subl*, e.g.,

- *subll* [1, 3, 4] [1, 2, 3, 4]
- *subll* [1,2] [1,2,3,4]
- *subll* [1, 3] [1, 2, 3, 4]

Also, e.g.,

- subll zeros zeros, in fact subll as as for any as
- $subll [0, 2, 4, 6, \ldots] [0, 1, 2, 3, \ldots]$

Say  $A = \mathbb{N}$ .

For finite lists, *subll* should behave just like *subl*, e.g.,

- *subll* [1, 3, 4] [1, 2, 3, 4]
- *subll* [1, 2] [1, 2, 3, 4]
- *subll* [1, 3] [1, 2, 3, 4]

Also, e.g.,

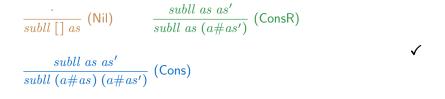
- subll zeros zeros, in fact subll as as for any as
- $subll [0, 2, 4, 6, \ldots] [0, 1, 2, 3, \ldots]$

subll as as' should hold if and only if: assuming as' has the form  $[a'_i]_{i < length \ as'}$  (with  $length \ as' \in \mathbb{N} \cup \{\infty\}$ ) there exists  $[j_p]_{p < length \ as}$  such that  $\forall p. \ p+1 < length \ as \longrightarrow j_p < j_{p+1}$ and  $as = [a'_{j_p}]_{p < length \ as}$ .

$$\frac{subll \ as \ as'}{subll \ as \ (a\#as')} \ (ConsR)$$

$$\frac{subll \ as \ as'}{subll \ (a\#as) \ (a\#as')}$$
 (Cons)

 $\frac{\cdot}{subll [] as}$  (Nil)



 $\checkmark$ 

$$\frac{subll \ as \ as'}{subll \ as \ (a\#as')} \ (ConsR)$$

 $\frac{\textit{subll as as'}}{\textit{subll } (a\#as) \ (a\#as')} \ (\mathsf{Cons})$ 

 $\frac{\cdot}{subll [] as}$  (Nil)

$$subll bs bs' \qquad \forall as. bs = [] \land bs' = as \longrightarrow P$$
  
$$\forall as, as', a. bs = as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$\forall as, as', a. bs = a\#as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$P$$
  
(Cases)

 $\checkmark$ 

$$\frac{subll \ as \ as'}{subll \ as \ (a\#as')} \ (ConsR)$$

 $\frac{\textit{subll as as'}}{\textit{subll } (a\#as) \ (a\#as')} \ (\mathsf{Cons})$ 

 $\frac{\cdot}{subll [] as}$  (Nil)

$$subll bs bs' \qquad \forall as. bs = [] \land bs' = as \longrightarrow P$$
  
$$\forall as, as', a. bs = as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$\forall as, as', a. bs = a\#as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$P$$
  
(Cases)

$$\frac{subll \ as \ as'}{subll \ as \ (a \# as')}$$
 (ConsR)

 $\frac{subll \ as \ as'}{subll \ (a\#as) \ (a\#as')} \ ({\sf Cons})$ 

$$subll bs bs' \qquad \forall as. bs = [] \land bs' = as \longrightarrow P$$
  
$$\forall as, as', a. bs = as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$\forall as, as', a. bs = a\#as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$P$$
  
(Cases)

or, equivalently ...

 $\frac{1}{subll [] as}$  (Nil)

subll bs  $bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\lor$   $\exists as, a, as'. bs = as \land bs' = a\#as' \land subll as as'$   $\lor$  $\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land subll as as'$ 

$$\frac{subll \ as \ as'}{subll \ as \ (a\#as')} \ (ConsR)$$

 $\frac{\textit{subll as as'}}{\textit{subll } (a\#as) \ (a\#as')} \ (\mathsf{Cons})$ 

$$subll bs bs' \qquad \forall as. bs = [] \land bs' = as \longrightarrow P$$
  
$$\forall as, as', a. bs = as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$\forall as, as', a. bs = a\#as \land bs' = a\#as' \land subll as as' \longrightarrow P$$
  
$$P$$
  
(Cases)  $\checkmark$ 

or, equivalently ...

 $\frac{1}{subll [] as}$  (Nil)

subll bs  $bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\lor$   $\exists as, a, as'. bs = as \land bs' = a\#as' \land subll as as'$   $\lor$  $\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land subll as as'$ 

#### How about induction?

$$\begin{array}{c} subll \ bs \ bs' & \forall as. \ P \ [] \ as \\ \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ as \ (a \# as') \\ \hline \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ (a \# as) \ (a \# as') \\ \hline P \ bs \ bs' \end{array}$$
(Induct)

#### How about induction?

$$\begin{array}{c} subll \ bs \ bs' & \forall as. \ P \ [] \ as \\ \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ as \ (a \# as') \\ \hline \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ (a \# as) \ (a \# as') \\ \hline P \ bs \ bs' \end{array}$$
(Induct)

It would allow us to prove, e.g., subll as as' implies as is finite.

#### How about induction?

$$\begin{array}{c} subll \ bs \ bs' & \forall as. \ P \ [] \ as \\ \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ as \ (a \# as') \\ \hline \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ (a \# as) \ (a \# as') \\ \hline P \ bs \ bs' \end{array}$$
(Induct)

It would allow us to prove, e.g., subll as as' implies as is finite.

This would imply, e.g.,

- $\neg$  subll zeros zeros
- $\neg$  subll  $[0, 2, 4, 6, \ldots] [0, 1, 2, 3, \ldots]$

#### How about induction?

$$\begin{array}{ccc} subll \ bs \ bs' & \forall as. \ P \ [] \ as \\ \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ as \ (a \# as') \\ \hline \forall as, \ as', \ a. \ subll \ as \ as' \ \land \ P \ as \ as' \longrightarrow P \ (a \# as) \ (a \# as') \\ \hline P \ bs \ bs' \end{array}$$
(Induct) ×

It would allow us to prove, e.g., subll as as' implies as is finite.

This would imply, e.g.,

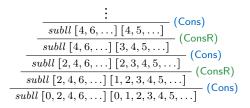
- $\neg$  subll zeros zeros
- $\neg$  subll  $[0, 2, 4, 6, \ldots] [0, 1, 2, 3, \ldots]$

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land subll as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land subll as as'$ 

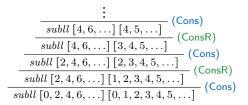
 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\overset{\vee}{\exists} as, a, as'. bs = a \land bs' = a\#as' \land subll as as'$   $\overset{\vee}{\exists} a, as, as'. bs = a\#as \land bs' = a\#as' \land subll as as'$ 

How can we prove  $subll [0, 2, 4, 6, \ldots] [0, 1, 2, 3, 4, 5, \ldots]?$ 

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land subll as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land subll as as'$ 

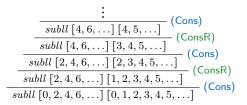


 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land subll as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land subll as as'$ 



So accepting infinite proofs with our introduction rules would solve our problem...

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\substack{\forall \\ \exists as, a, as'. bs = a \#as \land bs' = a \#as' \land subll as as'}}_{\forall \\ \exists a, as, as'. bs = a \#as \land bs' = a \#as' \land subll as as'}$ 



So accepting infinite proofs with our introduction rules would solve our problem... But how about something finitely expressible – a blueprint for an infinite proof?

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land subll as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land subll as as'$ 

$$\frac{\vdots}{subll [4, 6, \dots] [4, 5, \dots]} (Cons) \\ \frac{subll [4, 6, \dots] [3, 4, 5, \dots]}{subll [2, 4, 6, \dots] [2, 3, 4, 5, \dots]} (Cons) \\ \frac{subll [2, 4, 6, \dots] [1, 2, 3, 4, 5, \dots]}{subll [2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]} (ConsR) \\ \frac{subll [0, 2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]}{subll [0, 2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]} (ConsR)$$

So accepting infinite proofs with our introduction rules would solve our problem... But how about something finitely expressible – a <u>blueprint</u> for an infinite proof? *P* bs bs' =  $\exists k \in \mathbb{N}$ . bs =  $[2k, 2k + 2, 2k + 4 \dots] \land \overline{bs'} = [2k, 2k + 1, 2k + 2, \dots] \lor bs = [2k + 2, 2k + 4 \dots] \land bs' = [2k + 1, 2k + 2, \dots]$ 

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $subll bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land subll as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land subll as as'$ 

$$\frac{\vdots}{subll [4, 6, \dots] [4, 5, \dots]} (Cons) \\ \frac{subll [4, 6, \dots] [3, 4, 5, \dots]}{subll [2, 4, 6, \dots] [2, 3, 4, 5, \dots]} (Cons) \\ \frac{subll [2, 4, 6, \dots] [1, 2, 3, 4, 5, \dots]}{subll [2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]} (ConsR) \\ \frac{subll [0, 2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]}{subll [0, 2, 4, 6, \dots] [0, 1, 2, 3, 4, 5, \dots]} (ConsR)$$

So accepting infinite proofs with our introduction rules would solve our problem... But how about something finitely expressible – a <u>blueprint</u> for an infinite proof? *P* bs bs' =  $\exists k \in \mathbb{N}$ . bs =  $[2k, 2k + 2, 2k + 4 \dots] \land \overline{bs'} = [2k, 2k + 1, 2k + 2, \dots] \lor bs = [2k + 2, 2k + 4 \dots] \land bs' = [2k + 1, 2k + 2, \dots]$ 

 $\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$   $\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$   $P bs bs' \longrightarrow \exists as. bs = [] \land bs' = as$   $\bigvee_{\exists as, a, as'. bs} = as \land bs' = a\#as' \land P as as'$   $\bigvee_{\exists a, as, as'. bs} = a\#as \land bs' = a\#as' \land P as as'$ 

$$\begin{array}{c} \vdots \\ \hline (Cons) \\ \hline subll [4, 6, ...] [4, 5, ...] \\ \hline subll [4, 6, ...] [3, 4, 5, ...] \\ \hline (Subll [2, 4, 6, ...] [2, 3, 4, 5, ...] \\ \hline subll [2, 4, 6, ...] [1, 2, 3, 4, 5, ...] \\ \hline (Subll [0, 2, 4, 6, ...] [0, 1, 2, 3, 4, 5, ...] \\ \hline (Cons) \\ \hline \hline (Cons) \\ \hline (Cons) \\ \hline \hline (Cons) \\ \hline \hline (Cons) \\ \hline \hline (Cons) \\ \hline \hline (Con$$

So accepting infinite proofs with our introduction rules would solve our problem... But how about something finitely expressible – a <u>blueprint</u> for an infinite proof? *P* bs bs' =  $\exists k \in \mathbb{N}$ . bs =  $[2k, 2k + 2, 2k + 4 \dots] \land \overline{bs'} = [2k, 2k + 1, 2k + 2, \dots] \lor bs = [2k + 2, 2k + 4 \dots] \land bs' = [2k + 1, 2k + 2, \dots]$ 

(Coinduct)

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

$$\forall bs, bs'. P bs bs' \longrightarrow (\exists as. bs = [] \land bs' = as) \lor (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \lor (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as')$$

 $subll\ cs\ cs'$ 

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

$$\forall bs, bs'. P bs bs' \longrightarrow \begin{array}{l} (\exists as. bs = [] \land bs' = as) \lor \\ (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor \\ (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \end{array}$$

$$(\mathsf{Coinduct})$$

subll  $cs \ cs'$ 

Terminology:  $\underline{\text{consistent}}$  with some rules = "closed backwards" under these rules

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

$$\forall bs, bs'. P bs bs' \longrightarrow \begin{pmatrix} \exists as. bs = [] \land bs' = as \rangle \lor \\ (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor \\ (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \end{pmatrix}$$

$$(Coinduct)$$

subll  $cs \ cs'$ 

Terminology:  $\underline{\text{consistent}}$  with some rules = "closed backwards" under these rules

Coinduction says: If a relation P is consistent with the introduction rules (Nil), (ConsR) and (Cons), then P cs cs' implies *subll cs cs'* (for every cs, cs'), i.e.,  $P \leq subll$ .

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

$$\forall bs, bs'. P bs bs' \longrightarrow \begin{pmatrix} \exists as. bs = [] \land bs' = as \rangle \lor \\ (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor \\ (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \end{pmatrix}$$
(Coinduct)

subll  $cs \ cs'$ 

Terminology:  $\underline{\text{consistent}}$  with some rules = "closed backwards" under these rules

Coinduction says: If a relation P is consistent with the introduction rules (Nil), (ConsR) and (Cons), then P cs cs' implies *subll* cs cs' (for every cs, cs'), i.e.,  $P \leq subll$ . In other words, *subll* is the greatest (largest) relation that is consistent with the introduction rules.

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

$$\begin{array}{l} (\exists as. \ bs = [] \land bs' = as) \lor \\ \forall bs, bs'. \ P \ bs \ bs' \longrightarrow \\ (\exists as, a, as'. \ bs = as \land bs' = a\#as' \land P \ as \ as') \lor \\ (\exists a, as, as'. \ bs = a\#as \land bs' = a\#as' \land P \ as \ as') \end{aligned}$$

- (Coinduct)

subll cs cs'

This leads to the coinduction rule for *subll*:

 $P \ cs \ cs'$ 

 $\forall bs, bs'. P bs bs' \longrightarrow \begin{pmatrix} \exists as. bs = [] \land bs' = as \end{pmatrix} \lor \\ (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor \\ (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \end{cases}$ 

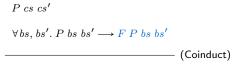
(Coinduct)



Remember the operator F on relations extracted from the intro rules:

$$\begin{array}{l} (\exists as. \ bs = [] \land bs' = as) \lor \\ F \ P = \lambda bs, \ bs'. \quad (\exists as, a, as'. \ bs = as \land bs' = a\#as' \land P \ as \ as') \lor \\ (\exists a, as, \ as'. \ bs = a\#as \land bs' = a\#as' \land P \ as \ as') \end{array}$$

This leads to the coinduction rule for *subll*:

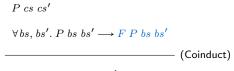




Remember the operator F on relations extracted from the intro rules:

$$\begin{array}{ll} (\exists as. bs = [] \land bs' = as) \lor \\ F P &= \lambda bs, bs'. & (\exists as, a, as'. bs = as \land bs' = a\#as' \land P as as') \lor \\ (\exists a, as, as'. bs = a\#as \land bs' = a\#as' \land P as as') \end{array}$$

This leads to the coinduction rule for *subll*:





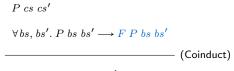
Remember the operator F on relations extracted from the intro rules:

$$\begin{array}{ll} (\exists as. \ bs = [] \land bs' = as) \lor \\ F \ P &= \lambda bs, \ bs'. & (\exists as, a, as'. \ bs = as \land bs' = a\#as' \land P \ as \ as') \lor \\ (\exists a, as, as'. \ bs = a\#as \land bs' = a\#as' \land P \ as \ as') \end{array}$$

Alternative formulation of the rule:

$$\frac{P \le F P}{P \le subll}$$
 (Coinduct)

This leads to the coinduction rule for *subll*:





Remember the operator F on relations extracted from the intro rules:

$$\begin{array}{ll} (\exists as. \ bs = [] \land bs' = as) \lor \\ F \ P &= \lambda bs, \ bs'. & (\exists as, a, as'. \ bs = as \land bs' = a\#as' \land P \ as \ as') \lor \\ (\exists a, as, as'. \ bs = a\#as \land bs' = a\#as' \land P \ as \ as') \end{array}$$

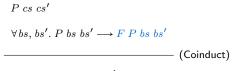
Alternative formulation of the rule:

$$\frac{P \le F P}{P \le subll}$$
 (Coinduct)

And since also  $subll \leq F$  subll, we have that subll is the largest post-fixpoint of F

#### Desired properties for the predicate subll

This leads to the coinduction rule for *subll*:





Remember the operator F on relations extracted from the intro rules:

$$\begin{array}{ll} (\exists as. \ bs = [] \land bs' = as) \lor \\ F \ P &= \lambda bs, \ bs'. & (\exists as, a, as'. \ bs = as \land bs' = a\#as' \land P \ as \ as') \lor \\ (\exists a, as, \ as'. \ bs = a\#as \land bs' = a\#as' \land P \ as \ as') \end{array}$$

Alternative formulation of the rule:

$$\frac{P \le F P}{P \le subll}$$
 (Coinduct)

And since also  $subll \leq F subll$ , we have that subll is the largest post-fixpoint of F – Knaster-Tarski again!

The relation subll: LazyList $(A) \rightarrow$  LazyList $(A) \rightarrow$  Bool specified coinductively by the rules:

$$\frac{\cdot}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$$
$$\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$$

The relation subll: LazyList $(A) \rightarrow$  LazyList $(A) \rightarrow$  Bool specified coinductively by the rules:

$$\frac{\cdot}{subll [] as} \text{ (Nil)} \qquad \frac{subll as as'}{subll as (a\#as')} \text{ (ConsR)}$$
$$\frac{subll as as'}{subll (a\#as) (a\#as')} \text{ (Cons)}$$

"Coinductively" means: greatest relation consistent with the given rules.

The relation subll: LazyList $(A) \rightarrow$  LazyList $(A) \rightarrow$  Bool specified coinductively by the rules:

$$\frac{1}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a#as')} (ConsR)$$
$$\frac{subll as as'}{subll (a#as) (a#as')} (Cons)$$

"Coinductively" means: greatest relation consistent with the given rules. More precisely: We define  $subll = J_F$ , where  $F : (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool) \rightarrow (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool)$  is defined as follows, for all  $R : LazyList(A) \rightarrow LazyList(A) \rightarrow Bool$ :

The relation subll: LazyList $(A) \rightarrow$  LazyList $(A) \rightarrow$  Bool specified coinductively by the rules:

$$\frac{1}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$$
$$\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$$

"Coinductively" means: greatest relation consistent with the given rules. More precisely: We define *subll* =  $J_F$ , where  $F : (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool) \rightarrow (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool)$  is defined as follows, for all  $R : LazyList(A) \rightarrow LazyList(A) \rightarrow Bool$ :

$$F R = \lambda bs, bs' \exists as. bs = [] \land bs' = as$$

$$\forall \exists as, a, as'. bs = as \land bs' = a\#as' \land R as as'$$

$$\forall \exists a, as, as'. bs = a\#as \land bs' = a\#as' \land R as as'$$

The relation subll: LazyList $(A) \rightarrow$  LazyList $(A) \rightarrow$  Bool specified coinductively by the rules:

$$\frac{1}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$$
$$\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$$

"Coinductively" means: greatest relation consistent with the given rules. More precisely: We define  $subll = J_F$ , where  $F : (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool) \rightarrow (LazyList(A) \rightarrow LazyList(A) \rightarrow Bool)$  is defined as follows, for all  $R : LazyList(A) \rightarrow LazyList(A) \rightarrow Bool$ :

$$F R = \lambda bs, bs'. \exists as. bs = [] \land bs' = as$$

$$\forall \exists as, a, as'. bs = as \land bs' = a\#as' \land R as as'$$

$$\forall \exists a, as, as'. bs = a\#as \land bs' = a\#as' \land R as as'$$

$$F \text{ is monotonic, so } J_F \text{ exists by Knaster-Tarski.}$$

| Thanks to                                      | we obtain                                     |
|--|---|
| subll being a pre-fixpoint of $F$              | the introduction rules (Nil), (ConsR), (Cons) |
| subll being a post-fixpoint of $F$             | the case distinction rule (Cases)             |
| $subll$ being $\geq$ all post-fixpoints of $F$ | the coinduction rule (Coinduct)               |

| Thanks to                                      | we obtain                                     |
|--|---|
| subll being a pre-fixpoint of $F$              | the introduction rules (Nil), (ConsR), (Cons) |
| subll being a post-fixpoint of $F$             | the case distinction rule (Cases)             |
| $subll$ being $\geq$ all post-fixpoints of $F$ | the coinduction rule (Coinduct)               |

$$\frac{\cdot}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$$
$$\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$$

| Thanks to  | we obtain                                     |  |
|--|---|--|
| subll being a pre-fixpoint of $F$  | the introduction rules (Nil), (ConsR), (Cons) |  |
| subll being a post-fixpoint of $F$   | the case distinction rule (Cases)             |  |
| $subll$ being $\geq$ all post-fixpoints of $F$   | the coinduction rule (Coinduct)               |  |
| $\frac{\cdot}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a \# as')} (ConsR)$  |   |  |
| $\frac{subll \ as \ as'}{subll \ (a\#as) \ (a\#as')} \ ({\sf Cons})$   |   |  |
| $subll \ bs \ bs' \qquad \forall as. \ bs = [] \land bs' = as \longrightarrow P$<br>$\forall as, \ as', \ a. \ bs = as \land bs' = a\#as' \land subll \ as \ as' \longrightarrow P$<br>$\forall as, \ as', \ a. \ bs = a\#as \land bs' = a\#as' \land subll \ as \ as' \longrightarrow P$<br>(Cases) |   |  |
| P  |   |  |

| Thanks to  | we obtain  |  |
|--|--|--|
| subll being a pre-fixpoint of $F$  | the introduction rules (Nil), (ConsR), (Cons)                    |  |
| subll being a post-fixpoint of $F$   | the case distinction rule (Cases)                                |  |
| $subll$ being $\geq$ all post-fixpoints of $F$   | the coinduction rule (Coinduct)                                  |  |
| $\frac{1}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$ $\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$   |  |  |
| $P \ cs \ cs'$   |  |  |
|  | $[] \land bs' = as) \lor$  |  |
| $\forall bs, bs'. P bs bs' \longrightarrow (\exists a, a, as) (\exists a, a, as) (\exists a, a, a) (\exists a, b, a) (a, b,$ | $A' \cdot bs = as \wedge bs' = a\#as' \wedge P \ as \ as') \vee$ |  |
| $\frac{(\exists a, as, as'. bs = a \# as \land bs' = a \# as' \land P as as')}{subll \ cs \ cs'} $ (Coinduct)  |  |  |

| Thanks to  | we obtain  |  |  |
|--|--|--|--|
| subll being a pre-fixpoint of $F$  | the introduction rules (Nil), (ConsR), (Cons)                                      |  |  |
| subll being a post-fixpoint of $F$   | the case distinction rule (Cases)  |  |  |
| $subll$ being $\geq$ all post-fixpoints of $F$   | the coinduction rule (Coinduct)  |  |  |
| $\frac{\cdot}{subll [] as} (Nil) \qquad \frac{subll as as'}{subll as (a\#as')} (ConsR)$ $\frac{subll as as'}{subll (a\#as) (a\#as')} (Cons)$ |  |  |  |
| $P \ cs \ cs'$   |  |  |  |
|  | $[] \land bs' = as) \lor$  |  |  |
|  | $(bs = as \land bs' = a \# as' \land (subll \ as \ as' \lor P \ as \ as')) \lor$   |  |  |
| $(\exists a, as, as)$  | $'. bs = a \# as \land bs' = a \# as' \land (subll \ as \ as' \lor P \ as \ as'))$ |  |  |
|  | subll cs cs'   |  |  |

*subll* is also the greatest (post-)fixpoint of  $G = \lambda P. F(subll \lor P) = \lambda P. F(\lambda as, as'. subll as as' \lor P as as').$ 

# Induction versus Coinduction

The semantic foundations for induction and coinduction are perfectly dual – via Knaster-Tarski:

- induction: smallest/least pre-fixpoint
- coinduction: largest/greatest post-fipoint

But they have quite different intuitions:

- induction whatever can be proved using a finite number of rule applications
- coinduction whatever can be proved using an infinite number of rule applications

# Induction versus Coinduction

The semantic foundations for induction and coinduction are perfectly dual – via Knaster-Tarski:

- induction: smallest/least pre-fixpoint
- coinduction: largest/greatest post-fipoint

But they have quite different intuitions:

- induction whatever can be proved using a finite number of rule applications
- coinduction whatever can be proved using an infinite number of rule applications

"Smallest versus largest" is mathematically elegant, but the "finite versus infinite proofs" is what gives us guidance when defining or proving things.

# Induction versus Coinduction

The semantic foundations for induction and coinduction are perfectly dual – via Knaster-Tarski:

- induction: smallest/least pre-fixpoint
- coinduction: largest/greatest post-fipoint

But they have quite different intuitions:

- induction whatever can be proved using a finite number of rule applications
- coinduction whatever can be proved using an infinite number of rule applications

"Smallest versus largest" is mathematically elegant, but the "finite versus infinite proofs" is what gives us guidance when defining or proving things.

Can we make this intuition precise?

Fix a set A. A <u>rule</u> over A is a pair r = (H, a),  $H \subseteq A$  is a finite set and  $a \in A$ .

Fix a set A. A rule over A is a pair r = (H, a),  $H \subseteq A$  is a finite set and  $a \in A$ . If H has the form  $\{a_1, \ldots, a_n\}$ , we can write the rule r as

 $\frac{a_1 \dots a_n}{a}$ 

Fix a set A. A rule over A is a pair r = (H, a),  $H \subseteq A$  is a finite set and  $a \in A$ . If H has the form  $\{a_1, \ldots, a_n\}$ , we can write the rule r as

 $\frac{a_1 \dots a_n}{a}$ 

If  $H = \emptyset$ , the rule r is called an axiom and can be written as follows:

 $\overline{a}$ 

Fix a set A. A rule over A is a pair r = (H, a),  $H \subseteq A$  is a finite set and  $a \in A$ . If H has the form  $\{a_1, \ldots, a_n\}$ , we can write the rule r as

 $\frac{a_1 \dots a_n}{a}$ 

If  $H = \emptyset$ , the rule r is called an axiom and can be written as follows:

 $\overline{a}$ 

We fix  $\mathcal{R}$ , a set of rules over A. We define/specify  $I_{\mathcal{R}}$  inductively by the rules in  $\mathcal{R}$ , namely:

$$(H, a) \in \mathcal{R} \qquad \forall b \in H. \ \mathsf{I}_{\mathcal{R}} \ b$$
$$\mathsf{I}_{\mathcal{R}} \ a$$

We define/specify  $J_{\mathcal{R}}$  coinductively by the same rules, namely:

$$\frac{(H,a)\in\mathcal{R}}{\mathsf{J}_{\mathcal{R}}\ a}\qquad \forall b\in H.\ \mathsf{J}_{\mathcal{R}}\ b$$

According to our semantic recipe, the above mean:

We define  $F : (A \rightarrow Bool) \rightarrow (A \rightarrow Bool)$ , the <u>operator associated to  $\mathcal{R}$ </u>, by applying the rules to its input predicate (like we did before in our examples):

$$F P = \lambda a. \exists H. (H, a) \in \mathcal{R} \land (\forall a \in H. P a)$$

F is monotonic, so  $I_F$  and  $J_F$  exist by Knaster-Tarski.

We define

- $I_{\mathcal{R}} = I_F$
- $J_{\mathcal{R}} = J_F$

An <u>*R*-proof tree</u>  $\pi$  is a (possibly infinite) tree whose nodes are labeled with elements of *A* and such that successor nodes correspond to rules; more precisely, if a node *N* is labeled with *a* and its successor nodes  $N_1, \ldots, N_k$  are labelled with  $a_1, \ldots, a_k$ , then  $(\{a_1 \ldots a_k\}, a) \in \mathcal{R}$ .

An <u>*R*-proof tree</u>  $\pi$  is a (possibly infinite) tree whose nodes are labeled with elements of *A* and such that successor nodes correspond to rules; more precisely, if a node *N* is labeled with *a* and its successor nodes  $N_1, \ldots, N_k$  are labelled with  $a_1, \ldots, a_k$ , then  $(\{a_1 \ldots a_k\}, a) \in \mathcal{R}$ .

If a labels the root of an  $\mathcal{R}$ -proof tree  $\pi$ , we say that  $\pi$  proves a.

An <u>*R*-proof tree</u>  $\pi$  is a (possibly infinite) tree whose nodes are labeled with elements of *A* and such that successor nodes correspond to rules; more precisely, if a node *N* is labeled with *a* and its successor nodes  $N_1, \ldots, N_k$  are labelled with  $a_1, \ldots, a_k$ , then  $(\{a_1 \ldots a_k\}, a) \in \mathcal{R}$ .

If a labels the root of an  $\mathcal{R}$ -proof tree  $\pi$ , we say that  $\pi$  proves a.

An  $\mathcal{R}$ -proof tree is said to be finite if its set of nodes is finite.

An <u>*R*-proof tree</u>  $\pi$  is a (possibly infinite) tree whose nodes are labeled with elements of *A* and such that successor nodes correspond to rules; more precisely, if a node *N* is labeled with *a* and its successor nodes  $N_1, \ldots, N_k$  are labelled with  $a_1, \ldots, a_k$ , then  $(\{a_1 \ldots a_k\}, a) \in \mathcal{R}$ .

If a labels the root of an  $\mathcal{R}$ -proof tree  $\pi$ , we say that  $\pi$  proves a.

An  $\mathcal{R}$ -proof tree is said to be finite if its set of nodes is finite.

Assuming  $(\{c, b\}, a)$ ,  $(\{d, b\}, b)$ ,  $(\emptyset, c)$ ,  $(\emptyset, d)$ ,  $(\emptyset, b) \in \mathcal{R}$ , here's an example of a finite  $\mathcal{R}$ -proof tree:

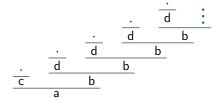
$$\frac{c}{c}$$
  $\frac{d}{b}$ 

An <u>*R*-proof tree</u>  $\pi$  is a (possibly infinite) tree whose nodes are labeled with elements of *A* and such that successor nodes correspond to rules; more precisely, if a node *N* is labeled with *a* and its successor nodes  $N_1, \ldots, N_k$  are labelled with  $a_1, \ldots, a_k$ , then  $(\{a_1 \ldots a_k\}, a) \in \mathcal{R}$ .

If a labels the root of an  $\mathcal{R}$ -proof tree  $\pi$ , we say that  $\pi$  proves a.

An  $\mathcal{R}$ -proof tree is said to be finite if its set of nodes is finite.

Assuming  $(\{c, b\}, a)$ ,  $(\{d, b\}, b)$ ,  $(\emptyset, c)$ ,  $(\emptyset, d)$ ,  $(\emptyset, b) \in \mathcal{R}$ , and here's an example of an infinite  $\mathcal{R}$ -proof tree:



#### IMO, crucial for the understanding of coinduction

# **Theorem.** For all $a \in A$ : (1) $I_{\mathcal{R}} a$ holds iff there exists a finite $\mathcal{R}$ -proof tree that proves a. (2) $J_{\mathcal{R}} a$ holds iff there exists a (possibly infinite) $\mathcal{R}$ -proof tree that proves a.

Note: It's not really about finite versus infinite – that is just a coincidence from using finitely branching rule systems.

Remove the restriction that rules (A, a) have the set of hypotheses A finite.

Say a tree is well-founded is it has no infinite paths.

#### More General Version. For all $a \in A$ :

(1)  $I_{\mathcal{R}} a$  holds iff there exists a well-founded  $\mathcal{R}$ -proof tree that proves a. (2)  $J_{\mathcal{R}} a$  holds iff there exists a (possibly non-well-founded)  $\mathcal{R}$ -proof tree that proves a.