sequent

Antecedents

consequents

$$\alpha_1, \ldots, \alpha_k \vdash \beta_1, \ldots, \beta_\ell$$

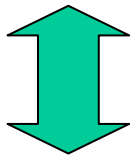
$$\alpha_1 \wedge \ldots \wedge \alpha_k \Rightarrow \beta_1 \vee \ldots \vee \beta_\ell$$

$$\{\neg\alpha_1 \vee \ldots \vee \neg\alpha_k \vee \beta_1 \vee \ldots \vee \beta_\ell\}$$

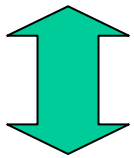
clausal form (disjunct)

antecedent/consequent exchange

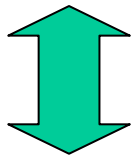
$$A, \alpha \vdash \beta, B$$



$$A \vdash \neg\alpha, \beta, B$$

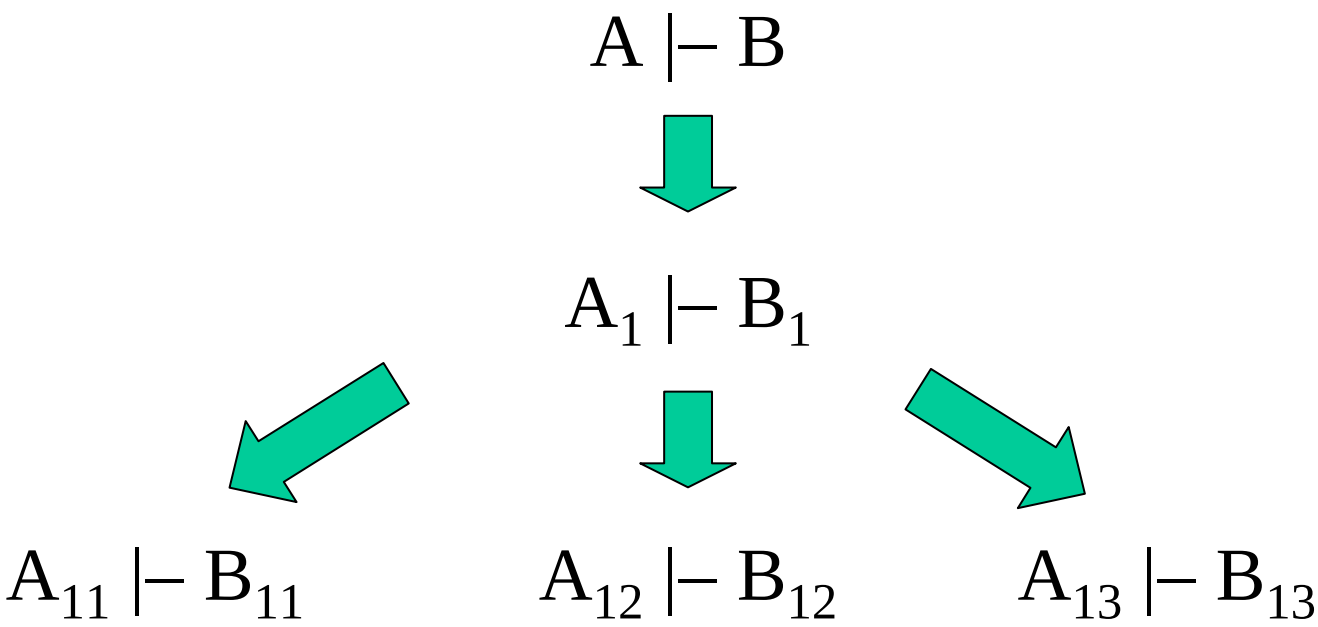


$$A, \neg\beta \vdash \neg\alpha, B$$



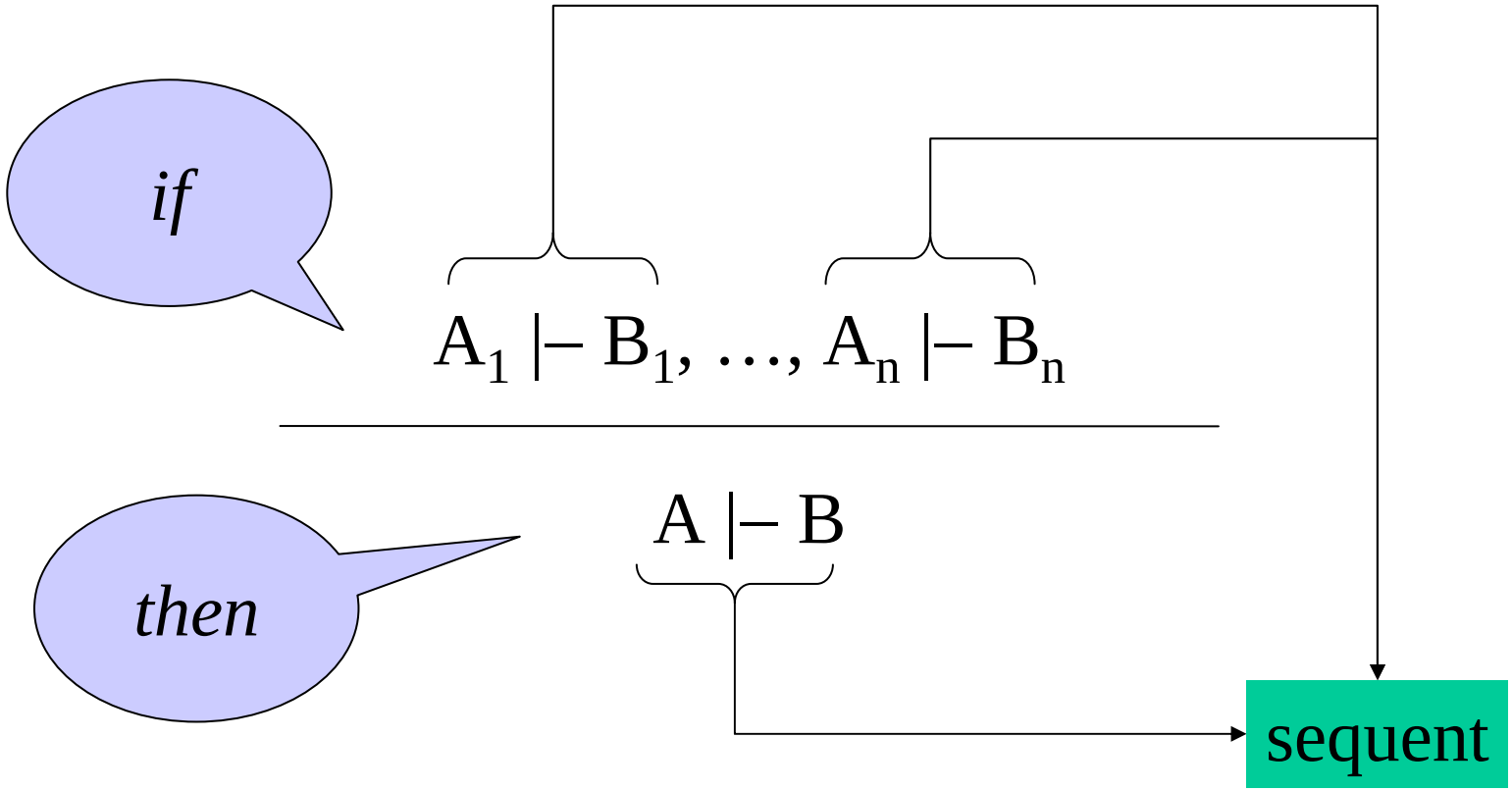
$$A, \alpha, \neg\beta \vdash B$$

proof tree

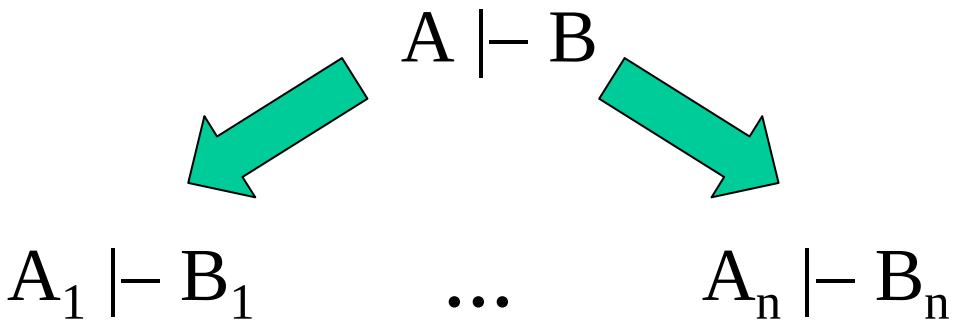


Validity of all subgoals must imply parent goal validity.
Each inference step is based on the use of an inference rule.

inference rule



inference step



Using inference rule

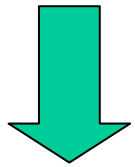
$$\frac{A_1 \vdash B_1, \ldots, A_n \vdash B_n}{A \vdash B}$$

PVS inference rules and decision procedures

hide
flatten, split, lift-if
skolem
inst, inst?
lemma, rewrite,
beta, expand, replace
assert, prop, ground
apply-extensionality
induct, generalize

hide rule
(forget some hypothesis)

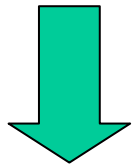
$$\alpha_1, \ldots, \alpha_{i-1}, \alpha_i, \alpha_{i+1}, \ldots, \alpha_k \vdash \beta_1, \ldots, \beta_\ell$$



$$\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_k \vdash \beta_1, \ldots, \beta_\ell$$

hide rule
(forget some conclusion)

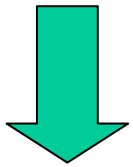
$$\alpha_1, \ldots, \alpha_k \vdash \beta_1, \ldots, \beta_{i-1}, \beta_i, \beta_{i+1}, \ldots, \beta_\ell$$



$$\alpha_1, \ldots, \alpha_k \vdash \beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_\ell$$

flatten rule
(flattening some antecedent conjunct)

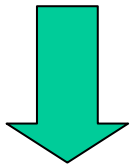
$$\mathbf{h_1 \wedge h_2}\ ,\ A\ \vdash\ B$$



$$\mathbf{h_1, h_2}\ ,\ A\ \vdash\ B$$

flatten rule
(flattening some consequent disjunct)

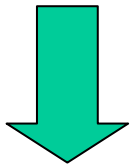
$$A \vdash \beta_1 \vee \beta_2, B$$



$$A \vdash \beta_1, \beta_2, B$$

flatten rule
(flattening some consequent implication)

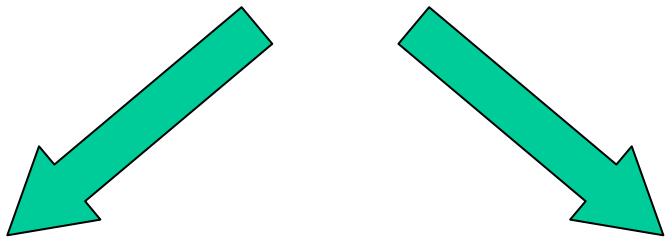
$$A \vdash \alpha \Rightarrow \beta, B$$



$$\alpha, A \vdash \beta, B$$

split rule
(consequent subgoaling)

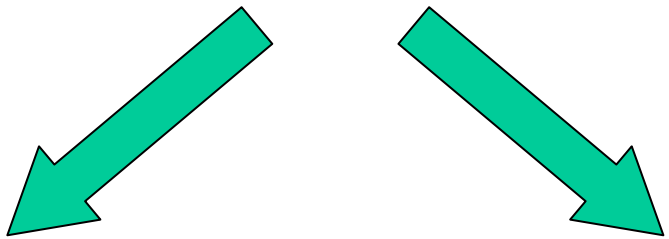
$$A \vdash \beta_1 \wedge \beta_2, B$$



$$A \vdash \beta_1, B \qquad\qquad A \vdash \beta_2, B$$

split rule
(antecedent subgoaling)

$\alpha_1 \vee \alpha_2$, A |– B



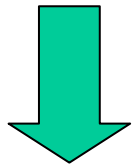
α_1, A |– B

α_2, A |– B

skolem rule

(skolemizing some consequent universal quantifier)

$$A \vdash \forall \mathbf{(x:T):}\ \beta, B$$



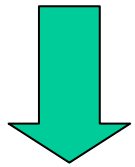
$$\boldsymbol{\xi}\mathbf{:T}, A \vdash \beta\mathbf{[x/\xi]}, B$$

ξ must be a new name, not occurring freely in the sequent.

skolem rule

(skolemizing some antecedent existential quantifier)

$$\exists(\mathbf{x:T}):\ \alpha,\ A\ \vdash\ B$$



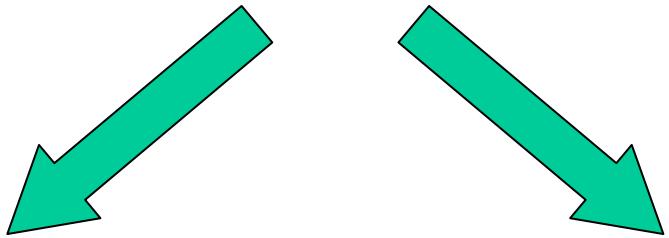
$$\alpha[\mathbf{x/\xi}],\ \xi\mathbf{:T},\ A\ \vdash\ B$$

ξ must be a new name, not occurring freely in the sequent.

inst rule

(instantiating existential variable in consequent)

$$A \vdash \exists(x{:}T){:}\ \beta, B$$



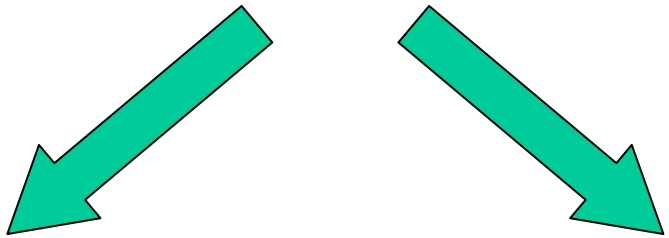
$$A \vdash \beta[x/e], B \qquad\qquad A \vdash e{:}T, B$$

finding a proper instance e may be hard!

inst rule

(instantiating universal variable in antecedent)

$\forall$**(x:T):** α, A |– B



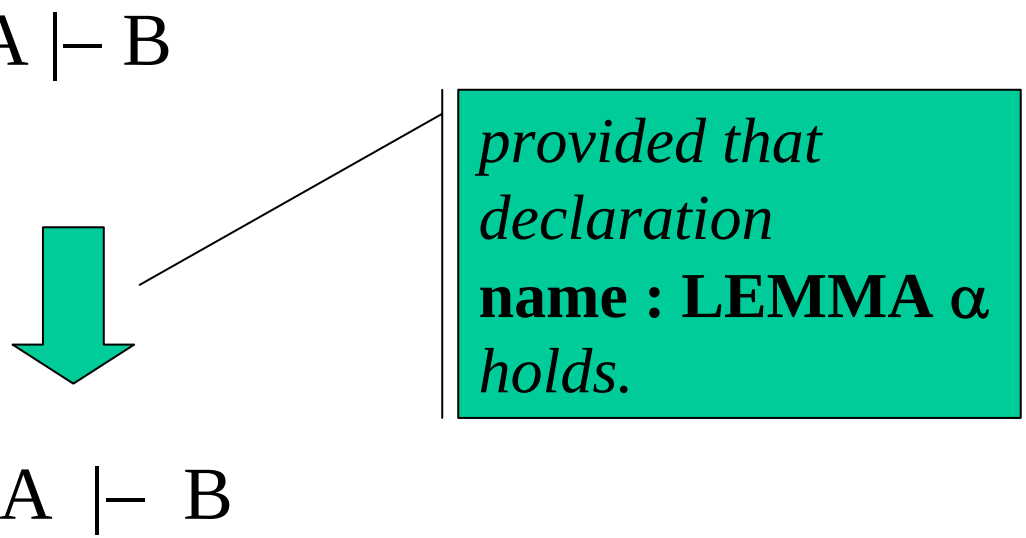
α[x/e], A |– B

$\forall$**(x:T):** α, A |– **e:T**, B

finding a proper instance e may be hard!

lemma rule

(introducing a lemma in the antecedents)

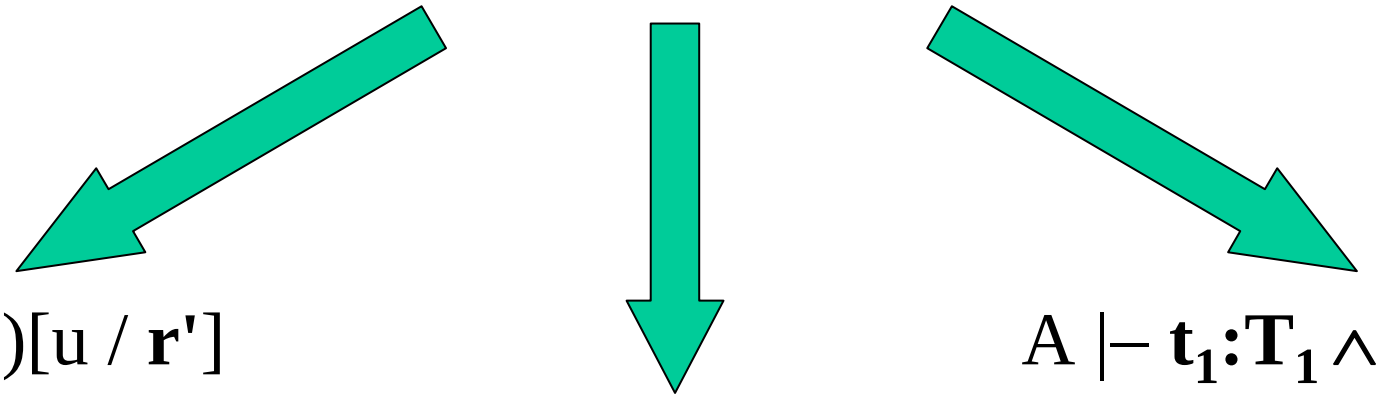


All lemmas or declarations from imported theories are hidden from antecedents.

rewrite rule

(rewriting some expression using a lemma)

$(A \vdash B)[u / \ell']$



$(A \vdash B)[u / \mathbf{r'}]$

$A \vdash \mathbf{t_1{:}T_1 \wedge t_k{:}T_k}, B$

$A \vdash \boldsymbol{\beta[x_1/t_1, \ldots, x_k/t_k]}, B$

assuming

name: LEMMA $\forall(x_1{:}T_1) \ldots \forall(x_k{:}T_k)$: $\beta \Rightarrow \ell{=}r$

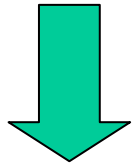
$\ell' = \ell[x_1/t_1, \ldots, x_k/t_k]$

$r' = r[x_1/t_1, \ldots, x_k/t_k]$

beta rule

(applying an anonymous function to its argument)

$$(A \mid\!- B)[u / \mathbf{(\lambda(x: T): body)(t)}]$$



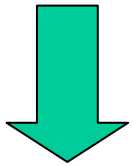
$$(A \mid\!- B)[u / \mathbf{body[x/t]}]$$

This is called β-reduction

expand rule

(rewriting a function application using a definition)

$$(A \vdash B)[u / \mathbf{f(t)}]$$



$$(A \vdash B)[u / \mathbf{body[x / t]}]$$

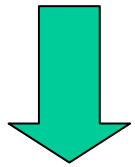
Assuming function definition

f(x:T): V = body

replace rule

(rewriting some expression using an equality)

$$(\ell = \mathbf{r},\ A \ |\!\!-\ B)[u\ /\ \ell]$$

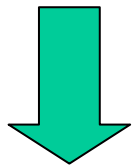


$$(\ell = \mathbf{r},\ A \ |\!\!-\ B)[u\ /\ \mathbf{r}]$$

The actual replace command may replace one or several (eventually all but one) ℓ occurrences altogether.

assert decision procedure
(proving some)

A |– B

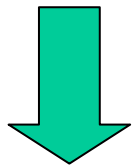


|– TRUE

provided that
A |– B can be proved using linear arithmetics

prop decision procedure
(proving a propositionally valid sequent)

A |– B

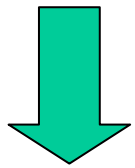


|– TRUE

provided that
A |– B
may be proved using propositional calculus only

ground decision procedure *(combination of assert, prop, and abstract datatype reasoning)*

A |– B

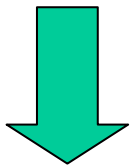


|– TRUE

provided that
A, linear_arithmetics_theory, abstract_datatype_theories |= B

apply-extensionality rule
(function equality)

$$A \vdash f = g$$



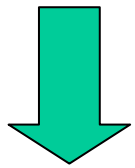
$$A \vdash \forall(x: T): f(x) = g(x)$$

assuming

$f, g: [T \rightarrow V]$

apply-extensionality rule
(datatype reasoning)

$$A \;|\!-\; \text{destructor}_i(\text{constructor}(t_1, \ldots, t_i, \ldots, t_k)) = t_i$$



$$A \;|\!-\; \text{TRUE}$$

assuming some datatype constructor declaration of the form
$\text{constructor}(\text{destructor}_1: T_1, \ldots, \text{destructor}_k: T_k): T$

well founded induction
(principle)

$$A \vdash \forall(x: T): p(x)$$

$$\xi : T, \forall(y:T): y << \xi \Rightarrow p(y), A \vdash p(\xi)$$

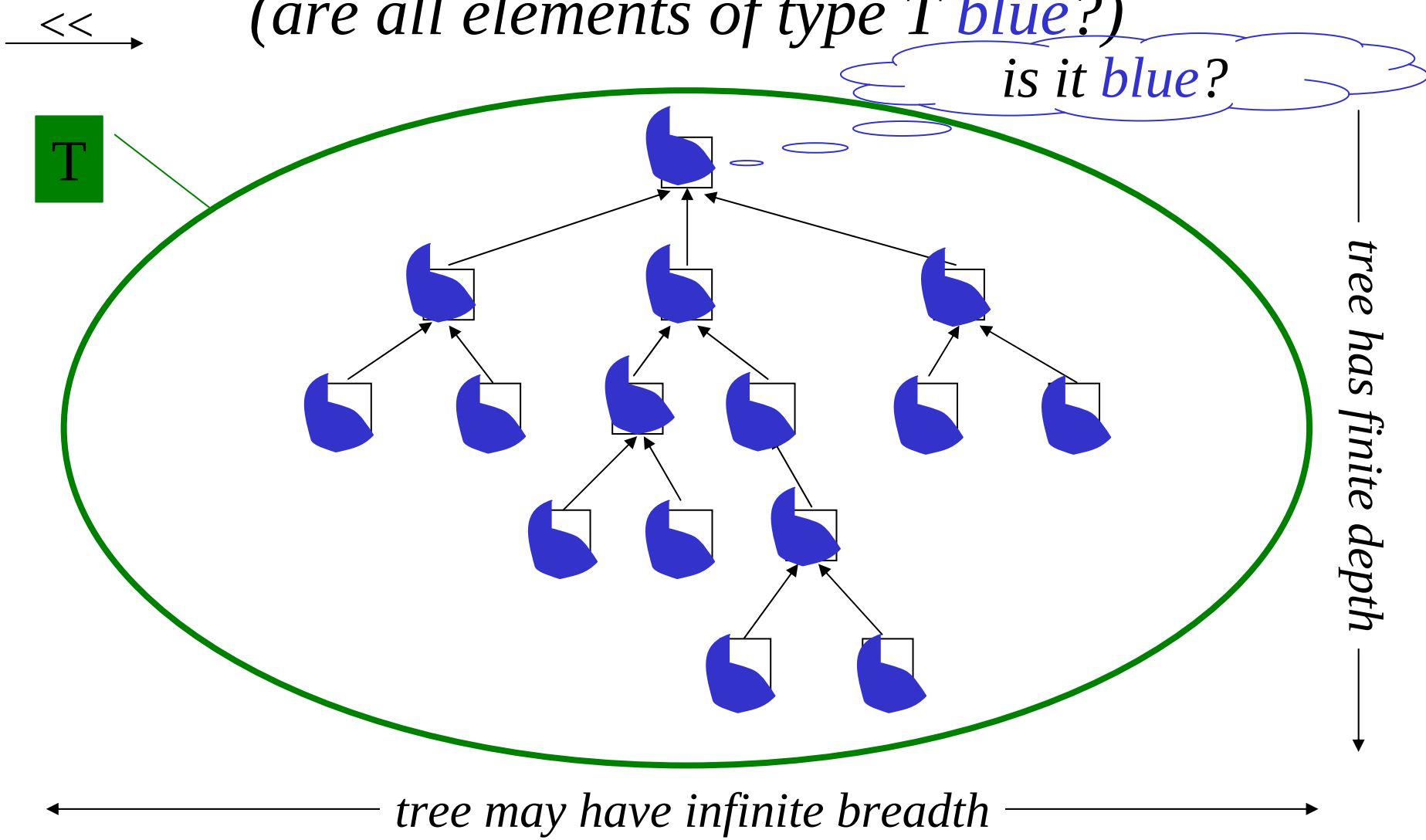
$$A \vdash well\_founded?[T](<<)$$

ξ must be a new name, not occurring freely in the sequent.

Well founded induction is based on "wf_induction[T]" prelude lemma; induct rule specializes the general principle, for each recursive type T.

proof idea of “wf_induction” lemma

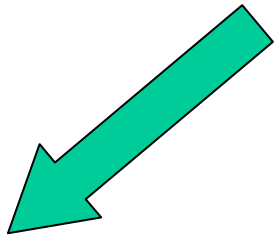
(are all elements of type T blue?)

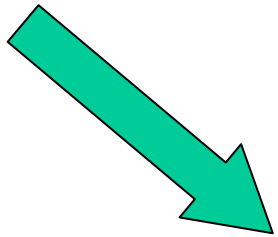


induct rule

(structural form of well founded induction: list example)

A |– ∀(ℓ: list[T]): p(ℓ)





A |– p(null[T])

t: T, ℓ: list[T], p(ℓ), A |– p(cons(t, ℓ))

***t** and ℓ are new names, not occurring freely in the sequent.*

Structural induction is just a special form of general well founded induction, based on some recursive datatype.

generalize rule
(generalization)

(A |– **c[u / t]**, B

A |– ∀**(x: T): c[u / x]**, B

A |– **t : T**, B

generalization often precedes induction;
generalization is somehow the inverse of instanciation.