Kruskal's tree theorem in Type Theory

Dominique Larchey-Wendling
TYPES team

LORIA – CNRS
Nancy, France

`http://www.loria.fr/~larchey/Kruskal`

Foundations for the Practical Formalization of Mathematics 2017

Well Quasi Orders (WQO) 1/2

- Important concept in Computer Science:
  - strenghtens well-foundedness, more stable
  - termination of rewriting (Dershowitz, RPO)
  - size-change termination, terminator (Vytiniostis, Coquand ...)
- Important concept in Mathematics:
  - Dickson's lemma, Higman's lemma
  - Higman's theorem, Kruskal's theorem
  - Robertson-Seymour theorem (graph minor theorem)
  - Unprovability result: Kruskal theorem not in PA (Friedman)

Well Quasi Orders (WQO) 2/2

- for $\leq$ a quasi order over X: reflexive & transitive binary relation
- several **classically** equivalent definitions (see e.g. JGL 2013)
  - almost full: each $(x_i)_{i\in\mathbb{N}}$ has a *good pair* ($x_i \leq x_j$ with $i < j$)
  - $\leq$ well-founded and no ∞ antichain
  - finite basis: $U = {\uparrow}U$ implies $U = {\uparrow}F$ for some finite F
  - $\{{\downarrow}U \mid U \subseteq X\}$ well-founded by $\subset$
- many of these equivalences **do not hold** intuitionistically

WQOs are stable under type constructs

- Given a WQO $\leq$ on X, we can lift $\leq$ to WQOs on:

  Higman lemma: $\texttt{list}(X)$ with $\texttt{subword}(\leq)$

  Higman thm: $\texttt{btree}(k, X)$ with $\texttt{emb\_product}(\leq)$ (any $k \in \mathbb{N}$)

  Kruskal theorem: $\texttt{tree}(X)$ with $\texttt{emb\_homeo}(\leq)$

- These theorem are *closure properties* of the class of WQOs

- Other noticable results:

  Dickson's lemma: $(\mathbb{N}^k, \leq)$ is a WQO

  Finite sequence thm: $\texttt{list}(\mathbb{N})$ WQO under $\texttt{subword}(\leq)$

  Ramsey theorem: $\leq_1$ and $\leq_2$ WQOs imply $\leq_1 \times \leq_2$ WQO

What Intuitionistic Kruskal Tree Theorem?

- The meaning of those closure theorems intuitionistically:
  - depends of what is a WQO (which definition?)
  - but not on e.g. `emb_homeo` which has an inductive definition
- What is a suitable intuitionistic definition of WQO ?
  - quasi-order does not play an important/difficult role
  - should be classically equivalent to the usual definition
  - should intuitionistically imply almost full
  - intuitionistic WQOs must be stable under liftings
- Allow the proof and use of Ramsey, Higman, Kruskal... results

Intuitionistic formulations of WQOs 1/2

- Almost full relations (Veldman&Bezem 93)
  - each $(x_i)_{i \in \mathbb{N}}$ has $x_i \; R \; x_j$ with $i < j$
  - works for Higman and Kruskal theorems (Veldman 04)
  - uses *stumps* over $\mathbb{N}$ which require *Brouwer's thesis*
- Bar induction (Coquand&Fridlender 93)
  - `bar extends` (`good` R) [ ]
  - works for the general Higman lemma (Fridlender 97)
- Well-foundedness (Seisenberger 2003)
  - $\texttt{extends}^{(-1)}$ is well-founded on $\texttt{Bad}(R)$
  - works for Higman lemma and Kruskal theorem
  - requires *decidability* of R

Intuitionistic formulations of WQOs 2/2

- Almost full relations (Vytiniostis&Coquand&Wahlstedt 12)
  - $\texttt{af}(R)$ inductively defined
  - works for Ramsey theorem
  - intuitionistically equivalent to $\texttt{bar extends}\ (\texttt{good}\, R)$ [ ]
- Seisenberger's definition not equiv. to Coquand&Fridlender for undecidable R
- Veldman&Bezem definition works for R over $\mathbb{N}$ (not over arbitrary types) but requires Brouwer's thesis
- Let us introduce
  - bar inductive predicates
  - Coquand et al. inductive definition of almost full

Bar inductive predicate, accessibility predicate (i)

- Given $\mathcal{T} : X \to X \to \texttt{Prop}$, $x : X$ and $Q : X \to \texttt{Prop}$
- x *bars* Q if every ∞ path from x meets Q
- x *is accessible* if every ∞ path from x meets $\_ \mapsto \texttt{False}$
- Inductive definitions (`Prop` or `Type`) are stronger (intui.)

$$\frac{Q\ x}{\texttt{bar}\ \mathcal{T}\ Q\ x} \qquad \frac{\forall y, \mathcal{T}\ x\ y \to \texttt{bar}\ \mathcal{T}\ Q\ y}{\texttt{bar}\ \mathcal{T}\ Q\ x} \quad \Big| \quad \frac{\forall y, \mathcal{T}\ x\ y \to \texttt{acc}\ \mathcal{T}\ y}{\texttt{acc}\ \mathcal{T}\ x}$$

- Axioms (like Brouwer's bar thesis) for equivalence
- Obviously: $\texttt{acc}\ \mathcal{T}\ x$ iff $\texttt{bar}\ \mathcal{T}\ (\_ \mapsto \texttt{False})\ x$

FAN theorem and bar over lists (ii)

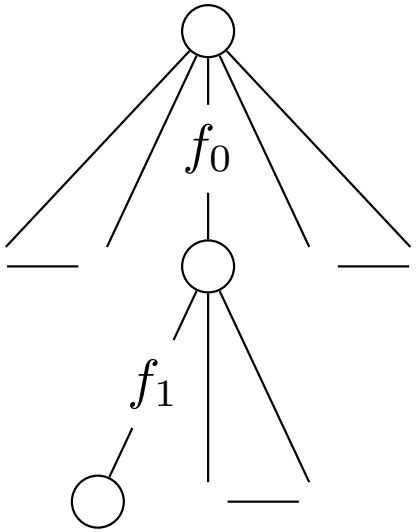
- inductive FAN theorem: $\boxed{\texttt{bar}\ \mathcal{T}\ Q\ x \to \texttt{bar}\ \mathcal{T}^\circ\ \forall Q\ [x]}$
  - for *monotonic* Q: $\forall x\, y, \mathcal{T}\ x\ y \to Q\ x \to Q\ y$
  - $\mathcal{T}^\circ\ l\ m$ iff $\forall y, y \in m \to \exists x, x \in l \wedge \mathcal{T}\ x\ y$ (direct image)
  - $(\forall Q)\ l$ iff $\forall x, x \in l \to Q\ x$ (finite quantification)
- We use $\texttt{bar}\ \mathcal{T}\ Q$ with $\mathcal{T} = \texttt{extends}$ and $Q = \texttt{good}\ R$
  - $\texttt{extends}\ l\ m$ iff $m = \_ :: l$
  - $\texttt{good}\ R\ ll$ iff $ll = l +\!\!+ \boxed{b} :: m +\!\!+ \boxed{a} :: r$ for some $a\ R\ b$
- $\texttt{bar}\ \texttt{extends}\ (\texttt{good}\ R)\ [\,]$ iff

  $\boxed{\text{iterated extensions of } [\,] \text{ must cross a good list}}$
- every infinite sequence contains a good pair (almost full)

Well-founded trees over a type X

- Well-founded trees $\mathtt{wft}(X)$
  - branching indexed by X
  - the least fixpoint of $\boxed{\mathtt{wft}(X) = \{\star\} + X \to \mathtt{wft}(X)}$
- Given a branch $f : \mathbb{N} \to X$, compute its height:

  - $f(1 + \cdot) = x \mapsto f(1 + x)$
  - $\mathtt{ht}(\mathtt{inl}\,\star, \_) = 0$
  - $\mathtt{ht}(\mathtt{inr}\,g, f) = 1 + \mathtt{ht}(g(f_0), f(1 + \cdot))$



- Veldman's stumps are sets of branches of trees in $\mathtt{wft}(\mathbb{N})$

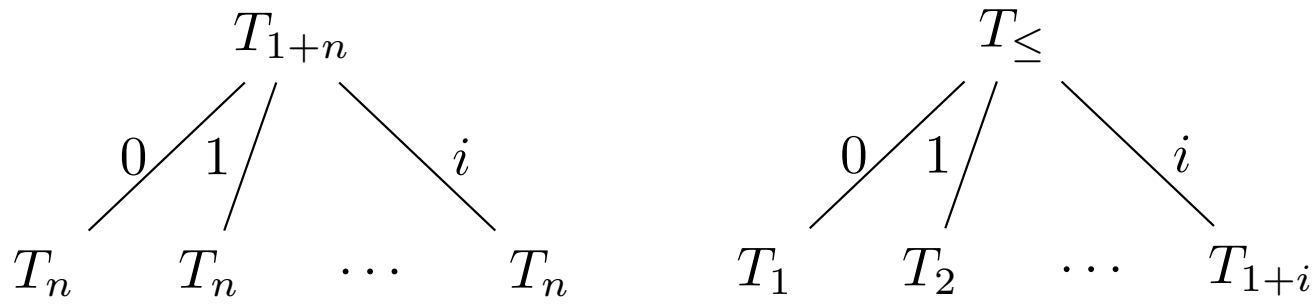
Coquand's Almost full relations, step by step

1. Veldman et al.: $\forall f : \mathbb{N} \to X,\ \exists i < j,\ f_i\ R\ f_j$
2. Logically eq. variant: $\forall f : \mathbb{N} \to X,\ \exists n,\ \exists i < j < n,\ f_i\ R\ f_j$
3. Partially informative: $\forall f : \mathbb{N} \to X,\ \{n \mid \exists i < j < n,\ f_i\ R\ f_j\}$
4. Variant: $\{h : (\mathbb{N} \to X) \to \mathbb{N} \mid \forall f,\ \exists i < j < h(f),\ f_i\ R\ f_j\}$
5. Variant: $\{t : \mathtt{wft}(X) \mid \forall f,\ \exists i < j < \mathtt{ht}(t, f),\ f_i\ R\ f_j\}$
6. Coquand et al.: is defined as an inductive predicate $\mathtt{af\_t}(R)$

- the prefix of length $\mathtt{ht}(t, f)$ of $f : \mathbb{N} \to X$ contains a good pair
- the *computational content* is (for every sequence $f : \mathbb{N} \to X$):
  - a $\boxed{\text{bound}}$ on the size of the search space for good pairs
  - and it is not a good pair

A well-founded tree for $(\mathbb{N}, \leq)$

- Property: $\forall f : \mathbb{N} \to \mathbb{N},\ \exists i < j < 2 + f_0,\ f_i \leq f_j$
- In $\mathtt{wft}(\mathbb{N})$, we define T_n the tree of uniform height n:
  - $T_0 = \mathtt{inl}(\star)$ and $T_{1+n} = \mathtt{inr}(\_ \mapsto T_n)$
  - for any $f : \mathbb{N} \to \mathbb{N}$, $\mathtt{ht}(T_n, f) = n$
- And $T_\leq = \mathtt{inr}(n \mapsto T_{1+n})$



- Hence $\mathtt{ht}(T_\leq, f) = 1 + \mathtt{ht}(T_{1+f_0}, f(1 + \cdot)) = 2 + f_0$

Almost full relations, inductively

- Lifted relation: $x\ (R \uparrow u)\ y = x\ R\ y \vee u\ R\ x$
  - in $R \uparrow u$, elements above u are forbidden in bad sequences
- $\mathtt{full} : \mathtt{rel}_2\ X \to \boxed{\mathtt{Prop}}$ and $\mathtt{af\_t} : \mathtt{rel}_2\ X \to \boxed{\mathtt{Type}}$

$$\frac{\forall x, y,\ x\ R\ y}{\mathtt{full}\ R} \qquad \frac{\mathtt{full}\ R}{\mathtt{af\_t}\ R} \qquad \frac{\forall u, \mathtt{af\_t}(R \uparrow u)}{\mathtt{af\_t}\ R}$$

- $\mathtt{af\_securedby} : \mathtt{wft}(X) \to \mathtt{rel}_2\ X \to \mathtt{Prop}$:
  - $\mathtt{af\_securedby}(\mathtt{inl}\ \star, R) = \mathtt{full}\ R$
  - $\mathtt{af\_securedby}(\mathtt{inr}\ g, R) = \forall u, \mathtt{af\_securedby}(g(u), R \uparrow u)$
- these are intuitionistically "equivalent" (hold in `Type`, not `Prop`):
  - $\mathtt{af\_t}\ R$ and $\left\{t : \mathtt{wft}(X) \;\middle|\; \mathtt{af\_securedby}(t, R)\right\}$
  - and $\left\{t : \mathtt{wft}(X) \;\middle|\; \forall f,\ \exists i < j < \mathtt{ht}(t, f),\ \ f_i\ R\ f_j\right\}$

Almost full relations, by bar inductive predicates

- $\texttt{good}\ R : \texttt{list}\ X \to \texttt{Prop}$
  - $\texttt{good}\ R\ ll$ iff $ll = l +\!\!+ \boxed{b} :: m +\!\!+ \boxed{a} :: r$ for some $a\ R\ b$
  - beware of the (implicit) use snoc lists
  - good has an easy inductive definition
- for $P : \texttt{list}\ X \to \texttt{Prop}$, we define $\texttt{bar\_t}\ P : \texttt{list}\ X \to \texttt{Type}$

$$\frac{P\ ll}{\texttt{bar\_t}\ P\ ll} \qquad \frac{\forall u,\ \texttt{bar\_t}\ P\ (u :: ll)}{\texttt{bar\_t}\ P\ ll}$$

- we show: $\texttt{af\_t}(R \uparrow a_n \uparrow \ldots \uparrow a_1)$ iff $\texttt{bar\_t}\ (\texttt{good}\ R)\ [a_1, \ldots, a_n]$
- another characterization: $\boxed{\texttt{af\_t}\ R \text{ iff } \texttt{bar\_t}\ (\texttt{good}\ R)\ [\,]}$

Almost full relations, some properties

- `af_t_refl`: if `af_t` R then $=_X \subseteq R$ (iff in case X is finite)
- `af_t_inc`: if $R \subseteq S$ and `af_t` R then `af_t` S
- `af_t_surjective` (easy but very useful):
  - for $f : X \to Y \to \texttt{Prop}$, $R : \texttt{rel}_2\, X$ and $S : \texttt{rel}_2\, Y$
  - if f surjective: $\forall y, \{x \mid f\ x\ y\}$
  - if f morphism: $f\ x_1\ y_1$ and $f\ x_2\ y_2$ and $x_1\ R\ x_2$ imply $y_1\ S\ y_2$
  - then `af_t` R implies `af_t` S
- Ramsey (Coquand): `af_t` R and `af_t` S imply `af_t`$(R \cap S)$
  - he deduces `af_t`$(R \times S)$ and `af_t`$(R + S)$
- *I stop because you may be almost full (but it is a MUST READ)*

Higman lemma and the subword relation

- Given $R : \mathtt{rel}_2\, X$ over a type X
- The subword relation $<_R^w : \mathtt{rel}_2\,(\mathtt{list}\, X)$ defined by 3 rules

$$\frac{}{[\,] <_R^w [\,]} \qquad \frac{l <_R^w m}{l <_R^w b :: m} \qquad \frac{a\ R\ b \quad l <_R^w m}{a :: l <_R^w b :: m}$$

- also write $\mathtt{subword}\, R$ for $<_R^w$
- Higman lemma (Fridlender 97, non informative version):

$$\mathtt{bar}\ (\mathtt{good}\, R)\ [\,] \quad \text{implies} \quad \mathtt{bar}\ (\mathtt{good}\,(\mathtt{subword}\, R))\ [\,]$$

- Nearly the same proof works for `bar_t` instead of `bar`
- But this proof cannot be generalized to finite trees...

The product tree embedding, Higman theorem

- trees with same type for all arities: $\texttt{tree}\, X = X \times \texttt{list}(\texttt{tree}\, X)$
- trees of breadth bounded by $k \in \mathbb{N}$:

$$\texttt{btree}\ k\ X = \left\{ t \;\middle|\; \texttt{tree\_fall}\ (\langle \_ | ll \rangle \mapsto \texttt{length}\, ll < k)\ t \right\}$$

- any $t \in T$ is $t = \langle x | t_1, \ldots, t_n \rangle$ with $n < k$, $x \in X$ and $t_i \in T$
- for a relation $R : \texttt{rel}_2\, X$, we define (needs some work...)

$$\frac{s <_R^\times t_i}{s <_R^\times \langle x_n | t_1, \ldots, t_n \rangle} \qquad \frac{x\ R\ y \quad s_1 <_R^\times t_1, \ldots, s_n <_R^\times t_n}{\langle x | s_1, \ldots, s_n \rangle <_R^\times \langle y | t_1, \ldots, t_n \rangle}$$

- also write $\texttt{emb\_tree\_product}\, R$ for $<_R^\times$
- Higman theorem: $\texttt{af\_t}\, R$ implies $\texttt{af\_t}(<_R^\times)$ on $\texttt{btree}\ k\ X$

The homeomorphic embedding, Krukal theorem

- one type X for all arities: $\texttt{tree}\,X = X \times \texttt{list}(\texttt{tree}\,X)$
- for $R : \texttt{rel}_2\,X$, we define $<_R^\star$ by nested induction

$$\frac{s <_R^\star t_i}{s <_R^\star \langle x_n | t_1, \ldots, t_n \rangle}$$

$$\frac{x_i\ R\ x_j \qquad [s_1, \ldots, s_i]\ (\texttt{subword} <_R^\star)\ [t_1, \ldots, t_j]}{\langle x_i | s_1, \ldots, s_i \rangle <_R^\star \langle x_j | t_1, \ldots, t_j \rangle}$$

- hand-written elimination scheme (nested induction)
- we also write $\texttt{emb\_tree\_homeo}\,R$ for $<_R^\star$
- Kruskal theorem: $\texttt{af\_t}\,R$ implies $\texttt{af\_t}(<_R^\star)$

Plan of the rest of the presentation

- high level and informal proof principles of Higman's theorem
  - with ideas from Veldman (mostly), Fridlender and Coquand
  - $\texttt{tree}(X_n)_{n<k}$, one type (and one relation) for each arity
- focus on several implementation chalenges of that proof
  - $\texttt{tree}(X_n)$ as a (decidable) subtype of $\texttt{tree}(\sum X_n)$
  - embed $\sum X_n$ in a (specialized) universe U
  - empty type grounded induction for `af_t`, ...
- what about the non-informative case `af` ?
  - beware `af` R is weaker than $\texttt{inhabited}(\texttt{af\_t}\ R)$
  - well-foundedness upto a projection
- from Higman theorem to Kruskal theorem (remarks)

The product tree embedding, Higman theorem

- $\texttt{tree}(X_n)_{n<k} = T$ where T is lfp of $T = \sum_{n=0}^{k-1} X_n \times T^n$
- one type X_n for each arity $n < k$
- any $t \in T$ is $t = \langle x_n | t_1, \ldots, t_n \rangle$ with $x_n \in X_n$ and $t_i \in T$
- for arity-indexed relations $R : \forall n < k, \texttt{rel}_2\,(X_n)$, we define

$$\frac{s <^h_R t_i}{s <^h_R \langle x_n | t_1, \ldots, t_n \rangle} \qquad \frac{x_n\ R_n\ y_n \quad s_1 <^h_R t_1, \ldots, s_n <^h_R t_n}{\langle x_n | s_1, \ldots, s_n \rangle <^h_R \langle y_n | t_1, \ldots, t_n \rangle}$$

- also write $\texttt{emb\_tree\_higman}\,R$ for $<^h_R$
- Higman thm.: $(\forall n < k, \texttt{af\_t}\,R_n)$ implies $\texttt{af\_t}(<^h_R)$

Higman theorem, based on (Veldman 2004)

- each $\texttt{af\_t}\, R_n$ is witnessed by w_n: $\texttt{af\_securedby}(w_n, R_n)$
- *easier* outermost induction on $[w_0, \ldots, w_{k-1}]$ (lexicographic)
- apply rule 2, hence prove: $\forall t, \texttt{af\_t}\,(<_R^h \uparrow t)$
- do this by structural induction on t
  - $t = \langle x_i | t_1, \ldots, t_i \rangle$ with $i < k$
  - we can assume $\texttt{af\_t}\,(<_R^h \uparrow t_1), \ldots, \texttt{af\_t}\,(<_R^h \uparrow t_i)$
  - we show $\texttt{af\_t}\,(<_R^h \uparrow \langle x_i | t_1, \ldots, t_i \rangle)$
  - depends on $i = 0$ or not, $w_i = \texttt{inl}\star$ or not

Higman thm, case of leaves ($i = 0$ and $w_0 = \mathtt{inr}\, g$)

- we have $t = \langle x_0 | \emptyset \rangle$ $(i = 0)$
- $R'_0 = R_0 \uparrow x_0$ is $\mathtt{af\_t}$, witnessed by $w'_0 = g(x_0)$
- $R'_j = R_j$ and $w'_j = w_j$ for $0 < j < k$
  - $w'_0 = g(x_0)$ is a sub-$\mathtt{wft}(X_0)$ of $w_0 = \mathtt{inr}\, g$, hence simpler
  - $[w'_0, w_1, \ldots, w_{k-1}]$ easier than $[w_0, w_1, \ldots, w_{k-1}]$
  - we deduce $\mathtt{af\_t}(<^h_{R'})$ by induction
- we show $<^h_{R'} \subseteq <^h_R \uparrow \langle x_0 | \emptyset \rangle$ (relatively easy to check)
- we conclude $\mathtt{af\_t}(<^h_R \uparrow \langle x_0 | \emptyset \rangle)$

Higman thm, case of leaves ($i = 0$ and $w_0 = \mathtt{inl}\,\star$)

- $t = \langle x_0 | \emptyset \rangle$
- $R_0 \uparrow x_0 = R_0$ because R_0 is (already) full ($w_0 = \mathtt{inl}\,\star$)
- but then we have $x_0\ R_0\ y$ for any y
- hence we deduce $\langle x_0 | \emptyset \rangle <_R^h \langle x_j | v_1, \ldots, v_j \rangle$
  - any (finite) tree contains a leaf $\langle y | \emptyset \rangle$
  - $\langle x_0 | \emptyset \rangle$ embeds into any leaf, e.g. $\langle y | \emptyset \rangle$
- we deduce $<_R^h \uparrow \langle x_0 | \emptyset \rangle$ is full (trivial to check)
- we conclude $\mathtt{af\_t}(<_R^h \uparrow \langle x_0 | \emptyset \rangle)$

Higman thm ($0 < i < k$ and $w_i = \mathtt{inr}\, g$) 1/2

- let $T = \mathtt{tree}(X_0, \ldots, X_{k-1})$
- we have $t = \langle x_i | t_1, \ldots, t_i \rangle$ with $0 < i < k$
- $X'_j = X_j$ and $R'_j = R_j$ for $j \notin \{i-1, i\}$
- $X'_i = X_i$ and $R'_i = R_i \uparrow x_i$ is $\mathtt{af\_t}$ for $w'_i = g(x_i)$ simpler than w_i
- $X'_{i-1} = X_{i-1} + \sum_{p=0}^{i-1} X_i \times T$ and $R'_{i-1} = R_{i-1} + \sum_{p=0}^{i-1} R_i \times (<^h_R \uparrow t_p)$
  - R'_{i-1} is $\mathtt{af\_t}$ by Ramsey, obtain w'_{i-1}
  - $[\ldots, w'_{i-1}, w'_i, \ldots]$ easier than $[\ldots, w_{i-1}, w_i, \ldots]$
  - we deduce $\mathtt{af\_t}(<^h_{R'})$ by induction
- we show $\mathtt{af\_t}(<^h_{R'})$ implies $\mathtt{af\_t}(<^h_R \uparrow \langle x_i | t_1, \ldots, t_i \rangle)$ (not easy)

Higman thm ($0 < i < k$ and $w_i = \mathtt{inr}\, g$) 2/2

- with $X'_{i-1} = X_{i-1} + \sum_{p=0}^{i-1} X_i \times T$, define an *evaluation map*
- $\mathtt{ev} : \mathtt{tree}(X_0, \ldots, X'_{i-1}, X_i, \ldots) \to \mathtt{tree}(X_0, \ldots, X_{k-1})$
  - $\mathtt{ev}(\langle y_j | t_1, \ldots, t_j \rangle) = \langle y_j | \mathtt{ev}\, t_1, \ldots, \mathtt{ev}\, t_j \rangle$ for $j \neq i-1$
  - $\mathtt{ev}(\langle y_{i-1} | t_1, \ldots, t_{i-1} \rangle) = \langle y_{i-1} | \mathtt{ev}\, t_1, \ldots, \mathtt{ev}\, t_{i-1} \rangle$
  - $\mathtt{ev}(\langle (p, y_i, t) | t_1, \ldots, t_{i-1} \rangle) = \langle y_i | \mathtt{insert}\ t\ p\ [\mathtt{ev}\, t_1, \ldots, \mathtt{ev}\, t_{i-1}] \rangle$
- `ev` (is surjective and) has finite inverse images
  - allows the use of `bar_t` induction and the FAN theorem
- use `ev` to show $\mathtt{af\_t}(<^h_{R'})$ implies $\mathtt{af\_t}(<^h_R \uparrow \langle x_i | t_1, \ldots, t_i \rangle)$
  - combinatorial principle: $\forall x \in X, P_x \vee Q_x \Rightarrow \forall x\, P_x \vee \exists x\, Q_x$
  - and more complex version (see later)
  - very technical part of Coq proof (largely absent from paper)

Higman thm ($0 < i < k$ and $w_i = \texttt{inl}\,\star$)

- $T = \texttt{tree}(X_0, \ldots, X_{k-1})$ and $t = \langle x_i | t_1, \ldots, t_i \rangle$ with $0 < i < k$
- $w_i = \texttt{inl}\,\star$ thus we have R_i is full on X_i
- X'_j and R'_j for $j \neq i$ as in case $w_i = \texttt{inr}\,g$
- $\boxed{X'_i = \emptyset}$ with any R'_i (only one exists) is $\texttt{af\_t}$
- $\boxed{\text{ensure case where } X'_i = \emptyset \text{ is simpler than } R_i \text{ is full on } X_i}$
  - $w'_i = \texttt{None}$ is simpler than $w_i = \texttt{Some}(\texttt{inl}\,\star)$
  - we deduce $\texttt{af\_t}(<^h_{R'})$ by induction
- we show $\texttt{af\_t}(<^h_{R'})$ implies $\texttt{af\_t}(<^h_R \uparrow \langle x_i | t_1, \ldots, t_i \rangle)$
  - similar to the case $w_i = \texttt{inr}\,g$
  - but not easy to factorize the Coq duplicated code

Higman thm ($i < k$ and $w_i = \texttt{None}$)

- $T = \texttt{tree}(X_0, \ldots, X_{k-1})$ and $t = \langle x_i | t_1, \ldots, t_i \rangle$ with $0 < i < k$
- but because $w_i = \texttt{None}$, we have $X_i = \emptyset$
- this contradicts $x_i \in X_i$; an easy case indeed

The induction principle of Veldman's proof

- lexicographic product (corresponds to nested induction)
- not grounded on full relations (witnessed by the empty `wft`)
- but grounded on empty types
- empty types are sub-cases of full relations

Remarks on the implentation of that proof

- Implements "well" for e.g. at most unary/binary trees

```
Theorem higman_abt_t : forall Z T, @af_t Z T
                     -> forall Y S, @af_t Y S
                     -> forall X R, @af_t X R
                     -> af_t (embed_abtree R S T).
Proof. do 3 (induction 1 using af_t_dep_rect); .... End.
```

- Thought it requires a dependent induction principle for `af_t`
- But that does not work for parameterized breadth k
  - `tree`$(X_n)_{n<k}$ VERY cumbersome to work with
  - $[\ldots, w'_{i-1}, w'_i, \ldots]$ "easier" than $[\ldots, w_{i-1}, w_i, \ldots]$
  - but the $w'_{i-1} :$ `wft` X'_{i-1} and $w_{i-1} :$ `wft` X_{i-1} not same type !!

A dependent induction principle for af_t

```
Section af_t_dep_rect.

  Variable (P : forall X, relation X -> Type).

  Hypothesis HP0 : P ER.
  Hypothesis HP1 : forall X R, full R -> P ER -> @P X R.
  Hypothesis HP2 : forall X R, (forall x, af_t (R rlift x))
                               -> (forall x, P (R rlift x))
                               -> @P X R.

  Theorem af_t_dep_rect : forall X R, af_t R -> @P X R.

End af_t_dep_rect.
```

Finite Trees in Coq

- Dependent types: nice way to represent complex data structures
- But too much dependency can make your life miserable
- Hence we represent $\texttt{tree}(X_n)_{n\in\mathbb{N}}$ by:

$$\left\{t : \texttt{tree}\left(\sum X_n\right) \;\middle|\; \texttt{tree\_fall}\ (x\ ll \mapsto \texttt{arity}\, x = \texttt{length}\, ll)\ t\right\}$$

- $\texttt{tree}\, X$ is the lfp of $\texttt{tree}\, X = X \times \texttt{list}(\texttt{tree}\, X)$:

```
Variable X : Type.
Inductive tree : Type := in_tree : X -> list tree -> tree.
```

- Can freely use the `List` library to deal with the forest of sons
- Nested definition does not generate a good elimination scheme

Finite Trees in Coq, a nice recursor

```
Variable P : tree -> Type.
Hypothesis f : forall a ll, (forall x, In x ll -> P x)
                            -> P (in_tree a ll).

Definition tree_rect t : P t := ... (* use Fix from Wf *)

Hypothesis f_ext : ...

Fact tree_rect_fix a ll :
  tree_rect (in_tree a ll) = f a ll (fun t _ => tree_rect t)
```

Finite trees in Coq, example definitions

```
Implicit Types (P : X -> list tree -> Prop)
               (Q : nat -> X -> Prop).

Definition tree_fall P : tree -> Prop.

Fact tree_fall_fix P x ll :
     tree_fall P (in_tree x ll)
 <-> P x ll
 /\ forall t, In t ll -> tree_fall P t.

Let btree k := tree_fall (fun x ll => length ll < k).
Let wfptree Q := tree_fall (fun x ll => Q (length ll) x).
```

Higman Embedding in Coq

```
Variables (X : Type) (R : nat -> X -> X -> Prop).

Inductive emb_tree_higman : tree X -> tree X -> Prop :=
  | in_emb_tree_higman_0 :  forall s t x ll,
                               In t ll
                            -> s <eh t
                            -> s <eh in_tree x ll
  | in_emb_tree_higman_1 :  forall x y ll mm,
                               R (length ll) x y
                            -> Forall2 emb_tree_higman ll mm
                            -> in_tree x ll <eh in_tree y mm
where "x <eh y" := (emb_tree_higman x y).
```

Higman Embedding in Coq, elimination Scheme

```
Variable S : tree X -> tree X -> Prop.
Infix "<<" := S (at level 70).

Hypothesis S_sub0 : forall s t x ll,
                      In t ll -> s <eh t
                  -> s << t  -> s << in_tree x ll.

Hypothesis S_sub1 : forall x y ll mm, R (length ll) x y
                   -> Forall2 emb_tree_higman ll mm
                   -> Forall2 S                ll mm
                   -> in_tree x ll << in_tree y mm.

Theorem emb_tree_higman_ind t1 t2 : t1 <eh t2 -> t1 << t2.
```

Almost Full predicate

```
Definition af_t R := { t : wft X | af_securedby R t }.

Inductive af_type : (X -> X -> Prop) -> Type :=
  | in_af_type0 : forall R, full R -> af_type R
  | in_af_type1 : forall R, (forall a, af_type (R rlift a))
                            -> af_type R.

Definition af_t_other R :=
   { t | forall f, good R (pfx_rev f (wft_ht t f)) }.

Thm af_t_eq : af_t R <-> af_type R <-> af_t_other R.
```

Inductive Bar predicates

```
Implicit Types (P : list X -> Prop) (R : X -> X -> Prop).

Inductive bar_t P : list X -> Type :=
  | in_bar_t0 : forall ll, P ll -> bar_t P ll
  | in_bar_t1 : forall ll, (forall a, bar_t P (a::ll))
                           -> bar_t P ll.

Inductive good R : list X -> Prop :=
    | in_good_0 : forall ll a b, In b ll
                                 -> R b a   -> good R (a::ll)
    | in_good_1 : forall ll a, good R ll -> good R (a::ll).

Thm af_t_bar_t R : af_t R <-> bar_t (good R) nil.
```

A universe tailored for Higman theorem

- Given a type $(X_i)_{i<k}$, a universe U is a post fixpoint of:

$$U = \{\star\} + \sum X_i + U + \mathbb{N} \times U \times \texttt{tree}\, U$$

- Then $X'_{i-1} = X_{i-1} + \sum_{p=0}^{i-1} X_i \times \texttt{tree}(X_0, \ldots, X_{k-1})$ can be viewed as a sub-type of U (in Veldman 2004, $U = \mathbb{N}$)

```
Variable X : Type.
Inductive htree_fix :=
  | in_htf_u : htree_fix                    (* undefined *)
  | in_htf_0 : X -> htree_fix               (* X embeds in U *)
  | in_htf_1 : htree_fix -> htree_fix (* U embed in U *)
  | in_htf_2 : nat -> htree_fix
            -> tree htree_fix -> htree_fix.
```

Higman theorem, the recursive statement

```
Definition owft X := option (wft X).
Variables (X : Type) (k : nat).
Notation U := (htree_fix X).

Theorem higman_htree_rec (s : nat -> owft U) :
   forall P : nat -> U -> Prop,
   (forall n, ~ P n (@in_htf_u X))
-> (forall n x, { P n x } + { ~ P n x })
-> (forall n, k < n -> P n = fun _ => False)
-> forall R,
   (forall n, n <= k -> afs_owft_sec (s n) (P n) (R n))
-> afs_t (wfptree P) (emb_tree_higman R).
```

What about the logical version

- $\texttt{af} : \texttt{rel}_2\, X \to \boxed{\texttt{Prop}}$ instead of $\texttt{af\_t} : \texttt{rel}_2\, X \to \texttt{Type}$
- Unlike provable/has a proof, $\texttt{af}\, R$ is NOT $\texttt{inhabited}(\texttt{af\_t}\, R)$
  - cannot use empty `wft` to decide when R is full or not !!
- To get $\texttt{af}\, R \Rightarrow \exists t, \texttt{af\_securedby}(t, R)$, you either need:
  - `FunctionalChoice_on`: $(\forall x \exists y, x\ R\ y) \Rightarrow \exists f \forall x, x\ R\ f(x)$
  - or *Brouwer's thesis* (Veldman 2004)
- How to replace lexicographic induction of `wft` sequences?
  - first idea: encode lex. product at `Prop` level instead of `Type`
  - new idea: use well-founded upto relations

wf. upto rels. are stable under lex. products

Well-founded upto relations

```
Variable (X Y : Type).
Implicit Type (f : X -> Y) (R : relation X)
              (P : X -> Prop) (Q : Y -> Prop).

Definition well_founded R :=
  forall P, (forall a, (forall b, R b a -> P b)
                    -> P a)
         -> forall a, P a.

Definition well_founded_upto f R :=
  forall Q, (forall a, (forall b, R b a -> Q (f b))
                    -> Q (f a))
         -> forall a, Q (f a).
```

Almost full rels and Wf upto 1/2

```
Inductive afw : Set := af_empty | af_full | af_rlift.
Let lt_afw : afw -> afw -> Prop. (* empty < full < rlift *)

Definition af_subrel := (af_w * ((X -> Prop) * relation X)).

Definition afsr_correct (c : af_subrel) :=
  match c with
    | (af_empty,(P,_)) => forall x, ~ P x
    | (af_full ,(P,R)) => forall x y, P x -> P y -> R x y
    | (af_rlift,(P,R)) => afs P R
  end.
```

Almost full rels and Wf upto 2/2

```
Definition lt_afsr (c1 c2 : af_subrel) :=
    match c1           , c2 with
      | (w1,(P1,R1)) , (w2,(P2,R2))
    => lt_afw w1 w2
    \/ w1 = w2
    /\ w1 = af_rlift
    /\ P1 = P2
    /\ exists p, P1 p
    /\ R1 = (R2 rlift p)
    end.  (* this relation has reflexive elements *)

Theorem lt_afsr_upto_wf :
  well_founded_upto (@snd _ _) afsr_correct lt_afsr.
```

What about Kruskal's tree theorem ?

- Shares the same structure as Higman theorem
- There are twice as many cases
- The proof uses both Higman lemma and Higman theorem
- The lexicographic product is a bit different: *more facile*
- The universe is not the same:

$$U = \{\star\} + X + U + U \times \texttt{list}(\texttt{list}(\texttt{tree}\, U))$$

- Replace `insert` with the more general `intercalate`

$$\texttt{intercalate}\ [a_1, \ldots, a_n]\ [l_0, \ldots, l_n] = l_0 +\!\!+ a_1 :: \cdots +\!\!+ a_n :: l_n$$

- `emb_tree_upto`: inbetween the product and the homeomorphic

Tree Embedding upto k

- $\texttt{tree}(X_n)_{n\in\mathbb{N}} = T$ where T is lfp of $T = \sum_{n=0}^{\infty} X_n \times T^n$
- $k \in \mathbb{N}$ and an arity-indexed relation $R : \forall n \in \mathbb{N}, \texttt{rel}_2\,(X_n)$
- one X_n for each arity, but $\boxed{X_k = X_n}$ as soon as $n \geq k$

$$\frac{s <^u_{k,R} t_i}{s <^u_{k,R} \langle x_n | t_1, \ldots, t_n \rangle}$$

$$\frac{n < k \quad x_n\ R_n\ y_n \quad s_1 <^u_{k,R} t_1, \ldots, s_n <^u_{k,R} t_n}{\langle x_n | s_1, \ldots, s_n \rangle <^u_{k,R} \langle y_n | t_1, \ldots, t_n \rangle}$$

$$\frac{k \leq i \quad x_i\ R_k\ x_j \quad [s_1, \ldots, s_i]\ (\texttt{subword} <^u_{k,R})\ [t_1, \ldots, t_j]}{\langle x_i | s_1, \ldots, s_i \rangle <^u_{k,R} \langle x_j | t_1, \ldots, t_j \rangle}$$

Coq code for emb_tree_upto

```
Variables (k : nat) (R : nat -> X -> X -> Prop).
Inductive emb_tree_upto : tree X -> tree X -> Prop :=
  | in_embut_0 : forall s t x ll, In t ll -> s <eu t
                            -> s <eu in_tree x ll
  | in_embut_1 : forall x y ll mm, length ll < k
                            -> R (length ll) x y
                            -> Forall2 emb_tree_upto ll mm
                            -> in_tree x ll <eu in_tree y mm
  | in_embut_2 : forall x y ll mm, k <= length ll
                            -> R k x y
                            -> subword emb_tree_upto ll mm
                            -> in_tree x ll <eu in_tree y mm
where "x <eu y" := (emb_tree_upto x y).
```

Kruskal's Tree Theorem, the recursive statement

```
Variables (X : Type).
Notation U := (ktree_fix X).

Theorem kruskal_ktree_rec (s : nt_stump U) :
   forall k, k = nts_char s
-> forall (P : nat -> U -> Prop),
          (forall n, ~ P n in_ktf_u)
       -> (forall n x, { P n x } + { ~ P n x })
       -> (forall n, k <= n -> P k = P n)
-> forall (R : nat -> relation U),
   (forall n, n <= k
           -> afs_owft_sec (nts_seq s n) (P n) (R n))
-> afs_t (wfptree P) (emb_tree_upto k R).
```