1.3: Inductive Definitions and Recursion

In this section, we will introduce and study ordered trees of arbitrary (finite) arity, whose nodes are labeled by elements of some set.

In later chapters, we will define regular expressions (in Chapter 3), parse trees (in Chapter 4) and programs (in Chapter 5) as restrictions of the trees we consider here.

1.3: Inductive Definitions and Recursion

In this section, we will introduce and study ordered trees of arbitrary (finite) arity, whose nodes are labeled by elements of some set.

In later chapters, we will define regular expressions (in Chapter 3), parse trees (in Chapter 4) and programs (in Chapter 5) as restrictions of the trees we consider here.

The definition of the set of all trees is our first example of an inductive definition—a definition in which we collect together all of the values that can be constructed using a set of rules.

1.3: Inductive Definitions and Recursion

In this section, we will introduce and study ordered trees of arbitrary (finite) arity, whose nodes are labeled by elements of some set.

In later chapters, we will define regular expressions (in Chapter 3), parse trees (in Chapter 4) and programs (in Chapter 5) as restrictions of the trees we consider here.

The definition of the set of all trees is our first example of an inductive definition—a definition in which we collect together all of the values that can be constructed using a set of rules.

In this section, we will also see how to define functions by recursion.

Inductive Definition of Trees

Suppose X is a set. The set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Inductive Definition of Trees

Suppose X is a set. The set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Ignoring the adjective "least" for the moment, some example elements of $\textbf{Tree}\,\mathbb{N}$ are:

Inductive Definition of Trees

Suppose X is a set. The set $\mathbf{Tree}\,X$ of X-*trees* is the least set such that, for all $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\,X)$, $(x, trs) \in \mathbf{Tree}\,X$.

Ignoring the adjective "least" for the moment, some example elements of $\mathbf{Tree}\,\mathbb{N}$ are:

- Since $3 \in \mathbb{N}$ and $[\,] \in \mathbf{List}(\mathbf{Tree}\,\mathbb{N})$, we have that $(3, [\,]) \in \mathbf{Tree}\,\mathbb{N}$.

Inductive Definition of Trees

Suppose X is a set. The set $\textbf{Tree}\, X$ of X-*trees* is the least set such that, for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\, X)$, $(x, trs) \in \textbf{Tree}\, X$.

Ignoring the adjective "least" for the moment, some example elements of $\textbf{Tree}\, \mathbb{N}$ are:

- Since $3 \in \mathbb{N}$ and $[\,] \in \textbf{List}(\textbf{Tree}\, \mathbb{N})$, we have that $(3, [\,]) \in \textbf{Tree}\, \mathbb{N}$.
- For similar reasons, $(1, [\,])$ and $(6, [\,])$ are in $\textbf{Tree}\, \mathbb{N}$.

Inductive Definition of Trees

Suppose X is a set. The set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

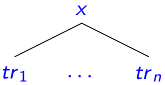
Ignoring the adjective "least" for the moment, some example elements of $\textbf{Tree}\,\mathbb{N}$ are:

- Since $3 \in \mathbb{N}$ and $[\,] \in \textbf{List}(\textbf{Tree}\,\mathbb{N})$, we have that $(3, [\,]) \in \textbf{Tree}\,\mathbb{N}$.
- For similar reasons, $(1, [\,])$ and $(6, [\,])$ are in $\textbf{Tree}\,\mathbb{N}$.
- Because $4 \in \mathbb{N}$, and $[(3, [\,]), (1, [\,]), (6, [\,])] \in \textbf{List}(\textbf{Tree}\,\mathbb{N})$, we have that $(4, [(3, [\,]), (1, [\,]), (6, [\,])]) \in \textbf{Tree}\,\mathbb{N}$.
- And we can continue like this forever.

Drawing Trees

Trees are often easier to comprehend if they are drawn.

We draw the X-tree $(x, [tr_1, \ldots, tr_n])$ as

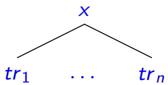


(just writing x when $n = 0$).

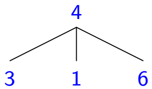
Drawing Trees

Trees are often easier to comprehend if they are drawn.

We draw the X-tree $(x, [tr_1, \ldots, tr_n])$ as



(just writing x when $n = 0$). For example,

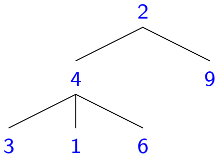


is the drawing of the $\mathbb{N}$-tree

$$(4, [(3, [\,]), (1, [\,]), (6, [\,])]).$$

Drawing Trees (Cont.)

And

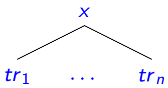


is the $\mathbb{N}$-tree

$$(2, [(4, [(3, [\,]), (1, [\,]), (6, [\,])]), (9, [\,])]).$$

Tree Terminology and Notation

Consider the tree



again.

The *root label* of this tree is x, and tr_1 is the tree's *first child*, etc. We write **rootLabel** tr for the root label of tr.

Tree Terminology and Notation (Cont.)

We often write a tree $(x, [tr_1, \ldots, tr_n])$ in a more compact, linear syntax:

- $x(tr_1, \ldots, tr_n)$, when $n \geq 1$, and
- x, when $n = 0$.

Thus

$$(2, [(4, [(3, [\,]), (1, [\,]), (6, [\,])]), (9, [\,])]).$$

can be written as

$$2(4(3, 1, 6), 9).$$

Understanding the Definition

Consider the definition of $\textbf{Tree}\,X$ again: the set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Understanding the Definition

Consider the definition of $\mathbf{Tree}\,X$ again: the set $\mathbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\,X)$, $(x, trs) \in \mathbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\mathbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \mathbf{List}\,U$, $(x, trs) \in U$.

Understanding the Definition

Consider the definition of $\textbf{Tree}\,X$ again: the set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\textbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \textbf{List}\,U$, $(x, trs) \in U$.

Thus the definition says that $\textbf{Tree}\,X$ is the least X-closed set, where we've yet to say just what "least" means.

Understanding the Definition

Consider the definition of $\textbf{Tree}\,X$ again: the set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\textbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \textbf{List}\,U$, $(x, trs) \in U$.

Thus the definition says that $\textbf{Tree}\,X$ is the least X-closed set, where we've yet to say just what "least" means.

From set theory, we have that, for all sets X, there is an X-closed set U.

Understanding the Definition

Consider the definition of $\mathbf{Tree}\,X$ again: the set $\mathbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\,X)$, $(x, trs) \in \mathbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\mathbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \mathbf{List}\,U$, $(x, trs) \in U$.

Thus the definition says that $\mathbf{Tree}\,X$ is the least X-closed set, where we've yet to say just what "least" means.

From set theory, we have that, for all sets X, there is an X-closed set U. (If X is a set, we can prove that there is a set U such that $X \subseteq U$, $U \times U \subseteq U$ and $\mathbf{List}\,U \subseteq U$.

Understanding the Definition

Consider the definition of $\mathbf{Tree}\,X$ again: the set $\mathbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\,X)$, $(x, trs) \in \mathbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\mathbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \mathbf{List}\,U$, $(x, trs) \in U$.

Thus the definition says that $\mathbf{Tree}\,X$ is the least X-closed set, where we've yet to say just what "least" means.

From set theory, we have that, for all sets X, there is an X-closed set U. (If X is a set, we can prove that there is a set U such that $X \subseteq U$, $U \times U \subseteq U$ and $\mathbf{List}\,U \subseteq U$. Thus if $x \in X$ and $trs \in \mathbf{List}\,U$, then $x \in U$ and $trs \in U$, so that $(x, trs) \in U$.)

Understanding the Definition

Consider the definition of $\textbf{Tree}\,X$ again: the set $\textbf{Tree}\,X$ of X-*trees* is the least set such that, (†) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$, $(x, trs) \in \textbf{Tree}\,X$.

Let's call a set U X-*closed* iff it satisfies property (†), where we have replaced $\textbf{Tree}\,X$ by U: for all $x \in X$ and $trs \in \textbf{List}\,U$, $(x, trs) \in U$.

Thus the definition says that $\textbf{Tree}\,X$ is the least X-closed set, where we've yet to say just what "least" means.

From set theory, we have that, for all sets X, there is an X-closed set U. (If X is a set, we can prove that there is a set U such that $X \subseteq U$, $U \times U \subseteq U$ and $\textbf{List}\,U \subseteq U$. Thus if $x \in X$ and $trs \in \textbf{List}\,U$, then $x \in U$ and $trs \in U$, so that $(x, trs) \in U$.)

If U is $\mathbb{Z}$-closed, then it will also be $\mathbb{N}$-closed. But if $\textbf{Tree}\,\mathbb{N}$ was U, it would have elements like $(-5, [\,])$, which are not wanted.

Understanding the Definition (Cont.)

To keep **Tree** X from having junk, we say that **Tree** X is the set U such that:

- U is X-closed, and

Understanding the Definition (Cont.)

To keep **Tree** X from having junk, we say that **Tree** X is the set U such that:

- U is X-closed, and
- for all X-closed sets V, $U \subseteq V$.

Understanding the Definition (Cont.)

To keep **Tree** X from having junk, we say that **Tree** X is the set U such that:

- U is X-closed, and
- for all X-closed sets V, $U \subseteq V$.

This is what we mean by saying that **Tree** X is the *least* X-closed set.

Understanding the Definition (Cont.)

To keep **Tree** X from having junk, we say that **Tree** X is the set U such that:

- U is X-closed, and
- for all X-closed sets V, $U \subseteq V$.

This is what we mean by saying that **Tree** X is the *least* X-closed set.

But how do we know that such a set U exists? (If U and U' are both X-closed sets that are subsets of all X-closed sets, then $U \subseteq U' \subseteq U$, and so $U = U'$. Thus there is at most one U with the above property.)

Justifying the Definition

We have that:

- There is an X-closed set V.

Justifying the Definition

We have that:

- There is an X-closed set V.
- If $\mathcal{W}$ is a nonempty set of X-closed sets, then $\bigcap \mathcal{W}$ is also an X-closed set.

Justifying the Definition

We have that:

- There is an X-closed set V.
- If $\mathcal{W}$ is a nonempty set of X-closed sets, then $\bigcap \mathcal{W}$ is also an X-closed set.

  (Because $\mathcal{W}$ is nonempty, $\bigcap \mathcal{W}$ is well-defined. To see that $\bigcap \mathcal{W}$ is X-closed, suppose $x \in X$ and $trs \in \textbf{List} \bigcap \mathcal{W}$. We must show that $(x, trs) \in \bigcap \mathcal{W}$, i.e.,

Justifying the Definition

We have that:

- There is an X-closed set V.
- If $\mathcal{W}$ is a nonempty set of X-closed sets, then $\bigcap \mathcal{W}$ is also an X-closed set.

  (Because $\mathcal{W}$ is nonempty, $\bigcap \mathcal{W}$ is well-defined. To see that $\bigcap \mathcal{W}$ is X-closed, suppose $x \in X$ and $trs \in \mathbf{List} \bigcap \mathcal{W}$. We must show that $(x, trs) \in \bigcap \mathcal{W}$, i.e., that $(x, trs) \in Z$, for all $Z \in \mathcal{W}$. Suppose $Z \in \mathcal{W}$. We must show that $(x, trs) \in Z$.

Justifying the Definition

We have that:

- There is an X-closed set V.
- If $\mathcal{W}$ is a nonempty set of X-closed sets, then $\bigcap \mathcal{W}$ is also an X-closed set.

  (Because $\mathcal{W}$ is nonempty, $\bigcap \mathcal{W}$ is well-defined. To see that $\bigcap \mathcal{W}$ is X-closed, suppose $x \in X$ and $trs \in \textbf{List} \bigcap \mathcal{W}$. We must show that $(x, trs) \in \bigcap \mathcal{W}$, i.e., that $(x, trs) \in Z$, for all $Z \in \mathcal{W}$. Suppose $Z \in \mathcal{W}$. We must show that $(x, trs) \in Z$. Because $Z \in \mathcal{W}$, we have that $\bigcap \mathcal{W} \subseteq Z$, so that $\textbf{List} \bigcap \mathcal{W} \subseteq \textbf{List}\, Z$. Thus, since $trs \in \textbf{List} \bigcap \mathcal{W}$, it follows that $trs \in \textbf{List}\, Z$.

Justifying the Definition

We have that:

- There is an X-closed set V.
- If $\mathcal{W}$ is a nonempty set of X-closed sets, then $\bigcap \mathcal{W}$ is also an X-closed set.

  (Because $\mathcal{W}$ is nonempty, $\bigcap \mathcal{W}$ is well-defined. To see that $\bigcap \mathcal{W}$ is X-closed, suppose $x \in X$ and $trs \in \mathbf{List} \bigcap \mathcal{W}$. We must show that $(x, trs) \in \bigcap \mathcal{W}$, i.e., that $(x, trs) \in Z$, for all $Z \in \mathcal{W}$. Suppose $Z \in \mathcal{W}$. We must show that $(x, trs) \in Z$. Because $Z \in \mathcal{W}$, we have that $\bigcap \mathcal{W} \subseteq Z$, so that $\mathbf{List} \bigcap \mathcal{W} \subseteq \mathbf{List}\, Z$. Thus, since $trs \in \mathbf{List} \bigcap \mathcal{W}$, it follows that $trs \in \mathbf{List}\, Z$. But Z is X-closed, and thus $(x, trs) \in Z$, as required.)

Justifying the Definition (Cont.)

Let $\mathcal{W}$ be the set of all subsets of V that are X-closed. Thus $\mathcal{W}$ is a nonempty set of X-closed sets, since $V \in \mathcal{W}$.

Justifying the Definition (Cont.)

Let $\mathcal{W}$ be the set of all subsets of V that are X-closed. Thus $\mathcal{W}$ is a nonempty set of X-closed sets, since $V \in \mathcal{W}$.

Let $U = \bigcap \mathcal{W}$. Then U is X-closed, and is a subset of all other X-closed sets (if T is an X-closed set, then $T \cap V \in \mathcal{W}$), i.e., it is the least X-closed set.

Principle of Induction on Trees

Because trees are defined via an inductive definition, we get an induction principle for trees almost for free:

Theorem 1.3.3 (Principle of Induction on Trees)
Suppose X *is a set and* $P(tr)$ *is a property of an element* $tr \in \textbf{Tree}\, X$.
If

> *for all* $x \in X$ *and* $trs \in \textbf{List}(\textbf{Tree}\, X)$,
> *if*
> *then* $P((x, trs))$,

then

$$\text{for all } tr \in \textbf{Tree}\, X,\ P(tr).$$

Principle of Induction on Trees

Because trees are defined via an inductive definition, we get an induction principle for trees almost for free:

Theorem 1.3.3 (Principle of Induction on Trees)
Suppose X *is a set and* $P(tr)$ *is a property of an element* $tr \in \mathbf{Tree}\, X$.
If

$$\begin{aligned}&\textit{for all } x \in X \textit{ and } trs \in \mathbf{List}(\mathbf{Tree}\, X),\\ &\textit{if } (\dagger) \textit{ for all } i \in [1 : |trs|],\ P(trs\ i),\\ &\textit{then } P((x, trs)),\end{aligned}$$

then

$$\textit{for all } tr \in \mathbf{Tree}\, X,\ P(tr).$$

We refer to (†) as the inductive hypothesis.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \textbf{Tree}\,X$, and

(‡) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$,
if for all $i \in [1 : |trs|]$, $P(trs\,i)$,
then $P((x, trs))$.

We must show that

$$\text{for all } tr \in \textbf{Tree}\,X,\ P(tr).$$

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \textbf{Tree}\,X$, and

(‡) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$,
if for all $i \in [1 : |trs|]$, $P(trs\,i)$,
then $P((x, trs))$.

We must show that

$$\text{for all } tr \in \textbf{Tree}\,X,\ P(tr).$$

Let $U = \{\, tr \in \textbf{Tree}\,X \mid P(tr) \,\}$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \textbf{Tree}\,X$, and

(‡) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$,
if for all $i \in [1 : |trs|]$, $P(trs\,i)$,
then $P((x, trs))$.

We must show that

$$\text{for all } tr \in \textbf{Tree}\,X,\ P(tr).$$

Let $U = \{\, tr \in \textbf{Tree}\,X \mid P(tr)\,\}$. We will show that U is X-closed.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \mathbf{Tree}\, X$, and

(‡) for all $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$,
if for all $i \in [1 : |trs|]$, $P(trs\, i)$,
then $P((x, trs))$.

We must show that

$$\text{for all } tr \in \mathbf{Tree}\, X,\ P(tr).$$

Let $U = \{\, tr \in \mathbf{Tree}\, X \mid P(tr)\,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \mathbf{List}\, U$. We must show that $(x, trs) \in U$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \mathbf{Tree}\, X$, and

$$(\ddagger)\ \text{for all } x \in X \text{ and } trs \in \mathbf{List}(\mathbf{Tree}\, X),\\ \text{if for all } i \in [1 : |trs|],\ P(trs\, i),\\ \text{then } P((x, trs)).$$

We must show that

$$\text{for all } tr \in \mathbf{Tree}\, X,\ P(tr).$$

Let $U = \{\, tr \in \mathbf{Tree}\, X \mid P(tr) \,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \mathbf{List}\, U$. We must show that $(x, trs) \in U$. It will suffice to show that $P((x, trs))$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \textbf{Tree}\,X$, and

(‡) for all $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$,
if for all $i \in [1 : |trs|]$, $P(trs\,i)$,
then $P((x, trs))$.

We must show that

$$\text{for all } tr \in \textbf{Tree}\,X,\ P(tr).$$

Let $U = \{\, tr \in \textbf{Tree}\,X \mid P(tr) \,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \textbf{List}\,U$. We must show that $(x, trs) \in U$. It will suffice to show that $P((x, trs))$. By (‡), it will suffice to show that, for all $i \in [1 : |trs|]$, $P(trs\,i)$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \mathbf{Tree}\, X$, and

$$(\ddagger)\ \text{for all } x \in X \text{ and } trs \in \mathbf{List}(\mathbf{Tree}\, X), \text{ if for all } i \in [1 : |trs|],\ P(trs\, i), \text{ then } P((x, trs)).$$

We must show that

$$\text{for all } tr \in \mathbf{Tree}\, X,\ P(tr).$$

Let $U = \{\, tr \in \mathbf{Tree}\, X \mid P(tr) \,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \mathbf{List}\, U$. We must show that $(x, trs) \in U$. It will suffice to show that $P((x, trs))$. By ($\ddagger$), it will suffice to show that, for all $i \in [1 : |trs|]$, $P(trs\, i)$. Suppose $i \in [1 : |trs|]$. We must show that $P(trs\, i)$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \textbf{Tree}\,X$, and

$$(\ddagger)\ \text{for all } x \in X \text{ and } trs \in \textbf{List}(\textbf{Tree}\,X), \\ \text{if for all } i \in [1 : |trs|],\ P(trs\,i), \\ \text{then } P((x, trs)).$$

We must show that

$$\text{for all } tr \in \textbf{Tree}\,X,\ P(tr).$$

Let $U = \{\, tr \in \textbf{Tree}\,X \mid P(tr)\,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \textbf{List}\,U$. We must show that $(x, trs) \in U$. It will suffice to show that $P((x, trs))$. By ($\ddagger$), it will suffice to show that, for all $i \in [1 : |trs|]$, $P(trs\,i)$. Suppose $i \in [1 : |trs|]$. We must show that $P(trs\,i)$. Because $trs \in \textbf{List}\,U$, we have that $trs\,i \in U$.

Proof of Principle of Induction on Trees

Proof. Suppose X is a set, $P(tr)$ is a property of an element $tr \in \mathbf{Tree}\,X$, and

$$
\begin{array}{ll}
(\ddagger) & \text{for all } x \in X \text{ and } trs \in \mathbf{List}(\mathbf{Tree}\,X), \\
& \text{if for all } i \in [1 : |trs|],\ P(trs\,i), \\
& \text{then } P((x, trs)).
\end{array}
$$

We must show that

$$\text{for all } tr \in \mathbf{Tree}\,X,\ P(tr).$$

Let $U = \{\, tr \in \mathbf{Tree}\,X \mid P(tr) \,\}$. We will show that U is X-closed. Suppose $x \in X$ and $trs \in \mathbf{List}\,U$. We must show that $(x, trs) \in U$. It will suffice to show that $P((x, trs))$. By ($\ddagger$), it will suffice to show that, for all $i \in [1 : |trs|]$, $P(trs\,i)$. Suppose $i \in [1 : |trs|]$. We must show that $P(trs\,i)$. Because $trs \in \mathbf{List}\,U$, we have that $trs\,i \in U$. Hence $P(trs\,i)$, as required.

Proof of Principle of Induction on Trees (Cont.)

Proof (cont.). Because U is X-closed, we have that

□

Proof of Principle of Induction on Trees (Cont.)

Proof (cont.). Because U is X-closed, we have that **Tree** $X \subseteq U$, as **Tree** X is the least X-closed set.

□

Proof of Principle of Induction on Trees (Cont.)

Proof (cont.). Because U is X-closed, we have that $\textbf{Tree}\,X \subseteq U$, as $\textbf{Tree}\,X$ is the least X-closed set. Hence, for all $tr \in \textbf{Tree}\,X$, $tr \in U$, so that, □

Proof of Principle of Induction on Trees (Cont.)

Proof (cont.). Because U is X-closed, we have that $\mathbf{Tree}\,X \subseteq U$, as $\mathbf{Tree}\,X$ is the least X-closed set. Hence, for all $tr \in \mathbf{Tree}\,X$, $tr \in U$, so that, for all $tr \in \mathbf{Tree}\,X$, $P(tr)$. □

Proposition 1.3.4
Suppose X is a set. For all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use to prove that, for all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$.

□

Destructing Trees

Proposition 1.3.4

Suppose X is a set. For all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$.

□

Proposition 1.3.4
Suppose X is a set. For all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$. Suppose $x \in X$, $trs \in \mathbf{List}(\mathbf{Tree}\, X)$, and assume the inductive hypothesis: for all $i \in [1 : |trs|]$,

$$trs\, i$$

□

Destructing Trees

Proposition 1.3.4
Suppose X is a set. For all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$. Suppose $x \in X$, $trs \in \textbf{List}(\textbf{Tree}\,X)$, and assume the inductive hypothesis: for all $i \in [1 : |trs|]$, there are $x' \in X$ and $trs' \in \textbf{List}(\textbf{Tree}\,X)$ such that $trs\,i = (x', trs')$.

□

Destructing Trees

Proposition 1.3.4
Suppose X is a set. For all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$. Suppose $x \in X$, $trs \in \mathbf{List}(\mathbf{Tree}\, X)$, and assume the inductive hypothesis: for all $i \in [1 : |trs|]$, there are $x' \in X$ and $trs' \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $trs\, i = (x', trs')$. We must show that
(x, trs)

□

Destructing Trees

Proposition 1.3.4
Suppose X is a set. For all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$.

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \mathbf{Tree}\, X$, there are $x \in X$ and $trs \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $tr = (x, trs)$. Suppose $x \in X$, $trs \in \mathbf{List}(\mathbf{Tree}\, X)$, and assume the inductive hypothesis: for all $i \in [1 : |trs|]$, there are $x' \in X$ and $trs' \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $trs\, i = (x', trs')$. We must show that there are $x' \in X$ and $trs' \in \mathbf{List}(\mathbf{Tree}\, X)$ such that $(x, trs) = (x', trs')$. And this holds, since

□

Proposition 1.3.4
Suppose X *is a set. For all* $tr \in \textbf{Tree}\,X$*, there are* $x \in X$ *and* $trs \in \textbf{List}(\textbf{Tree}\,X)$ *such that* $tr = (x, trs)$*.*

Proof. Suppose X is a set. We use induction on trees to prove that, for all $tr \in \textbf{Tree}\,X$, there are $x \in X$ and $trs \in \textbf{List}(\textbf{Tree}\,X)$ such that $tr = (x, trs)$. Suppose $x \in X$, $trs \in \textbf{List}(\textbf{Tree}\,X)$, and assume the inductive hypothesis: for all $i \in [1 : |trs|]$, there are $x' \in X$ and $trs' \in \textbf{List}(\textbf{Tree}\,X)$ such that $trs\,i = (x', trs')$. We must show that there are $x' \in X$ and $trs' \in \textbf{List}(\textbf{Tree}\,X)$ such that $(x, trs) = (x', trs')$. And this holds, since $x \in X$, $trs \in \textbf{List}(\textbf{Tree}\,X)$ and $(x, trs) = (x, trs)$. $\square$

Predecessor Relation on Trees

Suppose X is a set. Let the predecessor relation $\mathbf{pred}_{\mathbf{Tree}\,X}$ on $\mathbf{Tree}\,X$ be the set of all pairs of X-trees (tr, tr') such that there are $x \in X$ and $trs' \in \mathbf{List}(\mathbf{Tree}\,X)$ such that $tr' = (x, trs')$ and $trs'\,i = tr$ for some $i \in [1 : |trs'|]$.

Thus the predecessors of a tree $(x, [tr_1, \ldots, tr_n])$ are its children tr_1, . . . , tr_n.

Proposition 1.3.5

If X is a set, then $\mathbf{pred}_{\mathbf{Tree}\,X}$ is a well-founded relation on $\mathbf{Tree}\,X$.

Predecessor Relation on Trees

Suppose X is a set. Let the predecessor relation $\mathbf{pred}_{\mathbf{Tree}\,X}$ on $\mathbf{Tree}\,X$ be the set of all pairs of X-trees (tr, tr') such that there are $x \in X$ and $trs' \in \mathbf{List}(\mathbf{Tree}\,X)$ such that $tr' = (x, trs')$ and $trs'\,i = tr$ for some $i \in [1 : |trs'|]$.

Thus the predecessors of a tree $(x, [tr_1, \ldots, tr_n])$ are its children $tr_1, \ldots, tr_n$.

Proposition 1.3.5

If X is a set, then $\mathbf{pred}_{\mathbf{Tree}\,X}$ is a well-founded relation on $\mathbf{Tree}\,X$.

Proof. Suppose X is a set and Y is a nonempty subset of $\mathbf{Tree}\,X$. Mimicking Proposition 1.2.5, we can use the principle of induction on trees to prove that, for all $tr \in \mathbf{Tree}\,X$, if $tr \in Y$, then Y has a $\mathbf{pred}_{\mathbf{Tree}\,X}$-minimal element. Because Y is nonempty, we can conclude that Y has a $\mathbf{pred}_{\mathbf{Tree}\,X}$-minimal element. □

Recursion

Suppose R is a well-founded relation on a set A. We can define a function f from A to a set B by *well-founded recursion on* R.

The details are in the book, but the idea is simple: when f is called with an element $x \in A$, it may call itself recursively on as many of the predecessors of x in R as it wants.

Typically, such a definition will be concrete enough that we can regard it as defining an algorithm as well as a function.

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$,

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$, it can't make any recursive calls.

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$, it can't make any recursive calls.
- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on the predecessor relation $\mathbf{pred}_{\mathbb{N}}$, then when f is called with $n \in \mathbb{N}$, it may call itself recursively on $n - 1$, in the case when $n \geq 1$, and may make no recursive calls, when $n = 0$.

  Thus, if such a definition case-splits according to whether its input is 0 or not, it can be split into two parts:

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$, it can't make any recursive calls.
- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on the predecessor relation $\mathbf{pred}_{\mathbb{N}}$, then when f is called with $n \in \mathbb{N}$, it may call itself recursively on $n - 1$, in the case when $n \geq 1$, and may make no recursive calls, when $n = 0$.

  Thus, if such a definition case-splits according to whether its input is 0 or not, it can be split into two parts:

  - $f\,0 = \cdots$;

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$, it can't make any recursive calls.
- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on the predecessor relation $\mathbf{pred}_{\mathbb{N}}$, then when f is called with $n \in \mathbb{N}$, it may call itself recursively on $n - 1$, in the case when $n \geq 1$, and may make no recursive calls, when $n = 0$.

  Thus, if such a definition case-splits according to whether its input is 0 or not, it can be split into two parts:
  - $f\,0 = \cdots$;
  - for all $n \in \mathbb{N}$, $f(n + 1) = \cdots f\,n \cdots$.

Examples of Well-founded Recursion

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on $<$, then, when f is called with $n \in \mathbb{N}$, it may call itself recursively on any strictly smaller natural numbers. In the case $n = 0$, it can't make any recursive calls.

- If we define $f \in \mathbb{N} \to B$ by well-founded recursion on the predecessor relation $\mathbf{pred}_{\mathbb{N}}$, then when f is called with $n \in \mathbb{N}$, it may call itself recursively on $n - 1$, in the case when $n \geq 1$, and may make no recursive calls, when $n = 0$.

  Thus, if such a definition case-splits according to whether its input is 0 or not, it can be split into two parts:

  - $f\,0 = \cdots$;
  - for all $n \in \mathbb{N}$, $f(n+1) = \cdots f\,n \cdots$.

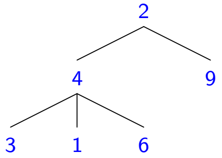
  We say that such a definition is by *recursion on* $\mathbb{N}$.

Examples of Well-founded Recursion (Cont.)

- If we define $f \in \textbf{Tree}\, X \rightarrow B$ by well-founded recursion on the predecessor relation $\textbf{pred}_{\textbf{Tree}\, X}$, then when f is called on an X-tree $(x, [tr_1, \ldots, tr_n])$, it may call itself recursively on any of tr_1, ..., tr_n. When $n = 0$, it may make no recursive calls. We say that such a definition is by *structural recursion*.

Examples of Well-founded Recursion (Cont.)

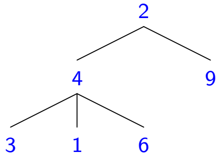
- For example, we may define the *size* of an X-tree $(x, [tr_1, \ldots, tr_n])$ by summing the recursively computed sizes of tr_1, ..., tr_n, and then adding 1. Then, e.g., the size of



is

Examples of Well-founded Recursion (Cont.)

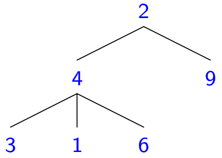
- For example, we may define the *size* of an X-tree $(x, [tr_1, \ldots, tr_n])$ by summing the recursively computed sizes of tr_1, ..., tr_n, and then adding 1. Then, e.g., the size of



is 6.

Examples of Well-founded Recursion (Cont.)

- For example, we may define the *size* of an X-tree $(x, [tr_1, \ldots, tr_n])$ by summing the recursively computed sizes of tr_1, ..., tr_n, and then adding 1. Then, e.g., the size of



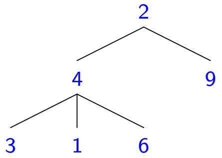
is 6.

This defines a function $\textbf{size} \in \textbf{Tree}\, X \to \mathbb{N}$.

Examples of Well-founded Recursion (Cont.)

- And we may define the *height* of an X-tree $(x, [tr_1, \ldots, tr_n])$ as
  - 0, when $n = 0$, and
  - 1 plus the maximum of the recursively computed heights of tr_1, . . . , tr_n, when $n \geq 1$.

  E.g., the height of

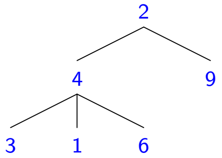


  is

Examples of Well-founded Recursion (Cont.)

- And we may define the *height* of an X-tree $(x, [tr_1, \ldots, tr_n])$ as
  - 0, when $n = 0$, and
  - 1 plus the maximum of the recursively computed heights of tr_1, ..., tr_n, when $n \geq 1$.

  E.g., the height of

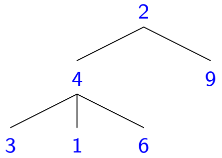


  is 2.

Examples of Well-founded Recursion (Cont.)

- And we may define the *height* of an X-tree $(x, [tr_1, \ldots, tr_n])$ as
  - 0, when $n = 0$, and
  - 1 plus the maximum of the recursively computed heights of tr_1, . . . , tr_n, when $n \geq 1$.

  E.g., the height of



  is 2.

  This defines a function $\textbf{height} \in \textbf{Tree}\, X \to \mathbb{N}$.

Examples of Well-founded Recursion (Cont.)

- Given a set X, we can define a well-founded relation $\mathbf{size}_{\mathbf{Tree}\,X}$ on $\mathbf{Tree}\,X$ by: for all $tr, tr' \in \mathbf{Tree}\,X$, $tr \; \mathbf{size}_{\mathbf{Tree}\,X} \; tr'$ iff $\mathbf{size}\,tr < \mathbf{size}\,tr'$.

  If we define a function $f \in \mathbf{Tree}\,X \to B$ by well-founded recursion on $\mathbf{size}_{\mathbf{Tree}\,X}$, when f is called with an X-tree tr, it may call itself recursively on any X-trees with strictly smaller sizes.

Examples of Well-founded Recursion (Cont.)

- Given a set X, we can define a well-founded relation $\textbf{size}_{\textbf{Tree}\,X}$ on $\textbf{Tree}\,X$ by: for all $tr, tr' \in \textbf{Tree}\,X$, $tr\ \textbf{size}_{\textbf{Tree}\,X}\ tr'$ iff $\textbf{size}\,tr < \textbf{size}\,tr'$.

  If we define a function $f \in \textbf{Tree}\,X \to B$ by well-founded recursion on $\textbf{size}_{\textbf{Tree}\,X}$, when f is called with an X-tree tr, it may call itself recursively on any X-trees with strictly smaller sizes.

- Given a set X, we can define a well-founded relation $\textbf{height}_{\textbf{Tree}\,X}$ on $\textbf{Tree}\,X$ by: for all $tr, tr' \in \textbf{Tree}\,X$, $tr\ \textbf{height}_{\textbf{Tree}\,X}\ tr'$ iff $\textbf{height}\,tr < \textbf{height}\,tr'$.

  If we define a function $f \in \textbf{Tree}\,X \to B$ by well-founded recursion on $\textbf{height}_{\textbf{Tree}\,X}$, when f is called with an X-tree tr, it may call itself recursively on any X-trees with strictly smaller heights.

Examples of Well-founded Recursion (Cont.)

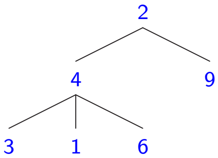
- Given a set X, we can define a well-founded relation $\mathbf{length}_{\mathbf{List}\,X}$ on $\mathbf{List}\,X$ by: for all $xs, ys \in \mathbf{List}\,X$, $xs\ \mathbf{length}_{\mathbf{List}\,X}\ ys$ iff $|xs| < |ys|$.

  If we define a function $f \in \mathbf{List}\,X \to B$ by well-founded recursion on $\mathbf{length}_{\mathbf{List}\,X}$, when f is called with an X-list xs, it may call itself recursively on any X-lists with strictly smaller lengths.

Paths in Trees

We can think of an $\mathbb{N} - \{0\}$-list $[n_1, n_2, \ldots, n_m]$ as a *path* through an X-tree tr: one starts with tr itself, goes to the n_1-th child of tr, selects the n_2-th child of that tree, etc., stopping when the list is exhausted.

Consider the $\mathbb{N}$-tree



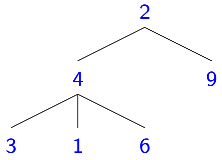
Then:

- $[\,]$ takes us to
- $[1]$ takes us to
- $[1, 3]$ takes us to
- $[1, 4]$ takes us to

Paths in Trees

We can think of an $\mathbb{N} - \{0\}$-list $[n_1, n_2, \ldots, n_m]$ as a *path* through an X-tree tr: one starts with tr itself, goes to the n_1-th child of tr, selects the n_2-th child of that tree, etc., stopping when the list is exhausted.

Consider the $\mathbb{N}$-tree



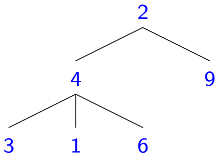
Then:

- $[\,]$ takes us to the whole tree.
- $[1]$ takes us to
- $[1, 3]$ takes us to
- $[1, 4]$ takes us to

Paths in Trees

We can think of an $\mathbb{N} - \{0\}$-list $[n_1, n_2, \ldots, n_m]$ as a *path* through an X-tree tr: one starts with tr itself, goes to the n_1-th child of tr, selects the n_2-th child of that tree, etc., stopping when the list is exhausted.

Consider the $\mathbb{N}$-tree



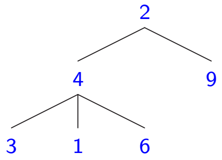
Then:

- $[\,]$ takes us to the whole tree.
- $[1]$ takes us to the tree $4(3, 1, 6)$.
- $[1, 3]$ takes us to
- $[1, 4]$ takes us to

Paths in Trees

We can think of an $\mathbb{N} - \{0\}$-list $[n_1, n_2, \ldots, n_m]$ as a *path* through an X-tree tr: one starts with tr itself, goes to the n_1-th child of tr, selects the n_2-th child of that tree, etc., stopping when the list is exhausted.

Consider the $\mathbb{N}$-tree



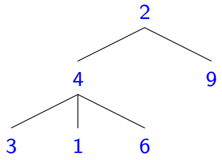
Then:

- $[\,]$ takes us to the whole tree.
- $[1]$ takes us to the tree $4(3, 1, 6)$.
- $[1, 3]$ takes us to the tree 6.
- $[1, 4]$ takes us to

Paths in Trees

We can think of an $\mathbb{N} - \{0\}$-list $[n_1, n_2, \ldots, n_m]$ as a *path* through an X-tree tr: one starts with tr itself, goes to the n_1-th child of tr, selects the n_2-th child of that tree, etc., stopping when the list is exhausted.

Consider the $\mathbb{N}$-tree



Then:

- $[\,]$ takes us to the whole tree.
- $[1]$ takes us to the tree $4(3, 1, 6)$.
- $[1, 3]$ takes us to the tree 6.
- $[1, 4]$ takes us to no tree.

Paths in Trees

We say that $xs \in \mathbf{List}(\mathbb{N} - \{0\})$ *is a valid path for* an X-tree tr iff following the directions of xs takes us from the top of tr to some tree.

Paths in Trees

We say that $xs \in \mathbf{List}(\mathbb{N} - \{0\})$ *is a valid path for* an X-tree tr iff following the directions of xs takes us from the top of tr to some tree.

The trees that can be reached from an X-tree tr by following valid paths are the *subtrees* of tr. And *subtrees* are *leaves* when they have no children.

Paths in Trees

We say that $xs \in \mathbf{List}(\mathbb{N} - \{0\})$ *is a valid path for* an X-tree tr iff following the directions of xs takes us from the top of tr to some tree.

The trees that can be reached from an X-tree tr by following valid paths are the *subtrees* of tr. And *subtrees* are *leaves* when they have no children.

If xs is a valid path for an X-tree tr, and tr' is an X-tree, then we can form a new tree by replacing the subtree at position xs in tr by tr'.

Paths in Trees

We say that $xs \in \mathbf{List}(\mathbb{N} - \{0\})$ *is a valid path for* an X-tree tr iff following the directions of xs takes us from the top of tr to some tree.

The trees that can be reached from an X-tree tr by following valid paths are the *subtrees* of tr. And *subtrees* are *leaves* when they have no children.

If xs is a valid path for an X-tree tr, and tr' is an X-tree, then we can form a new tree by replacing the subtree at position xs in tr by tr'.

For example, replacing the subtree at position $[1, 2]$ in $4(3(2, 1(7)), 6)$ by $3(7, 8)$ gives us $4(3(2, 3(7, 8)), 6)$.