# Computer Supported Modeling and Reasoning

David Basin, Achim D. Brucker, Jan-Georg Smaus, and Burkhart Wolff

April 2005

http://www.infsec.ethz.ch/education/permanent/csmr/

# Higher-Order Logic: Fixpoints and Inductive Definitions

Jan-Georg Smaus and Burkhart Wolff

# Recursion in HOL

Current stage of our course:

- On the basis of conservative extensions, set theory can be built safely.
- But: our mathematical language is still quite small.
  Conservative extensions rule out recursive definitions,
  be it recursive programs or recursive set equations, . . . ).

How can we benefit from set theory to introduce some form of recursion?

# Recursion and General Fixpoints

Naïve Approach: One could axiomatize fixpoint combinator $Y$ as

$$\frac{}{Y = \lambda F.F(Y\,F)}\ \text{fix}$$

This axiom is not a constant definition.

Then we could easily derive

$$\forall F^{\alpha \Rightarrow \alpha}.Y\ F = F\,(Y\ F).$$

- Why are we interested in $Y$?
- What's the problem with such a definition?

# Why Are We Interested in $Y$?

First, why are we interested in recursion (solutions to recursive equations)?

- Recursively defined functions are solutions of such equations (example: $fac$).
- Inductively defined sets are solutions of such equations (example: $Fin\ A$, all finite subsets of $A$).

We are interested in $Y$ because it is the mother of all recursions. With $Y$, recursive axioms can be converted into constant definitions.

# What's the Problem with such an Axiom?

- an axiom would lead to inconsistency. (Consider $\neg$).
- This is not surprising because not all functions have a fixpoint.
- Therefore we only consider special forms of fixpoint combibinators, the two approaches
  - Least fixpoints
  - well-founded orderings.

The following section is devoted to least fixpoints, which happen to be closely related to inductive definitions.

# More Detailed Explanations

# Axiom is not a Definition

The axiom

$$Y = \lambda F.F(Y\,F)$$

is not a constant definition, since $Y$ occurs again on the right-hand side.

$$\forall F^{\alpha \Rightarrow \alpha}. Y\ F = F\ (Y\ F)$$

In words, this says that $Y\ F$ is a fixpoint of $F$.

# Recursive Equation

By a recursive equation, we mean an equation of the form

$$f = e$$

where $f$ occurs in $e$. Such an equation does not satisfy the requirements of a constant definition.

# Converting Recursive Axioms

Any recursive function can be defined by an expression (functional) which is not itself recursive, but instead relies on the recursive equation defining $Y$.

Consider $fac$ or $Fin\ A$ as an example.

# Defining $fac$

In the following explanations, any constants like $1$ or $+$ or **if-then-else** are intended to have their usual meaning.

A fixpoint combinator is a function $Y$ that returns a fixpoint of a function $F$, i.e., $Y$ must fulfill the equation $YF = F(YF)$. Doing $\lambda$-abstraction over $F$ on both sides and $\eta$-conversion (backwards) on the left-hand side, we have

$$Y = \lambda F.F(YF)$$

This is a recursive equation. We will now demonstrate how a definition of a function $fac$ (factorial) using a recursive equation can be transformed to a definition that uses $Y$ instead of using recursion directly.

In a functional programming language we might define

$$fac\ n = (\mathbf{if}\ n = 0\ \mathbf{then}\ 1\ \mathbf{else}\ n * fac\ (n-1)).$$

We now massage this equation a bit. Doing $\lambda$-abstraction on both sides we get

$$\lambda n.\ fac\ n = (\lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n * fac(n-1))$$

which is the $\eta$-conversion of

$$fac = (\lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n * fac(n-1))$$

which in turn is a $\beta$-reduction of

$$fac = (\underline{(\lambda f.\ \lambda n.\ \textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n * f(n-1))}\ fac) \qquad (1)$$

We are looking for a solution to (1). We abbreviate the underlined expression by $Fac$. We claim $fac = Y\ Fac$, i.e., it is a solution to (1).

Simply replacing $fac$ with $Y\ Fac$ in (1) we get

$$Y\ Fac = Fac\ (Y\ Fac)$$

which holds by the definition of $Y$.

Thus we see that a recursive definition of a function can be transformed so that the function is the fixpoint of an appropriate functional (a function taking a function as argument).

# Defining $Fin\ A$

We want to define a function $Fin$ such that $Fin\ A$ is the set of all finite subsets of $A$.

How do you construct the set of all finite subsets of $A$? The following pseudo-code suggests what you have to do:

$$
\begin{array}{l}
S := \{\{\}\};\\
\textbf{forever do}\\
\quad \textbf{foreach } a \in A \textbf{ do}\\
\quad\quad \textbf{foreach } B \in S \textbf{ do}\\
\quad\quad\quad \textbf{add } (\{a\} \cup B) \textbf{ to } S\\
\textbf{od od od}
\end{array}
$$

This means that you have to add new sets forever (however, when you actually do this construction for a finite set $A$, it will indeed reach a

fixpoint, i.e., adding new sets won't change anything).

Generally (even if $A$ is infinite), $Fin\ A$ is a set such that adding new sets as suggested by the pseudo-code won't change anything. Written as recursive equation:

$$Fin\ A = \{\{\}\} \cup \bigcup x \in A.((\texttt{insert}\ x)\ `\ Fin\ A)$$

Recall that ` is nice syntax for $image$, defined in `Set.thy`.

The above is a $\beta$-reduction of

$$Fin\ A = \underline{(\lambda X.\ \{\{\}\} \cup \bigcup x \in A.((\texttt{insert}\ x)\ `\ X))}\ (Fin\ A) \qquad (2)$$

We are looking for a solution to (2). We abbreviate the underlined expression by $FA$. We claim

$$Fin\ A = Y\ FA,$$

i.e., it is a solution to (2). Simply replacing $Fin\ A$ with $Y\ FA$ in (2) we get

$$Y\ FA = FA(Y\ FA),$$

which holds by the definition of $Y$.

You should compare this to what we said about $fac$. Note that in this example, there is no such thing as a recursive call to a "smaller" argument as in $fac$ example.

# Least Fixpoints

Jan-Georg Smaus and Burkhart Wolff

# The Roadmap

We are still looking at how the different parts of mathematics are encoded in the Isabelle/HOL library.

- Orders
- Sets
- Functions
- (Least) fixpoints and induction
- (Well-founded) recursion
- Arithmetic
- Datatypes

# A Conservative Approach for Least Fixpoints

- Restriction: $F$ is not of arbitrary type $(\alpha \Rightarrow \alpha)$ but of set type $(\alpha\ set \Rightarrow \alpha\ set)$.
- Instead of $Y$ define $lfp$ by an equation which is not recursive.
- $lfp$ is fixpoint combinator, but only under additional condition that $F$ is monotone, and: this is not obvious (requires non-trivial proof)!

This leads us towards recursion and induction.

# Lfp.thy

**constdefs**

lfp :: ['a set ⇒ 'a set] ⇒ 'a set

" lfp(f) == ⋂({u. f(u) ⊆u})"

Note: The definition of *lfp* is conservative.

That's fine. But is it a fixpoint combinator?

# Knaster-Tarski's Fixpoint Theorem

**Theorem 1 (Knaster-Tarski):**

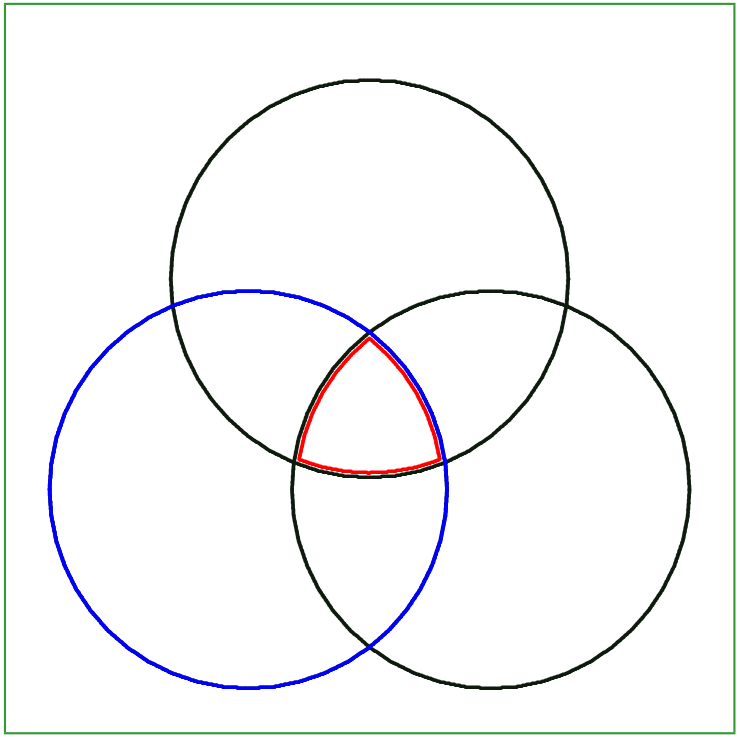
If $f$ is monotone, then $lfp\ f = f\ (lfp\ f)$.

The proof has four steps.

# Knaster-Tarski's Fixpoint Theorem (1)

Claim 1 ("$lfp$ lower bound"): If $f\ A \subseteq A$ then $lfp\ f \subseteq A$.

The box denotes "the set" $\alpha$. The three circles denote the sets $A$ for which $f\ A \subseteq A$.
By Lfp.thy, $lfp\ f$ is the intersection.
Pick an $A$ for which $f\ A \subseteq A$.
Clearly, $lfp\ f \subseteq A$.

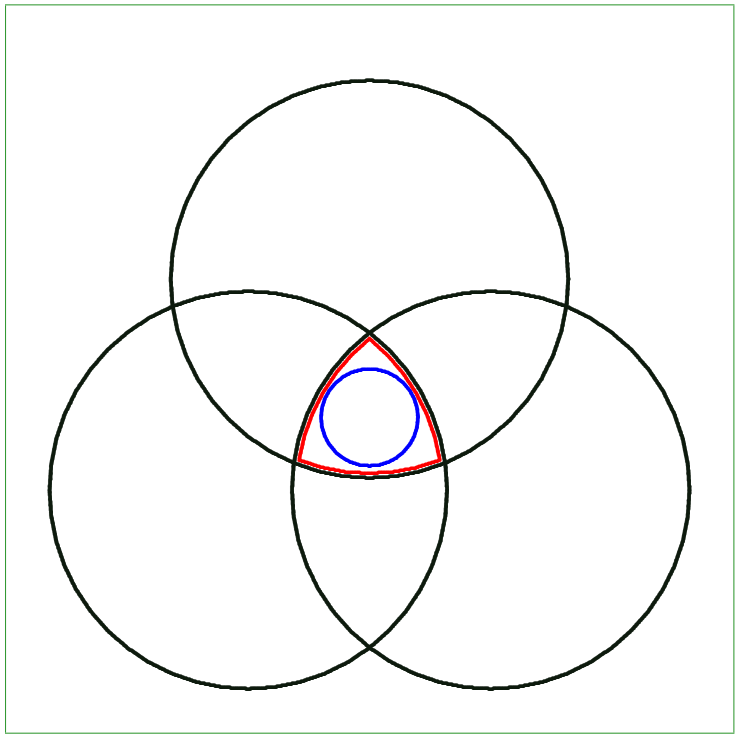


Or as proof tree.

# Knaster-Tarski's Fixpoint Theorem (2)

Claim 2 ("$lfp$ greatest"): For all $A$, if
for all $U$, $f\ U \subseteq U$ implies $A \subseteq U$, then $A \subseteq lfp\ f$.

The three circles denote the sets $U$ for which $f\ U \subseteq U$.
By hypothesis, $A \subseteq U$ for each $U$ (1st, 2nd, 3rd . . . ).
By definition, $lfp\ f$ is the intersection.
Clearly, $A \subseteq lfp\ f$.
Or as proof tree.

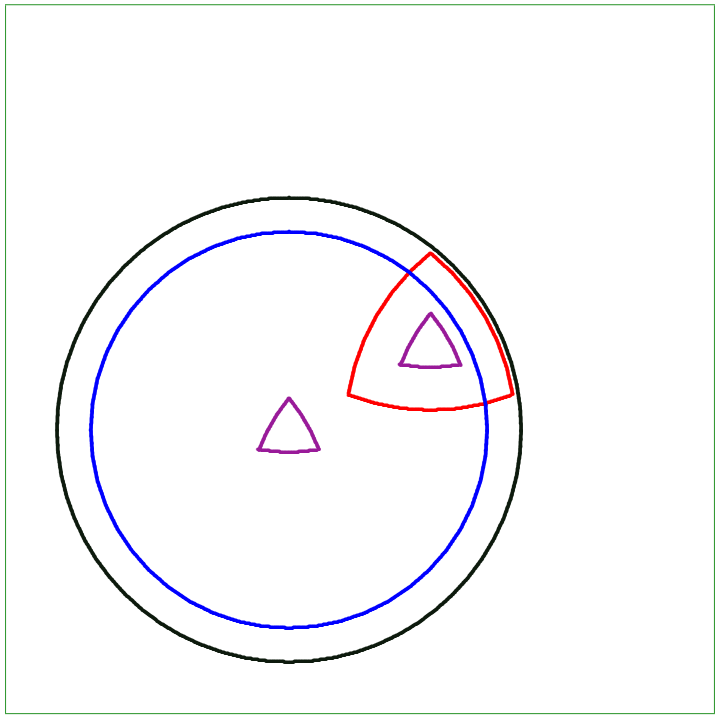


# Knaster-Tarski's Fixpoint Theorem (3)

Claim 3: If $f$ is monotone then $f(lfp\ f) \subseteq lfp\ f$.

First show Claim 3*: $f\ U \subseteq U$ implies $f(lfp\ f) \subseteq U$.

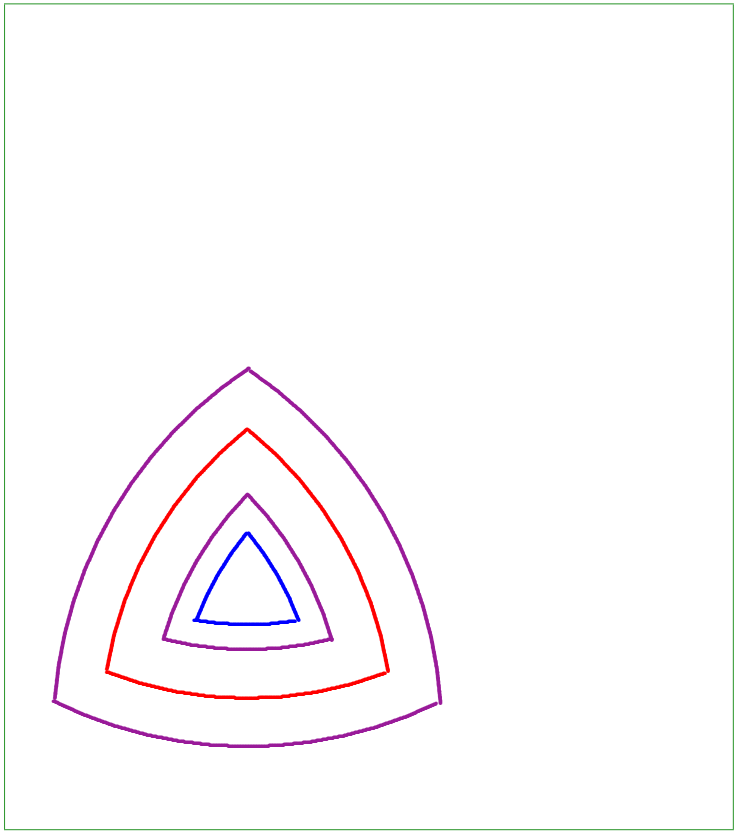
Let the circle be such a $U$. By Claim 1, $lfp\ f \subseteq U$.
$f\ U \subseteq U$ (hypothesis).
$f(lfp\ f) \subseteq f\ U$ (monotonicity).
$f(lfp\ f) \subseteq U$ (transitivity of $\subseteq$).
Claim 3* shown.
By Claim 2 (letting $A := f(lfp\ f)$), $f(lfp\ f) \subseteq lfp\ f$. Or as proof tree.



# Knaster-Tarski's Fixpoint Theorem (4)

Claim 4: If $f$ is monotone then $lfp\ f \subseteq f(lfp\ f)$.

By Claim 3, $f(lfp\ f) \subseteq lfp\ f$.
By monotonicity, $f(f(lfp\ f)) \subseteq f(lfp\ f)$.
By Claim 1 (letting $A := f(lfp\ f)$), $lfp\ f \subseteq f(lfp\ f)$.



Or as proof tree.

# Knaster-Tarski's Fixpoint Theorem: QED

Claim 3 ($lfp\ f \subseteq f(lfp\ f)$) and Claim 4 ($f(lfp\ f) \subseteq lfp\ f$) together give the result:

If $f$ is monotone, then $lfp\ f = f\ (lfp\ f)$.

So under appropriate conditions, $lfp$ is a fixpoint combinator.

Or as proof tree.

We will reuse Claim 1 also for the proofs on induction.

# Alternative: A Natural-Deduction Style Proof

The proof can also be presented in natural deduction style.

# Knaster-Tarski's Fixpoint Theorem (1)

Claim 1 ("$lfp$ lower bound"): If $f\ A \subseteq A$ then $lfp\ f \subseteq A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[f\ A \subseteq A]^1}{A \in \{u.f\,u \subseteq u\}}\ \text{CollectI}}{\bigcap\{u.f\,u \subseteq u\} \subseteq A}\ \text{Inter\_lower}}{lfp\ f \subseteq A}\ \text{Def. } lfp}{f\ A \subseteq A \longrightarrow lfp\ f \subseteq A}\ \longrightarrow\text{-}I^1$$

# Knaster-Tarski's Fixpoint Theorem (2)

Claim 2 ("*lfp* greatest"): For all $A$, if for all $U$, $f\ U \subseteq U$ implies $A \subseteq U$, then $A \subseteq lfp\ f$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x.fx \subseteq x \longrightarrow A \subseteq x]^1}{\forall x.x \in \{u.fu \subseteq u\} \longrightarrow A \subseteq x}\ \textit{subst}, \text{CollectI}}{A \subseteq \cap\{u.fu \subseteq u\}}\ \text{Inter\_greatest}}{A \subseteq lfp\,f}\ \text{Def. } \textit{lfp}}{(\forall x.fx \subseteq x \longrightarrow A \subseteq x) \longrightarrow A \subseteq lfp\,f}\ \longrightarrow\text{-}I^1$$

# Knaster-Tarski's Fixpoint Theorem (3)

Claim 3: If $f$ is monotone then $f(lfp\ f) \subseteq lfp\ f$.

$$\dfrac{\dfrac{\dfrac{\dfrac{[mono\ f]^1 \quad \dfrac{[fx \subseteq x]^2}{lfp\ f \subseteq x}}{f(lfp\ f) \subseteq f\ x} \qquad [fx \subseteq x]^2}{f(lfp\ f) \subseteq x}\ \text{order\_trans}}{\forall x.fx \subseteq x \longrightarrow f(lfp\ f) \subseteq x}\ \forall\text{-}I,\ \longrightarrow\text{-}I^2}{f(lfp\ f) \subseteq lfp\ f}\ \text{Claim2},\ \longrightarrow\text{-}E}{mono\ f \longrightarrow f(lfp\ f) \subseteq lfp\ f}\ \longrightarrow\text{-}I^1$$

# Knaster-Tarski's Fixpoint Theorem (4)

Claim 4: If $f$ is monotone then $lfp\ f \subseteq f(lfp\ f)$.

$$\dfrac{\dfrac{\dfrac{[mono\ f]^1 \quad \dfrac{[mono\ f]^1}{f(lfp\ f) \subseteq lfp\ f}\ \text{Claim 3, } \rightarrow\text{-}E}{f(f(lfp\ f)) \subseteq f(lfp\ f)}\ \text{monoD}}{lfp\ f \subseteq f(lfp\ f)}\ \text{Claim1, } \rightarrow\text{-}E}{mono\ f \rightarrow lfp\ f \subseteq f(lfp\ f)}\ \rightarrow\text{-}I^1$$

# Completing Proof Tree

$$\dfrac{\dfrac{\dfrac{[mono\ f]^1}{lfp\ f \subseteq f(lfp\ f)}\ \text{Claim 4} \qquad \dfrac{[mono\ f]^1}{f(lfp\ f) \subseteq lfp\ f}\ \text{Claim 3}}{lfp\ f = f(lfp\ f)}\ \text{equalityI}}{mono\ f \longrightarrow lfp\ f = f(lfp\ f)}\ \longrightarrow\text{-}I^1$$

# More Detailed Explanations

# Algorithmic Version of $lfp$

The theorem

$$(\forall S.\ f(\bigcup S) = \bigcup(f\ `\ S)) \Longrightarrow \bigcup_{n\in N}(f^n\{\}) = lfp\ f$$

says that under a certain condition, $lfp\ f$ can be computed by applying $f$ to the empty set over and over again:

- although the expression uses the union over all natural numbers, which is an infinite set, this can sometimes effectively be computed. Under certain conditions, there exists a $k$ such that $f^k\ \{\} = f^{k+1}\{\}$.
- Even if $\bigcup n \in Nat.f^n\ \{\}$ cannot be effectively computed, it can still be approximated: for any $k$, we know that $\bigcup i \leq k.f^i\ \{\} \subseteq \bigcup n \in Nat.f^n\ \{\}$.

# Monotone Functions

A function $f$ is monotone w.r.t. a partial order $\leq$ if the following holds: $A \leq B$ implies $f(A) \leq f(B)$.

In particular, we consider the order given by the subset relation.

# "The Set" $\alpha$

$\alpha$ is not a set but a type (variable). But we can consider the set of all terms of that type (`UNIV` of type $\alpha$).

The polymorphic constant `UNIV` was defined in `Set.thy`. `UNIV` of type $\tau$ `set` is the set containing all terms of type $\tau$.

# Three Circles?

In general, needless to say, there could be any number of such sets, but the picture is to be understood in the sense that the three circles are all the sets $A$ with the property $f\ A \subseteq A$.

# Different Phrasings

The theorem is phrased a bit differently in the "mathematical" version we give here and in the Isabelle version (see Lfp.thy). This is convenient for the graphical illustration of the proof.

The "mathematical phrasing" corresponding closely to the Isabelle version would be the following:

**Theorem 2 (Induct (alternative)):**

If

- $a \in \mathit{lfp}\ f$, and
- $f$ is monotone, and
- for all $x$, $x \in f(\mathit{lfp}\ f \cap \{x \mid P\,x\})$ implies $P\,x$

then $P\,a$ holds.

Other phrasings, which may help to get some intuition about the

theorem:

**Theorem 3 (Induct (alternative)):**

If

- $a \in lfp\ f$, and
- $f$ is monotone, and
- $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq \{x \mid P\,x\}$

then $P\,a$ holds.

**Theorem 4 (Induct (alternative)):**

If

- $f$ is monotone, and
- $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq \{x \mid P\,x\}$

then for all $x$ in $lfp\ f$, we have $P\,x$.

# Detail on Monotonicity

$lfp\ f \cap \{x \mid P\,x\} \subseteq lfp\ f$, so by monotonicity,
$f(lfp\ f \cap \{x \mid P\,x\}) \subseteq f(lfp\ f)$.

# Use of Claim 1

We have just seen $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq lfp\ f \cap \{x \mid P\,x\}$.

By Claim 1 (setting $A := lfp\ f \cap \{x \mid P\,x\}$), this implies

$lfp(f) \subseteq lfp\ f \cap \{x \mid P\,x\}$.

# Antisymmetry of $\subseteq$

We have $lfp\ f \cap \{x \mid P\,x\} \subseteq lfp(f)$ and $lfp(f) \subseteq lfp\ f \cap \{x \mid P\,x\}$, and so $lfp(f) = lfp\ f \cap \{x \mid P\,x\}$ by the antisymmetry of $\subseteq$.

# Recursion in a Definitional Way

Recall why we were interested in fixpoints.

The problem with $Y$ is that it leads to inconsistency (and of course, the definition of $Y$ is not a constant definition/conservative extension.).

The definition of $lfp$ is conservative.

And in appropriate situations, it can be used to define recursive functions.

Compared to $Y$, the type of $lfp$ is restricted.

This restriction means that there is no obvious way to use $lfp$ for defining recursive numeric functions such as $fac$.

# Finite Sets in Isabelle

Above, we defined the set of finite subsets of a set $A$. Alternatively, one could define "the set of all finite sets whose elements have type $\tau$". In this case, no fixed set $A$ is involved, and it is closer to what actually happens in Isabelle.

In Finite_Set.thy a constant $Finites$ is defined. It has polymorphic type $\alpha\, set\, set$. We have $A \in Finites$ if and only if $A$ is a finite set. However, it would be wrong to think of $Finites$ as one single set that contains all finite sets. Instead, for each $\tau$, there is a polymorphic instance of $Finites$ of type $\tau\, set\, set$ containing all finite sets of element type $\tau$.

In Finite_Set.thy we find the inductive definition:

```
inductive " Finites"
  intros
    emptyI [simp,  intro !]:  "{} ∈ Finites"
```

insertI [simp, intro !]: "A : Finites $\Longrightarrow$ *insert* a A $\in$ Finites"

The Isabelle mechanism of interpreting the keyword `inductive` translates this into the following definition: $Finites = lfp\ G$ where

$$G \equiv \lambda S.\ \{x \mid x = \{\} \vee (\exists A\ a.\ x = insert\ a\ A \wedge A \in S)\}$$

As a sanity-check, consider the type of this expression. The expression $insert\,a\,A$ forces $A$ to be of type $\tau\ set$ for some $\tau$ and $a$ to be of type $\tau$. Next, $insert\,a\,A$ is of type $\tau\ set$, and hence $x$ is also of type $\tau\ set$. Moreover, the expression $A \in S$ forces $S$ to be of type $\tau\ set\ set$. The expression $\{x \mid x = \{\} \vee (\exists A\,a.\ x = insert\,a\,A \wedge A \in S)\}$ is of type $\tau\ set\ set$. Next, $G$ is of type $\tau\ set\ set \rightarrow \tau\ set\ set$, and so finally, $Finites$ is of type $\tau\ set\ set$. But actually, since $\tau$ is arbitrary, we can replace it by a type variable $\alpha$.

Note that there is a convenient syntactic translation

**translations** " finite A" == "A : Finites"

# $F$ is monotone

This proof is of course done in Isabelle.

# Induction for Finite Sets

The theorem

$$[\![xa \in Fin\ A; P\ \{\}; \bigwedge ab.[\![a \in A; b \in Fin\ A; P\ b]\!] \Longrightarrow P\ (insert\ a\ b)]\!] \Longrightarrow P\ xa$$

is an instance of the general induction scheme. That is to say, if we take the general induction scheme `lfp_induct`

$$[\![a \in lfp\ f; mono\ f; \bigwedge x.x \in f\ (lfp\ f \cap \{x.P\ x\}) \Longrightarrow P\ x]\!] \Longrightarrow P\ a$$

and instantiate $f$ to $\lambda X.\{\{\}\} \cup \bigcup x \in A.((\texttt{insert}\ x)\ `\ X)$ then some massaging using the definitions will give us the first theorem.

Note here that monotonicity has disappeared from the assumptions. This is because the monotonicity of $F$ is shown by Isabelle once and for all.

This is one aspect of what we mean by special proof support for inductive definitions.

The least fixpoint of the functional is $Fin\ A$ (the set of finite subsets of $A$) in this case.

# Co-induction

Co-induction is a construction analogous to induction but using greatest fixpoints. This is a useful mechanism when defining infinite sequences, streams, and similar structures inductively.

# Inductive Definitions

Jan-Georg Smaus and Burkhart Wolff

# Induction Based on Lfp.thy

**Theorem 5 (lfp induction):**

If

- $f$ is monotone, and
- $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq \{x \mid P\,x\}$,

then $lfp\ f \subseteq \{x \mid P\,x\}$.

In Isabelle, it is called `lfp_induct`:

$$[\![a \in lfp\,f; mono\,f; \bigwedge x.x \in f(lfp\,f \cap \{x.P\,x\}) \Longrightarrow P\,x]\!] \Longrightarrow P\,a$$

We now show the theorem similarly as Tarski's Theorem.

# Showing `lfp_induct`

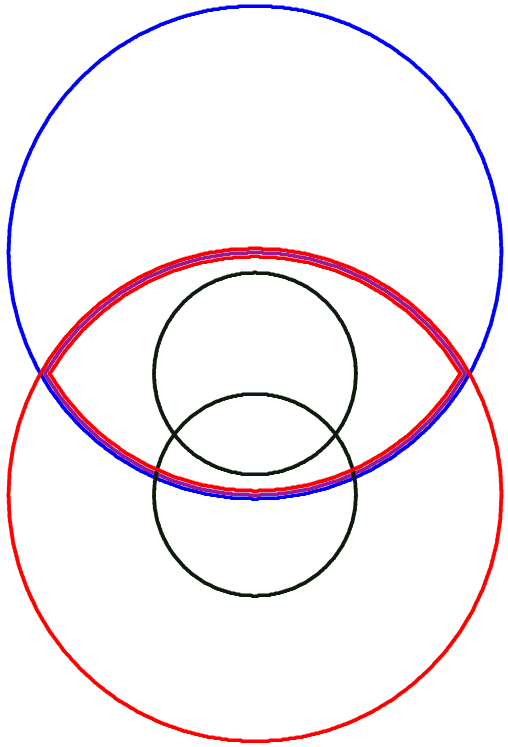
Circles denote $lfp\ f$ and $\{x \mid P\,x\}$.

By monotonicity, $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq f(lfp\ f)$. By Tarski, $lfp\ f = f(lfp\ f)$. Hence $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq lfp\ f$.

By hypothesis, $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq \{x \mid P\,x\}$, and so we must adjust picture: $f(lfp\ f \cap \{x \mid P\,x\}) \subseteq lfp\ f \cap \{x \mid P\,x\}$.

By Claim 1, $lfp\ f \subseteq lfp\ f \cap \{x \mid P\,x\}$ and so $lfp\ f = lfp\ f \cap \{x \mid P\,x\}$.

Conclusion: $lfp\ f \subseteq \{x \mid P\,x\}$.



# Approximating Fixpoints

Looking ahead: Suppose we have the set $N$ of natural numbers (the type is formally introduced later). The theorem `approx`

$$(\forall S.\ f(\bigcup S) = \bigcup(f\ `\ S)) \Longrightarrow \bigcup_{n\in N}(f^n\{\}) = lfp\ f$$

shows a way of approximating $lfp$, which is important for algorithmic solutions (e.g. in program analysis).

You will show this as an exercise.

# Where Are We Going? Induction and Recursion

Let's step back: What is an inductive definition of a set $S$?

It has the form: $S$ is the smallest set such that:

- $\emptyset \subseteq S$;
- if $S' \subseteq S$ then $F(S') \subseteq S$ (for some appropriate $F$).

At the same time, $S$ is the smallest solution of the recursive equation $S = F(S)$.

Induction and recursion are two faces of the same coin.

# Lfp.thy for Inductive Definitions

Least fixpoints are for building inductive definitions of sets in a definitional way: $S := lfp\ F$.

This is obviously well-defined, so why this fuss about monotonicity and Tarski?

Tarski allows us to exploit the equation $lfp\ f = f(lfp\ f)$ in proofs about $S$! That's what $lfp$ is all about.

# Example (Revisited)

The set of all finite subsets of a set $A$:

$$Fin\ A = lfp\ F$$

where $F = \lambda X.\{\{\}\} \cup \bigcup x \in A.((\texttt{insert}\, x)\, `\, X)$.

Thus we can do using $lfp$ what we would have wanted to do using $Y$.

To show: $F$ is monotone!

There will be an exercise on this.

# Example: Transitive Closures

The transitive closure of a relation $R :: ('a \times' a)set$ can be defined as follows:

$$R^* = lfp\ F$$

where $F = \lambda X.Id \cup r\ O\ X)$.

To show: $F$ is monotone!

There is also an exercise on this.

# The Package for Inductive Sets

Since inductive definitions are so ubiquitious, Isabelle provides a special compiler (a "package") with an own front-end syntax.

It generates conservative definitions and derives introduction and induction rules from them.

# The Package for Inductive Sets

Example:

```
consts Fin :: 'a set => 'a set set
inductive "Fin(A)"
  intrs
    emptyI "{} : Fin(A)"
    insertI "[| a: A; b: Fin(A) |] ==>
insert a b : Fin(A)"
```

# Technical Support for Inductive Definitions

Support important in practice since many constructions are based on inductively defined sets (datatypes, . . . ) Support provided for:

- Automatic proof of monotonicity
- Automatic proof of induction rule, for example:

$$[\![xa \in Fin\ A; P\ \{\}; \bigwedge ab.[\![a \in A; b \in Fin\ A; P\ b]\!] \Longrightarrow P\ (insert\ a\ b)]\!] \Longrightarrow P\ xa$$

This works also for mutually recursive definitions, co-inductive definitions, . . .

# Summary on Least Fixpoints

- We are interested in recursion because we need recursively defined sets.
- It turns out that inductively defined sets are solutions to recursive equations.
- We cannot have general fixpoint operator $Y$, but we can have a conservatively defined least fixpoints operator.
- There is an induction scheme (lfp induction) for proving theorems about an inductively defined sets.