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: Well-Founded Recursion

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

Motivation

Motivation(1)

After least fixpoints, well-founded recursion is our second concept of recursion represented by another fixpoint combinator.

Idea: Modeling “terminating” recursive functions, i.e. recursive definitions that use “smaller” arguments for the recursive call.

Claim: An axiom like:

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

is no problem since “it terminates” !

Motivation(2)

However: Logic talks about validity, not execution !

Moreover: is this true? What does this mean precisely ?

1. Consider: fac :: int → int !
2. Consider:

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

Motivation(3)

1) shows that arguments must be ordered wrt. to a well-founded ("terminating") ordering,
2) shows that the context of the recursive call ("the function body") must be coherent, i.e. it must supply only arguments to the recursive call which are lesser w.r.t. this ordering.

How can this be modeled?

Motivation(4)

One aspect of the problem: In HOL we can represent the "context of a recursive call". Reconsider:

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

Abstracting the recursive call yields:

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

We say: Fac is the body of fac.

Recall that a general fixpoint combinator can define fac by its body by $Y\ Fac$ and thus solve $fac = Fac\, fac$.

In the sequel, we will define and explore the

- concept of well-founded ordering
- concept of coherence of a body

Prerequisite: Relations

We need some standard operations on binary relations (sets of pairs), such as converse, composition, image of a set and a relation, the identity relation, . . .

These are provided by Relation.thy.

Relation.thy (Fragment)

constdefs

```
  converse  ::  ('a×'b)set ⇒ ('b×'a) set    ("(_^−1)" ..)
  r^−1       ≡{(y, x). (x, y) ∈ r}
  rel_comp  ::  [('b×'c)set, ('a×'b)set] ⇒ ('a×'c)set
                                                ("(_O_)" ..)
  r O s ≡{(x,z). ∃ y. (x, y)∈s ∧ (y, z)∈r}
  Image ::  [('a×'b)set, 'a set] ⇒ 'b set   ("(_“_)" ..)
  r “ s ≡{y. ∃ x∈s. (x,y)∈r}
  Id      ::  ('a×'a) set
  Id      ≡{p. ∃ x. p = (x,x)}
```

As can be expected, these notions are similar to Fun.thy.

Prerequisite: Closures

We need the transitive, as well as the reflexive transitive closure of a relation.

These are provided by Transitive_Closure.thy.

How would you define those inductively

Transitive_Closure.thy (Fragment)

```
consts
  rtrancl :: ('a × 'a) set ⇒ ('a × 'a) set
                                          ("(_^*)"  ..)
inductive "r^*"
 intros
   rtrancl_refl    [...]:
          (a, a) ∈ r^*
   rtrancl_into_rtrancl    [...]:
          ⟦(a, b) ∈ r^*; (b, c) ∈ r ⟧⟹ (a, c) ∈ r^*
```

Transitive_Closure.thy (Fragment Cont.)

consts

trancl :: ('a × 'a) set ⇒ ('a × 'a) set ("(_^+)" ..)

inductive "r^+"

intros

r_into_trancl [...]:

$(a, b) \in r \Longrightarrow (a, b) \in r\hat{}+$

trancl_into_trancl [...]:

$(a, b) \in r\hat{}+ \Longrightarrow (b, c) \in r \Longrightarrow (a,c) \in r\hat{}+$

Well-Founded Orderings

Defined in Wellfounded_Recursion.thy.

Wellfounded_Recursion = Transitive_Closure +

constdefs

$$\begin{array}{lll} \mathsf{wf} & :: & (\text{'a} \times \text{'a})\ \mathsf{set} \Rightarrow \mathsf{bool} \\ \mathsf{wf(r)} & \equiv & (\forall\ P.\ (\forall\ x.\ (\forall\ y.\ (y,x) \in r \longrightarrow P(y)) \\ & & \quad \longrightarrow P(x)) \longrightarrow (\forall\ x.\ P(x))) \end{array}$$

In other words . . . A relation r is well-founded iff well-founded (Noetherian) induction based on r is a valid proof scheme. This is conservative, fine. But does it meet our intuition of “termination”?

Gaining Intuition of Well-Foundedness

A first reality-check: Is $\emptyset$ well-founded?

The definition of wf is:

Let's instantiate r to $\emptyset$.

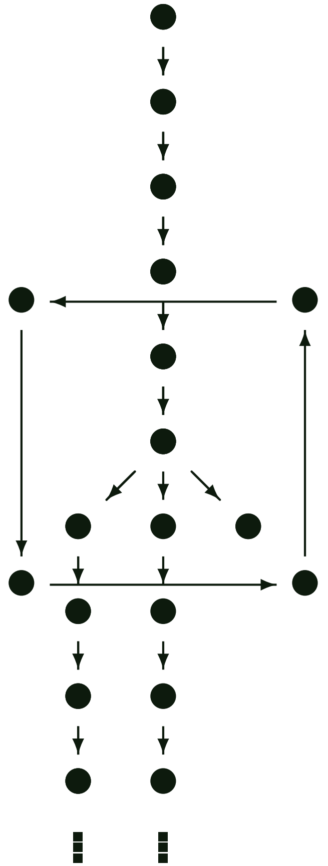
$$wf(\emptyset) \equiv \forall P. (\forall x. (\forall y. (y, x) \in \emptyset \to P(y)) \to P(x)) \to (\forall x. P(x))$$

So the empty set is well-founded.

Gaining Intuition of Well-Foundedness

Intuition of wf: All descending chains are finite.

But: concept of “finite chain” is difficult to express; we therefore look for for alternatives.



- Not symmetric: $(x, y) \in r \longrightarrow (y, x) \notin r$?
- No cycles: $(x, x) \notin r^+$?
- r has minimal element: $\exists x.\forall y.(y, x) \notin r$?
  Note: Trivial for $r = \emptyset$.
- Any subrelation must have minimal element: $\forall p.p \subseteq r \longrightarrow \exists x.\forall y.(y, x) \notin p$? “Minimal element” badly formalized (already in previous point).

The Characterisation

All these attempts are just necessary but not sufficient conditions for well-foundedness.

Here is a characterization:

$$wf\ r = \forall r'.\, r' \neq \{\} \wedge r' \subseteq r \longrightarrow (\exists x \in Domain\, r'.\forall y.(y, x) \notin r')$$

Here is an alternative characterization:

$$wf\ r = (\forall Q x.\ x \in Q \longrightarrow (\exists x \in Q.\ \forall y.(y, x) \in r \longrightarrow y \notin Q))$$

Let's see some theorems to confirm our intuition, including the statements just shown.

A Theorem for Induction

By massage of the definition of well-foundedness

$$\forall P.(\forall x.(\forall y.(y,x) \in r \longrightarrow P\,y) \longrightarrow P\,x) \longrightarrow (\forall x.P\,x)$$

one obtains the theorem wf_induct

$$[\![wf\ r; \bigwedge x.\forall y.(y,x) \in r \longrightarrow P\,y \Longrightarrow P\,x]\!] \Longrightarrow P\,a.$$

This is a form suitable for doing induction proofs in Isabelle.

Induction Theorem as Proof Rule

The Isabelle theorem wf_induct

$$\llbracket wf\ r; \bigwedge x. \forall y. (y, x) \in r \longrightarrow P\,y \Longrightarrow P\,x \rrbracket \Longrightarrow P\,a.$$

as proof rule:

$$\frac{wf\ r \qquad\qquad \begin{array}{c}[\forall y. (y, x) \in r \longrightarrow P\,y]\\ \vdots \\ P\,x\end{array}}{P\,a}\ \texttt{wf\_induct}$$

A Theorem on Antisymmetry

wf_not_sym: \<lbrakk wf r; (a, x) **in** r⟧ $\Longrightarrow$ (x, a) ∈ r

Proof sketch:

$$\dfrac{wf\ r \qquad \begin{array}{c}[\forall y.(y,x)\in r \to (\forall z.(y,z)\in r \to (z,y)\notin r)]\\ \vdots \\ \forall z.(x,z)\in r \to (z,x)\notin r\end{array}}{\forall z.(a,z)\in r \to (z,a)\notin r}\ \texttt{wf\_induct}$$

Rest routine though not so trivial (needs classical reasoning).

A variation will be done as exercise.

Theorems on Absence of Cycles

wf_not_refl : wf r $\Longrightarrow$ (a, a) $\notin$ r
wf_trancl : wf r $\Longrightarrow$ wf (r^+)
wf_acyclic : wf r $\Longrightarrow$ acyclic r

(where $acyclic\, r \equiv \forall x.(x, x) \notin r^+$)

Proof sketch:

wf_not_refl : Corollary of wf_not_sym.
wf_trancl : Uses induction.
wf_acyclic : Apply wf_not_refl and wf_trancl .

Ergo: Definition of wf meets our intuition of “no cycles”.

Another Theorem ("Exists Minimal Element")

wf_minimal: wf r $\Longrightarrow \exists$ x. $\forall$ y. (y,x) $\notin$ r^+

Proof sketch, abbreviating $\phi \equiv (\exists x.\forall y.(y,x) \notin r^+)$:

$$
\dfrac{\dfrac{\mathtt{wf}(r)}{\mathtt{wf}(r^+)}\bullet \quad \dfrac{\overline{\phi \vee \neg\phi}\,\bullet \quad [\phi]^2 \quad \dfrac{\dfrac{\dfrac{[\neg\phi]^2}{\forall x.\exists y.(y,x) \in r^+}\ \cdots \quad \dfrac{[\neg\phi]^2 \quad [\forall w.(w,v) \in r^+ \to \phi]^1}{\exists w.(w,v) \notin r^+}\bullet}{False}\ \cdots}{\phi}\,FalseE}{\phi}\,disjE^2}{\phi}\,\mathtt{wf\_induct}^1
$$

[illegible] check!

A Characterization of wf

The theorem wf_eq_minimal is characterization of well-foundedness.:

$$wf\ r = (\forall Q x. x \in Q \longrightarrow (\exists z \in Q. \forall y. (y, z) \in r \longrightarrow y \notin Q))$$

Proof uses iffI $=$, use `wf_def`, rest routine.

Ergo: Definition of `wf` meets textbook definitions “every non-empty set Q has a minimal element in r” (more or less standard textbook).

A Theorem on Subsets

`wf_subset` $\llbracket wf\ r; p \subseteq r \rrbracket \Longrightarrow wf\ p$

Proof sketch:

wf_subset: simplification tactic using wf_eq_minimal.

A Theorem on Subrelations

wf r $\Longrightarrow$ $\forall$ p. p $\subseteq$ r $\longrightarrow$ $\exists$ x. $\forall$ y. (y,x) $\notin$ p^+

Proof sketch: Combine wf_minimal and wf_subset.
This implies $wf\ r \Longrightarrow \forall p.p \subseteq r \rightarrow \exists x.\forall y.(x, y) \notin p$.

Ergo: `wf` fulfills the conditions of second attempt of characterizing well-foundedness using minimal elements.
Note this is not a characterization: The subrelation must be non-empty, and minimum must be in the domain of p in order to rule out an isolated element, unrelated to the subrelation. (see characterizations)

Defining Recursive Functions

Coherent Function Bodies

A function body H is coherent w.r.t. $<$ if all recursive calls are supplied with arguments “smaller” than the original argument.

This means that $H\,f\,a$ and $H\,f'a$ are equal provided that that $f x = f'x$ for all $x < a$.

This allows us to use an approximation f' instead of a “perfect” f when recursively defining a function.

Using Approximating f's

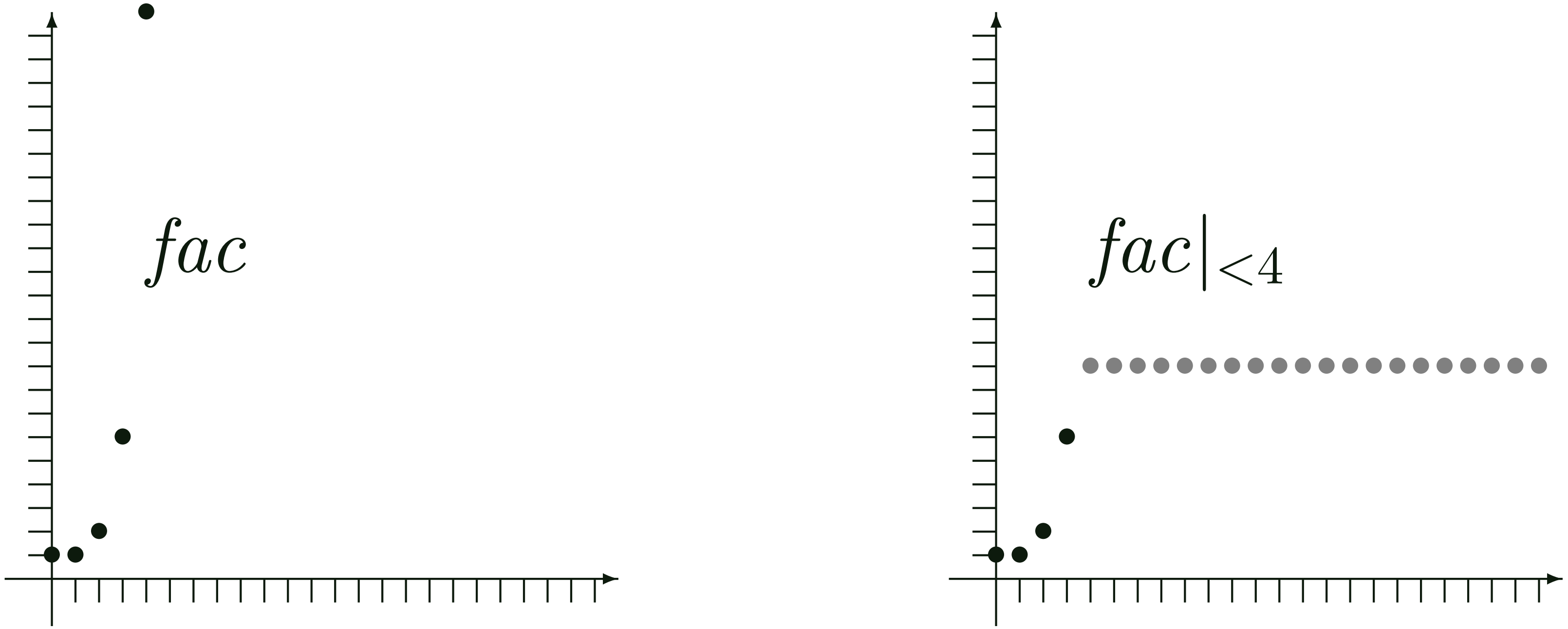
Let $f|_{<a}$ be a function that is like f on all values $< a$, and arbitrary elsewhere. $f|_{<a}$ is an approximation, a “bad” f.

Now we can define coherence of H by:

$$H\ f\ a = H\ (f|_{<a})\ a. \tag{1}$$

Approximating f's: Example

Consider fac. On the right-hand side, we show one possibility for $fac|_{<4}$):



cut (in Wellfounded_Recursion.thy)

Technically, the function $f|_{<x}$ is defined as follows:

constdefs

```
cut :: ('a ⇒ 'b) ⇒ ('a × 'a)set ⇒ 'a ⇒ 'a ⇒ 'b
cut f r x ≡ λy. if (y,x)∈r then f y else arbitrary
```

The unspecified constant `arbitrary` is declared in HOL.thy.

The function cut f r x is therefore unspecified for arguments y where (y,x)∉ r, but for each such argument, (cut f r x) y must be the same in any particular model.

Theorems Involving cut

Properties of cut:

cuts_eq (cut f r x = cut g x) =
(∀ y. (y,x)∈r ⟶ f y = g y)

cut_apply (x,a)∈r ⟹ cut f r a x = f x

Or, using the previous textbook notation:

`cuts_eq` $(f|_{<x} = g|_{<x}) = (\forall y. y < x \longrightarrow f\,y = g\,y)$

`cut_apply` $x < a \Longrightarrow f|_{<a}\,x = f\,x$

wfrec_rel (in Wellfounded_Recursion.thy)

construction: “approximate” f by a relation wfrec_rel R F.

$$\text{wfrec\_rel} :: (\text{'a} \times \text{'a})\ \text{set} \Rightarrow ((\text{'a} \Rightarrow \text{'b}) \Rightarrow \text{'a} \Rightarrow \text{'b}) \Rightarrow (\text{'a} \times \text{'b})\ \text{set}$$

inductive ” wfrec_rel R F”

intrs

$$\text{wfrecI} \quad \forall z.\ (z, x) \in R \longrightarrow (z, g\ z) \in \text{wfrec\_rel}\ R\ F \Longrightarrow (x, F\ g\ x) \in \text{wfrec\_rel}\ R\ F$$

More on wfrec_rel

Assume the ordering on natural numbers pred_nat and assume wf pred_nat.

Question: Which elements do we have in wfrec_rel pred_nat Fac ?

(0, Fac g 0) ∈ wfrec_rel pred_nat Fac
(1, Fac (Fac g) 1) ∈ wfrec_rel pred_nat Fac
(2, Fac (Fac (Fac g)) 2) ∈ wfrec_rel pred_nat Fac
. . .

wfrec (in Wellfounded_Recursion.thy)

Now we turn the relation wfrec_rel into a function:

$$\begin{aligned} \text{wfrec} ::\ & (\text{'a} \times \text{'a})\ \text{set} \Rightarrow \\ & ((\text{'a} \Rightarrow \text{'b}) \Rightarrow \text{'a} \Rightarrow \text{'b}) \Rightarrow \text{'a} \Rightarrow \text{'b} \end{aligned}$$

$$\begin{aligned} \text{wfrec R F} \equiv \lambda\text{x. THE y.} \\ & (\text{x, y}) \in \text{wfrec\_rel R} (\lambda\text{f x. F(cut f R x)x}) \end{aligned}$$

Note that the type of wfrec R is again an instance of the type of the Y-combinator (similar lfp).

THE x. P x picks the unique a such that P a holds, if it exists. Otherwise (see HOL.thy) it is arbitrary.

The Fixpoint Theorem

Theorem: wfrec satisfies the fixpoint property:

$$\text{wfrec}:\ \text{wf r} \Longrightarrow \text{wfrec r H a} = \text{H (cut wfrec r H r a) a}$$

Note that wfrec is used here both as a name of a constant (defined above) and a theorem. So if R is well-founded and the body H is coherent, we have

$$\text{wfrec r H a} = \text{H (wfrec r H) a}$$

Example for *wfrec*: Natural Numbers

The constant wfrec provides the mechanism/support for defining recursive functions. We illustrate this using nat, the type of natural numbers.

wfrec is applied to a well-founded order and a body to define a function.

First, define predecessor relation:

constdefs

pred_nat :: $(\text{nat} \times \text{nat})$ set

pred_nat $\equiv \{(\text{m},\text{n}).\ \text{n} = \text{Suc m}\}$

How would you define addition or subtraction?

Defining Division and Modulus

```
div  ::  ['a :: div,  'a] ⇒ 'a             ( infixl  70)
m div n ≡wfrec (pred_nat^+)
                (λf j.  if  j<n ∨n=0 then 0
                    else  Suc (f  (j−n))) m

mod ::  ['a :: div,  'a] ⇒ 'a             ( infixl  70)
m mod n ≡wfrec (pred_nat^+)
                 (λf j.  if  j<n ∨n=0 then j
                     else  f  (j−n)) m
```

Here, `div` is a syntactic class for which division is defined.

We assume a definition for −(subtract).

The functions are recursive in one argument (just like `add`).

Theorems of the Example

wf_pred_nat : wf pred_nat

m mod n = if m < n then m else (m − n) mod n
m div n = if m < n then 0 else Suc((m − n) div n)

This is very similar to functional programming code and hence lends itself to real computations (rewriting), as opposed to only doing proofs.

Package for Primitive Recursion

For primitive recursion, finding a well-founded ordering is simple enough for automation!

Examples (use `nat` and `case`-syntax): . . .

Recursion and Arithmetic

Isabelle provides a syntactic front-end for defining an important subclass of well-founded recursions, namely primitive recursive functions:

primrec

add_0: $0 + n = n$

add_Suc: $\mathrm{Suc}\ m + n = \mathrm{Suc}\ (m + n)$

primrec

diff_0 : $m - 0 = m$

diff_Suc : $m - \mathrm{Suc}\ n = (\mathrm{case}\ m - n\ \mathbf{of}$

$$\begin{array}{lll} & 0 & => 0 \\ | & \mathrm{Suc}\ k & => k) \end{array}$$

Recursion and Arithmetic

recdef statement is more general and requires a mesure-function (involving a proof of well-foundedness potentially requiring user interaction).

Example:

```
consts posDivAlg :: "int*int => int*int"
recdef posDivAlg "inv_image less_than
                  (λ(a,b). nat(a − b + 1))"
 "posDivAlg (a,b) = (if (a<b | b≤0) then (0,a)
                     else adjust b (posDivAlg(a, 2*b)))"
```

Conclusion

- We can model recursively defined functions conservatively!
- Together with the theory of least fixpoints, we can avoid a general fixpoint combinator Y.
- There is a further powerful induction principle wf_induct.
- The methodological overhead can be faced by powerful mechanical support.

More Detailed Explanations

Bad Formalization of “Minimal Element”

In this attempt, we formalized the “minimal element in p” as an x such that there is no y with $(x, y) \in p$. But this is a bad formalization since an isolated element, i.e., one that is completely unrelated to p, or even to r, would meet the definition.

In fact, this problem was already present for the previous attempt where we just required $\exists x. \forall y. (y, x) \notin r$ (i.e., r has a minimal element).

No Infinite Descending Chains

The final condition

$$(\forall Q x . x \in Q \longrightarrow (\exists z \in Q . \forall y . (y, z) \in r \longrightarrow y \notin Q))$$

expresses the absence of infinite descending chains without explicitly using the concept of infinity.

It is a characterization of well-foundedness. One could say that the above formula expresses what well-foundedness is, while the “offical” definition is somewhat indirect since it defines well-foundedness by an induction principle. As we have seen, both repesentations are equivalent.

`induct_wf`

As far as the induction principle is concerned, `induct_wf` states the same as the very definition of `wf`. All that happens is that some explicit universal object-level quantifiers are removed and the according variables are (implicitly) universally quantified on the meta-level, and some shifting from object-level implications to meta-level implications using `mp`. This is why we dare say “logical massage”. See `Wellfounded_Recursion.ML`.

Elementary Equivalences

For example $\neg\forall x.\phi = \exists x.\neg\phi$ or $\neg\neg\phi = \phi$, which hold because our reasoning is classical.

$\neg\exists w.(w, v) \in r^+$ in Detail

In the proof of $\exists x.\forall y.(y, x) \notin r^+$ we had the sub-proof

$$\frac{\neg\phi \quad \forall w.(w, v) \in r^+ \rightarrow \phi}{\neg\exists w.(w, v) \in r^+}$$

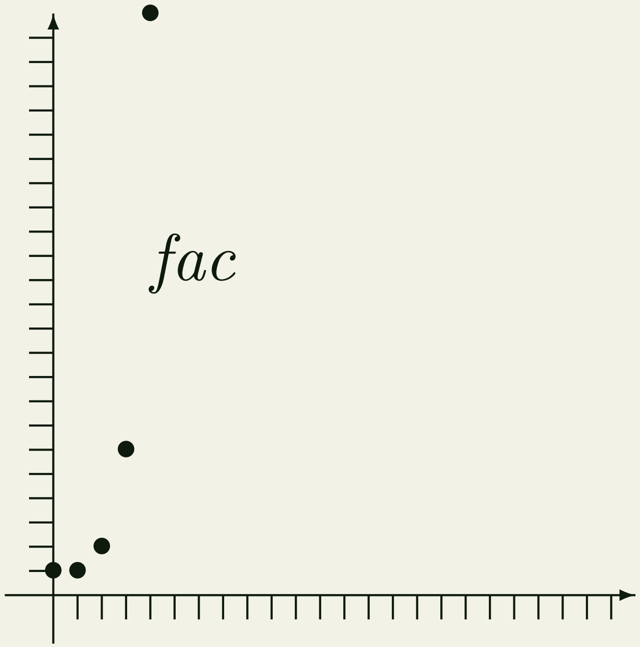
This sub-proof does not actually depend on ϕ, it would hold no matter what ϕ is (unlike the entire proof)

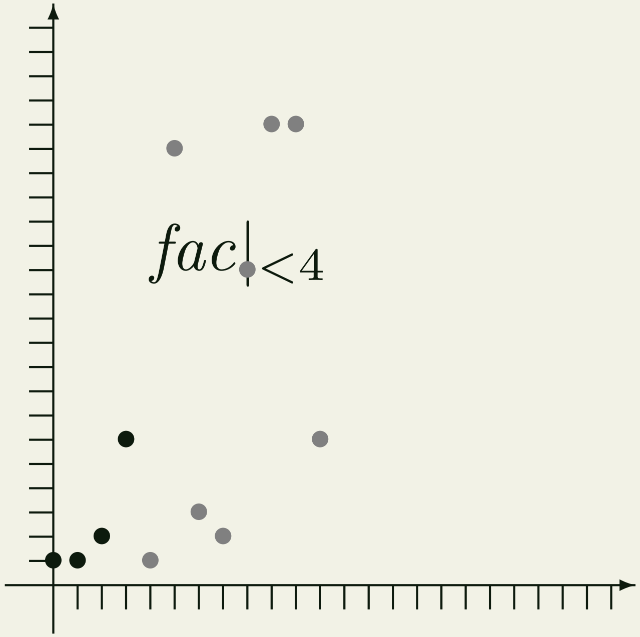
In detail, the sub-proof looks as follows:

$$\cfrac{\cfrac{\neg\phi \qquad \cfrac{[\exists w.(w,v)\in r^{+}]^{1} \qquad \cfrac{[(w,v)\in r^{+}]^{2} \qquad \cfrac{\forall w.(w,v)\in r^{+}\longrightarrow\phi}{(w,v)\in r^{+}\longrightarrow\phi}\,\mathit{spec}}{\phi}\,\mathit{mp}}{\phi}\,\mathit{exE}^{2}}{\mathit{False}}\,\mathit{notE}}{\neg\exists w.(w,v)\in r^{+}}\,\mathit{notI}^{1}$$

Appoximating Functions by cut?

For the construction we have in mind, it would be fine that $f|_{<a}$ be a function that is like f on all values $< a$, and arbitrary elsewhere. E.g., $fac|_{<4}$ could be





However, such a $fac|_{<4}$ could not be in a model for HOL. Since arbitrary is an uninterpreted constant declared in HOL.thy, it turns out that in any model and for each type, there must be one specific element in the semantic domain for it. Since the value of $fac|_{<4}$ is "arbitrary" for all arguments ≥ 4, this means that in each model, this value must be the same for all arguments ≥ 4.

Relation is a Function

When we say that a binary relation $r : \tau \times \sigma$ is in fact a function, we mean that for $t : \tau$, there is exactly one $s : \sigma$ such that $(t, s) \in r$.

Define Addition and Subtraction

```
add :: [nat, nat] ⇒ nat                    ( infixl 70)
m add n ≡wfrec (pred_nat^+)
               (λ f j. if j=0 then n
                       else Suc(f(pred j))) m
```

Here we suppose that we have a predecessor function `pred` (which can be defined using the Hilbert-operator).

Note that `add` is a function of type $nat \to nat \to nat$ (written infix), but it is only recursive in one argument, namely the first one.

You may be confused about this and wonder: how do I know that it is the first? Is this some Isabelle mechanism saying that it is always the first? The answer is: no. You must look at the two sides in isolation. On the right-hand side, we have

```
wfrec (pred_nat^+)
      (λ f j. if j=0 then n else Suc(f(pred j)))
```

By the definitions (of *wfrec* most importantly), this expression is a function of type $nat \to nat$, namely the function that adds n (which is not known looking at this expression alone; it occurs on the left-hand side) to its argument. The function is recursive in its argument (and hence not in n). Now, this function is applied to m. Therefore we say that the final function `add` is recursive in m but not in n.

Now look at subtraction:

```
  subtract :: [nat, nat] ⇒ nat                 ( infixl 70)
  m subtract n ≡wfrec (pred_nat^+)
            (λf j. if j=0 then m else pred (f (pred j))) n
```

Note that `subtract` is recursive in its second argument, simply because the right-hand side of the defining equation was constructed in a different way that for `add`.

Similar considerations apply for other binary functions defined by recursion in one argument.

Primitive Recursion

A function is primitive recursive if the recursion is based on the immediate predecessor w.r.t. the well-founded order used (e.g., the predecessor on the natural numbers, as opposed to any arbitrary smaller numbers).

This is not the same concept as used in the context of computation theory, where primitive recursive is in contrast to μ-recursive [LP81].

Automated Support of Recursive Functions

The **primrec** syntax provides a convenient front-end for defining primitive recursive functions.

Isabelle will guess a well-founded ordering to use. E.g. for functions on the natural numbers, it will use the usual $<$ ordering. The ordering is limited, but the proof will be automatic.

recdef statement is more general and requires a mesure-function (involving a proof of well-foundedness potentially requiring user interaction). Example:

```
consts posDivAlg :: "int*int => int*int"
recdef posDivAlg "inv_image less_than (λ(a,b). nat(a − b + 1))"
 "posDivAlg (a,b) = (if (a<b | b≤ 0) then (0,a)
                      else adjust b (posDivAlg(a, 2*b)))"
```

References

[LP81] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.