Proof construction with Yarrow

Preliminaries

This tutorial is intended for people who are familiar with Pure Type Systems, but not necessarily with proof-assistants. Start up Yarrow, and type `option +p`. This means Yarrow shows the proofterm that is under construction, so we can see what happens. Note that Yarrow has an extensive help-system. E.g. if the syntax we use below is not clear, type `help syntax`.

EXAMPLE 1

We construct a proof of $\forall P,Q,R:*.\ (P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow (P \Rightarrow R)$. The command `prove` starts a new proof-task, with as parameters the proposition we want to prove and its name.

```
> prove example1 : @P,Q,R:*. (P->Q->R) -> (P->Q) -> (P->R)
Proofterm = ?1

---------------------------------------------------
?1 : @P,Q,R:*. (P->Q->R)->(P->Q)->P->R
```

">" is the prompt of Yarrow when in main-mode (no proof-tasks). We have started a proof-task, the prompt has changed to "$". Yarrow indicates that the proof is a completely unknown term `?1`, that should have the given type.
The command `show` shows the current goal(s).

```
$ show
Proofterm = ?1

---------------------------------------------------
?1 : @P,Q,R:*. (P->Q->R)->(P->Q)->P->R
```

Using the command `intro`, the context is extended with a variable `P` of type `*`.

```
$ intro
?1 := \P:*. ?2
Proofterm = \P:*. ?2

P : *
---------------------------------------------------
?2 : @Q,R:*. (P->Q->R)->(P->Q)->P->R
```

Yarrow shows that for `?1` the term `\P:*. ?2` is substituted, and also shows the complete proofterm. Above the dashed line `-----------` is the local context: all variables (hypotheses) we may use, so for `?2` we may use the variable `P`.
As long as the goal belongs to a Π-type, we can repeat the command `intro`.

```
$ intro
?2 := \Q:*. ?3
Proofterm = \P,Q:*. ?3

P : *
Q : *
---------------------------------------------------
?3 : @R:*. (P->Q->R)->(P->Q)->P->R
```

The command `intros` repeats `intro` as often as possible.

```
$ intros
?3 := \R:*.\H:P->Q->R.\H1:P->Q.\H2:P. ?7
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. ?7

P : *
Q : *
R : *
H : P->Q->R
H1 : P->Q
H2 : P
--------------------------------------------------
?7 : R
```

An ordinary mathematician would proceed as follows:

1. First conclude `Q` from `H2:P` and `H1:P->Q`
2. Then conclude `R` from `Q`, `H2:P` and `H:P->Q->R`

In Yarrow we work in the other way around (*goal-directed*). First we use `H`, with the command `apply H`.

```
$ apply H
?7 := H ?11 ?9
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. H ?11 ?9

2 goals

P : *
Q : *
R : *
H : P->Q->R
H1 : P->Q
H2 : P
--------------------------------------------------
?11 : P

2) ?9 : Q
```

We see the proofterm for `R` is of the form `(H ?11 ?9)`, where `?11 : P` and `?9 : Q`.
Only the local context of the *first* goal is printed. By typing `show 2` we see the local context of the second goal (which is, in this case, the same as the local context of the first goal).

```
$ show 2
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. H ?11 ?9

2 goals

P : *
Q : *
R : *
H : P->Q->R
H1 : P->Q
H2 : P
--------------------------------------------------
?9 : Q

1) ?11 : P
```

Each command will be applied to the first goal. The second goal can be selected with the command `focus 2`.
We have an inhabitant of `P`, viz. `H2`.

```
$ exact H2
?11 := H2
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. H H2 ?9

P : *
Q : *
R : *
H : P->Q->R
H1 : P->Q
H2 : P
--------------------------------------------------
?9 : Q
```

Goal `P` has now been proved, and only goal `Q` remains. `Q` follows from `H1` and `H2`.

```
$ apply H1
?9 := H1 ?13
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. H H2 (H1 ?13)

P : *
Q : *
R : *
H : P->Q->R
H1 : P->Q
H2 : P
--------------------------------------------------
?13 : P

$ exact H2
?13 := H2
Proofterm = \P,Q,R:*.\H:P->Q->R.\H1:P->Q.\H2:P. H H2 (H1 H2)

Goal proved!
```

Now we have proved our theorem.
With `exit` we end the proof-task.

```
$ exit
Prove example1 : @P,Q,R:*. (P->Q->R)->(P->Q)->P->R
Intro
Intro
Intros
Apply H
Exact H2
Apply H1
Exact H2
Exit

example1 := .. : @P,Q,R:*. (P->Q->R)->(P->Q)->P->R
>
```

The proof is stored in the context as definition of `example1`. A summary of the proof is given, for your convenience. Yarrow is insensitive to the case of the letters in a command (e.g. you can type `iNtRoS` instead of `intros`, but Yarrow prints them in a standard form.

EXAMPLE 2

It is possible to extend the context with declarations and definitions:

```
> var nat : *
nat : *
> var zero : nat
zero : nat
> var succ : nat->nat
succ : nat->nat
> var IS : @A:*. A->A->*
IS : @A:*. A->A->*
> var refl : @A:*. @z:A. IS A z z
refl : @A:*.@z:A. IS A z z
> def two := succ (succ zero) : nat
two := .. : nat
> def f := \x:nat. succ (succ x) : nat->nat
f := .. : nat->nat
> prove example2 : IS nat (f zero) two
Proofterm = ?1

--------------------------------------------------
?1 : IS nat (f zero) two

$ apply refl
?1 := refl nat (f zero)
Proofterm = refl nat (f zero)

Goal proved!
```

Yarrow sees that `(IS nat (f zero) two)` is an instantiation of `(IS A z z)`. For `A` he substitutes `nat`. For `z` he substitutes `f zero` (this is β-equal to `two`).
With the command `abort` a proof-task is stopped, and the proof is discarded.

```
$ abort
Prove example2 : IS nat (f zero) two
Apply refl
Abort

Goal not proved
>
```

EXAMPLE 3

```
> def false := @P:*. P
false := .. : *
> def not := \P:*. P->false
not := .. : *->*
> var classic : @P:*. not (not P) -> P
classic : @P:*. not (not P) -> P
```

Under this assumption we can prove that $\forall P, Q : *. (\neg P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow P)$

```
> prove example3 : @P,Q:*. (not P -> Q) -> (not Q -> P)
Proofterm = ?1
```

```
-------------------------------------------------
@P,Q:*. (not P -> Q) -> not Q -> P

$ intros
?1 := \P,Q:*.\H:not P -> Q.\H1:not Q. ?5
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. ?5

P : *
Q : *
H : not P -> Q
H1 : not Q
-------------------------------------------------
P

$ apply classic
?5 := classic P ?7
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P ?7

P : *
Q : *
H : not P -> Q
H1 : not Q
-------------------------------------------------
not (not P)
```

`(not (not P))` is the same as `(P->false)->false`. This is an arrow-type, so we can introduce a new hypothesis with `intro`.

```
> intro
Not an @-type or definition (error)
```

Yarrow doesn't see that `(not (not P))` is an arrow-type. We have to unfold the definition first. With `unfold 1 not` the first occurrence of `not` is unfolded.

```
$ unfold 1 not
?7 := ?9
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P ?9

P : *
Q : *
H : not P -> Q
H1 : not Q
-------------------------------------------------
not P -> false

$ intro
?9 := \H2:not P. ?10
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P (\H2:not P. ?10)

P : *
Q : *
H : not P -> Q
H1 : not Q
H2 : not P
-------------------------------------------------
false
```

The ordinary way to proceed is:

1. first conclude `Q` from `H:(not P)->Q` and `H2:(not P)`
2. then derive a contradiction from `Q` and `H0:(not Q)`.

Again, we work the other way around in Yarrow. First we use `H1`, and only later `H` and `H2`. `(not Q)`, the type of `H1`, is the same as `Q->false`:

```
$ unfold not in H1
?10 := ?11
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P (\H2:not P. ?11)

P : *
Q : *
H : not P -> Q
H1 : Q->false
H2 : not P
--------------------------------------------------
false

$ apply H1
?11 := H1 ?13
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P (\H2:not P. H1 ?13)

P : *
Q : *
H : not P -> Q
H1 : Q->false
H2 : not P
--------------------------------------------------
Q
```

It isn't necessary to unfold the definition of not; we could have done `apply H1` right away. We prove `Q` with `H` and `H2`.

```
$ apply H
?13 := H ?15
Proofterm= \P,Q:*.\H:not P -> Q.\H1:not Q. classic P (\H2:not P. H1 (H ?15))

P : *
Q : *
H : not P -> Q
H1 : Q->false
H2 : not P
--------------------------------------------------
not P
```

There is already a variable with type `(not P)` in the context, viz. `H2`. So we use the command `exact`.

```
$ exact H2
?15 := H2
Proofterm= \P,Q:*.\H:not P -> Q.\H1:not Q. classic P (\H2:not P. H1 (H H2))

Goal proved!
```

With the command `restart` we throw away the proof we have given so far, and start the proof from scratch.

```
$ restart
Proofterm = ?1

--------------------------------------------------
@P,Q:*. (not P -> Q) -> not Q -> P

$ intros
?1 := \P,Q:*.\H:not P -> Q.\H1:not Q. ?5
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. ?5

P : *
Q : *
H : not P -> Q
H1 : not Q
--------------------------------------------------
P

$ apply classic
?5 := classic P ?7
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. classic P ?7

P : *
Q : *
H : not P -> Q
H1 : not Q
--------------------------------------------------
not (not P)
```

Using `undo` we can retrace the last step of the current goal:

```
$ undo
Proofterm = \P,Q:*.\H:not P -> Q.\H1:not Q. ?5

P : *
Q : *
H : not P -> Q
H1 : not Q
--------------------------------------------------
P

prove example3 : @P,Q:*. (not P -> Q) -> not Q -> P
intros
abort

Goal not proved
>
```

EXAMPLE 4

We prove $\forall A : *.\ \forall P, Q : A \to *.\ (\forall x : A.\ Px \Rightarrow Qx) \Rightarrow (\forall y : A.\ Py) \Rightarrow (\forall z : A.\ Qz)$.

```
> prove example4 : @A:*. @P,Q:A->*. (@x:A. P x -> Q x) -> (@y:A. P y) ->
.                                        (@z:A. Q z)
```

```
Proofterm = ?1

--------------------------------------------------
?1 : @A:*.@P,Q:A->*. (@x:A. P x -> Q x)->(@y:A. P y)->(@z:A. Q z)
$ intros
?1 := \A:*.\P,Q:A->*.\H:@x:A. P x -> Q x.\H1:@y:A. P y.\z:A. ?7
Proofterm = \A:*.\P,Q:A->*.\H:@x:A. P x -> Q x.\H1:@y:A. P y.\z:A. ?7

A : *
P : A->*
Q : A->*
H : @x:A. P x -> Q x
H1 : @y:A. P y
z : A
--------------------------------------------------
?7 : Q z
```

The ordinary way to proceed is:

1. first conclude `P z` from `H1 :  (@y:A. P y)` and `z:A`
2. then conclude `Q z` from `P z` and `H : (@x:A. P x -> Q x)`

Again, we work the other way around. First use `H`, with the command `apply H`.

```
$ apply H
?7 := H z ?9
Proofterm = \A:*.\P,Q:A->*.\H:@x:A. P x -> Q x.\H1:@y:A. P y.\z:A. H z ?9

A : *
P : A->*
Q : A->*
H : @x:A. P x -> Q x
H1 : @y:A. P y
z : A
--------------------------------------------------
?9 : P z
```

Yarrow determined itself that `H` has to have `z` as first argument. We prove `P z` with `H1`.

```
$ apply H1
?9 := H1 z
Proofterm = \A:*.\P,Q:A->*.\H:@x:A. P x -> Q x.\H1:@y:A. P y.\z:A. H z (H1 z)

Goal proved!
```

Again, `H1` has got argument `z` automatically.

```
$ exit
Prove example4 : @A:*.@P,Q:A->*. (@x:A. P x -> Q x)->(@y:A. P y)->(@z:A. Q z)
Intros
Apply H
Apply H1
Exit

example4 := .. : @A:*.@P,Q:A->*. (@x:A. P x -> Q x)->(@y:A. P y)->(@z:A. Q z)
>
```