Types depending on terms: the system λP ← "predicate"

General form: $\lambda x:A.M$ where M is a type (type constructor)

Application in logic: sets and propositions

(1) Sets: Let S_n be a set for all $n:nat$. (may think of sets as types)
Then $\lambda n:nat.S_n$ is a type (type constructor) depending on the term n (set-valued function)
$\lambda n:nat.S_n$ maps n to a set S_n
Other interpretations: $\lambda n:nat.S_n$ is a "family of types" or an "indexed type"

Examples
a) If $S_n = \{0, n, 2n, 3n, \dots\}$, then $(\lambda n:nat.S_n)\,0 \to_\beta \{0\}$, $(\lambda n:nat.S_n)\,1 \to_\beta \{1,2,3,\dots\}$, ...
What is the type of $\lambda n:nat.S_n$?
Should be $nat \to *$ $(n:nat,\ S_n:*)$
b) Let $V_n := \{(v_1,\dots,v_n) \mid v_i \in \mathbb{N}\}$ ... set of all sequences of natural numbers of length n
Then $\lambda n:nat.V_n : nat \to *$

(2) Prop: Let P_n be a proposition for all $n:nat$ (i.e. P_n is a type by PAT)
Then $\lambda n:nat.P_n$ is again a type (proposition-valued function)
$\lambda n:nat.S_n$ maps every n to a proposition depending on n, i.e. $\lambda n:nat.P_n$ is a predicate
Example: P_n ... "n is a prime number"
Then $\lambda n:nat.P_n$ is the predicate "to be a prime number"
$\lambda n:nat.P_n : nat \to *$
$(\lambda n:nat.P_n)\,3 \to_\beta$ true
$(\lambda n:nat.P_n)\,4 \to_\beta$ false

Derivation rules of λP

cf. derivation rules of $\lambda\omega$

(axiom) $\vdash *:\square$ — (axiom) id.

(var) $\dfrac{\Gamma \vdash A:s}{\Gamma, x:A \vdash x:A}$ if $x \notin \Gamma$, $s \in \{*,\square\}$ — (var) id.

(weak) $\dfrac{\Gamma \vdash A:B \quad \Gamma \vdash C:s}{\Gamma, x:C \vdash A:B}$ — (weak) id.

(form) $\dfrac{\Gamma \vdash A:* \quad \Gamma, x:A \vdash B:s}{\Gamma \vdash \Pi x:A.B:s}$ — (form) $\dfrac{\Gamma \vdash A:s \quad \Gamma \vdash B:s}{\Gamma \vdash A \to B:s}$

(appl) $\dfrac{\Gamma \vdash M:\Pi x:A.B \quad \Gamma \vdash N:A}{\Gamma \vdash MN:B[x:=N]}$ — (appl) $\dfrac{\Gamma \vdash M:A \to B \quad \Gamma \vdash N:A}{\Gamma \vdash MN:B}$

(abst) $\dfrac{\Gamma, x:A \vdash M:B \quad \Gamma \vdash \Pi x:A.B:s}{\Gamma \vdash \lambda x:A.M:\Pi x:A.B}$ — (abst) $\dfrac{\Gamma, x:A \vdash M:B \quad \Gamma \vdash A \to B:s}{\Gamma \vdash \lambda x:A.M:A \to B}$

(conv) $\dfrac{\Gamma \vdash A:B \quad \Gamma \vdash B':s}{\Gamma \vdash A:B'}$ if $B =_\beta B'$ — (conv) id.

Differences between λP and $\lambda\omega$:
(i) upgrade of $\to$-types to Π-types in (form), (appl), (abst)
(ii) downgrade of implicit types in (form) (only $A:*$ instead of $A:s$, so x must be a term), no $\Pi C:*.C \to C$ in λP
(iii) extend context of second premiss in (form) and choose proper output type in (appl) ($B[x:=N]$)

Double rule of (form): $s = * \Rightarrow A:*, B:*, \text{ or } \Pi x:A.B:*$ (example: $\Pi n:nat.nat:*$)

$s = \square \Rightarrow A:*, B:\square, \text{ or } \Pi x:A.B:\square$ (example: $\Pi n:nat.*:\square$)

inhabitants: $\lambda n:nat.1(1n) : \Pi n:nat.nat$

$\lambda n:nat.S_n : \Pi n:nat.*$ or $\lambda n:nat.P_n : \Pi n:nat.*$

Notation: Non-$\rightarrow$-types in λP, but nice notation $A \rightarrow B$ for a product type $\Pi x:A.B$ if x does not occur free in B

λP has all "nice properties" of $\lambda\underline{w}$

Remark: The formation rule is also called "product rule" and a Π-type is considered a Cartesian product of a family of types. If A is a finite type consisting of two elements a_1 and a_2, then $\Pi x:A.B = B[x:=a_1] \times B[x:=a_2]$. Π-types are thus a generalization of the Cartesian product and of the space of functions ($x \notin FV(B)$, then $\Pi x:A.B$ is just $A \rightarrow B$)

Example derivation

(1) $*:\square$ (axiom)

$A:*$

(2) $A:*$ (var) on (1) ($s=\square$)

(3) $*:\square$ (weak) on (1), (1) ($s=\square$)

$x:A$

(4) $*:\square$ (weak) on (3), (2) ($s=*$) ($s=\square$)

(5) $A \rightarrow *:\square$ (form) on (2), (4) (not derivable in $\lambda\underline{w}$)

$= \Pi x:A.*$

$A \rightarrow *$ is a kind depending on a term and $P:A \rightarrow *$ is a type depending on a term

(assume inhabitant of new type $A \rightarrow *$)

$P:A \rightarrow *$

(6) $P:A \rightarrow *$ (var) on (5) ($s=\square$)

(7) $A:*$ (weak) on (2), (5) ($s=\square$)

(8) $*:\square$ (weak) on (3), (5) ($s=\square$)

$x:A$

(9) $x:A$ (var) on (7) ($s=*$)

(10) $P:A \rightarrow *$ (weak) on (6), (7) ($s=*$)

(11) $Px:*$ (appl) on (10), (9)

(12) $\Pi x:A.Px:*$ (form) on (7), (11) ("real" dependent type) ((12)–(18): $s=*$)

$x:A$

$y:Px$

(13) $Px:*$ (weak) on (11), (11)

(14) $Px \rightarrow Px:*$ (form) on (11), (13)

(15) $\Pi x:A.Px \rightarrow Px:*$ (form) on (7), (14)

(final inhabitant of $\Pi x:A.Px \rightarrow Px$)

$x:A$

$x:Px$

(16) $y:Px$ (var) on (11)

(17) $\lambda y:Px.y:Px \rightarrow Px$ (abst) on (16) (11)

(18) $\lambda x:A.\lambda y:Px.y : \Pi x:A.Px \rightarrow Px$ (abst) on (17), (15)

PAT: $A:*, P:A \rightarrow * \vdash \Pi x:A.Px \rightarrow Px:*$ if A is a type (set) and P is a predicate on A, then $\Pi x:A.Px \rightarrow Px$ is a valid proposition ($\forall x \in A: Px \Rightarrow Px$)

PAT: $\lambda x:A.\lambda y:Px.y$ is a proof of $\Pi x:A.Px \rightarrow Px$

Shortened derivation: axiom (ax), (var), (weak), (form), 2nd generation (abst)

(a) $A:*$

(b) $P:A \rightarrow *$

(c) $x:A$

(19) $Px:*$ (appl) on (b), (c)

(d) $y:Px$

(16) $y:Px$ (var) on (19)

(17) $\lambda y:Px.y:Px \rightarrow Px$ (abst) on (16)

(18) $\lambda x:A.\lambda y:Px.y : \Pi x:A.Px \rightarrow Px$ (abst) on (17)