MRI masterclass Type Theory & Proof Assistants
Spring 2007
Nijmegen NL

Herman Geuvers

Chapter 11 (S & U): Second Order Logic and Polymorphism

Why Polymorphic λ-calculus?

- Simple type theory $\lambda\!\rightarrow$ is not very expressive
- In simple type theory, we can not 'reuse' a function.
  E.g. $\lambda x{:}\alpha.x : \alpha\rightarrow\alpha$ and $\lambda x{:}\beta.x : \beta\rightarrow\beta$.

We want to define functions that can treat types polymorphically: add types $\forall\alpha.\sigma$:
Examples

- $\forall\alpha.\alpha\rightarrow\alpha$
  If $M : \forall\alpha.\alpha\rightarrow\alpha$, then M can map any type to itself.
- $\forall\alpha.\forall\beta.\alpha\rightarrow\beta\rightarrow\alpha$
  If $M : \forall\alpha.\forall\beta.\alpha\rightarrow\beta\rightarrow\alpha$, then M can take two inputs (of arbitrary types) and return a value of the first input type.

Derivation rules for Weak (ML-style) polymorphism,
Typ : add $\forall\alpha_1.\ldots\forall\alpha_n.\sigma$ for σ a $\lambda\!\rightarrow$-type.

1. Curry style:

$$\frac{\Gamma\vdash M:\sigma}{\Gamma\vdash M:\forall\alpha.\sigma}\ \alpha\notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma\vdash M:\forall\alpha.\sigma}{\Gamma\vdash M:\sigma[\tau/\alpha]}\ \text{for } \tau \text{ a } \lambda\!\rightarrow\text{-type}$$

2. Church style:

$$\frac{\Gamma\vdash M:\sigma}{\Gamma\vdash \lambda\alpha.M:\forall\alpha.\sigma}\ \alpha\notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma\vdash M:\forall\alpha.\sigma}{\Gamma\vdash M\tau:\sigma[\tau/\alpha]}\ \text{for } \tau \text{ a } \lambda\!\rightarrow\text{-type}$$

- $\forall$ only occurs on the outside and is therefore often left out: "all type variables are implicitly universally quantified"
- With weak polymorphism, type checking is still decidable: the principal types algorithm still works.

Derivation rules for Weak (ML-style) polymorphism,
Typ : add $\forall\alpha_1.\ldots\forall\alpha_n.\sigma$ for σ a $\lambda\!\rightarrow$-type.
NB: Also the abstraction rule is restricted to $\lambda\!\rightarrow$-types:

1. Curry style: $\dfrac{\Gamma, x:\tau\vdash M:\sigma}{\Gamma\vdash \lambda x.M:\tau\rightarrow\sigma}\ \tau$ a $\lambda\!\rightarrow$-type
2. Church style: $\dfrac{\Gamma, x:\tau\vdash M:\sigma}{\Gamma\vdash \lambda x{:}\tau.M:\tau\rightarrow\sigma}\ \tau$ a $\lambda\!\rightarrow$-type

Derivation rules for Weak (ML-style) polymorphism,
Typ : add $\forall\alpha_1.\ldots\forall\alpha_n.\sigma$ for σ a $\lambda\!\!\rightarrow$-type.

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ a } \lambda\!\!\rightarrow\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M\tau : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ a } \lambda\!\!\rightarrow\text{-type}$$

Examples:

- $\lambda 2$ à la Curry: $\lambda x.\lambda y.x : \forall\alpha.\forall\beta.\alpha\rightarrow\beta\rightarrow\alpha$.
- $\lambda 2$ à la Church: $\lambda\alpha.\lambda\beta.\lambda x{:}\alpha.\lambda y{:}\beta.x : \forall\alpha.\forall\beta.\alpha\rightarrow\beta\rightarrow\alpha$.

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ}\rightarrow\mathsf{Typ}) \mid \forall\alpha.\mathsf{Typ}$$

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M\tau : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

- $\forall$ can also occur deeper in a type.
- With full polymorphism, type checking becomes undecidable! [Wells 1993]

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ}\rightarrow\mathsf{Typ}) \mid \forall\alpha.\mathsf{Typ}$$

NB: In the abstraction rule all types are $\lambda 2$-types:

1. Curry style: $\dfrac{\Gamma, x:\tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau\rightarrow\sigma}\ \sigma, \tau$ are $\lambda 2$-types
2. Church style: $\dfrac{\Gamma, x:\tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau\rightarrow\sigma}\ \sigma, \tau$ are $\lambda 2$-types

From $\lambda 2$ à la Church to $\lambda 2$ à la Curry.
erasure map:

$$\begin{array}{rclrcl} |x| & := & x & & & \\ |\lambda x{:}\sigma.M| & := & |\lambda x.M| & |MN| & := & |M|\,|N| \\ |\lambda\alpha.M| & := & |M| & |M\sigma| & := & |M| \end{array}$$

Proposition
For M an untyped λ-term,

$$\Gamma \vdash_{\mathsf{Curry}} M : \tau \iff \exists P(\Gamma \vdash_{\mathsf{Church}} P : \tau \wedge |P| \equiv M)$$

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\text{Typ} := \text{TVar} \mid (\text{Typ}{\to}\text{Typ}) \mid \forall\alpha.\text{Typ}$$

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\ \alpha \notin \text{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.M : \forall\alpha.\sigma}\ \alpha \notin \text{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M\tau : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

Examples valid only with full polymorphism:

- $\lambda 2$ à la Curry: $\lambda x.\lambda y.x : (\forall\alpha.\alpha){\to}\sigma{\to}\tau$.
- $\lambda 2$ à la Church: $\lambda x{:}(\forall\alpha.\alpha).\lambda y{:}\sigma.x\tau : (\forall\alpha.\alpha){\to}\sigma{\to}\tau$.

Recall: Important Properties

$\Gamma \vdash M : \sigma?$	TCP
$\Gamma \vdash M : ?$	TSP
$\vdash ? : \sigma$	TIP

Properties of polymorphic λ-calculus
TIP is undecidable.

TIP is equivalent to provability in second order intuitionistic propositional logic, which is undecidable (Löb, Gabbay).

Properties of polymorphic λ-calculus
TCP and TSP are equivalent & decidable.

TCP	à la Church	à la Curry
ML-style	decidable	decidable
System F-style	decidable	undecidable [Wells]

With full polymorphism (system F), untyped terms contain too little information to compute the type. (No principal types.)

Some examples of typing in $\lambda 2$: Abbreviate $\bot := \forall\alpha.\alpha$, $\top := \forall\alpha.\alpha{\to}\alpha$.

- Curry $\lambda 2$: $\lambda x.xx : \bot{\to}\bot$
- Church $\lambda 2$: $\lambda x{:}\bot.x(\bot{\to}\bot)x : \bot{\to}\bot$.
- Church $\lambda 2$: $\lambda x{:}\bot.\lambda\alpha.x(\alpha{\to}\alpha)(x\alpha) : \bot{\to}\bot$.
- Curry $\lambda 2$: $\lambda x.xx : \top{\to}\top$
- Church $\lambda 2$: $\lambda x{:}\top.x\top x : \top{\to}\top$.
- Church $\lambda 2$: $\lambda x{:}\top.\lambda\alpha.x(\alpha{\to}\alpha)(x\alpha) : \top{\to}\top$.

Formulas-as-types for $\lambda 2$:
There is a formulas-as-types isomorphism between $\lambda 2$ and second order proposition logic, PROP2

Derivation rules of PROP2:

$$\frac{\Gamma \vdash \sigma}{\Gamma \vdash \forall \alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash \forall \alpha.\sigma}{\Gamma \vdash \sigma[\tau/\alpha]}$$

NB This is constructive second order proposition logic:

$$\forall \alpha.\forall \beta.((\alpha{\rightarrow}\beta){\rightarrow}\alpha){\rightarrow}\alpha \text{ Peirce's law}$$

is not derivable.

Definability of the other connectives:

$$\begin{aligned} \bot &:= \forall \alpha.\alpha \\ \sigma{\wedge}\tau &:= \forall \alpha.(\sigma{\rightarrow}\tau{\rightarrow}\alpha){\rightarrow}\alpha \\ \sigma{\vee}\tau &:= \forall \alpha.(\sigma{\rightarrow}\alpha){\rightarrow}(\tau{\rightarrow}\alpha){\rightarrow}\alpha \\ \exists \alpha.\sigma &:= \forall \beta.(\forall \alpha.\sigma{\rightarrow}\beta){\rightarrow}\beta \end{aligned}$$

and all the standard constructive derivation rules are derivable.

Example ($\wedge$-elimination):

$$\frac{\dfrac{\forall \alpha.(\sigma{\rightarrow}\tau{\rightarrow}\alpha){\rightarrow}\alpha}{(\sigma{\rightarrow}\tau{\rightarrow}\sigma){\rightarrow}\sigma} \qquad \dfrac{\dfrac{[\sigma]^1}{\tau{\rightarrow}\sigma}}{\sigma{\rightarrow}\tau{\rightarrow}\sigma}\,1}{\sigma}$$

Definability of connectives and derivation rules:

$$\begin{aligned} \bot &:= \forall \alpha.\alpha \\ \sigma{\wedge}\tau &:= \forall \alpha.(\sigma{\rightarrow}\tau{\rightarrow}\alpha){\rightarrow}\alpha \\ \sigma{\vee}\tau &:= \forall \alpha.(\sigma{\rightarrow}\alpha){\rightarrow}(\tau{\rightarrow}\alpha){\rightarrow}\alpha \\ \exists \alpha.\sigma &:= \forall \beta.(\forall \alpha.\sigma{\rightarrow}\beta){\rightarrow}\beta \end{aligned}$$

Example ($\wedge$-elimination) with λ-terms:

$$\frac{\dfrac{M : \forall \alpha.(\sigma{\rightarrow}\tau{\rightarrow}\alpha){\rightarrow}\alpha}{M\sigma : (\sigma{\rightarrow}\tau{\rightarrow}\sigma){\rightarrow}\sigma} \qquad \dfrac{\dfrac{[x:\sigma]^1}{\lambda y{:}\tau.x : \tau{\rightarrow}\sigma}}{\lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma{\rightarrow}\tau{\rightarrow}\sigma}\,1}{M\sigma(\lambda x{:}\sigma.\lambda y{:}\tau.x) : \sigma}$$

So the following term is a 'witness' for the $\wedge$-elimination.

$$\lambda z{:}\sigma{\wedge}\tau.z\sigma(\lambda x{:}\sigma.\lambda y{:}\tau.x) : (\sigma \wedge \tau){\rightarrow}\sigma$$

Data types in $\lambda 2$

$$\mathsf{Nat} := \forall \alpha.\alpha{\rightarrow}(\alpha{\rightarrow}\alpha){\rightarrow}\alpha$$

This type can be used as the type of natural numbers, using the encoding of $\mathbb{N}$ as Church numerals in the λ-calculus.

$$n \mapsto \lambda x.\lambda f.f(\ldots(fx)) \quad n\text{-times } f$$

- $0 := \lambda \alpha.\lambda x{:}\alpha.\lambda f{:}\alpha{\rightarrow}\alpha.x$
- $S := \lambda n{:}\mathsf{Nat}.\lambda \alpha.\lambda x{:}\alpha.\lambda f{:}\alpha{\rightarrow}\alpha.f(n\alpha x f)$
- Iteration: if $c : \sigma$ and $g : \sigma{\rightarrow}\sigma$, then $\mathsf{It}\, c\, g : \mathsf{Nat}{\rightarrow}\sigma$ is defined as
  $$\lambda n{:}\mathsf{Nat}.n\,\sigma\,c\,g$$
  Then
  $$\mathsf{It}\, c\, g\, n = g(\ldots(gc)) \ \ (n \text{ times } g)$$

Examples:

- Addition

$$\text{Plus} := \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.\text{It}\, m\, S\, n$$

or $\text{Plus} := \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.n\, \text{Nat}\, m\, S$

- Multiplication

$$\text{Mult} := \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.\text{It}\, 0\, (\lambda x{:}\text{Nat}.\text{Plus}\, m\, x)\, n$$

- Predecessor is difficult!
  This requires defining primitive recursion in terms of iteration.
  As a consequence:

$$\text{Pred}(n+1) \twoheadrightarrow_\beta n$$

in a number of steps of $O(n)$.

Data types in $\lambda 2$ ctd.

$$\text{List}_A := \forall \alpha.\alpha \to (A \to \alpha \to \alpha) \to \alpha$$

represents the type of lists over the type A, using the following encoding of lists in the untyped λ-calculus.

$$[a_1, a_2, \ldots, a_n] \mapsto \lambda x.\lambda f.f\, a_1(f\, a_2(\ldots(f\, a_n x))) \quad n\text{-times } f$$

- $\text{Nil} := \lambda \alpha.\lambda x{:}\alpha.\lambda f{:}A \to \alpha \to \alpha.x$
- $\text{Cons} := \lambda a{:}A.\lambda l{:}\text{List}_A.\lambda \alpha.\lambda x{:}\alpha.\lambda f{:}A \to \alpha \to \alpha.f\, a(l\, \alpha\, x\, f)$
- Iteration: if $c : \sigma$ and $g : A \to \sigma \to \sigma$, then $\text{It}\, c\, g : \text{List}_A \to \sigma$ is defined as

$$\lambda l{:}\text{List}_A.l\, \sigma\, c\, g$$

Then, for $l = [a_1, a_2, \ldots, a_n]$,

$$\text{It}\, c\, g\, l = g\, a_1(g\, a_2(\ldots(g\, a_n\, c))) \quad (n \text{ times } g)$$

Example:

- Map, given $f : \sigma \to \tau$, $\text{Map}\, f : \text{List}_\sigma \to \text{List}_\tau$ applies f to all elements in a list.

$$\text{Map} := \lambda f{:}\sigma \to \tau.\text{It Nil}(\lambda x{:}\sigma.\lambda l{:}\text{List}_\tau.\text{Cons}(f\, x)l).$$

Many data-types can be defined in $\lambda 2$:

- Product of two data-types: $\sigma \times \tau := \forall \alpha.(\sigma \to \tau \to \alpha) \to \alpha$
- Sum of two data-types: $\sigma + \tau := \forall \alpha.(\sigma \to \alpha) \to (\tau \to \alpha) \to \alpha$
- Unit type: $\text{Unit} := \forall \alpha.\alpha \to \alpha$
- Binary trees with nodes in A and leaves in B:
  $\text{Tree}_{A,B} := \forall \alpha.(B \to \alpha) \to (A \to \alpha \to \alpha) \to \alpha$

One can also do primitive recursion (à la system T) in $\lambda 2$:
For $g : \sigma \to \text{Nat} \to \sigma$ and $d : \sigma$,

$\text{Rec}\, dg : \text{Nat} \to \sigma$
$\text{Rec}\, dg : \lambda n{:}\text{Nat}.\pi_1(n(\sigma \times \text{Nat})(\lambda v{:}\sigma \times \text{Nat}.\langle g(\pi_1\, v)(\pi_2\, v), S(\pi_2\, v)\rangle)\langle d, 0\rangle).$

where S is the successor and 0 is the zero of Nat and π_1 and π_2 are the projections.
Then

$$\begin{aligned} \text{Rec}\, dg0 &=_\beta c \\ \text{Rec}\, dg(S\, n) &=_\beta g(\text{Rec}\, dgn)n \end{aligned}$$

Properties of $\lambda 2$.

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$.
- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash N : \sigma$.
- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all $\beta\eta$-reductions from M terminate.

Subject Reduction also holds for η-reduction in $\lambda 2$ à la Church, but not in $\lambda 2$ à la Curry !

$$x : \alpha{\to}\forall\beta(\beta{\to}\beta) \vdash \lambda y.xy : \alpha{\to}\beta{\to}\beta$$

but not

$$x : \alpha{\to}\forall\beta(\beta{\to}\beta) \vdash x : \alpha{\to}\beta{\to}\beta$$

In $\lambda 2$ à la Church:

$$x : \alpha{\to}\forall\beta(\beta{\to}\beta) \vdash \lambda y{:}\alpha.xy\beta : \alpha{\to}\beta{\to}\beta$$

which is not an η-redex.

Strong Normalization of β for $\lambda 2$.
Note:

- There are two kinds of β-reductions
  - $(\lambda x{:}\sigma.M)P \longrightarrow_\beta M[P/x]$
  - $(\lambda\alpha.M)\tau \longrightarrow_\beta M[\tau/\alpha]$
- The second doesn't do any harm, so we can just look at $\lambda 2$ à la Curry

Recall the proof of Strong Normalization for $\lambda{\to}$ à la Curry
This is proved by constructing a model of $\lambda{\to}$.
Definition

- $[\![\alpha]\!] :=$ SN (the set of strongly normalizing λ-terms).
- $[\![\sigma{\to}\tau]\!] := \{M \mid \forall N \in [\![\sigma]\!](MN \in [\![\tau]\!])\}$.

Lemma (both by induction on σ)

- $[\![\sigma]\!] \subseteq$ SN
- If $M[N/x]\vec{P} \in [\![\tau]\!]$, $N \in [\![\sigma]\!]$, then $(\lambda x.M)N\vec{P} \in [\![\tau]\!]$.

Proposition

$$\left.\begin{array}{l} x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash M : \sigma \\ N_1 \in [\![\tau_1]\!], \ldots, N_n \in [\![\tau_n]\!] \end{array}\right\} \Rightarrow M[N_1/x_1, \ldots N_n/x_n] \in [\![\sigma]\!]$$

Proof By induction on the derivation of $\Gamma \vdash M : \sigma$.

Proposition

$$\left.\begin{array}{l} x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash M : \sigma \\ N_1 \in [\![\tau_1]\!], \ldots, N_n \in [\![\tau_n]\!] \end{array}\right\} \Rightarrow M[N_1/x_1, \ldots N_n/x_n] \in [\![\sigma]\!]$$

Corollary $\lambda\!\rightarrow$ is SN

Proof By taking $N_i := x_i$ in the Proposition.

Crucial part of the proof for $\lambda\!\rightarrow$:

- $[\![\alpha]\!] :=$ SN.
- $[\![\sigma\!\rightarrow\!\tau]\!] := \{M \mid \forall N \in [\![\sigma]\!](MN \in [\![\tau]\!])\}$.

Question:
How to define $[\![\forall\alpha.\sigma]\!]$??

$$[\![\forall\alpha.\sigma]\!] := \Pi_{X\in U}[\![\sigma]\!]_{\alpha:=X}??$$

Strong Normalization of β for $\lambda 2$.

Question:
How to define $[\![\forall\alpha.\sigma]\!]$??

$$[\![\forall\alpha.\sigma]\!] := \Pi_{X\in U}[\![\sigma]\!]_{\alpha:=X}??$$

- What is U?
  The collection of all 'possible' interpretations of types (?)
- $\Pi_{X\in U}[\![\sigma]\!]_{\alpha:=X}$ may get very (too?) big.

Girard:

- $[\![\forall\alpha.\sigma]\!]$ should be small

$$\bigcap_{X\in U}[\![\sigma]\!]_{\alpha:=X}$$

- Characterization of U.

U := SAT, the collection of saturated sets of (untyped) λ-terms.
$X \subset \Lambda$ is saturated if

- $xP_1 \ldots P_n \in X$ (for all $x \in$ Var, $P_1, \ldots, P_n \in$ SN)
- $X \subseteq$ SN
- If $M[N/x]\vec{P} \in X$ and $N \in$ SN, then $(\lambda x.M)N\vec{P} \in X$.

Let ρ : TVar $\longrightarrow$ SAT be a valuation of type variables.
Define the interpretation of types $[\![\sigma]\!]_\rho$ as follows.

- $[\![\alpha]\!]_\rho := \rho(\alpha)$
- $[\![\sigma\to\tau]\!]_\rho := \{M | \forall N \in [\![\sigma]\!]_\rho (MN \in [\![\tau]\!]_\rho)\}$
- $[\![\forall\alpha.\sigma]\!]_\rho := \cap_{X\in\mathsf{SAT}}[\![\sigma]\!]_{\rho,\alpha:=X}$

Proposition

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash M : \sigma \Rightarrow M[P_1/x_1, \ldots, P_n/x_n] \in [\![\sigma]\!]_\rho$$

for all valuations ρ and $P_1 \in [\![\tau_1]\!]_\rho, \ldots, P_n \in [\![\tau_n]\!]_\rho$

Proof
By induction on the derivation of $\Gamma \vdash M : \sigma$.

Corollary $\lambda 2$ is SN

(Proof: take P_1 to be x_1, ..., P_n to be x_n.)

NB

- SN for $\lambda 2$ cannot be proved in second-order arithmetic!
- SN for a fixed term (involving only finitely many types) can be proved in second-order arithmetic.

Theorem[Girard] A function f : $\mathbb{N}^k \longrightarrow \mathbb{N}$ is definable in $\lambda 2$ iff f is provably total in second-order Peano arithmetic.
Proof:
$\Rightarrow$ Using the fact that f can be proved SN in second order arithmetic.
$\Leftarrow$ By extending the Dialectica interpretation.

There are SN terms that are not typable in $\lambda 2$.
Here are two examples:

$$(\lambda zy.y(z\mathbf{I})(z\mathbf{K}))(\lambda x.xx)$$

$$c_2 c_2 \mathbf{K}$$

Exercises

1. 12.7.4
2. 12.7.5
3. 12.7.6
4. 12.7.8
5. 12.7.13
6. 12.7.18