Lambda calculus and Functional Programming, exercises ; v. 14. ledna 2018

Solve for credit:
4d
5a SIIx
6b,c2
6.1
7a,d2
13d
14a
10b
10e
11a
9c using 8

1st seminar

1. (HW) Prove:
a) $\lambda \vdash SKK = I$
b) $\lambda \vdash KI = K_*$

1.1 (HW) a) Write explicitely as lambda-abstractions (including parentheses, dots, etc.), with all parentheses, with only necessary parentheses, and reduce: a.1) KII ; a.2) $K(IK_*)I$
b) Explain: $KA \neq \lambda xy.xA$; (A lesson learned: you can't use a text or string substitution (naively)
c) Explain a difference between relations $=$ a $\equiv$; c.1) If F is defined as $C[x, y]$ and particularly $\lambda x.yx$, then which pairs of following terms are in the relation $=$ and which are in the relation $\equiv$: FI, $F[y := I]$, $C[x, I]$, $C[I, I]$, $\lambda x.Ix$, $\lambda x.x$, $(\lambda x.yx)I$?

2. Prove (by structural induction) for arbitrary λ-terms $s, t, u \in \Lambda$ and (nonequal) variables x and y:
a) if $s = t$, then $s[x := u] = t[x := u]$
b) if $s = t$, then $u[x := s] = u[x := t]$
c) $(s[x := u])[y := t] = s[y := t]([x := u[y := t]])$, if $x \notin freevar(t)$

3. Find a closed λ-term F:
a) $F \in \Lambda$, s.t. $FGHX = G(HX)(HX)$.
b) $F \in \Lambda$, s.t. $FX = XXX$.

4. Prove:
a) $\exists F\ \forall X\ FX = FF$
b) $\exists F\ \forall X\ FX = XF$
c) $\exists F\ \forall X\ FX = F$, sometimes denoted K_∞
d) $\exists F\ \forall X\ FX = SFX$ (and η-reduced expression)
e) $\exists F\ \forall X\ FX = FXX$
(q) It does not exist such F that $\forall X, Y\ F(XY) = X$
z) (Can you find a solution of a), and d) without a fixed point combinator?)
Hint: Write an equation in a form $F = (\lambda f.C[f, x])F$ and use the fixed point theorem to find F.
Note: We can reduce the λ-terms "under" λ-abstraction, but such transformations cannot be used (in functions) in classical programming languages. It can be used in optimisation and in partial evaluation.

4.1 (HW) How can be a term substituted (for example) to the second argument of a function? We want to substitute the term T for the second argument y of the function $F = \lambda xy.C[x, y]$. Write such a term Z that $ZFT = \lambda x.C[x, T] = \lambda x.(C[x, y][y := T])$
Note: Partial evaluation; a relation to lambda-lifting (and to supercombinators).

5. Simplify:
a) $SIIx$, $(SII(SII))$
(b) $SIIt$ for $t = S(Ku)(SII)$
c) $S(KK)I$

6. Prove (1st + 3rd seminar)
a) If Z is a fixed point combinator, then $Z(SI)$ is also a fixed point combinator.
b) The combinator $Z = VVVV$, where $V = \lambda helo.o(hello)$ (respectively $V = \lambda ehlo.o(hello)$), is a f.p.combinator.
b1) Question: Is it valid for Z: $ZF \rightarrow_\beta^* F(ZF)$?
c) Turing's f.p.comb. $\Theta = AA$, where $A = \lambda xy.y(xxy)$ is a f.p.combinator
c1) Is it valid that $\Theta F \rightarrow_\beta^* F(\Theta F)$?
c2) Is it valid that $Y(SI) \rightarrow_\beta^* \Theta$? (where Y is a f.p.comb. from a lecture)
c3) Decide: true or false?: $Z(SI)F \rightarrow_\beta^* F(Z(SI)F)$ (TODO/Check)
Note: (p. 79) Turing's combinator Θ is in a form of a supercombinator, but Y is not. (The argument F is placed after the reduction of ΘF as an argument.)
d) Turing's Θ, call-by-value version: $\Theta_v = A'A'$, where $A' = \lambda xy.y(\lambda z.xxyz)$ is a f.p.comb.
d2) Y, call-by-value version: $Y_v = \lambda f.A'A'$, where $A' = \lambda x.f(\lambda v((xx)v))$ is a f.p.comb.
d.3) Explain a relation of Θ, respectively Y, to Θ_v, respectively Y_v, using η-expansion.
e) Can a reduction of YF be finite (at least in some cases)? Explain and/or give an example.

2nd seminar

6.1 Find examples of λ-terms (S2.2,2.6):
A term is *strongly normalizing*, if all reduction strategies are finite. A theory is strongly normalizing, if all terms are strongly normalizing.
a) Pairs of terms, which show that the relations $\rightarrow_\beta$, $\rightarrow_\beta^*$, and $=_\beta$ are different.
b) in β-normal form
c) strongly normalizing, but not in the β-reduced form
d) normalizing, but not strongly normalizing
e) are not normalizing

7. Definition. The rule η (eta). For an arbitrary λ-term F and a variable x which does not occur free in F, the following equality is valid

$$\lambda x.Fx = F$$

It is possible to show that the rule η is consistent with the axioms of λ-calculus. A calculus, which includes the η rule is denoted $\lambda\eta$.

The rule η is called an extensionality rule because it allows to prove for any two λ-terms F and G and a variable x, which is not free in F nor in G, that a) is valid, that means from the equality $Fx = Gx$ (1) to entail the equality $F = G$. (2)
a) Show that in $\lambda\eta$-calculus the equality $F = G$ follows from $Fx = Gx$.
b) Is the opposite implication valid? `:-)` :-)

If we take F and G as functions of a single argument, then (1) says that these functions have equal values for all *arguments*. The rule η enables to derive the equality (2) of *functions* F and G.
Extensionality of functions expresses the fact that functions which have all results equal are the same.
Def. The rule *ext*: If $Nx = Mx$ for $x \notin FV(MN)$, then $M = N$. (!only for a variable x)
Def. Let T be a formal theory with formulas in a form of equations between terms. We say that T is *extensional* if

$$T \vdash ML = NL \text{ for all } L, \text{ then } T \vdash M = N$$

c) Show that the theory λ is not extensional

d) Prove:
d.1) The theory $\lambda + ext$ is extensional
d.2) The theory $\lambda\eta$ is extensional
d.3) $\lambda + ext \vdash M = N \Leftrightarrow \lambda\eta \vdash M = N$
d.4) The theories $\lambda + ext$ and $\lambda\eta$ are the smallest extensional extensions of the theory λ.
Note: The rule η (eta) enables to consider any term M for the function $\lambda x.Mx$, but this need not be desirable in some context of usage. This is the reason why the rule was not included by Church to the theory λ.
Note: The rule ξ (ksi, xi), p. 17, $M = M' \to (\lambda x.M = \lambda x.M')$ is the *weak extensionality*: if there are convertible bodies of functions, then the functions are convertible as well.
Also: the reduction η; η is strongly normalizing and is Church-Rosser (CR) (it has CR property); the reduction $\beta\eta$, it is CR, a relation to $\lambda\eta$-calculus, consistency of $\lambda\eta$, $\beta\eta$-NF $\Leftrightarrow$ β-NF (NF for Normal Form), "completeness" of $\lambda\eta$: for M, N with $\beta\eta$-NF: $M = N \vee M\#N$ (incompatibility of terms, later)
Note: Extensionality is in contrast to the concept of *intensionality*, which compares internal definitions of objects. E. g. Turing Machines are intensional.

13. Prove that for $n, m \in N$, $c_n \equiv \lambda fx.f^n(x)$:
a) for $A_+ \equiv \lambda xypq.xp(xpq)$ holds $A_+ c_n c_m = c_{n+m}$
b) for $A_* \equiv \lambda xyz.x(yz)$ holds $A_* c_n c_m = c_{n*m}$
c) for $A_{exp} \equiv \lambda xy.yx$ holds $A_{exp} c_n c_m = c_{n^m}$, except for $m = 0$
d) find a term A_{succ} which fulfills $A_{succ} c_n = c_{n+1}$ (two possibilities)
(e) find a term A_{pred} which fulfills $A_{pred} c_{n+1} = c_n$
(f) express (and reduce) functions $2 + n$, $n + 2$, $2 * n$, n^2, 2^n, tower(n,m)

14. (HW) a) Formulate and prove an analogical theorem to "Double Fixed Point Theorem" for n terms in n (mutually recursive) equations. (p. 40)
b) Formulate and prove an analogical corollary (p. 43) for n mutually recursive equations (given by n contexts).
c.1) Find combinator(s) analogical to the Turing's combinator Θ (p. 80) for the "Double F.P.Th.", which allow *β-reductions* from left side to right side.
c.2) similarly for Second F.P.Th.

3rd seminar

6. c), d) see above

6.2 Find examples of situations (or what Church-Rosser property does not imply):
a) $M =_\beta N$, $M \not\equiv N$, M has NF and has an infinite reduction sequence as well.
a.1) moreover $M \not\twoheadrightarrow_\beta N$
a.2) moreover M has an infinite reduction sequence of *different* terms.
b) $M =_\beta N$, M and N have NF, but (some) common term L is not in NF
c) $M =_\beta N$, $M \not\equiv N$, M doesn't have NF
d) M has (at least) two infinite reduction sequences of *different* terms.
z) Def.: The weak Church-Rosser property (for a relation R): if $M_1 \leftarrow^1_R M \rightarrow^1_R M_2$, then exists M_3 s.t. $M_1 \rightarrow^*_R M_3 \leftarrow^*_R M_2$.
z.1) The weak C-R property is a weaker assumption than the (strong) C-R property. They are the same, when the reduction R is always finite (i.e. for strongly normalizing theories).

10. Definition. We say that λ-terms s and t are incompatible (with $\lambda\beta$), if an addition of an axiom $s = t$ to $\lambda\beta$ creates an inconsistent theory. We denote the incompatibility of terms by $s\#t$.
It holds $s = t$ for all two λ-terms s and t in an inconsistent theory.
Show:
a) $S\#K$. Hint: Apply the both sides of the equation $S = K$ to well chosen terms p, q, and r and show

that $I = t$ for any term t.
b) $I \# K$. (Also a direct proof.)
c) $I \# S$.
d) $K \# K_*$.
e) $s \# t \leftrightarrow \lambda + (s = t) \vdash K = K_*$
(If $true \equiv K$ and $false \equiv K_*$, then the right equality on the right side means $true = false$.)
(f) Find a λ-term F, s.t. $FI = x$ and $FK = y$.

12. Let $G = C[f, n]$ is $\lambda fn.if\ Zero\ n\ then\ 1\ else\ 2 * f(Pred\ n))$
a) What does compute a (recursive) function F given by the definition $FN = GFN$?
b) Find an explicit expression for F.
c) Reduce F 2.
d) Which function does compute a finite number of k applications of a function G to the everywhere undefined function $\perp$ (i.e. bottom). That is $c_k G \Omega$, expressed using a Church numeral c_k.

11. (HW) Supercombinators are combinators where each λ-abstraction (in all its arguments) is again a (closed) supercombinator. So each term S in a form $(\lambda x_1 \lambda x_2 \ldots \lambda x_n.E)$, where $n \geq 0$, is supercombinator (of the arity n) if it is true, that S does not have free variables, E does not begin with λ-abstraction, and every λ-abstraction in E is again a supercombinator. That means that also embedded functions (in all their arguments) are closed, so they don't have nonlocal variables. (For remembering: combinators are closed λ-terms.) A constant is a supercombinator as well.
Ex.: $\lambda x.x(\lambda y.\underline{x}y) \to \lambda x.x((\underline{\lambda z}\lambda y.zy)\underline{x})$
a) Transform to a supercombinator: $\lambda fgh.f(\lambda y.g(hy))$
b) Transform Y , Y_v, and Θ_v to a supercombinator.
c) Are the terms S, K, and I supercombinators?
Implementation notes: lambda-lifting, (supercombinators do not need environment/evaluation): functions with free variables get additional arguments and original free variables are transferred as values of new arguments; the *closure* is created.

15. Definition. A set A of λ-terms is *closed to the equality* if for any two terms M, N holds: if $M \in A$ and $M =_\beta N$, then $N \in A$.
Prove a generalization of the Scott Theorem (p. 54) and corollaries:
a) [Zl.104] Let A and B are nonempty sets of Λ -terms closed to the equality. Then A and B are not recursively separable. That means, it does not exists such recursive set R that $A \subset R$ and $B \subset -R$.
a.1) another way of using diagonalization/a fixed point theorem: if the separating function is total, then a fixed point (of an analogical functional to the one on p. 55) does not belong nor to A neither to B.
Applications/Corollaries:
b) Scott (p. 54): if A is a set closed to the equality, $A \neq \emptyset$, $A \neq \Lambda$, then A is not recursive.
c) Church (p. 60): The set $\{M | M$ has a NF $\}$ is not recursive (and is recursively enumerable).
d) The relation of convertibility is not recursive. Hint: $A = \{M | M = I\}$.
e) Let E is a consistent set of equations. Then the relation $=_E$ of E-convertibility in the theory $\lambda + E$ is not recursive. (Consistent extensions of λ-calculus are not decidable.)
f) Halting problem is not decidable. Hint: $A = \{M | M$ has a head normal form$\}$.

4th seminar

8. Combinatory logic, (SK calculus), an elimination of abstraction
It holds in λ-calculus
a.1) $\lambda x.x = SKK\ (= I)$
a.2) $\lambda x.M = KM$, for $x \notin FV(M)$
a.3) $\lambda x.MN = S(\lambda x.M)(\lambda x.N)$
The rules provide an algorithm for terms translation to applications of S and K. They cover all ;-) structures

of terms and are disjoint. We denote terms on the right side as $\lambda^* x.x$, $\lambda^* x.M$, and $\lambda^* x.MN$.
Def. *Combinatory logic* (CL) has axioms for K and S: $KMN = M$, $SMNL = ML(NL)$. Then it has schemas of axioms for the equality and assignment in an application (p. 17, except the rule ξ (ksi, xi), which does not hold in CL: $M =_{CL} N \not\Rightarrow \lambda x.M =_{CL} \lambda x.N$) The CL doesn't have an abstraction, bound variables, and α-conversion. *Terms of combinatory logic* are S, K, (free) variables x, and they are closed to an application. (`T::=V|<prim>|TT`; Primitive terms may differ in other variants.)
We can define reductions in CL (analogically to p. 65): base rules of a weak w-reduction (i.1) are $KMN \to_w M$ and $SMNL \to_w ML(NL)$, and then we can define a single-step reduction, a (multi-step) reduction, and a convertibility.
It holds:
b.1) $SKK \neq_{CL} I$, for I with $IM = M$
b.2) $SKKx =_{CL} x$ $(= Ix)$
b.3) $CL \not\vdash SKK = SKS$, but $SKK =_\beta SKS$
Note: We can see from b.3 that provability (and a weak w-convertibility) in CL does not coincide with provability in λ (and β-convertibility). (A missing coincidence can be "eliminated" by supplementary axioms, but the system gets more complicated. For example the axiom $K = \lambda^* xy.Kxy$ can be added (as $CL \not\vdash K = \lambda^* xy.Kxy$), so in this case, the problem is missing arguments on the left side.) *(?Critical pairs.)*
Ex.: Expressing conditions using combinators: we want to express "F is commutative " using a combinator C: instead of $\forall x, y : Fxy = Fyx$, we demand an equality $Cfxy = fyx$ for the combinator C and then the commutativity of F is expressed as $CF = F$.
Ex.: F is associative: $\forall x, y, z : Fx(Fyz) = F(Fxy)z$. We introduce A_1 and A_2 with $A_1 fxyz = fx(fyz)$ and $A_2 fxyz = f(fxy)z$ and demand an equality $A_1 F = A_2 F$ for F.
(c) A translation of λ-terms to the combinators S and K; a completeness.
d) A coding of λ-terms using λ^* notations/transformation to terms of CL (according to b).
Note: An introduction of $\lambda^* : \Lambda \to \mathcal{C}$ notation to CL is only syntactic sugar and not an introduction of a λ-abstraction to CL.
(e) Definitions of B, C (and W, K); a completeness. $Bfgz = f(gz)$, $Cfxz = fzx$. The combinators B and C are similar to $Sxyz$, but the argument z is not propagated to x and y, respectively. ($Wfxx = fxx$)
(e.1) A strict version of CL: CL$_I$ (versus CL$_K$) for strict functions (λ_I: the variable x from $\lambda x.E$ has at least one occurence in E.): CL$_I$ uses S, B, C, and I (without K).
(f) "Extreme programming": A single combinator $X \equiv \lambda x.xKSK$ is enough and it holds $XXX = K$ a $X.XX = S$. (Note to Curry-Howard isomorphism: S and K are in X in assumptions.)
Note: The rule $M = N \wedge N = L \Rightarrow \underline{L = M}$ includes and replaces the rules for symmetry and transitivity.

9. Transform to CL (with S, K, I):
a) $\lambda x.xx$, that means, write $\lambda^* x.xx$
b) $(K=)\lambda xy.x$, b2) $(K_*=)\lambda xy.y$
c) $\lambda xy.yx$
d) d1) $\lambda xy.xx$, d2) $\lambda xy.yy$, d3) $\lambda xy.xy$
e) Transform to combinators S, I, B, and C: e1) $\lambda xy.yx$, e2) $\lambda xy.xy$, e3) $\lambda x(\lambda y.y)x$

10.-15 see above

16. a) How does it appear $[D \to D]$ for a singleton D?
b) Does it hold $S \neq K$ in a single point model?
c) A corollary for consistency?

5th seminar

Typed lambda calculus
17. Prove, that I, K, S, K* have types in $\lambda \to_{Curry}$
18. Show type of a Church numeral c_2.

TO DO: improve

Errata. Calculus 1.

54, down. $\#A = \{\#M|M \in A\}$ instead of $\#A = \{\#M|M \in \underline{\Lambda}\}$
60, up. $NF = \{M|M$ <u>has</u> a normal form$\}$ instead of $NF = \{M|M$ a normal form$\}$
109 down rightmost: missing a right parenthesis: ... $= \|M\|_{\rho(x:=\|N\|_\rho\underline{)}}$

Calculus 2.
30. Ex. $S = (\lambda xyz.x\underline{z}(yz))$ instead of $S = (\lambda xyz.xy(yz))$
32. under line in footnote. $\lambda uv\underline{.u}$: instead of λuv :
33(,34). $M' - >>_\beta M$, instead of $M - >>_\beta M'$, see teaching materials chap.5.12
42 l.2: <u>and</u> and
45 c) $\underline{x : \sigma} \vdash (\lambda y : \tau.x) : (\tau \to \sigma)$
82 b) $\vdash (\lambda xy.y) : (\forall \alpha\beta.\alpha \to \beta \to \underline{\beta})$
(99 terminology (in Czech): preorder: kvaziuspoĹdn, pĹeduspoĹdn)
99 (i) l.7: $\sigma \leq \tau, \sigma \leq \rho \Rightarrow \sigma \leq \overline{\underline{\tau \cap \rho}}$
103 a) $\vdash (\lambda x.xx) : ((\sigma \to \tau) \cap \underline{\sigma}) \to \tau$
114 (i) $\Gamma \vdash x : \sigma \Rightarrow \exists \sigma' \geq \sigma((x : \underline{\sigma'}) \in \Gamma\underline{)}$
182 iii. includin<u>g</u>

Calculus 3.
53 l.2 conjun<u>c</u>tion

Errata. In Czech ex. (Check!): Ex. 4z Hint: $F = C[f, x]F$ -¿ $F = (\lambda f.C[f, x])F$

KomentĹe:
s 21. PĹm rekurze pomoc komb. pevnŠho bodu, vzjemn rekurze pozdÄji pomoc dvojitŠho pevnŠho bodu. (PodobnÄ: pevn bod pro n rovnic.) Impl: Jeden pevn bod pro cel program vs. (stratifikovan) definice pro komponenty silnŠ souvislosti v grafu zvislost.
116: prvn varianty kombinatorickŠ logiky a lambda kalkulu byly nekonzistentn. :-(
Lambda 2, 99: kontravariantnÄ v argumentu, kovariantnÄ ve vsledku

Pozn.

A.1 PouĹžit Y a Mu(μ) v datatypu (alias `Fix`, `FixF`), zpracovn po rovnch; fold, unfold, refold

A.2 Zl 103: rekurzivnÄ neoddÄlitelnŠ mnoĹžiny

A.3 Hlavov NF: nelze redukovat funkci (i pod λ-abstrakc), ale nestarme se o redukci argumentĹ; m tvar $M \equiv \lambda x_1...x_n.yL_1...L_m$. Termy bez hlavovŠ NF lze povaĹžovat za oznaÄ‰en nedefinovanch vrazĹ.

A.3.1 Zl 96: nelze ztotoĹžnit termy bez NF; ZL 138: pro M, N v NF(!) je v lambda,eta teorii buÄ $M = N$ nebo $M\#N$

A.4 PĹepisovac systŠmy (rewriting s.), Knuth-BendixĹv alg. pro dokazovn v rovnostnch teorich, neselhvajc varianta KB; rippling; konfluence, slab konfluence;

A.5 v druhŠm semestru: Curry-HowardĹv izomorfizmus (a "proof assistants"), zvislŠ typovŠ systŠmy (pĹ. vektory s dŠlkou);
DokazovaÄ‰e jsou killer application pro OCAML, (ale: soutÄĹž dokazovaÄ‰Ĺ vyhrvaj ty rychlŠ)

A.5.1 implicitn typovn v Haskellu; pĹi rozĹĹench explicitn typ pouze u urÄ‰itch konstrukc; (statickŠ) typy taky umoĹžĹuj optimalizace (v runtime se (nepouĹžvaj, a proto) netestuj tagy/typy); Haskell jako typov laboratoĹ;
typy: (velmi) rĹznŠ pouĹžit (polyTypickŠ, fantomovŠ, multistage, BTA v PE: Binding Time Anotation) ..., statickŠ typy vyvolaj typovou chybu pĹi kompilaci (1. v netestovanch Ä‰stech, 2. v jinŠ rovni (generovan string, on-the-fly, DSEL - ale: kryptickŠ chybovŠ hlĹky))

A.5.2 Vztah k teorii (k teorii kategori); funktory, mondy, komondy, Arrows, `Fix`, `FixF` pro datovŠ typy, fold, unfold;
… a nstroje: GADT (Generalized Abstract Data Types), aplikativn funktory, aktivn konstruktory;
… a dĹvĂjĹ prci: s-vrazy $\approx$ XML;
… parametricity; catamorphism, anamorphism, paramorphism (cata/fold se zachovnm hodnoty), apomorphism, hylomorphism = cata(f).ana(g) (je pĹpad deforestation), metamorphism …

A.5.3 implementaĂ‰n triky: superkombintory, grafovŠ pĹepisovn, lambda-lifting - pĹemĂna loklnch procedur na globln (viz jinŠ pĹednĹky) - a lambda dropping, closure conversion; defunkcionalizace (s pomoc apply); full laziness (pro Ă‰steĂ‰nĂ aplikovanŠ funkce)

A.5.4 HOAS: higher-order abstract syntax, repr. abstraktnch syntaktickch stromĹ s vzanmi promĂnnmi, prom. nemaj jmŠna a jejich vskyty ukazuj na vzac msto; moĹžn impl. HOAS: de Bruijn indexy; FOAS: first-order abstract syntax;

A.6 pull (FP) vs. push (OO) alg./styl, prce s celmi dat. strukturami, "vzory" rekurze: map, fmap, fold, unfold, scan, …; Filter-Map-Reduce; lambda abstrakce (FP: lambda funkce, closures) umoĹžĹuje dynamickou vazbu, hooks, (implementaci TVM, parametrizaci pomoc "vmĂny" procedur, metaparametry; srv. Lua: metatabulky);
vhody a pĹnosy Lispu (Paul Graham, viz);
abstraktn interpretace: poĂ‰tn s jinou, typicky omezenou sŠmantikou; pouĹžitelnŠ pro globln analzu programu pĹed kompilac; (impl.: …); pĹ: msto standardn sŠmantiky regulrnch vrazĹ/BKG nad jazyky poĂ‰tm first, empty a follow (v koneĂ‰nch domŠnch);

A.7 Manipulace s programy: odvozovn typĹ, Ă‰steĂ‰nŠ vyhodnocovn (partial evaluation, PE), Futamurovy projekce, optimalizaĂ‰n transformace (`map f.map g=map(f.g)`), poĂ‰tn s programy (fusion laws, P. Wadler: Theorems for free!; pointfree vs. pointwise), (taky run-time code generation), deforestation;
JinŠ styly/druhy psan: (continuation passing style); polytypickŠ programy (pro pĹedstavu: umoĹžn vygenerovat definici funkce map pro uĹživatelsk datatyp pomoc strukturln rekurze podle struktury typu), generic programming: Generic Haskell; (multi-)staged programming (vcerovĹovŠ programovn): MetaOCaml, (MetaML), (pro pĹedstavu: generuje typovĂ bezpeĂ‰n kd za bĂhu (typesafe run-time code generation), napĹ. `.<int->int>.`; nepĹesnĂ/zjednoduĹenĂ: typovĂ sprvn makra), splicing - vkldn kdu (a dat) - typovĂ sprvnĂ, (operace: `.< >.`, `~` splice, lift, run);
meta-jazyk a objektov jazyk; pĹstup k zdrojku (string nebo syntax tree): lispovskŠ eval (A.8b), quasiquote; Template Haskell `[| |]`;

A.7.1a Type-directed partial evaluation (TDPE);

A.7.1b Normalization by evaluation (NBE): vyuĹžva pĹevod do domŠn a zptky - fce reflect a reify; reifikace; eta-dlouh forma;

A.7.2 Dokazovn vlastnost programu/modulĹ, zapisovn vlastnost (napĹ. pro unit testy, `rev(rev(x))=x`); QuickCheck - testovn vlastnost pomoc nhodnch vstupĹ (srv: redukce tĹdcch st na 0-1 vstupy) - pozdĂji pĹeveden do jinch jazykĹ; (navc Hs: (nĂjak) genertor nhodnch vstupĹ, i strukturovanch dat (seznamy, stromy, funkce …) vygenerovn automaticky pomoc typovch tĹd (pĹi troĹce opatrnosti))

A.8a pohled zvrchu na hierarchii jazykĹ (Paul Graham);
A.8b Greenspun's tenth rule: Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.
(MorrisĹv dodatek: :-) … vĂ‰etnĂ Common Lispu)

A.9 vĂtĹ abstrakce = kratĹ programy (=mŠnĂ chyb=lepĹ udrĹžovatelnost), abstrakce dovol lepĹ/obecnĂjĹ nvrh; (nvrhovŠ vzory (skoro) jako kd: co je (ve FP) vzor Strategie? apod.; ale: FP m jinŠ nvrhovŠ vzory), znovupouĹžitelnost a parametrizace, vĂ‰etnĂ metaparam.;
Pro "uĹživatele": tvorba DSEL: Domain Specific (Embedded) Languages, domenovŠ kombintory (napĹ. parsery …), (napĹ. generovn sprvnch (DTD valid) XML/HTML dat, SQL dotazĹ); prototypy, psan interpretĹ, fantomovŠ typy;

A.10 Kombinace FP a LP: funkcionln logickŠ programovn, napĹ. jazyky Curry, Mercury; obsahuje vpoÄ‰tov mechanizmus Narrowing (zuĹžovn), pouĹžv substituce, rewriting; vstup: funkcionln term s logickmi promÄnnmi;
FP: OCAML; FP+OOP: Scala (nad JVM), F# (nad .NET); nÄkterŠ rysy maj skriptovac jazyky (ale: dynamickŠ typovn)