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 explicitly 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 \( = \) and \( \equiv \):
      a.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 \( \lambda \)-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 \( x \not\in freevar(t) \)

3. Find a closed \( \lambda \)-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 \( K_\infty \)
   d) \( \exists F \forall X FX = SXF \) (and \( \eta \)-reduced expression)
   e) \( \exists F \forall X FX = FX \)
   (q) It does not exist such \( F \) that \( \forall X, Y FXY = X \)
   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 \( \lambda \)-terms ”under” \( \lambda \)-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 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 \( V = \lambda helo.o(hello) \)), is a f.p.combinator.
   b1) Question: Is it valid for \( Z \):
      \[ Z \rightarrow^* \beta F \rightarrow^* Z(SI)F \]?
   c) Turing’s f.p.combinator \( \Theta = AA \), where \( A = \lambda xy.y(xxy) \) is a f.p.combinator
   c1) Is it valid that \( \Theta \rightarrow^* \beta F \rightarrow^* \Theta(F(\Theta F)) \)?
   c2) Is it valid that \( Y(SI) \rightarrow^* \beta \Theta(F(\Theta F)) \)? (where \( Y \) is a f.p.combinator. 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 \( \Theta \) is in a form of a supercombinator, but \( Y \) is not. (The argument \( F \) is placed after the reduction of \( \Theta F \) as an argument.)
   d) Turing’s \( \Theta \), call-by-value version: \( \Theta_v = AA' \), where \( A' = \lambda xy.y(\lambda z.xxyz) \) is a f.p.combinator.
   d2) \( Y \), call-by-value version: \( Y_v = \lambda f.A' \), where \( A' = \lambda x.f((xx)v) \) is a f.p.combinator.
   d3) Explain a relation of \( \Theta \), respectively \( Y \), to \( \Theta_v \), respectively \( Y_v \), using \( \eta \)-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 \( \lambda \)-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 \( \beta \)-normal form
   c) strongly normalizing, but not in the \( \beta \)-reduced form
   d) normalizing, but not strongly normalizing
   e) are not normalizing

7. Definition. The rule \( \eta \) (eta). For an arbitrary \( \lambda \)-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 \( \eta \) is consistent with the axioms of \( \lambda \)-calculus. A calculus, which includes the \( \eta \) rule is denoted \( \lambda \eta \).

The rule \( \eta \) is called an extensionality rule because it allows to prove for any two \( \lambda \)-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 \( \eta \) 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 \( \lambda \) is not extensional
d) Prove:
d.1) The theory $\lambda + \text{ext}$ is extensional
d.2) The theory $\lambda \eta$ is extensional
d.3) $\lambda + \text{ext} \vdash M = N \iff \lambda \eta \vdash M = N$
d.4) The theories $\lambda + \text{ext}$ and $\lambda \eta$ are the smallest extensional extensions of the theory $\lambda$.

Note: The rule $\eta$ (eta) enables to consider any term $M$ for the function $\lambda x. M x$, 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 $\lambda$.

Note: The rule $\xi$ (ksi, xi), p. 17, $M = M' \rightarrow (\lambda x. M = \lambda x. M')$ is the weak extensionality: if there are convertible bodies of functions, then the functions are convertible as well.

Also: the reduction $\eta$; $\eta$ 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 $\iff \beta$-NF (NF for Normal Form), ”completeness” of $\eta$; for $M$, $N$ with $\beta \eta$-NF: $M = N \lor M \# N$ (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 $n, m \in N$, $c_n \equiv \lambda f x. f^n(x)$:
   a) for $A_+ \equiv \lambda x. y pq. x p y q$ holds $A_+ c_n c_m = c_{n+m}$
   b) for $A_* \equiv \lambda x. y z x (y z)$ holds $A_* c_n c_m = c_{n*m}$
   c) for $A_{exp} \equiv \lambda x. y. y x$ holds $A_{exp} c_n c_m = c_{n*m}$, except for $m = 0$
   d) find a term $A_{s u c c}$ which fulfills $A_{s u c c} c_n = c_{n+1}$ (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 $\Theta$ (p. 80) for the ”Double F.P.Th.”, which allow $\beta$-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) $M =_\beta N$, $M \not= N$, $M$ has NF and has an infinite reduction sequence as well.
      a.1) moreover $M \not= _\beta N$
      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) $M =_\beta N$, $M \not= N$, $M$ 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 $M_1 \leftarrow R \vdash M \rightarrow R M_2$, 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 $\lambda$-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 $s \not= t$.

   It holds $s = t$ for all two $\lambda$-terms $s$ and $t$ in an inconsistent theory.
   Show:
   a) $s \not= K$. 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 \( \lambda \)-term \( F \), s.t. \( FI = x \) and \( FK = y \).

12. Let \( G = C[f, n] \) is \( \lambda f n i f \ Zero n \ then \ then \ 1 \ else \ 2 * f(Pred n) \)
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 \( \lambda \)-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 \( \lambda \)-abstraction, and every \( \lambda \)-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 \( \lambda \)-terms). A constant is a supercombinator as well.

Ex.: \( \lambda x.x(\lambda y.\overline{z}y) \rightarrow \lambda x.((\lambda z\lambda y.\overline{z}y)x) \)
a) Transform to a supercombinator: \( \lambda fgh.f(\lambda y.g(hy)) \)
b) Transform \( Y \), \( Y_\varepsilon \), and \( \Theta_\varepsilon \) 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 \( \lambda \)-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 \( \lambda \)-terms closed to the equality. Then \( A \) and \( B \) are not recursively separable. That means, it does not exist 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 \( \{ M \mid M \text{ has a NF} \} \) is not recursive (and is recursively enumerable).

d) The relation of convertibility is not recursive. Hint: \( A = \{ M \mid M = I \} \).

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 \( \lambda \)-calculus are not decidable.)

f) Halting problem is not decidable. Hint: \( A = \{ M \mid M \text{ has a head normal form} \} \).

4th seminar

8. Combinatory logic, (SK calculus), an elimination of abstraction

It holds in \( \lambda \)-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 $\lambda x.M.N$.

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 $\xi$ (ksi, xi), which does not hold in CL: $M =_{CL} N \neq \lambda x.M =_{CL} \lambda x.N$) The CL doesn’t have an abstraction, bound variables, and $\alpha$-conversion. Terms of combinatory logic are $S$, $K$, (free) variables $x$, and they are closed to an application. (T: =\forall Tprim\in\{IT\}; 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 $KMN \rightarrow_w M$ and $SMNL \rightarrow_w ML(NL)$, 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) $CL \nvdash SKK = SKS$, 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 $\lambda$ (and $\beta$-convertibility). (A missing coincidence can be "eliminated" by supplementary axioms, but the system gets more complicated. For example the axiom $K = \lambda x.y.Kxy$ can be added (as $CL \nvdash K = \lambda x.y.Kxy$), so in this case, the problem is missing arguments on the left side.) (\textit{Critical pairs.})

Ex.: Expressing conditions using combinators: we want to express \textit{"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: $\forall x, y, z : Fxy = F(Fyx)z$. We introduce $A_1$ and $A_2$ with $A_1fxyz = f(x(yz))$ and $A_2fxyz = f(fxyz)$ and demand an equality $A_1F = A_2F$ for $F$.

(c) A translation of $\lambda$-terms to the combinators $S$ and $K$; a completeness.

d) A coding of $\lambda$-terms using $\lambda^*$ notations/transformation to terms of CL (according to b).

Note: An introduction of $\lambda^* : \Lambda \rightarrow \mathcal{C}$ notation to CL is only syntactic sugar and not an introduction of a $\lambda$-abstraction to CL.

(e) Definitions of $B$, $C$ (and $W$, $K$); a completeness. $Bfyz = f(gz)$, $Cfxyz = f(xz)$. The combinators $B$ and $C$ are similar to $Sxyz$, but the argument $z$ is not propagated to $x$ and $y$, respectively. ($Wfxx = fxx$)

(e1) A strict version of CL: $CL_I$ (versus $CL_K$) for strict functions ($\lambda_I$: the variable $x$ from $\lambda x.E$ has at least one occurrence in $E$): $CL_I$ uses $S$, $B$, $C$, and $I$ (without $K$).

(f) "Extreme programming": A single combinator $X \equiv \lambda x.x KS K$ 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 $M = N \land N = L \Rightarrow L = M$ 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 $[D \rightarrow D]$ 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 $\lambda \rightarrow \text{Carry}$

18. Show type of a Church numeral $c_2$. 

5
TO DO: improve

Errata. Calculus 1.

54, down. \(\#A = \{\#M | M \in A\}\) instead of \(\#A = \{M | \#M \in A\}\)
56, up. \(NF = \{M | M \text{ has a normal form}\}\) instead of \(NF = \{M | M \text{ a normal form}\}\)
109 down rightmost: missing a right parenthesis: \(\ldots = \|M\|_{\rho(x = \|N\|_{\rho}}\)

Calculus 2.

30. Ex. \(S = (\lambda xyz.xz(yz))\) instead of \(S = (\lambda x y z.x y (y z))\)
32. under line in footnote. \(\lambda uv \ y : \) instead of \(\lambda uv :\)
33(,34). \(M' > >_\beta M\), instead of \(M' > >_\beta M\), see teaching materials chap.5.12
42 l.2: and and
45 c) \(x : \sigma \vdash (\lambda y : \tau.x) : (\tau \to \sigma)\)
82 b) \(\vdash (\lambda x y : \omega \to \beta. \alpha \to \beta)\)
(99 terminology (in Czech): preorder: kvaziuspoĹdn, pĹeduspoĹdn)
99 (i) 1.7: \(\sigma \subseteq \tau, \sigma \subseteq \rho \Rightarrow \sigma \subseteq \tau \cap \rho\)
103 a) \(\vdash (\lambda x : \tau xx) : ((\sigma \to \tau) \cap \sigma) \to \tau\)
114 (i) \(\Gamma \vdash x : \sigma \Rightarrow \exists \sigma' \geq \sigma((x : \sigma') \in \Gamma)\)
182 iii. including

Calculus 3.

53 l.2 conjunction

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

KomentĹe:

s 21. PĹm rekurse pomoc komb. pevnŠho bodu, vzjemn rekurse 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: prv varianty kombinatorickŠ logiky a lambda kalkulu byly nekonzistentn. :-(" Lambda 2, 99: kontravariantnÁ v argumentu, kovariantnÁ ve vsledku

Pozn.

A.1 PouĹžit Y a Mu(\(\mu\)) v datatypu (alias Fix, FixF), zpracovn po rovnch; fold, unfold, refold
A.2 Zl 103: rekurzivnÁ neoddÄlitelnŠ mnoĹžiny
A.3 Hlavov NF: nelze redukovat funkci (i pod \(\lambda\)-abstrakc), ale nestarome se o redukci argumentĹ; m tvar \(M \equiv xyz...x\ y zur L1...LM\). Termny bez hlavovŠ NF lze povaĹžovat za oznaĹa‰en nedefinovaných vrazĹ.
A.31 Zl 96: nelze ztotoĹžnít termny bez NF; ZL 138: pro M, N v NF(!) je v lambda,eta teorii buĹ
\(M = N\) nebo \(M \neq N\)
A.4 PĹepisovac systĹmy (rewriting s.), Knuth-BendixĹv alg. pro dokazovn v rovnostnch teorich, nese-

lhvajc varianta KB; rippling; konfluence, slab konfluence;
A.5 v druhŠm semestru: Curry-HowardĹv izomorfizmus (a "proof assistants"), zvislŠ typovŠ systĹmy
(pĹ. vektory s dŠlkou); DokazovaĹa‰ej jsou killer application pro OCAML, (ale: soutĹž dokazovaĹa‰L vyhrvaj ty rychlŠ)
A.51 implicitn typovn v Haskellu; pĹí rozĹčch explicitn typ pouze u urĹa‰nich konstrukc; (statickŠ)
typy taky umoĹžĹnit optimalizace (v runtime se (ne)pouĹživaj, a proto) netestuj tagy/typy); Haskell jako

typov laboratoĹ;
typy: (velmi) rĹžnŠ pouĹžit (polyTypickŠ, fantomovŠ, multistage, BTA v PE: Binding Time Anotation)
...; statickŠ typy vyvolaj typovou chybu pĹí kompilaci (1. v netestovanych A‰stech, 2. v jinŠ rovni
(generovan string, on-the-fly, DSEL - ale: kryptickŠ chybovŠ hĹky))
A.5.2 Vztah k teorii (k teorii kategori); funktory, mondy, komondy, Arrows, Fix, FixF pro datové typy, fold, unfold; ... a nstroje: GADT (Generalized Abstract Data Types), aplikativní funkctory, aktivní konstruktory; ... a dřívá příč: s-vrazy ≈ XML; ... parametricity; catamorphism, anamorphism, paramorphism (cata/fold se zachovánm hodnoty), apomorphism, hylohomorphism = cata(f).ana(g) (je případ deorestation), metamorphism ... 
A.5.3 implementána%tn triky: superkombinatory, grafovéS pEpsion, lambda-lifting - pLemÁna lokálních procedur na globln (viz jinŠ pLednLky) - a lambda dropping, closure conversion; defunkcionalizace (s pomoc apply); full laziness (pro ěste%tnÁ%tnÁ aplikovanÁ funkce) 
A.5.4 HOAS: higher-order abstract syntax, repr. abstraktých syntaktických stromů s vzánní promÁn- ní, prom. nemaj jinŠn a jejich vstupy ukazuj na vzác msto; moLo zimpl. 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) umoLoŽuje dynamickou vazbu, hooks, (implementaci TVM, parametrizaci pomoc "vmÁny" procedur, metaparametry; srv. Lua: metatabulky); vhody a pLnozy Lispu (Paul Graham, viz); abstraktní interpretace: poÅ%tn s jinou, typicky omezenou sŠmantikou; pouĹžitelnÁ pro globln analzu programu pLed kompilac; (impl.: ...); pL: msto standardn sŠmantiky regulrních vrazL/BKG nad jazyky poÅ%tn first, empty a follow (v kone%tních domÁnách)
A.7 Manipulace s programy: odvozovn typL, ěste%tnÁ vyhodnocovn (partial evaluation, PE), Futamurovy projekce, optimalizaÁ%tn transformace (map f.map g=map(f.g)), poÅ%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); polytypickÁ programy (pro pLedstavu: umoLo zgenerovat definici funkce map pro uĹživatelsk datatyp pomoc strukturln rekurze podle struktury typu), generic programming; Generic Haskell; (multi-staged programming (ucerovLovŠ programovn): MetaO-Caml, (MetaML), (pro pLedstavu: generuje typovÁ bezpeÅ%tn kdo za bÅhu (typesafe run-time code generation), napL. .<int->int>; nepLesnÁ/zjednoduĹenÁ: typovÁ sprvn makra), splicing - vklO kdu (a dat) - typovÁ sprvnÁ, (operate: .< >., * splice, lift, run); meta-jazyk a objektov jazyk; pLstup k zdrojku (string nebo syntax tree): lispovskÁ eval (A.8b), quasi-quote; Template Haskell [1 [1];
A.7.1a Type-directed partial evaluation (TDPE); A.7.1b Normalization by evaluation (NBE): vyuĹživa pLevod do domnÁ a zptky - fce reflect a reify; reifikace; eta-dlouÅh forma;
A.7.2 Dokazovn vlastnost programu/modul, zapisovn vlastnost (napL. pro unit testy, rev(rev(x))=x); QuickCheck - testovn vlastnost pomoc nhodnch vstupL (srv: redukce tLdch st na 0-1 vstupy) - pozdÄji pĽeden do jiných jazykL; (navc Hs: (nÁjak) generotor nhodnch vstupL, i struktuvanchn dat (seznamy, strony, funkce ...) vygenerovn automaticky typovch tLd (pĹti troĹce opatrnosti))
A.8a pohled zvrchu na hierarchii jazykL (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. (MorrisL vy dodatek: :-) ... vÄ%etnÁ Common Lispu) A.9 vÄtL abstrakce = kratL programy (=mŠnÁ chyb=lepL udrĹžovatelnost), abstrakce dovol lepL/obecnÁ L nvrh; (nvrhovŠ vzory (skoro) jako kd: co je (ve FP) vzor Strategie? apod.; ale: FP m jinŠ nvrhovŠ vzory), znovoupĹžitelnost a parametrizace, vÄ%etnÁ metaparam.; Pro "uĹživatele": tvorba DSEL: Domain Specific (Embedded) Languages, domenovŠ kombinatory (napL. parsery ...), (napL. generovn sprvnch (DTD valid) XML/HTML dat, SQL dotazL); prototypy, psan interpretL, fantomovŠ typy;
A.10 Kombinace FP a LP: funkcionáln logické programování, např. jazyky Curry, Mercury; obsahuje vpočítov mechanizmus Narrowing (zuzavín), používá substituce, rewriting; vstup: funkcionální term s logickmi proměnními;
FP: OCAML; FP+OOP: Scala (nad JVM), F# (nad .NET); některé rysy mají skriptovací jazyky (ale: dynamické typování)