Subject ۹۴،۶،۲۵
Date

مقدمه

زبان‌های برنامه‌سازی پیشرفته

عنوان درس: Advanced Programming Languages / Advanced Topics in Programming Languages

* نسبت درس PL به Programming: PL شاخه‌ای از Programming است که به زبان‌های برنامه‌سازی می‌پردازد.
* ماهیت درس PL: Formalism (رسمیت و دقت)

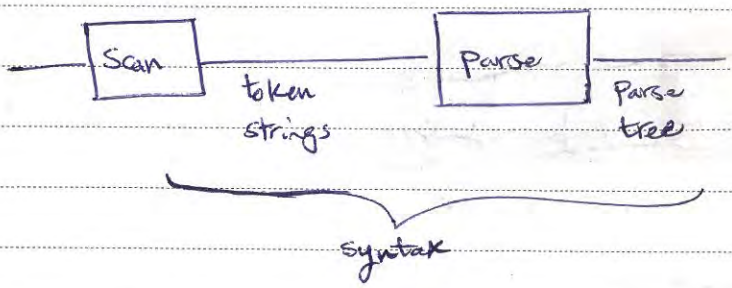
Language: a medium for expression (زبان برای بیان)
Program: specifying a computation

① Semantics: در این درس عمدتاً به معنای برنامه می‌پردازیم.

to give meaning to sentences (programs) ↓ Semantics

برنامه: ساخت و معنای آن

برنامه واقعاً چیست؟



برنامه‌ها در برنامه‌نویسی ابتدا syntactic هستند ولی مراحل بعدی را بررسی

طی مراحل: دلیل استفاده از Context-Free Grammars برای تعریف syntax زبان این برنامه‌هاست

دلیل استفاده: tradition و قابلیت پیاده‌سازی آسان و همچنین کارکرد خوب آن‌ها

Context-Sensitive هم هست، این برنامه‌ها قابل بررسی هستند.

Subject
Date

int i, j;
char c;
j = i + c; → این عمل در این context معنی دارد.
چگونه؟ از c و i، char و int تشخیص داده شد.

Static Analyzer مثل parse و scan بخشی از کامپایلر است که روی زبان، بعد از (Static Semantics) کار می‌کند.
Static Analyzer با Type System کار می‌کند. از طریق آن ارورها را می‌دهد.
↓
Type Mismatch مثلاً این ارور را می‌دهد.

* Key Concept های این درس، Type هاست.

Static Semantics (Types, Typing) : این درس دو بخش دارد ②

سوال این درس: چطور می‌توان به زبان طراحی کرد که برنامه‌های مدنظر را اجرا کند؟
↓
- طراحی زبان طراحی شده به گونه‌ای که stuck نشود.
هرباره
- زبان طراحی شده دارای خاصیت امنیتی باشد.
(Built-in Security)

- زبان طراحی شده Preserve Requirement را داشته باشد.

PaPCo

Subject ۹۹،۷،۲۵
Date APL

Textbooks:

- Foundations for Programming Languages, John Mitchell, MIT Press, 1996.
  (The Bible of Semantics)
- Practical Foundations for Programming Languages, Robert Harper, Cambridge University Press, 2013.
- Types and Programming Languages, Benjamin Pierce, MIT Press, 2003.
- The Formal Semantics of Programming Languages, Glynn Winskel, Cambridge University Press, 1993.

Functional ... application, function ...

Haskell : ...

Lambda Calculus : ...
(λ)

Evaluation: Final 12
midterm 6
Assignments 2

Subject

Date ۹۹/۹/۲۷

Induction:

(Inductive Definitions)

تعاریف و اثبات‌های مبتنی بر این روش را به نام استقرا می‌شناسیم.

(Winskel – فصل ۲) (Harper – فصل ۴) (Foundations – Mitchell – فصل ۱۰ Page 27 – 42)

* Introduction to Bisimulation and Co-induction, Davide Sangiorgi)

این روش در اثبات خاصیت‌های مبتنی بر دنباله‌ها کاربرد دارد. فصل ۴ هریت – روش‌های دوگان را

(تنها استقرا و co-induction می‌دهد)

استقرا روی اعداد طبیعی:

0, 1, 2, ...

↓

numeral ⇐ عددی که برای نمایش اعداد استفاده می‌کنیم.

$\forall n \in \mathbb{N} \cdot P(n)$

e.g. $P(n) \triangleq 0+1+2+\cdots+n = \frac{n(n+1)}{2}$

Form I)

$$\forall n \in \mathbb{N} \cdot P(n) \Leftrightarrow \underbrace{P(0)}_{\text{Induction Basis}} \wedge \underbrace{\forall K \in \mathbb{N} \cdot P(K)}_{\text{Induction Hypothesis}} \rightarrow \underbrace{P(K+1)}_{\text{Induction Conclusion}}$$

Induction Step

برای اثبات proof principle روی اعداد طبیعی داریم می‌نویسیم:

۱ – نشان دهیم P(0) برقرار است.

۲ – نشان دهیم برای هر $K \in \mathbb{N}$ اگر P(K) برقرار باشد، P(K+1) نیز برقرار است.

↳ K ای را فرض کنید و بدون هیچ فرض خاصی دیگر ثابت کنید.

Subject APL
Date ۹۹,۴,۲۷

ثبت استقرایی: اثبات $P(k) \to P(k+1)$ برای $P(k)$ دلخواه، و سپس اثبات $P(k+1)$ برای $P(k)$ دلخواه. سپس نتیجه ثبت برای اثبات این عبارت با استفاده از این دو مرحله اثبات شد.

A simple generalization:

$$\forall n \in \mathbb{Z}_{\geq m} \cdot P(n) \Leftrightarrow P(m) \wedge \forall k \in \mathbb{Z}_{\geq m} \cdot P(k) \to P(k+1).$$

(این قضیه با تغییر متغیر به صورت زیر ثابت می‌شود:)

$$Q(n) = P(n+m)$$

$$\forall n \in \mathbb{N} \cdot Q(n) \Leftrightarrow Q(0) \wedge \forall k \in \mathbb{N} \cdot Q(k) \to Q(k+1).$$

(

Form II) (The strong form of mathematical induction on natural numbers)

$$\forall n \in \mathbb{N} \cdot P(n) \Leftrightarrow \forall m \in \mathbb{N} \cdot (\forall k \in \mathbb{N} \cdot k < m \to P(k)) \to P(m)$$

OR $\forall n \in \mathbb{N} \cdot P(n) \Leftrightarrow \forall m \in \mathbb{N} \cdot (\forall k < m \cdot P(k)) \to P(m)$

مثال شبیه (Strong form) Form II ، (Weak form) Form I اثبات می‌شود.

↓

دلیل استفاده از صورت این strong این است که

می‌توان برای اثبات $P(k)$ فرض کرد که چند حالت قبل از آن درست است.

یعنی می‌توان فرض کرد که همه حالت‌های قبل از آن درست است.

$P(k-1)$ $P(k-1), P(k-2), \cdots, P(0)$

↳ پس برای اثبات نادرست است.

Subject

Date

$\forall n \in \mathbb{Z}^{\geq 2} \cdot \exists p_1, \dots, p_r \in Primes \cdot n = p_1 \cdot p_2 \cdots p_r$ (تجزیه به اعداد اول) (اثبات)

(Factorization)

$P(n)$

$P(2) \triangleq \exists p_1, \dots, p_r \in Primes \cdot 2 = p_1 \cdots p_r \Rightarrow 2 = 2$ ✓

$\left(\forall k < m \cdot P(k)\right) \rightarrow P(m)$

$$\begin{cases} \text{if } m \text{ is prime} : & m = m \quad ✓ \\ \text{if } m \text{ is composite} : & m = m_1 \cdot m_2 \;;\; m > m_1, m_2 \geq 2 \end{cases}$$

فرض قوی عدد غیر اول

$\xrightarrow{I.H.} m = (p_1 \cdots p_r)(p'_1 \cdots p'_s)$

m_1 $\quad$ m_2

پس هر m به صورت حاصل ضرب اعداد اول قابل بیان است.

آیا این برهان اثبات یک خاصیت دیگری برای همه اعداد طبیعی غیر از صفر یا یک هم می‌تواند باشد؟

هر عضو مجموعه‌ی دلخواه A یک ویژگی را دارد؟

$\forall a \in A \cdot P(A)$

برای اثبات چنین گزاره‌هایی از استقرا استفاده نمی‌کنیم!

$f : A \rightarrow \mathbb{N}$ تعریف یک تابع روی A

total function

Subject APL
Date 94,9,27 Hafta.

$Q(n) \triangleq f(a) = n \longrightarrow P(a)$

$\forall n \in \mathbb{N}.\ Q(n)$

↳ حال این را ثابت کنیم.

(روابطی از آن نوع f بین well-ordering مجموعه‌ها، بین مجموعه A)

Example – A binary tree is either empty, or an internal node with two subtrees.

(خود همین تعریف را به تعریف استوارتر کنید)

T : The set of all binary trees

Prove that $\forall t \in T.$ t has at most one more leaf than internal nodes.

(برگ) — (درونی)

$P(t)$

height : $T \longrightarrow \mathbb{N}$

طول طولانی‌ترین مسیر از ریشه تا برگ

$Q(n)$: height(t) = n $\longrightarrow$ t has at most ...

$P(t)$

$\forall n \in \mathbb{N}.\ Q(n)$

Subject
Date

$Q(0) \triangleq height(t) = 0 \rightarrow p(t)$

اثبات کنیم فقط درخت‌های تهی چنین خاصیتی دارند.

empty

$leaves(empty) = 0$

$inode(empty) = 0$

$leaves(empty) \leq inodes(empty) + 1$ ✓

درخت یک‌برگی →

$leaves = 1$ ✓

$inodes = 0$

پس $Q(0)$ برقرار است.

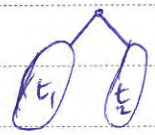
حرکت به سوی استقرای ریاضی، به جای استقرای قوی (فرم دوم)

$\forall m \in \mathbb{N}. \left(\forall k < m . Q(k) \right) \rightarrow Q(m)$.

برای $Q(0)$ را با بالا نشان دادیم، برقرار است.

برای $m \geq 1$،

درخت به صورت زیر است، پس ما درباره درخت‌هایی داریم.



$height(t_1) < m$ $\quad$ $height(t_2) < m$

I.H. → $leaves(t_2) \leq inodes(t_2) + 1$

I.H. → $leaves(t_1) \leq inodes(t_1) + 1$

root ریشه

$\Rightarrow leaves(t_1) + leaves(t_2) \leq inodes(t_1) + inodes(t_2) + 1 + 1$ ✓

$leaves(t)$ $\quad$ $inodes(t)$

Subject APL
Date ۹۴/۶/۲۷

* این تعریف استوار پایه (ساختار بنیادی) ، پس ما می‌توانیم تابع f را ایده‌ی .

روی ساختن یک ساختار بنیادی ، یعنی n ، از تعریف (قاعده) استوار استفاده می‌کند. (تابع height)

پس می‌توان تابع را روی ساختارهای پیچیده‌تر از قاعده استوار استفاده کرد.

حال این طریقت تعریف استوار (یعنی) می‌تواند بدون اینکه بدانیم f چیست، تعریف استوار شود.

پس بنابراین، برای تابع f ما، می‌توانیم استوار روی ساختار بنیادی تعریف کنیم.

Induction on Expressions (Structural Induction):

(Induction on the structure of expressions)
استوار روی ساختار عبارت‌ها

عبارت‌ها از نوع syntactic object یا ساختار بنیادی هستند. ما expression می‌گوییم.

expression → ترکیب بنیادی

پس این عبارت یک درخت (derivation tree) parse خواهد داشت.

ما تابع‌ها را استوار روی انواع درخت‌های بنیادی ، روی انواع درخت‌های parse هم استوار کردیم.

در واقع ، ما برای هر expression یک rule داریم. پس تابع‌ها را روی این rule ها استوار کردیم.

PAPCO

Subject

Date 94/7/1

"is defined to be"

Example - $e ::= 0 \mid 1 \mid x \mid e+e \mid e*e$

x: variable

$Q(n) \triangleq \forall t \in Trees . (height(t) = n \wedge t \text{ is a parse tree of } e) \rightarrow P(e)$

$\forall n \in \mathbb{N}. Q(n) \Leftrightarrow \forall e. P(e)$

(برای اثبات $\forall e. P(e)$ می‌توانیم به روش استقرای ریاضی عمل کنیم: یعنی ابتدا $P(0)$، $P(1)$، $P(x)$ را اثبات کنیم:)

Ind. Base: $P(0), P(1), P(x)$

Ind. step: $P(e_1) \wedge P(e_2) \Rightarrow P(e_1 + e_2)$

$P(e_1) \wedge P(e_2) \Rightarrow P(e_1 * e_2)$

(اثبات Base)

Structural Induction - To prove that $P(e)$ is true for every expressions e generated by some grammar, it is sufficient to prove $P(e)$ for every atomic expression and, for any compound

Subject APL
Date 4/4/21

expressions e with immediate subexpressions $e_1, \ldots, e_k$, prove that if $P(e_i)$ for $i = 1, \ldots, k$, then $P(e)$.

Example –

$$e ::= 0 \mid 1 \mid v \mid e + e \mid e \times e$$

prove that every expression given by the grammer above defines a multi-variate function bounded by a certain form of polynomial.

$P(e) \triangleq$ for any list $v_0, \ldots, v_n$ of variables containing all the variables in e, there is a polynomial $c v_0^k v_1^k \ldots v_n^k$ such that for all natural number values of $v_0, \ldots, v_n$ greater than zero, the value of e is less than the value of the polynomial.

$P(0)$, $P(1)$, $P(v_i)$: ✓

$P(0)$: $0 < v_0 \cdot \ldots \cdot v_n$ ✓ $(c = k = 1)$

$P(1)$: $1 < 2 v_0 \cdot \ldots \cdot v_n$ ✓ $(c = 2, k = 1)$

Subject
Date

$P(v_i)$: $v_i < 2^{v_1} \cdot \ldots \cdot v_i \cdot \ldots \cdot v_n$ ✓

v_i : حرف‌های v_1 تا v_n خواهیم.

: Induct.on step با استقرا ادامه

$P(e_1)$: $e_1 < c\, v_0^K v_1^K \cdots v_n^K$ فرض استقرا

$P(e_2)$: $e_2 < c'\, v_0^{K'} v_1^{K'} \cdots v_n^{K'}$

با این فرض v_1 تا v_n یک جمله‌ی معتبر در e_1 و e_2 هست.

$$\Rightarrow e_1 + e_2 < (c+c')\, v_0^{\max\{K,K'\}} \cdot v_1^{\max\{K,K'\}} \cdot \ldots \cdot v_n^{\max\{K,K'\}}$$

$\Rightarrow P(e_1+e_2)$ ✓

حالت ۲ :

I.H.

$$\Rightarrow e_1 * e_2 < c.c'.\, v_0^{(K+K')} \cdot \ldots \cdot v_n^{(K+K')} \Rightarrow P(e_1 * e_2)$$ ✓

پس ثابت شد $P(e)$ برای همه‌ی e ها برقرار است.

✓ استقرای ریاضی روی اعداد طبیعی مثالی از استقرای روی ساختمان است.

$n ::= 0 \mid \text{Succ } n$

$\forall n.\, P(n)$

Subject APL
Date 99/1/1

Induction on Proofs: (Structural Induction on Proofs)

Proof System ... Proof ...

A Hilbert-style proof system consists of 'axioms' and 'proof rules'.

... Hilbert-style ... proof system ...

... equality (...

$e = e$ (axiom)

$$\frac{e_1 = e_2 \quad e_2 = e_3}{e_1 = e_3}$$ (proper rule) (inference rule)
(proof rule)

(... proof rule ... axiom ...)

$$\frac{A_1 \quad \cdots \quad A_n}{B}$$: proof Rule ...

... B ... A_1 ... A_n ...

B ... premises ... antecedents ... A_1 ... A_n ...

... Conclusion ! Consequent

... Proof Rule ... axiom ... Proof Rule ...

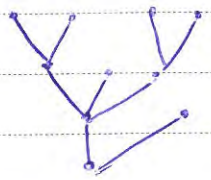
antecedent ...

Subject
Date

(Sequence)

◆ تعریف proof : دنباله‌ای از فرمول‌ها که هرکدام یا از فرضیات یا از axiom یا از rule های سیستم نتیجه شده‌اند.

می‌خواهیم ثابت کنیم:



Structural Induction on Proofs - To prove that $P(\pi)$ is true for every proof π in some proof system, it is sufficient to show that P holds for every axiom of the proof system, and then assuming that P holds for proofs $\pi_1, \ldots, \pi_k$, Prove that $P(\pi)$ for any proof that ends by extending one or more of the proofs $\pi_1, \ldots, \pi_k$ with one inference rule.

(برای اثبات به این شکل می‌توان از loproof rule استفاده کرد.)

◆ منظور از Soundness of a proof system این است که هیچ فرمول اثبات‌ناپذیری را نمی‌توان ثابت کرد.

هر فرمول اثبات‌پذیر را می‌توان ثابت کرد.

PAPCO:

Subject APL
Date ۹۹/۷/۱

Example - (inequality)

$$e ::= 0 \mid 1 \mid v \mid e_1 + e_2 \mid e_1 * e_2$$

given: A simple proof system for inequalities $e \leq e'$.

Axioms
$$e \leq e$$
$$0 \leq e$$

Proof Rules

$$\frac{e \leq e' \quad e' \leq e''}{e \leq e''} \text{ (Transitivity)}$$

$$\frac{e \leq e' \quad e'' \leq e'''}{e + e'' \leq e' + e'''} \text{ (plus-monotonicity)}$$

$$\frac{e_1 \leq e_2 \quad e_3 \leq e_4}{e_1 * e_3 \leq e_2 * e_4} \text{ (Times-monotonicity)}$$

✓

[illegible] soundness [illegible]

$P(\pi) \triangleq$ if π is a proof of $e \leq e'$, then for all values of variables, the value of e is less than or equal to the value of e'.

(arithmetic)

Subject
Date

ممکن است تعابیر ضعیف و قوی برای این رشته‌ی استقراها هم داشت.

ابتدا باید اثبات کنیم $e \leq e$ و $e \leq \hat{e}$ و $\hat{e} \leq e$ تنها یک درخت است.

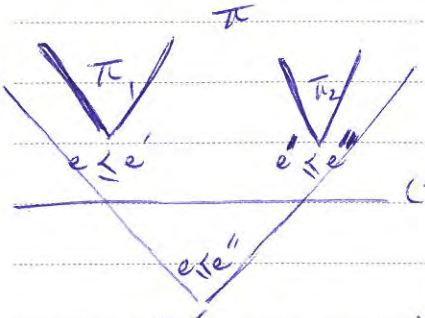
$e \leq e$ ✓

بدیهی است. چون مقدار است صفر، و با
اثبات در همان ریشه‌ی مقدار برمی‌گردد.

برای اثبات $0 \leq e$ استقرا روی ساختار e. مثلاً $0 \leq e$
به این شکل اثبات می‌شود. ✓

حال برای حالت‌ها:

از اثبات $e_1 \leq e_2$ رشته‌ی (π_1) را داریم:



از soundness برای اثبات $e \leq e'$ و $e' \leq e''$ داریم پس واقعاً مقدار e کوچکتر مساوی

e'' خواهد بود در همه‌ی مدل‌ها. پس اثبات $e \leq e''$ هم درست است.

$$p(\pi_1) \wedge p(\pi_2) \Rightarrow p(\pi)$$

$p(\pi_1)$: proof of $e \leq e'$

$p(\pi_2)$: proof of $e' \leq e''$

$p(\pi)$: proof of $e \leq e''$

Subject APL
Date ۹۷/۷/۲۰

Well-founded Induction:

(استقرای خوش‌بنیاد)

A well-founded relation on a set A is a binary relation $\prec$ on A with the property that there is no infinite descending sequence $a_0 \succ a_1 \succ \dots$.

example - $i \prec j \Leftrightarrow j = i+1$ on natural numbers.

i, j اعداد طبیعی هستند.

رابطه‌ای خوش‌بنیاد است که دنباله نزولی بی‌نهایت ندارد.

A well-founded relation is irreflexive.

رابطه انعکاسی نمی‌تواند خوش‌بنیاد باشد: $a \prec a \prec a \prec \dots$

Lemma - Let $\prec$ be a binary relation on set A. Then, $\prec$ is well-founded iff every nonempty set of A has a minimal element.

minimal یعنی

$B \subseteq A \cdot b \in B \cdot \forall x \in A \cdot x \neq b \rightarrow x \not\prec b$.

(b is a minimal element of B)

Proof - As an exercise!

Subject
Date

Theorem_ (Well-founded Induction)

Let $\prec$ be a well-founded relation on A and let P be some property on A. If P(a) holds whenever we have P(b) for all $b \prec a$, then P(a) is true for all $a \in A$.

$$\forall a.\Big(\big(\forall b.\ b \prec a \rightarrow P(b)\big) \rightarrow P(a)\Big) \Leftrightarrow \forall a.\ P(a)$$

(اثبات: هم ارز با استقرای ریاضی است)

Proof_

(از طرف راست به چپ: بدیهی است. از طرف چپ: فرض خلف می‌گیریم $\forall a.P(a)$ و ثابت می‌کنیم)

از طرف چپ به راست:

Proof by Contradiction:

If there is some $x \in A$ with $\neg P(x)$, then the set $B = \{a \in A \mid \neg P(a)\}$ is nonempty. According to the previous lemma, B has a minimal element $a \in B$. Since we ~~b~~ therefore have: $\forall b \prec a.\ P(b)$ (minimal چون a است) which implies P(a). A contradiction!

Subject APL

Date 94/۷/۳

Induction on natural numbers: $m \prec n$ if $m+1=n$

(weak)

Induction on natural numbers: $m \prec n$ if $m < n$

(strong)

Structural Induction on expressions: $e \prec e'$ if e is an "immediate" subexpression of e'.

(Form I) (weak)

Structural Induction on expressions: $e \prec e'$ if e is a subexpression of e'.

(Form II) (strong)

Induction on Proofs: $\pi \prec \pi'$ if π is the subproof for some antecedents of the last inference rule in proof π'.

Lexiographic ordering

$(m,n) \prec (m',n')$

iff $m \prec m'$ or

$(m = m'$ and $n \prec n')$

PAPCO

Proof principle

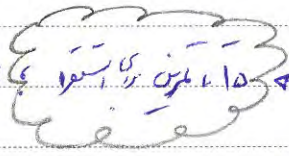
1/ · lexiographic induction

Subject
Date

example - $(m_1, m_2, \dots, m_k) < (n_1, n_2, \dots, n_\ell)$

iff

$k < \ell$ V (or) $(k = \ell \wedge \exists i \leq k . \forall j < i . m_j = n_j \wedge m_i < n_i)$



فصل ۱۰۸ کتاب ← مبنی Foundation ← کاملترین برای استوار کاملاً مبنی است.

(از این به بعد کتاب Harper.)

Binding Syntax, Abstract Syntax Tree ← مبنی Syntactic object ← فصل اول

parse Tree of a program

مثلاً: AST کتاب؛ علاوه بر این علم Scope و استخراج binding های متغیرها در درخت.

فصل دوم و سوم کتاب Harper در مورد تعریف استوار است.

Inductive Definitions :

هر چیزی که به صورت استوار تعریف شود، حتماً می‌توان رابطه well-founded ای روی آن تعریف کرد.

حتماً می‌توان از استوار بودن اثبات به دست آورد که عناصر است.

Judgment (حکم) :

خبر

declarative sentence ← proposition

که می‌تواند درست یا غلط باشد.

meaningful ← نحوه درست یا غلط بودن آن

مبنی معنا است.

Subject APL
Date ۹۷/۷/۳

Judgment یعنی این!

example - n nat → n is a natural number

$n = n_1 + n_2$ → n is the sum of n_1 and n_2

τ type → τ is a type

e : τ → expression e has type τ

e ⇓ v → expression e evaluates to v.

Surprisingly

2 nat

5 = 3 + 2

unary predicates n nat

b nat ، 2 nat

Subject
Date

Judgment در مورد true یا false ثابت کرد.

برای تعریف judgment ابتدا آن را تعریف کنید.

An inductive definition of a judgment form consists of a collection of "rules" of the form

$$\frac{J_1 \quad \cdots \quad J_k}{J}$$

(inference)

$J_1 \cdots J_k$: premises نامیده می‌شوند و J: Conclusion

Conclusion طبق premise ها از inference rule استنتاج می‌شود.

هر premise ها از بالا Conclusion را نتیجه می‌دهد.

اگر premise های یک rule همگی از قبل ثابت شده باشند، آن judgment نیز ثابت می‌شود.

- if $k = 0$ → judgment is an "axiom". (called)
- if $k \neq 0$ → judgment is called a "proper rule".

Example: "a nat" judgment form

(تعریف برای مجموعه اعداد طبیعی)

مثال: (تعریف برای اعداد طبیعی)

$$\frac{}{\text{zero nat}} \qquad \frac{a \text{ nat}}{\text{succ}(a) \text{ nat}}$$

Subject APL
Date ۹۴/۷/۱۰

۳۰ Bisimulation and Co-induction (Co-induction و تعریف استقرایی)

تعریف استقرایی ← finiteness (تناهی بودن) ← برای چنین چیزهایی نمی‌توان از Co-induction استفاده کرد.

* در judgment ها finite derivation یعنی finite proof.

judgment برای proof نوشتیم.

judgment ها assertion هم می‌گویند.

judgment ها را با Inference Rule تعریف می‌کنیم یا استقرایی یا هم‌استقرایی.

تعریف استقرایی با axiom های محدود و rule های متناهی. ← proper rule

Rules define the strongest judgment closed under the rules.

in an inductive definition ← forward

Rules in co-inductive definition ← weakest judgment closed backward under the rules (largest set)

judgment ای که با rule ها بسته است (هیچ مجموعه‌ای از آن با rule ها قابل گسترش نیست).

strongest یعنی smallest set

(necessary) ← هیچ چیز بزرگ‌تری از این نمی‌توان ساخت که با rule ها بسته باشد.

کمترین (closed) این است که sufficient هم هست.

Subject
Date

sufficient ← closeness
necessary ← strongest

strongest, closeness : ...

"تعریف" استقرایی

مثال - تعریف درخت :

empty tree

a_1 tree a_2 tree

node (a_1, a_2) tree

The set of trees = { empty, node(empty, empty), node(empty, node(empty, empty)) ... }

strongest ... proof principle ...

PaPCO

Subject APL
Date ۹۴/۷/۱۰

مثال - تعریف ساری :

$$\frac{}{zero = zero \quad nat}$$

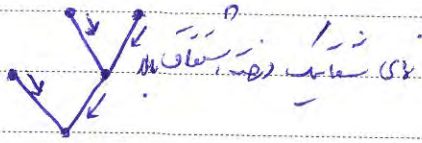
$$\frac{a = b \quad nat}{succ(a) = succ(b) \quad nat}$$

derivation tree ، judgment

empty tree empty tree
---
node(empty, empty) tree
---
empty tree
---
node(empty, node(empty, empty)) tree

$$\frac{V_1 \quad \cdots \quad V_k}{J}$$

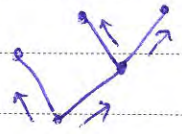
$V_1, \cdots, V_k$: derivations

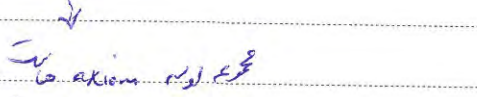


Bottom-up derivation

دو نوع استنتاج ماشینی :

Top-down (Backward chaining) ، Bottom-up (Forward chaining)





PqPCO

۵/

Subject

Date

in general

Semi-decidable ← ... semi-decidable ... in general

... Backward ... Goal ... Sub-goal ...

... in general

Rule Induction:

... $\frac{J_1 \quad \cdots \quad J_k}{J}$ ...

... judgment ...

... $P(J)$ whenever J is derivable (holds) ← (P holds on judgment J)

... judgment form ...

Induction Hypothesis — Induction Conclusion

$$\forall J_1, \ldots, J_k \cdot P(J_1) \wedge \cdots \wedge P(J_k) \Rightarrow P(J)$$

...

Subject APL
Date 99/1/10

مثال ـ برای نوشتن استقرایی خواص خاصیت P را ثابت کنیم و ثابت کنیم:

1) P(zero nat)

2) ∀a nat . P(a nat) ⇒ P(Succ(a) nat)

(Structural Induction on a nat Inference Rules)

Lemma – If a nat, then a=a nat.

P(a nat) ≜ a nat ⇒ a=a nat. (implies) — فرض Induction Hypothesis

∀a nat. P(a nat)

Thus, 1) P(zero nat) : پس استقرا برای قاعده nat ثابت کنیم. (تعریف استقرایی ثابت کنیم):

2) ∀a nat.(P(a nat) ⇒ P(Succ(a) nat))

proof 1) P(zero nat) ≜ zero nat ⇒ zero = zero nat ✓

vacuously holds!

چون این axiom قرائد مساوی است.

proof 2) حال فرض می‌کنیم ثابت کنیم پس با درست دانستن فرض، کلی ثابت می‌کنیم. این را ثابت کردن

تمرین: ثبت سراغ اثبات کلی که از فرض سوم و ششم استفاده کند.

P4PCO

Subject Induction Conclusion (فصل آزاد - هادی)
Date

$P(Succ(a)\ nat) \triangleq Succ(a)\ nat \Rightarrow Succ(a) = Succ(a)\ nat$

a nat $\xrightarrow{I.H.}$ a = a nat

① ② ③

مرحله ① را Taking apart (آوردن) می‌نامند. مرحله ③ را Building up می‌نامند (ساختن)

Iterated and Simultaneous Definitions

Example -

nil list

a nat b list

Cons(a;b) list

Subject APL

Date ۹۴/۷/۱۵

Example - مثال برای Simultaneous

zero Even

a Odd

Succ(a) Even

a Even

Succ(a) Odd

بنابراین Odd و Even عوض میشن با هم.

این یک تعریف استقرایی است.

تابع استقرایی:

تابع را به شیء میبریم

تابع sum را تعریف میکنیم.

Sum(a; b; c)

b nat

Sum(zero; b; b)

sum(a; b; c)

Sum(succ(a); b; succ(c))

Theorem - For every 'a nat' and 'b nat', there exists a unique 'c nat' such that Sum(a; b; c).

PdPCO ∃!

(یکتایی) uniqueness , there exists

۹/

Subject
Date

Proof - 1) Existence :

By induction on a nat

$P(a\ nat) \triangleq a\ nat \Rightarrow \forall b\ nat.\ \exists c\ nat.\ sum(a;b;c)$.

$P(zero\ nat) \triangleq zero\ nat \Rightarrow \forall b\ nat.\ \exists c\ nat.\ sum(zero;b;c)$

Take $c = b$. ✓

$P(a\ nat) \Rightarrow P(succ(a)\ nat)$

$succ(a)\ nat \Rightarrow \forall b\ nat.\ \exists c\ nat.\ sum(succ(a);b;c)$

(1) ↙ ↗ (2)

$a\ nat \Rightarrow \forall b\ nat.\ \exists c'\ nat.\ sum(a;b;c')$

I.H.

Take $c = succ(c')$

2) Uniqueness :

$P(a\ nat) \triangleq a\ nat \Rightarrow \forall b\ nat.\ \forall c_1\ nat.\ \forall c_2\ nat.\ sum(a;b;c_1) \wedge sum(a;b;c_2) \Rightarrow c_1 = c_2)$

PqPCO

Subject APL
Date ۹۲/۷/۱۵

Judgment → Categorical
↳ Hypothetical

Hypothetical Judgments:

Formalism

Context

Conclusion

1) Derivability 2) Admissibility

1) Derivability:

$$\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n \vdash_{\mathcal{R}} \mathcal{K}$$

(V-dash)

judgment

The form of a derivability judgment is as above.

Subject
Date

حکم K با فرض حصول $J_1, \dots, J_n$ را به کمک قواعد R_0 (مجموعه‌ی axiom و قواعد استنتاج) از آن نتیجه می‌گیریم.

از این مجموعه قاعده

$$R_0[J_1, \dots, J_n]$$

و

Γ را یک مجموعه از basic judgment می‌نامیم. هر basic judgment را با $J_1, \dots, J_n$ و

$$R_0[\Gamma]$$

نشان می‌دهیم.

$\Gamma \vdash_{R_0} K$ means that K is derivable from rules $R_0[\Gamma]$.

(اگر $\Gamma = \emptyset$، عبارت را Categorical می‌گوییم)

the rule $\dfrac{J_1 \quad \dots \quad J_n}{K}$ is derivable. : یعنی

اصطلاحاً یعنی خود قاعده باید را می‌توان از R_0 به دست آورد.

(یعنی حکم از خود قاعده!)

Example -

$$\frac{}{zero\ nat} \qquad (*)$$

$$\frac{a\ nat}{succ(a)\ nat}$$

این دو تا قاعده را داریم.

$a\ nat \vdash_{*} succ(succ(a))\ nat$

$$\frac{\dfrac{a\ nat}{succ(a)\ nat}}{succ(succ(a))\ nat}$$

(Gentzen)

Subject APL

Date ۹۷/۷/۱۵

ویژگی های همچون derivability :

Theorem (Stability) - If $\Gamma \vdash_R J$, then $\Gamma \vdash_{R \cup R'} J$.

اضافه شدن قواعد جدید به R باعث derivability judgment ها میشود.

(می توانید از قواعد جدید برای derivation استفاده کنید.)

Structural Proportion:

ویژگی های همه قواعد همچون derivability

- Reflexivity: $\Gamma, J \vdash_R J$

- Weakening: If $\Gamma \vdash_R J$, then $\Gamma, K \vdash_R J$.

اضافه کردن فرضیه های همه K استفاده نکنیم، همچنان اثبات برقرار است.

که در مقدمه های غیرضروری وزن دارد.

- Transitivity: If $\Gamma \vdash_R K$ and $\Gamma, K \vdash_R J$, then $\Gamma \vdash_R J$.

(Cut)

2) Admissibility Judgments:

(a weaker form)

$$\Gamma \models_R K \triangleq \vdash_R \Gamma \Rightarrow \vdash_R K$$

ارسطو از ویژگی ~~derivability~~ همچون R، Γ، K استفاده کرد، یعنی از روی قاعده R، K را نتیجه گرفت.

اگر همه فرضیات Γ، از قواعد R، derivable باشند، آنگاه judgment نیز derivable است!

(derivability is stronger than admissibility)

Subject
Date

Theorem - $\Gamma \vdash_R K$, then $\Gamma \models_R K$.

(derivability, transitivity)

$\Gamma = \{J_1, \dots, J_n\}$ $\quad \Gamma \vdash_R K \wedge J_2, \dots, J_n \vdash_R J_1 \Rightarrow J_2, \dots, J_n \vdash_R K$

$\vdash_R K$ ... $J_3, \dots, J_n \vdash_R K$

(Repeated application of the transitivity of derivability shows that, if $\Gamma \vdash_R J$ and $\vdash_R \Gamma$, then $\vdash_R J$.)

Example - $\mathrm{Succ}(a)$ nat $\models_*$ a nat.

Stable derivability, admissibility

Example - $* : \mathrm{nat}$

$\dfrac{}{\mathrm{Succ}(j\text{-}k)\ \mathrm{nat}}$

$\mathrm{Succ}(a)$ nat $\not\models$ a nat.

$*$ $\cup$ $\dfrac{}{\mathrm{Succ}(j\text{-}k)\ \mathrm{nat}}$

$\mathrm{Succ}(j\text{-}k)$ nat

Subject APL
Date ۹۹/۷/۱۵

(Transitivity ، Weakening ، (Reflexivity دارد) Structural Properties ✓

دارای Admissibility خواهد بود.

- Reflexivity: $\Gamma, J \vdash_R J$

- Weakening: $\Gamma \vdash_R J \Rightarrow \Gamma, K \vdash_R J$

- Transitivity: $(\Gamma, K \vdash_R J \wedge \Gamma \vdash_R K) \Rightarrow \Gamma \vdash_R J$

✓ حال برای بیان قواعد، تعریف استقرایی، استفاده می‌شود: (Hypothetical Inductive Definitions)

$$\frac{\Gamma\Gamma_1 \vdash J_1 \quad , \cdots , \quad \Gamma\Gamma_n \vdash J_n}{\Gamma \vdash J}$$

global context Γ

local context $\Gamma_1, \ldots, \Gamma_n$

هر قاعده دارای تعدادی فرض (صفر یا بیشتر) و یک نتیجه است.

✓ این قاعده Uniformity دارد (یعنی Γ در همه جا یکسان است) و Reflexivity، Transitivity، Weakening نیز دارد.

✓ این قاعده برای all for Γ برقرار است.

✓ این قاعده دارای Generic Judgment است.

PqPCO

(بقیه مطالب فصل ۲ و ۳ در ۵ برگه بعدی)

۲۵/

Subject APL
Date ۹۴/۷/۱۷

$\vec{x} \mid \Gamma \vdash J$: Generic Judgment

$\vec{x}$: a sequence of variables (identifiers)

(placeholder)

(Scope)

binding

let x be 2 in x+x

let x be x*x in x+x

let x be 2 in let x be x*x in x+x

α-renaming

let y be 2 in let x be y*y in x+x

abstract syntax tree

Sort

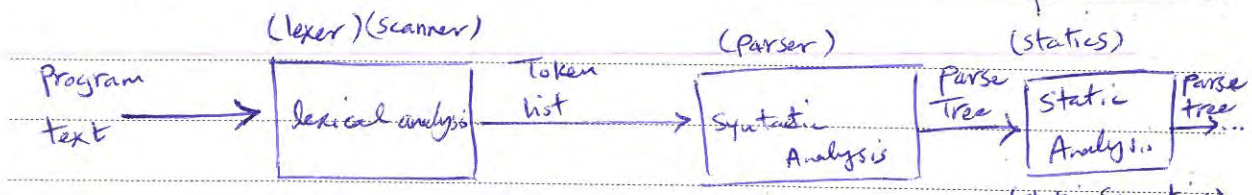
x+2

plus (x, num[2])

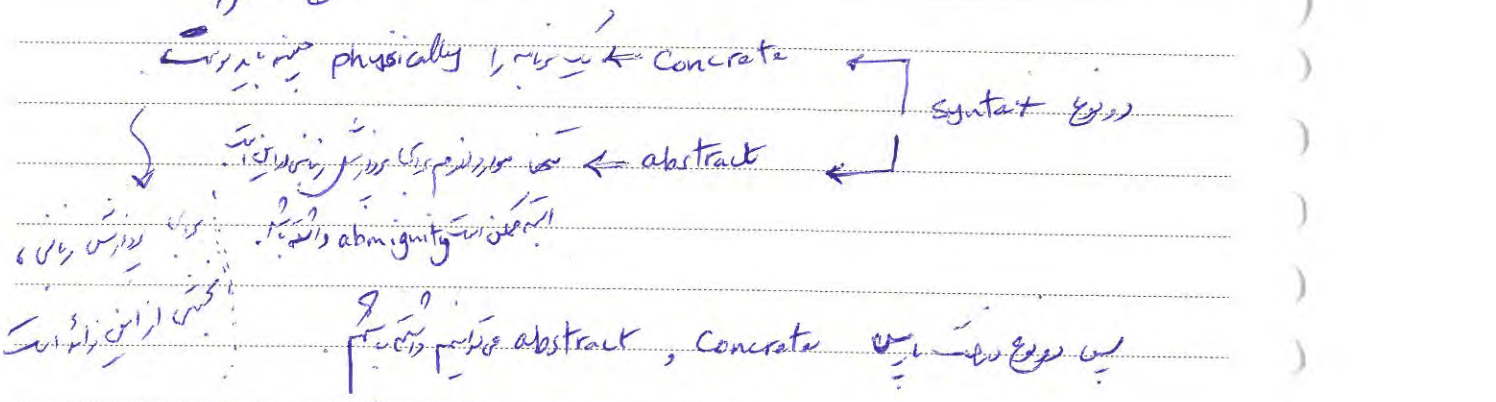
plus

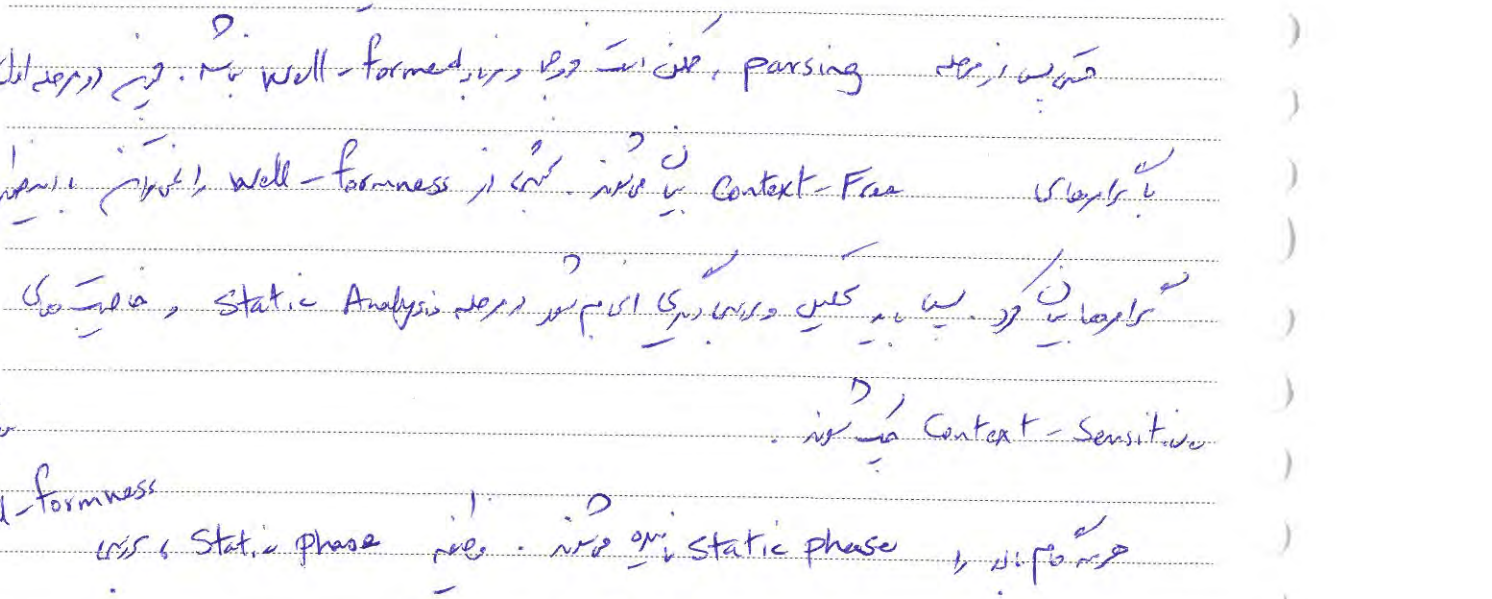
x num[2]

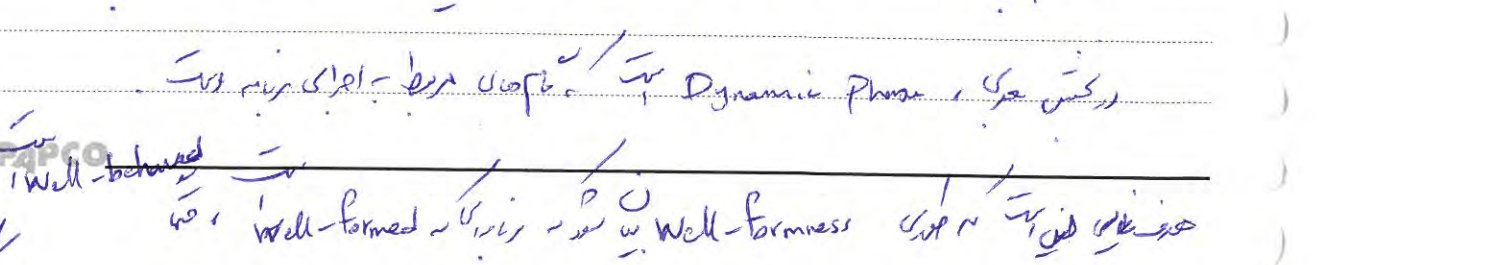
Subject APL
Date 99,7,17



smallest unit of significants







Subject

Date

Well-formed برای ، static

Safety برای ، well-behaved

(Stack overflow برای) safe

well-formedness ، well-behaved

(Safety)

Well-formed ⟹ Well-behaved

(Safety ثابت می‌کند که برنامه خوش‌رفتار باشد، خوش‌رفتاری ثابت می‌کند که برنامه درست باشد)

(model language) : running example

$\mathcal{L}\{\text{numstr}\}$

Sort یعنی syntactic object

Typ ، Exp

Typ $\tau ::=$

Exp $e ::=$

abstract ، form

Concrete .

Subject APL
Date 94/11/11

Sort		Abstract Form	Concrete Form
TYP	T ::=	num str	num str
EXP	e ::=	x	x
		num[n]	n
		str[s]	"s"
		plus(e_1; e_2)	$e_1 + e_2$
		times(e_1; e_2)	$e_1 * e_2$
		cat(e_1; e_2)	$e_1 \hat{} e_2$
		len(e_1)	$\lvert e \rvert$
		let(e_1; $x.e_2$)	let x be e_1 in e_2

(abstract syntax ... operator ...)

(EXP, EXP(EXP)) : ... let(e_1; $x.e_2$) sort ...

... EXP ... EXP ... EXP ...

... bind ...

(EXP, EXP) ... plus sort

times

cat

PAPCO

9/

Subject
Date

کتاب pierce برای خواندن type و مفهوم آن بسیار کتاب خوبی است.

Static Semantics اعم از کتب Typing است. Typing و Type System چیزی است که به phrase ها اعمال می‌شود (impose)

مثال)

plus (x ; num [2])

در عبارت بالا x باید از نوع number باشد.

در Type System این را چطور بنویسیم، این را باید با قوانین بنویسیم.

نوع عبارت‌های مورد prediction اعم از اطلاعات زمان اجرا. عبارت‌های مورد

این کار را انجام بدهیم که این کار undecidable است؛ این نوع عبارت‌ها قابل حل نیست.

در Type System و static Semantics از قوانین و یا عباراتی به نام rule استفاده می‌شود

(Typing rules)

$$\vec{x} \mid \Gamma \vdash e : \tau$$

context

$$\Gamma = \{ x:\tau, y:\tau', \dots \}$$

هر متغیر باید در Γ دقیقا یک بار (only once) تعریف شده باشد.

PaPCO

(در ادامه توانایی نوع دهی زبان را اضافه می‌کنیم: (توضیح زبان، چیزی که type annotation ها)

Subject APL
Date 9/4/V/IV

1) $\Gamma, x:\tau \vdash x:\tau$

2) $\Gamma \vdash str[s] : str$

3) $\Gamma \vdash num[n] : num$

4) $\dfrac{\Gamma \vdash e_1 : num \qquad \Gamma \vdash e_2 : num}{\Gamma \vdash plus(e_1; e_2) : num}$

5) $\dfrac{\Gamma \vdash e_1 : num \qquad \Gamma \vdash e_2 : num}{\Gamma \vdash times(e_1; e_2) : num}$

6) $\dfrac{\Gamma \vdash e_1 : str \qquad \Gamma \vdash e_2 : str}{\Gamma \vdash cat(e_1; e_2) : str}$

7) $\dfrac{\Gamma \vdash e : str}{\Gamma \vdash len(e) : num}$

8) $\dfrac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x:\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash let(e_1; x.e_2) : \tau_2}$

Subject
Date

let(num[5]; x . plus(x; num[1])) (مثال

Bottom-up

Top-down

(3) $x:num \vdash x:num$ —— (4) $x:num \vdash num[1]:num$ ——

(2) $\vdash num[5]:num$ —— $x:num \vdash plus(x;num[1]):num$ ——

(1) $\vdash let(num[5]; x.plus(x;num[1])):num$

$\Gamma = \emptyset$

(guess) حدس

از نوع‌بندی برای حدس type می‌گیریم، آن را typable یا well-typed می‌نامند.

Well-behaved برای int یا string است؟ این برای ما مهم است!

برای prediction به value یا type نیاز داریم؟ type

(Type Computation) محاسبات را انجام می‌دهد.

در این جلسه می‌خواهیم well-formed بودن statics را تعریف کنیم.

Type System: Machinery که well-form بودن را بررسی می‌کند.

PAPCO

این صفحه را برای آموزش خوش‌خط رها کنید!

Subject APL

Date ۹۷/۷/۲۲

Lemma (Unicity of Typing) – For every context Γ and expression e, there exists at most one τ such that Γ ⊢ e : τ.

Syntax-Directed

Bottom-Up

Lemma (Inversion for Typing) – Suppose that Γ ⊢ e : τ. If e = plus(e₁; e₂) then τ = num, Γ ⊢ e₁ : num, Γ ⊢ e₂ : num.

structural properties

Subject
Date

Lemma (Weakening) - If $\Gamma \vdash e' : \tau'$, then $\Gamma, x:\tau \vdash e' : \tau'$.

Proof - Induction on the typing judgment $\Gamma \vdash e' : \tau'$.

Rule

$P(\Gamma \vdash e' : \tau') \triangleq \Gamma \vdash e' : \tau' \Rightarrow \Gamma, x:\tau \vdash e' : \tau'$.

a fresh variable

Base case (axiom) $P(\Gamma, x:\tau \vdash x:\tau) \triangleq \Gamma, x:\tau \vdash x:\tau \Rightarrow \Gamma, x:\tau, y:\tau \vdash x:\tau$.

(Trivially)

$P(\Gamma \vdash e_1 : num) \wedge P(\Gamma \vdash e_2 : num) \Rightarrow P(\Gamma \vdash plus(e_1; e_2) : num)$

$\triangleq \Gamma \vdash plus(e_1; e_2) : num \Rightarrow \Gamma, x:\tau \vdash plus(e_1; e_2) : num$

Take apart

$\Gamma \vdash e_1 : num \wedge \Gamma \vdash e_2 : num$

I.H. ↓ I.H. ↓

$\Gamma, x:\tau \vdash e_1 : num$ $\Gamma, x:\tau \vdash e_2 : num$

$\Rightarrow \Gamma, x:\tau \vdash plus(e_1; e_2) : num$

Subject APL
Date ۹۹/۷/۲۲

برای let به صورت:

$$P(\Gamma \vdash e_1 : \tau_1) \wedge P(\Gamma, x:\tau_1 \vdash e_2 : \tau_2) \Rightarrow P(\Gamma \vdash let(e_1; x.e_2) : \tau_2)$$

$$\triangleq \Gamma \vdash let(e_1; x.e_2) : \tau_2 \Rightarrow \Gamma, y:\tau \vdash let(e_1; x.e_2) : \tau_2$$

Taking apart

$$\Gamma \vdash e_1 : \tau_1 \wedge \Gamma, x:\tau_1 \vdash e_2 : \tau_2$$

I.H. ⇓ I.H. ⇓

$$\Gamma, y:\tau \vdash e_1 : \tau_1 \qquad \Gamma, x:\tau_1, y:\tau \vdash e_2 : \tau_2$$

$$\Rightarrow \Gamma, y:\tau \vdash let(e_1; x.e_2) : \tau_2$$

(Rule-up)

Lemma (Substitution) — If $\Gamma, x:\tau \vdash e' : \tau'$ and $\Gamma \vdash e : \tau$, then

$\Gamma \vdash [e/x]e' : \tau'$.

↓

Generic Typing Judgment ← Structural

e' و x و e هر سه متغیر، τ و τ' هم متغیر

Substitute e for x in e' for every free variable x.

Modularity and linking: این دو ویژگی این زبان است

PoPCO

(abstraction) برای ماژولاریتی لازم است بعضی از متغیرها آزاد باشند

و در زمان linking و پیاده‌سازی نیاز است آنها را جایگزاری کنیم

Subject
Date

در مقابل از linking time ، ثابت نگه داشته می‌شود ، زمان linking می‌کند

e ، e' ، client ، implementation ، e' ، e ، D

(int به float ، int به عدد + ، عدد صحیح) implementation

$[x \mapsto e]e'$ ، $[e/x]e' : \tau'$: substitution روی عبارات

Proof - Induction on the derivation of $\Gamma, x:\tau \vdash e' : \tau'$.

(Induction on Proof)

↳ [اثبات روی ساختار درخت اثبات]

$\Phi(\Gamma, x:\tau \vdash e' : \tau') \triangleq \Gamma, x:\tau \vdash e' : \tau' \wedge \Gamma \vdash e : \tau$

$\Rightarrow \Gamma \vdash [e/x]e' : \tau'$.

Proof - Exercise!

Lemma (Decomposition) - If $\Gamma \vdash [e/x]e' : \tau'$, then for every type τ such that $\Gamma \vdash e : \tau$, we have $\Gamma, x:\tau \vdash e' : \tau'$.

(برعکس substitution است ، یعنی از implementation ، abstraction می‌سازیم)

Subject APL
Date ۹۷/۷/۲۴

eliminatory و introductary ترجمه‌ی : ... ۵

Values (Canonical forms)

... ۸

Dynamic Semantics ، Dynamics ... ۴

Dynamics :

how programs are to be executed.

Transition System ... Structural Dynamics

Dynamics ...

Subject
Date

(PCF مثل) Plotkin روش‌های معنای ← از

Abstract Machine یا Transition System

Command و program و state

Equational Dynamics ، Contextual Dynamics structural dynamics

Judgment Transition System

1) s state
2) s final
3) s initial
4) $s \mapsto s'$

Final state

s final $\Rightarrow \nexists s'$ state $. s \mapsto s'$

stuck state

$\nexists s'$ state $. s \mapsto s'$

s' is said to be 'stuck'.

stuck final

Subject APL

Date ۹۹/۷/۲۴

$\forall s$ state . $\exists^{\leq 1}$ s' state . $s \mapsto s'$

↓

Deterministic

otherwise it is non-deterministic

Transition Sequence is a sequence of states

$s_0, s_1, \dots, s_n$

such that

s_0 initial and $s_i \mapsto s_{i+1}$ for every $0 \leq i < n$.

A maximal transition sequence is a transition sequence with

$\nexists s$ state . $s_n \mapsto s$.

↓

Stuck (در state)

A complete transition sequence is a ~~Transition sequence~~ maximal one such that

s_n is a final state.

$s\downarrow$: There is a complete transition starting from s.

۹

Subject
Date

Iteration of transition judgment = $s \mapsto^* s'$
(multi-step transition)

$$\frac{}{s \mapsto^* s}$$

$$\frac{s \mapsto s' \quad s' \mapsto^* s''}{s \mapsto^* s''}$$

K-times step transition:
(K-times-iterated transition judgment)

$$\frac{}{s \mapsto^0 s}$$

$$\frac{s \mapsto s' \quad s' \mapsto^k s''}{s \mapsto^{k+1} s''}$$

Theorem - For all states s and s', $s \mapsto^* s'$ iff $s \mapsto^k s'$ for some $k \geq 0$.

Proof - By induction on Rules.

Subject APL
Date ۹۹/۷/۱۴

free variable ... abstractor binder

1) states: closed expressions

2) final states: (closed) values

open value

numstr to value

$$\frac{}{num[n]\ val}$$

$$\frac{}{str[s]\ val}$$

3) initial states: all states

4) Transition:

$$\frac{n_1 + n_2 = n \quad nat}{plus(num[n_1]; num[n_2]) \mapsto num[n]} \quad (*)$$

$$\frac{e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)}$$

$$\frac{e_1\ val \qquad e_2 \mapsto e_2'}{plus(e_1; e_2) \mapsto plus(e_1; e_2')}$$

۵۱/

(pierce [author])

Subject: a typed arithmetic expression

Date

(cat)

$$\frac{s_1 \hat{} s_2 = s \quad str}{cat(str[s_1]; str[s_2]) \mapsto str[s]} \quad (*)$$

$$\frac{e_1 \mapsto e_1'}{cat(e_1; e_2) \mapsto cat(e_1'; e_2)}$$

$$\frac{e_1 \ val \qquad e_2 \mapsto e_2'}{cat(e_1; e_2) \mapsto cat(e_1; e_2')}$$

$$\frac{e_1 \mapsto e_1'}{let(e_1; x.e_2) \mapsto let(e_1'; x.e_2)}$$

$$\frac{e_1 \ val}{let(e_1; x.e_2) \mapsto [e_1/x]e_2} \quad (*)$$

Example -

$$let(plus(num[1]; num[2]); x.\ plus(plus(x; num[3]); num[4]))$$

$$\frac{\dfrac{1+2=3 \quad nat}{plus(num[1]; num[2]) \mapsto num[3]}}{let(plus(num[1]; \dots)) \mapsto let(num[3]; x.\ plus(plus(\dots)) \longmapsto} \quad num[3]\ val$$

$$[num[3]/x](plus(plus(x; num[3]); num[4])) \mapsto plus(num[6]; num[4]) \mapsto^{num[6]}$$

Subject APL
Date ۹۴/۱۰/۲۴

قواعدی به ترتیب (*) جزء نیست ، اصطلاح instruction transition نشان‌دهنده یکی قواعد

قاعده نیست Search Transition

که حاوی دنبال ادامه ترتیب قابل حذف است.

نیست قوانین تعریف نیستند اولا حذف است.

قاعده By-value dynamics جزئی است که دو قاعده اخیر حذف شده است. یعنی اول تعریف ادامه

تعریف نمی‌شود.

که در By-name dynamics است ⟸ معادل قاعده نیست:

$let(e_1; x.e_2) \mapsto [e_1/x]e_2$

در اینجا از By-value استفاده می‌شود.

(Contextual Semantics ، یا Dynamics) که استاتیک برای operational Semantics ، hole

هر استیت را الزامی تعریف کنیم ، می‌خواهد از آن

قاعده middle-weight است که در Contextual Semantics جزء

Subject:
Year. Month. Date. ۹۶/۷/۲۹

فصل مرتبط

در فصل سوم کتاب (Untyped Arithmetic Exp) pierce

در فصل چهارم کتاب (An ML implementation of —) pierce

در این فصل دستور interpreter برای زبان ML ارائه شده است.

به * ها ... instruction transition گفته می‌شود ← یعنی واقعاً یک گام محاسبه انجام شود.
(primitive computation)

به لیست search transition گفته می‌شود ← برای اینکه محاسبه کنیم، ترتیب را بگیرد باید در مورد ها rule بکار بگیرید.

اگر نوع rule ——— ودر دینامیک (by-name dynamics)
$let(e_1 ; x . e_2) \longmapsto [e_1/x]e_2$
نام دارد.

$$\frac{e_1 \longmapsto e'_1}{plus(e_1 ; e_2) \longmapsto plus(e'_1 ; e_2)} \longleftarrow$$
تا جایی e_1 ساده می‌شود ساده کنید.

$$\frac{e_1 \; val \quad e_2 \longmapsto e'_2}{plus(e_1 ; e_2) \longmapsto plus(e_1 ; e'_2)} \longleftarrow$$
وقتی e_1 به نتیجه رسید برو در سراغ e_2

Lemma (finality of values)

For no expression e do we have both e val and $e \longmapsto e'$ for some e'.

Subject:
Year. Month. Date. ( )

Lemma (Determinacy)

If $e \mapsto e'$ and $e \mapsto e''$ then e' and e'' are α-equivalent.

↳ یعنی اگر نام bound-variable را عوض کنیم هردو e' و e'' یکی میشوند.

مثال

$$\begin{cases} \text{let } x \text{ be } 2 \text{ in } x+2 \\ \text{let } y \text{ be } 2 \text{ in } y+2 \end{cases} \equiv_\alpha \quad (\alpha\text{-equivalent})$$

← چگونگی اثبات با استقرا

$P(e \mapsto e') \triangleq e \mapsto e' \wedge e \mapsto e'' \Longrightarrow e' \equiv_\alpha e''$

در مراحل اثبات این استقرا کمتر است استقرا درون استقرا روی دو عدد.

$\forall\ m, n \in \mathbb{N} \quad P(m, n)$

$\forall m \in \mathbb{N} \Longrightarrow \forall n \in \mathbb{N} \quad P(0, n)$

Contextual Dynamics (Contextual Operational Semantics)

یعنی گام به گام انجام میگیرد.

برای Contextual Dynamics یک middle weight Java (MJ) Semantics نوشته‌اند.

یک document 150 صفحه‌ای است.

در Contextual dynamic قبلی تعریف داریم:

بگذارید:

$$\frac{n_1 + n_2 = n \quad nat}{plus(num[n_1]; num[n_2]) \longrightarrow num[n]}$$

$$\frac{s_1 \hat{} s_2 = s \; str}{cat(str[s_1]; str[s_2]) \longrightarrow str[s]}$$

$$\frac{}{let(e_1; x.e_2) \longrightarrow [e_1/x]e_2}$$

به جای search transition، تعریف evaluation context می‌پردازیم:

یعنی محلی که قرار است بایکی از سه rule بالایی کار شود.

یک evalutionatxt است که قرار است محل بعدی دستور کار باشد:

$$\mathcal{E} \; ectxt \quad (evaluation \; Context)$$

"Hole" $\circ$

یک $\mathcal{E}$ را تعریف می‌کنیم (جایی که محاسبه باید انجام گردد).

یعنی $\mathcal{E}_1$ داری یک حفره است:

$$\frac{}{\circ \; ectxt} \qquad \frac{\mathcal{E}_1 \; ectxt}{plus(\mathcal{E}_1; e_2) \; ectxt}$$

$$\frac{e_1 \; val \quad \mathcal{E}_2 \; ectxt}{plus(e_1; \mathcal{E}_2) \; ectxt}$$

یعنی اگر یک عبارت بزرگ داشته باشیم

می‌توانیم با استفاده از این rule ها، کل بخش حفره دار محاسبه را مشخص کرد.

Subject:
Year. Month. Date. ( )

$e' = \mathcal{E}\{e\}$

The expression e' is the result of filling the hole in the evaluation context $\mathcal{E}$ with the expression e.

بصورت استقرایی آن را تعریف می‌کنیم:

$$\frac{}{e = 0\{e\}} \qquad \frac{e_1 = \mathcal{E}_1\{e\}}{plus(e_1; e_2) = plus(\mathcal{E}_1; e_2)\{e\}}$$

$$\frac{e_1 \ val \quad e_2 = \mathcal{E}_2\{e\}}{plus(e_1; e_2) = plus(e_1; \mathcal{E}_2)\{e\}}$$

The Contextual Dynamics for $\mathcal{L}\{num\ str\}$ is defined by a single rule

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \to e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \longmapsto e'} \ (1) \qquad \frac{e_0 \to e_0'}{\mathcal{E}\{e_0\} \to \mathcal{E}\{e_0'\}}$$

هر گام را قبل، از نوع گام‌هستند تری judgement ی را تعریف کنیم.
(شناسی!)

- هرگاه خواستی یک عبارت را در یک گام ساده کنی

مثال:

$$plus(num[1]; plus(num[2]; num[3])) \overset{(1)}{=} plus(num[1]; 0)\{plus(num[2]; num[3])\}$$

$$plus(num[2]; num[3]) \overset{(2)}{\to} num[5] \qquad plus(num[1]; num[5]) \overset{(3)}{=} plus(num[1]; 0)\{num[5]\}$$

Subject:
Year. Month. Date. ( )

همه قبلی ها، بالای خط Judgement بود حال زیر خط Judgement ↙

---

plus (num[1] ; plus (num[2] ; num[3]) ⟼ plus (num[1] ; num[5])

- میتوانیم به جای تمرین آن let Rule را با Structural به کردیم را بازگیر با Contextual انجام دهیم.

Contextual یعنی $\longmapsto_C$ یعنی استقرار از

Structural یعنی $\longmapsto_S$ یعنی استقرار از

Theorem: $e \longmapsto_S e'$ if and only if $e \longmapsto_C e'$.

در اصطلاح این ۲ سمنتیک، coincide کردند یعنی هم هستند.

- تمام قضایایی که اثبات نکردیم جزو تمرین ها هستند که باید انجام دهیم.

{ Domain Theory, Axiomatic (کتاب Winskel)
Denotational Semantics ← مقاله ایشان

Equational Dynamics (Definitional Semantics)

یعنی برخواست) دو یک برنامه داده شده معادل چه چیزی است

(کمک اتمادهای جبری حل)

Subject APL

Date ۹۴/۸/۱

Equational Dynamics:

زبان‌های برنامه‌سازی را می‌توان از دیدگاه Definitional Semantics یعنی dynamics تعریف کرد.

این دیدگاه خود به دو دسته Operational Semantics و Axiomatic Semantics تقسیم می‌شود.

در این دیدگاه می‌خواهیم بفهمیم که دو برنامه معادل هستند یا خیر. برای این کار باید مفهوم برنامه‌های معادل را تعریف کنیم.

برنامه‌های معادل ← Structural

چه زمانی دو عبارت با هم معادل هستند؟

$$p \to (q \to \neg p) \Leftrightarrow \neg p \vee (q \to \neg p)$$
$$\Leftrightarrow \neg p \vee (\neg q \vee \neg p)$$
$$\Leftrightarrow (\neg p \vee \neg p) \vee \neg q$$
$$\Leftrightarrow \neg p \vee \neg q$$
$$\Leftrightarrow \neg(p \wedge q)$$
$$\Leftrightarrow \dots$$

این Judgment در Equational Dynamics به صورت زیر است:

$$\Gamma \vdash e \equiv e' : \tau$$

در این حالت e از نوع τ است، e' از نوع τ است، و e و e' معادل هستند.

we tacitly assume that $\Gamma \vdash e : \tau$, $\Gamma \vdash e' : \tau$.
(implicitly)

Subject APL
Date ۹۴,۸,۱

Equational Dynamics

Equivalence

Reflexivity, Symmetric, Transitivity

$$\frac{}{\Gamma \vdash e \equiv e : \tau} \text{(ref)}$$

$$\frac{\Gamma \vdash e' \equiv e : \tau}{\Gamma \vdash e \equiv e' : \tau} \text{(sym.)}$$

$$\frac{\Gamma \vdash e \equiv e' : \tau \quad \Gamma \vdash e' \equiv e'' : \tau}{\Gamma \vdash e \equiv e'' : \tau} \text{(trans)}$$

Congruence

Congruence : A relation that is compatible with all expression forming constructs.

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \text{num} \quad \Gamma \vdash e_2 \equiv e_2' : \text{num}}{\Gamma \vdash \text{plus}(e_1; e_2) \equiv \text{plus}(e_1'; e_2') : \text{num}}$$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \text{str} \quad \Gamma \vdash e_2 \equiv e_2' : \text{str}}{\Gamma \vdash \text{cat}(e_1; e_2) \equiv \text{cat}(e_1'; e_2') : \text{str}}$$

P4PCO

Subject
Date

$\Gamma, x:\tau_1$

$$\frac{\Gamma \vdash e_1 \equiv e_1' : \tau_1 \qquad \Gamma \vdash e_2 \equiv e_2' : \tau_2}{\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv \mathtt{let}(e_1'; x.e_2') : \tau_2}$$

این رابطه یک congruence است.

: به primitive constructor ها (۲

$$\frac{n_1 + n_2 = n \quad \mathsf{nat}}{\Gamma \vdash \mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \equiv \mathtt{num}[n] : \mathtt{num}}$$

$$\frac{s_1 \hat{} s_2 = s \quad \mathsf{str}}{\Gamma \vdash \mathtt{cat}(\mathtt{str}[s_1]; \mathtt{str}[s_2]) \equiv \mathtt{str}[s] : \mathtt{str}}$$

$$\Gamma \vdash \mathtt{let}(e_1; x.e_2) \equiv [e_1/x]e_2 : \tau$$

primitive constructs / equational dynamics / این ها کمک می‌کنند

(*) The strongest congruence closed under rules of primitive constructs.

$$\mathtt{let}(\mathtt{plus}(\mathtt{num}[1]; \mathtt{num}[2]); x.\ \mathtt{plus}(x; \mathtt{plus}(\mathtt{num}[3]; \mathtt{num}[4]))) \equiv \mathtt{num}[10] : \mathtt{num}$$

Subject APL
Date 94,11,1

$1+2 = 3 \quad nat$ (1)

$plus(num[1]; num[2]) \equiv num[3] : num$

Context

(2)

$x : \tau_1 \vdash plus(x; plus(num[3]; num[4])) \equiv plus(x; plus(num[3]; \ldots)) : num$

(1) (2)

$let(plus(num[1]; num[2]); x.\, plus(x; \ldots)) = let(num[3]; x.\, plus(x; \ldots)) : num$

(ثابت)

Definitional equality (Symbolic Evaluation) تعریف

Theorem — $e \equiv e' : \tau$ iff there exists e_0 val such that $e \mapsto^* e_0$ and $e' \mapsto^* e_0$.

(این قضیه رابطه بین structural Semantics و Equational را بیان می‌کند)

$e \equiv e'$ یعنی $e_0 \equiv e'$ و $e_0 \equiv e$ برای یک e_0

PdPCO

If $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e \equiv e' : \tau$, then whenever $e_1 : \tau_1, \ldots, e_n : \tau_n$,
$[e_1, \ldots, e_n / x_1, \ldots x_n] e \equiv [e_1, \ldots, e_n / x_1, \ldots, x_n] e' : \tau \Rightarrow$

37

Subject

Date

Cost semantics ←

Small step semantics ، Operational semantics ، Big step semantics

$\exists e_0 \; val. \quad [e_1, \ldots, e_n / x_1, \ldots, x_n] e \mapsto^* e_0$

$\wedge$

$[e_1, \ldots, e_n / x_1, \ldots, x_n] e' \mapsto^* e_0$

closed ، Context free

$\vdash e = e' : \tau$

Semantics

Type Safety :

strongly typed ، type Safe ، Safe

language ← semantics

program ← meaning

Run time ، Compile time ، Safety

Type System

PAPCO

ML ، Safety run-time ، lisp

Subject Apr
Date ۹۴/۸/۱

Safety می‌شود pierce

high level : Safe

Abstraction

mid level : Safe

Abstract Data Type

abstract data type

low level

stuck

progress

Type System

Type Safe

well-behaved ، well-typed (Type Safety)

(well-formed)

Subject Advanced Programing languages

Date ۱۳۹۹/۸/۵

Safety

Well-formed (Syntactic) ⟹ Well-behaved (Semantic)

Concat, plus, number و String ← تعریف عمل روی اعداد

(illegal operation)

null pointer dereferencing ←

multiple thread starting ←

functional و non-functional ←

Static, Dynamic, Strong, Weak

Subject APL
Date ۹۴/۸/۵

*Meta : حتی نوع داده را وسترنگ می‌کند.

این بخش از مبحث را حتماً کمی دقت بیشتر کنید.

در بعضی موارد ما نیاز به Type Computation داریم. مثلاً نوع یک تابع به ورودی‌اش بستگی دارد.

در این موارد Type ها را به صورت یک تابع در نظر می‌گیریم.

Type System ها برای Enforcement, verification و Modal Logic ها استفاده می‌شوند.

این مبحث را می‌توان light-weight بررسی کرد.

(Type Industry در ۶۴ بیت)

Type Safety = Preservation + Progress

برای اثبات Safety باید Progress و preservation را بررسی کنیم.

preservation: اگر یک term یک type داشته باشد و یک مرحله اجرا شود، حاصل هم type دارد.

Progress: یک term که type دارد یا value است یا می‌تواند یک مرحله اجرا شود.

$e : \tau \wedge e \mapsto e' \Rightarrow e' : \tau$

اثبات این موارد در جلسه بعد می‌آید.

$e : \tau \Rightarrow e \text{ val} \vee \exists e'.\ e \mapsto e'$ (or)

P4PCO

Subject

Date

typeable ، well-typed ، type

stuck typeable

state value state

Typeable ≠ ill-typed
(well-typed)

(*) well-typed states do not get stuck.

Theorem - If $e : \tau$ and $e \mapsto e'$, then $e' : \tau$.
(preservation)

Proof - (By induction)
0- Transition Rules

$$\frac{e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)}$$

Assume that $plus(e_1; e_2) : \tau \Longrightarrow e_1 : num \wedge e_2 : num \wedge \tau : num$
(inversion lemma, typing)

$\Longrightarrow e_1' : num$ (I.H.)

$\Longrightarrow plus(e_1'; e_2) : num$

Subject Apl
Date ۹۶/۸/۵

$let(e_1; x.e_2) \mapsto [e_1/x]e_2$

$let(e_1; x.e_2) : \tau_2 \xrightarrow[\text{lemma}]{\text{inversion}} e_1 : \tau_1 \wedge x : \tau_1 \wedge e_2 : \tau_2$

$\xrightarrow[\text{lemma}]{\text{substitution}} [e_1/x]e_2 : \tau_2$

Lemma (Canonical forms) - If e val and $e : \tau$, then

1. if $\tau = num$, then $e = num[n]$ for some number n.

2. if $\tau = str$, then $e = str[s]$ for some string s.

Theorem (Progress) - If $e : \tau$, then either e val or there exists e' such that $e \mapsto e'$.

proof - (By induction on typing derivation $e : \tau$)

$P(e : \tau) \triangleq e : \tau \Rightarrow e \text{ val} \underset{\text{or}}{\vee} \exists e'. e \mapsto e'$

$$\frac{e_1 : num \qquad e_2 : num}{plus(e_1; e_2) : num}$$

Subject
Date

Case 1: if $e_1 \mapsto e_1'$, then $plus(e_1; e_2) \mapsto plus(e_1'; e_2)$

Case 2: if e_1 val → if $e_2 \mapsto e_2'$ then $plus(e_1; e_2) \mapsto plus(e_1; e_2')$

→ if e_2 val then $e_1 = num[n_1]$ and $e_2 = num[n_2]$

↓ and

Canonical form → $plus(e_1; e_2) \mapsto num[n_1 + n_2]$

type-safe, $\vdash$ humistr ?

dynamics

stuck

stuck

safety

dynamics

PdPCO

Type System dynamics

Subject APL
Date ۹۴/۸/۵

Runtime Errors

چطور می‌توان با وجود خطاهای زمان اجرا، safety را تضمین کنیم.

فرض کنیم به زبان L{num str} ، عمل تقسیم هم اضافه کنیم.

$$\frac{e_1 : num \quad e_2 : num}{div(e_1 ; e_2) : num}$$

طبق این قاعده: div(num[3] ; num[0]) : num صحیح است اما در زمان اجرا stuck می‌شود.

دو راه داریم: ۱) تغییر type system بطوریکه هیچ زمان تقسیم بر صفر نداشته باشیم.
← که پیچیده است
به سادگی صفر بودن مخرج را نمی‌توان تشخیص داد

۲) اضافه کردن dynamic check

← که در زمان اجرا چک کند که آیا تقسیم بر صفر رخ داده است error به عنوان خروجی برگرداند.

Error های زمان اجرا به دو دسته checked و unchecked تقسیم می‌شوند.

unchecked ← مربوط به type system است.

checked ← خطاهای زمان اجرا

در قواعد [illegible] و evaluation

PAPCO

Subject
Date

error.e

اینا میسا : هدف error حاصل نمیشه. یعنی specify

$$(1.)\quad \frac{e_1 \ \text{val}}{\text{div}(e_1; \text{num}[0]) \ \text{err}} \rightarrow \text{error state}$$

$$(2)\quad \frac{e_1 \ \text{err}}{\text{plus}(e_1; e_2) \ \text{err}}$$

$$(3)\quad \frac{e_1 \ \text{val} \qquad e_2 \ \text{err}}{\text{plus}(e_1; e_2) \ \text{err}}$$

type system

$$\frac{}{\Gamma \vdash \text{error} : \tau}$$

dynamics

$$(4.)\quad \frac{}{\text{error} \ \text{err}}$$

Theorem (Program with error)

$$e : \tau \Rightarrow e \ \text{val} \ \underset{\text{or}}{\vee} \ e \ \text{err} \ \vee \ \exists e'.\ e \mapsto e'.$$

Evaluation Dynamics : (Big-step Semantics)

Subject APL

Date ۹۴/۸/۱۸

$e \Downarrow v$

(down bar) (evaluates.)

$$\frac{}{num[n] \Downarrow num[n]}$$

$$\frac{}{str[s] \Downarrow str[s]}$$

$$\frac{e_1 \Downarrow num[n_1] \qquad e_2 \Downarrow num[n_2] \qquad n_1 + n_2 = n \quad nat}{plus(e_1; e_2) \Downarrow num[n]}$$

$$\frac{e_1 \Downarrow str[s_1] \qquad e_2 \Downarrow str[s_2] \qquad s_1 \hat{} s_2 = s \quad str}{cat(e_1; e_2) \Downarrow s}$$

$$\frac{[e_1/x]e_2 \Downarrow v_2}{let(e_1; x.e_2) \Downarrow v_2}$$

$$\frac{e_1 \Downarrow v_1 \qquad [v_1/x]e_2 \Downarrow v_2}{let(e_1; x.e_2) \Downarrow v_2}$$

Subject

Date

Lemma - If $e \Downarrow v$, then v val.

Proof - Induction on $e \Downarrow v$. (It is clear!)

Theorem - For all closed expressions e and values v,

$$e \mapsto^* v \text{ iff } e \Downarrow v.$$

Proof - Right-to-Left: $e \Downarrow v \Rightarrow e \mapsto^* v$.

(Induction on $e \Downarrow v$)

As an example:

$$plus(e_1; e_2) \Downarrow num[n] \Rightarrow plus(e_1; e_2) \mapsto^* num[n]$$

$$\text{I.H. } P(e_1 \overset{num[n_1]}{\Downarrow}) \triangleq e_1 \overset{num[n_1]}{\Downarrow} \Rightarrow e_1 \mapsto^* num[n_1]$$

$$P(e_2 \overset{num[n_2]}{\Downarrow}) \triangleq e_2 \overset{num[n_2]}{\Downarrow} \Rightarrow e_2 \mapsto^* num[n_2]$$

$$plus(e_1; e_2) \mapsto^* plus(num[n_1]; e_2) \mapsto^* plus(num[n_1]; num[n_2]) \mapsto num[n]$$

(structural)

Subject APL

Date

Left-to-Right: $e \mapsto^* v \Rightarrow e \Downarrow v$

: ($\mapsto^*$ Transitive, Reflexive ...)

$P(e \mapsto^* v) \triangleq e \mapsto^* v \Rightarrow e \Downarrow v$

Induction on Ref and Trans closure $\mapsto$ ($\mapsto^*$)

Rules

instantiation

$$\frac{}{v \mapsto^* v}$$

$$\frac{e \mapsto e' \quad e' \mapsto^* v}{e \mapsto^* v}$$

1) $v \mapsto^* v \Rightarrow v \Downarrow v$

Trivial! holds w/o any assumptions

2) $e \mapsto e' \wedge P(e' \mapsto^* v) \Rightarrow P(e \mapsto^* v)$

N.I. (#)

Thus

$e \mapsto e' \wedge e' \Downarrow v \dashrightarrow e \Downarrow v$

?

$Q(e \mapsto e') \triangleq e \mapsto e' \wedge e' \Downarrow v \Rightarrow e \Downarrow v$

PdPCO

Subject
Date

$e_1 \mapsto e_1'$

$plus(e_1; e_2) \mapsto plus(e_1'; e_2)$

$Q(plus(e_1; e_2) \mapsto plus(e_1'; e_2)) \triangleq (plus(e_1; e_2) \mapsto plus(e_1'; e_2) \wedge plus(e_1'; e_2) \Downarrow v) \Rightarrow plus(e_1; e_2) \Downarrow v.$

I.H. : $Q(e_1 \mapsto e_1') \triangleq e_1 \mapsto e_1' \wedge e_1' \Downarrow v_1 \Rightarrow e \Downarrow v_1.$

(I) $e_1 \mapsto e_1'$: مبنای inversion

: مبنای inversion

(II) $e_1' \Downarrow num[n_1]$

$\wedge$

(III) $e_2 \Downarrow num[n_2]$

$\wedge$

(3) $n_1 + n_2 = n$ nat

$\wedge$

n nat

(I), (II) $\xrightarrow{I.H.}$ $e_1 \Downarrow num[n_1]$ (1)

(I), (III) $\xrightarrow{I.H.}$ $e_2 \Downarrow num[n_2]$ (2)

(1), (2), (3) $\rightarrow plus(e_1; e_2) \Downarrow num[n]$

Subject APL
Date 94/11/18

Type Safety برای structural Dyn. و Big-step Dyn. بررسی

preservation: $e : \tau \wedge e \Downarrow v \Rightarrow v : \tau$

progress: $e : \tau \Rightarrow e \Downarrow v$

(a naive attempt!)

more than required

value

Big-step Semantics

progress cannot be discussed correctly in the context of Big-step.

Dynamic Type Errors

Subject
Date

راه حل اول: با مدل Big-step ی program ی.
راه حل دوم: با تعریف Goes Wrong

Dynamic Type Error ⇑
(Goes Wrong)

$$\frac{}{\mathtt{plus}(\mathtt{str}[s]; e_2) \Uparrow}$$

$$\frac{e_1 \ \mathtt{val}}{\mathtt{plus}(e_1; \mathtt{str}[s]) \Uparrow}$$

Explicit checks for Dynamic Type Errors

⋮

Expressiveness ... Runtime ... Error ...

... Structural Dyn ... Big-step ...

Safety ... safety ...

Theorem - If $e\Uparrow$, then there is no τ such that $e:\tau$.

$$e:\tau \Longrightarrow \underset{\text{not}}{\neg}(e\Uparrow)$$

... program ...

Subject APL
Date ۹۹/۸/۸

Cost Dynamics :

num [n] ⇓° num [n]

str [s] ⇓° str [s]

Subject

Date 99/8/12

$$\frac{e_1 \Downarrow^{k_1} num[n_1] \qquad e_2 \Downarrow^{k_2} num[n_2] \qquad n_1 + n_2 = n \; nat}{plus(e_1; e_2) \Downarrow^{k_1+k_2+1} num[n]}$$

⋮

(By-Value) let

$$\frac{e_1 \Downarrow^{k_1} v_1 \qquad [v_1/x]e_2 \Downarrow^{k_2} v_2}{let(e_1; x.e_2) \Downarrow^{k_1+k_2+1} v_2}$$

(Operational Dyn. Semantics)

* Turing Complete Recursion Recursion function

Gödel , Functional Type

(System T)

plotkin

built-in

(primitive) (base)

head , tail

type

nonfunctional

Subject APL
Date 97/8/14

(فرمولهای ما - جاری)

Function Types:

...

... specify ... Abstraction ...

... function ... Rapply ... Rnumstrfun ...

... extension ... Lnumstr → Lnumstrfun

... (ABT)

* function definition: binding a name to an Abstract Binding Tree t, a bound variable that serves as the argument of the function.

* function application (Subitition):

... Abstract Syntax Tree

... First-order functions ...

... higher-order ... imperative ...

... Dynamic Binding ...

... functional ...

Subject
Date

Application

L {num str fun} :

Abstract | Concrete

Expressions e ::= call [f](e) | f(e)

fun [τ1 ; τ2] (x1 . e2 ; f.e) | fun f(x1 : τ1) : τ2 = e1 in e

definition

Domain Type (Argument Type)

Range (Result) Type

type checking

e : τ judgment

f(τ1) : τ2

Typing

$$\frac{\Gamma, x_1 : \tau_1 \vdash e_2 : \tau_2 \qquad \Gamma, f(\tau_1) : \tau_2 \vdash e : \tau}{\Gamma \vdash \text{fun}[\tau_1 ; \tau_2](x_1 . e_2 ; f.e) : \tau}$$

PdPCO

Subject APL
Date ۹۴/۸/۱۴

؟ Enum Struct, recursion, high-order function

$$\frac{\Gamma \vdash f(\tau_1) : \tau_2 \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash call\ (f)(e_1) : \tau_2}$$

Function Substitution:

$[\![x.e/f]\!]e'$

$x.e$ ... e' ... f ...

... call(f) ...

$[\![x.e/f]\!]\, call(f)(e') = let([\![x.e/f]\!]e'; x.e)$

$f_1(f_2(x))$ ...

example: $double(num(2) + double(num(3)))$

$= let([\![x.x+x/double]\!])(num(2) + double(num(3)))$

$x.x+x$

$; x.x+x)$

$= let(let([\![x.x+x/double]\!](num(2) + double(num(3)));$

$x.x+x); x.x+x) = \ldots$

P4PCO

$call(f)(e') = let(e'; x.e)$ ...

Subject
Date

(dynamics)

: Semantics ی static

$$fun[\tau_1;\tau_2](x.e_1;f.e_2) \mapsto [x.e_1/f]e_2$$

Application

higher-order

ref : ( )

Abstract Concrete

Typ $\tau ::= arr(\tau_1;\tau_2)$ $\tau_1 \to \tau_2$

Exp $e ::= lam[\tau](x.e)$ $\lambda(x:\tau)\,e$

$ap(e_1;e_2)$ $e_1(e_2)$

definition, abstraction

application

$e \cdot 3 \cdot ap(lam[num](x.\ plus(num[2];x));\ num[3])$

Subject APL

Date 98/9/15

e.g. $lam[num \to num](x.ap(x; num[2]))$

type: $(num \to num) \to num$

$ap(lam[num \to num](x.ap(x; num[2])); lam[num](x.plus(x; num[3])))$

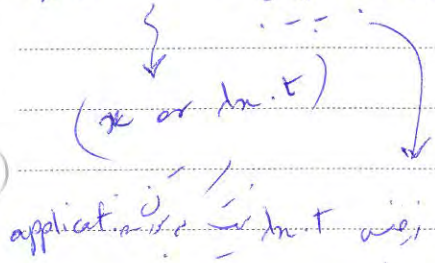
$\mapsto^* num[5]$

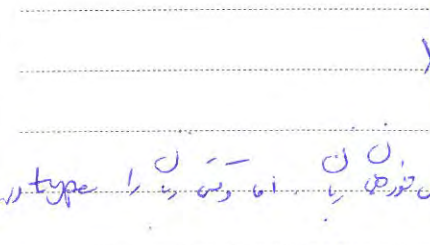
Lambda Calculus:

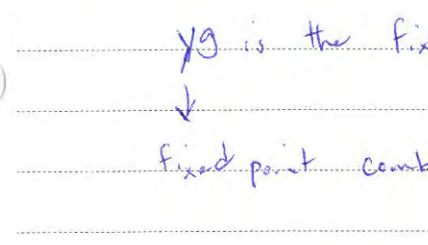
Syntax $e ::= x \mid \lambda x.e \mid ee$

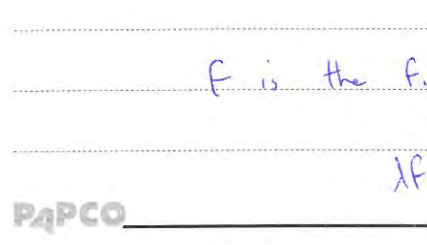
Semantics $(\lambda x.e)e' \mapsto [e'/x]e$

Subject
Date









Subject APL
Date ۹۹,۸,۱۵

$Y(\lambda f.\lambda n. \ldots)$ ... Lambda calculus ... f ...

... :

$Y(\lambda f.\lambda n.\ \text{if } n=0 \text{ then } 1 \text{ else } n * f(n-1))$

↓

... F ... n-1 ...

... Type System ... xy ... xy ... λ ...

... Y ... (fixed point Combinator) ... Primitive ... Recursion ... plotkin ... 1975 ... PCF ...

... plotkin ... Gödel ... Recursion ...

... :

... :

$$\frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash lam[\tau_1](x.e) : arr(\tau_1;\tau_2)} \quad (ABST)$$

PaPCO

Subject
Date

$$\frac{\Gamma \vdash e_1 : arr(\tau_2; \tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash ap(e_1; e_2) : \tau} \text{ (APP)}$$

Example - $lam[str](x.\ len(x))$

$$\frac{x : str \vdash len(x) : num}{lam[str](x.\ len(x)) : arr(str; num)}$$

: Arrow is Right Associative

$$(\tau_1 \to (\tau_2 \to (\tau_3 \to \cdots (\tau_{n-1} \to \tau_n))))$$

e.g. $num \to num \to num = num \to (num \to num)$

$lam[num](x.\ lam[num](y.\ plus(x; y)))$

e.g. $(num \to num) \to num$

$lam[num \to num](f.\ f\ num[a])$

حالا طبق این Type System و قاعده App و Var ، باید ثابت بدیم که برنامه زیر

Subject Apl
Date ۹۴،۸،۱۵

CSF 2017

Theorem (Substitution). If $\Gamma, x:\tau \vdash e' : \tau'$ and $\Gamma \vdash e : \tau$,

then $\Gamma \vdash [e/x]e' : \tau'$.

Proof - By induction...

Semantics:

$$\frac{}{\text{lam}[\tau](x.e) \ \text{val}}$$

$$\frac{e_1 \mapsto e_1'}{\text{ap}(e_1; e_2) \mapsto \text{ap}(e_1'; e_2)}$$

$$\left[\frac{e_1 \ \text{val} \qquad e_2 \mapsto e_2'}{\text{ap}(e_1; e_2) \mapsto \text{ap}(e_1; e_2')}\right] \ (\text{Call-by-value})$$

$$\frac{[e_2 \ \text{val}]}{\text{ap}(\text{lam}[\tau_2](x.e_1); e_2) \mapsto [e_2/x]e_1}$$

Call-by-name (Lazy) Call-by-value (Eager)

Subject
Date

Progress , preservation دارد safe یعنی

Theorem (Preservation) — If $e : \tau$ and $e \mapsto e'$, then $e' : \tau$.

Theorem (Progress) — If $e : \tau$ then either e val or there exists e' such that $e \mapsto e'$.

$(\lambda x.\,(x\,x))(\lambda x.\,(x\,x))$ مثال بارز

$\mathrm{lam}[\tau](x.\ x\ x)\left(\mathrm{lam}[\tau'](x.\ x\ x)\right)$

$\tau = \tau' \to \tau''$

arrow

(Recursive Types) Recursion

untyped ، typed

"unit"

Harper! Recursive Types

Hungry Function: $\tau = \mu p.\ \mathrm{num} \to p$

PAPCO

Subject APL

Date ۹۴/۸/۱۵

Safety ... Recursion ...

Recursion ... Constructs

Recursion ... Type ...

(i Gödel's T

(Primitive Recursion)

(2 Plotkin's PCF

(General Recursion)

↓

Turing Complete ... divergence ... partial function

Turing Complete ... PCF ... Gödel's T

Dynamic Scoping, Static Scoping ... function

Algol 60

Call-by-name, capture

Subject AP L → logician mathematician

Date ۹۴/۸/۲۰

Gödel's T :

$\mathcal{L}\{nat \rightarrow\}$

primitive recursion, plus, times

Sort	Abstract Form	Concrete Form
Typ	$\tau ::= nat$	nat
	$arr(\tau_1; \tau_2)$	$\tau_1 \rightarrow \tau_2$
Exp	$e ::= x$	x
	z	z
	$s(e)$	$s(e)$
	$rec(e; e_0; x.y.e_1)$	$rec\ e\ \{z \Rightarrow e_0 \mid s(x)\ with\ y \Rightarrow e_1\}$
	$lam[\tau](x.e)$	$\lambda(x:\tau)\ e$
	$ap(e_1; e_2)$	$e_1(e_2)$

Primitive recursion construct :

$$rec\ e\ \{z \Rightarrow e_0 \mid s(x)\ with\ y \Rightarrow e_1\}$$

PdPCO

اگر e برابر z شد، e_0 اجرا می‌شود. اگر e برابر $s(e)$ شد، y را برابر e قرار می‌دهیم و e_1 را اجرا می‌کنیم.

Subject APL
Date ۹۴, ۸, ۲۵

$\bar{n} \triangleq S(S(\dots S(Z)))$ (n times)

n is a numeral

(Syntax, Statics, Dynamics)

Statics:

$$\frac{}{\Gamma, x:\tau \vdash x:\tau}$$

$$\frac{}{\Gamma \vdash Z : nat}$$

$$\frac{\Gamma \vdash e : nat}{\Gamma \vdash S(e) : nat}$$

$$\frac{\Gamma \vdash e : nat \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x:nat, y:\tau \vdash e_1 : \tau}{\Gamma \vdash rec(e; e_0; x.y.e_1) : \tau}$$

$$\frac{\Gamma, x:\rho \vdash e:\tau}{\Gamma \vdash lam[\rho](x.e) : \rho \to \tau}$$

$$\frac{\Gamma \vdash e_1 : arr(\tau_2; \tau) \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash ap(e_1; e_2) : \tau}$$

Subject
Date

Dynamics:

$$\frac{}{z \ \text{val}}$$

Call-by-value

$$\frac{[e \ \text{val}]}{S(e) \ \text{val}}$$

$$\frac{}{\text{lam}[\tau](x.e) \ \text{val}}$$

$$\left[\frac{e \mapsto e'}{S(e) \mapsto S(e')}\right]$$

$$\frac{e_1 \mapsto e_1'}{\text{ap}(e_1; e_2) \mapsto \text{ap}(e_1'; e_2)}$$

$$\left[\frac{e_1 \ \text{val} \quad e_2 \mapsto e_2'}{\text{ap}(e_1; e_2) \mapsto \text{ap}(e_1; e_2')}\right]$$

$$\frac{[e_2 \ \text{val}]}{\text{ap}(\text{lam}[\tau](x.e); e_2) \mapsto [e_2/x]e}$$

substitution

(Recursor)

$$\frac{e \mapsto e'}{\text{rec}(e; e_0; x.y.e_1) \mapsto \text{rec}(e'; e_0; x.y.e_1)}$$

PqPCO

Subject APL

Date ۹۹/۸/۲۰

(Basis)

$$rec(z; e_0; x.y.e_1) \mapsto e_0$$

s(e) val

$$rec(s(e); e_0; x.y.e_1) \mapsto [e, rec(e; e_0; x.y.e_1)/x, y]e_1$$

Example -

$$lam[nat](u.rec(u; z; x.y.s(s(y))))$$

$$rec(\bar{2}; z; x.y.s(s(y))) \mapsto [\bar{1}, rec(\bar{1}; z; x.y.s(s(y))/x, y]s(s(y))$$

$$\mapsto s(s(rec(\bar{1}; z; x.y.s(s(y)))) \mapsto$$

$$s(s([z, rec(z; z; x.y.s(s(y))/x, y]s(s(y))))$$

$$\mapsto s(s(s(s(rec[z; z; x.y.s(s(y))])))) \mapsto s(s(s(s(z))))$$

(double)

Subject
Date

Definability: A mathematical function $f: \mathbb{N} \to \mathbb{N}$ is definable in $\mathcal{L}\{\text{nat} \to\}$ iff there exists an expression e_f of type $\text{nat} \to \text{nat}$ such that for every $n \in \mathbb{N}$, $e_f(\bar{n}) \equiv \overline{f(n)} : \text{nat}$.

example: $e_f = \text{lam}[\text{nat}](u.\text{rec}(u; z; x.y.s(s(y))))$

Thus, $e_f(\bar{n}) \equiv \overline{2 \times n} : \text{nat}$

definitional equality $\longrightarrow$ $\Gamma \vdash e \equiv e' : \tau$ $\forall$ { ref. trans. sym.

The strongest congruence containing

$$\Gamma \vdash \text{ap}(\text{lam}[\tau_1](x.e_2); e_1) \equiv [e_1/x]e_2 : \tau$$

$$\Gamma \vdash \text{rec}(z; e_0; x.y.e_1) \equiv e_0 : \tau$$

$$\Gamma \vdash \text{rec}(s(e); e_0; x.y.e_1) \equiv [e, \text{rec}(e; e_0; x.y.e_1)/x,y]e_1 : \tau$$

Subject APL
Date 94/A/Y۵

Theorem (Termination) - If e : τ then there exists v val such that

e ≡ v : τ.

(value) definitally equal ... Ldnat → ...

(... →* ...)

Total function ... → ...

partial total

Gödel Numbering:

diagonalization

halting problem

Subject (Midterm, Seminar Topic)

↳ e.g. Monads, Monadic Computatit Functional, Protocol Analysis, Refinement Typing (what? and Applications), ~~Haskell~~ concept mapping

Date ۹۹/۸/۲۴

۹۹/۹/۱۱

Total function ... Gödel T ... recursion ...

... total ...

... untyped λ-calculus ... typed λ-calculus ...

... normalization ...

... type ... (Stuck, Value) ... safety ... value ...

... Gödel T ... primitive recursion ... partial ...

... P ...

... Gödel T ... Ackermann ...

... Universal ... computability ... diagonalization ...

... program ... input ... interpreter ...

... Universal ...

PqPCO

(... interpreter ...)

Subject APL

Date 94/8/22

Subject
Date

Example

ast a = plus(plus($\bar{2}$, $\bar{3}$), $\bar{5}$)

operator plus ⟶ ❌ 2

هر عدد هم عوض میشه با ؟

$\ulcorner a \urcorner = 2^{2} \cdot 3^{2} \cdot 5^{3} = 2^{2} \cdot 3^{2} \cdot 5^{3}$

(Factorization این یه عدد ست، ast یه عدد ست این هم یه عدد ست.

ست یه عدد برمیگردونه، یکی هم میگه برو این عدد رو ازش فاکتور بگیر و بهت اعداد رو بده.

برای هر عبارت ast ای یه عدد طبیعی وجود داره (یکتا) و برعکس هر عدد طبیعی یه ast یکتا وجود داره.

ast آن عدد.

تفسیر میکنه: interpreter

$f_{univ} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

for any $e : nat \to nat$, $f_{univ}(\ulcorner e \urcorner)(m) = n$ iff $e(\bar{m}) \equiv \bar{n} : nat$

e رو بده

(وجود f_{univ} یعنی total، برای هر عدد $\ulcorner e \urcorner$ یک عدد طبیعی وجود داره که اگر e رو بهش بدی n رو برمیگردونه، هست.)

PdPCO

Subject APL
Date ۹۹/۸/۲۲

تابع d: اصطلاحاً قطری‌سازی (diagonalization)

$d : \mathbb{N} \to \mathbb{N}$

$d(m) = f_{univ}(m)(m)$

$d(\ulcorner e \urcorner) = f_{univ}(\ulcorner e \urcorner)(\ulcorner e \urcorner) = e(\overline{\ulcorner e \urcorner})$

↓ apply

تابع d، T Gödel نامیده می‌شود.

فرض کنید e_d یک عبارت بسته باشد که تابع d را محاسبه می‌کند.

$e_d(\overline{\ulcorner e \urcorner}) \equiv e(\overline{\ulcorner e \urcorner}) : nat$ (I)

Let $e_D = \lambda(x : nat)\ S(e_d(x)) : nat \to nat$

Now

$e_D(\overline{\ulcorner e_D \urcorner}) = S(e_d(\overline{\ulcorner e_D \urcorner})) = S(e_D(\overline{\ulcorner e_D \urcorner}))$

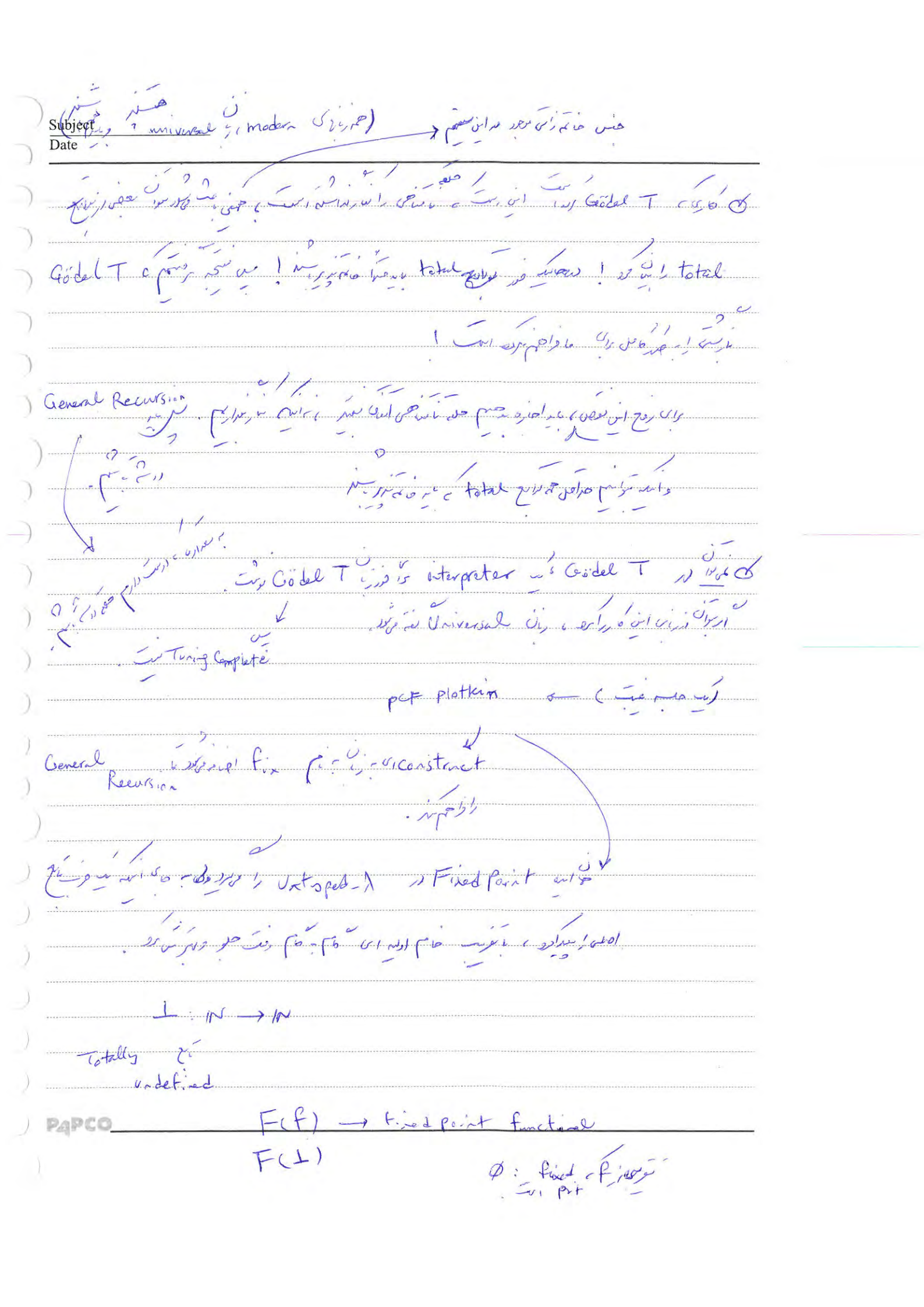
↓ طبق (I)

این تناقض به این دلیل است که فرض کردیم e_D یک عبارت بسته با مقدار termination است.

پس $e_D(\overline{\ulcorner e_D \urcorner})$ یک عبارت بسته نیست!!! (می‌توان نشان داد که یک عبارت بسته است)

لذا تابع e_dای که تابع d را محاسبه می‌کند، تابع total نیست.

Subject
Date



Subject:
Year. Month. Date. 94/05/27

- Gödel's T تابع را کامل در نظر گرفتیم تا termination داشته باشیم.
infinite loop

همین کامل در نظر گرفتن تابعی باعث شد که بعضی توابع نمی‌توانیم total را تعریف کنیم.

- Gödel's T بنابراین Universal نیست!

بنابراین ما نمی‌توانیم اجازه دهیم loop تابع صورت بگیرد.

Gödel's T دارای General Recursion نیست.

یعنی به هر تعداد که دوست داریم recurse کند.

- اگر زبانی که می‌خواهیم طراحی کنیم باید General Recursion داشته باشیم.

یعنی دلیل است که نمی‌توانیم بیاییم از خود زبان برای آن زبان interpreter بنویسیم؟

- ماشین های تورینگ می‌توانند توابع Partial را محاسبه کنند یعنی ممکن است برای ورودی‌هایی ماشین هیچگاه Halt نکند.

- همین termination موجود در سیستم Gödel مانع از این می‌شود که تمام توابع total را پوشش دهد.

- هم چنین یک عدد Gödel دارد! عکس این قضیه نیست!

Gödel's T is not Turing Complete.

- در دهه 70، Plotkin، PCF را پیشنهاد داد. (Plotkin's PCF)

Year. Month. Date. ( )

- plotkin توانست زبانی را ارائه دهد که Turing Complete باشد. در این تحقیق تئوری recursion نیز پیشرفت زیادی کرده بود.

- او توانست General recursion را در زبان خودش داشته باشد.

- ایده اصلی در Plotkin's PCF= وجود دارد از untyped λ-Calculus آمده است.

- اگر بخواهیم مساله

- بخواهیم تابع فاکتوریل را در نظر بگیریم:

$$f: \mathbb{N} \to \mathbb{N}$$

$$f: n \longrightarrow \begin{cases} 1 & n=0 \\ n \times f(n-1) & \text{otherwise} \end{cases}$$

- بیایید Functional در نظر بگیریم

که تابعی است که تابع میگیرد و تابع را میدهد.

$$F(f) = n \longrightarrow \begin{cases} 1 & n=0 \\ n \times f(n-1) & \text{otherwise} \end{cases}$$

- در اینجا F تابعی است که تابع f را میگیرد و تابع ! را میدهد.

- حال فرض کنید $g: n \longmapsto n+1$ (تابعی است که n را به $n+1$ میبرد)، حالا با جایگذاری میکنیم:

$$F(g) = n \longrightarrow \begin{cases} 1 & n=0 \\ n \times n & \text{otherwise} \end{cases}$$

در اینصورت $F(g) = \{(0,1), (1,1)(2,4), (3,9) \dots\}$

Year. Month. Date. ( )

فرض کنید $g : n \to 2n+3$ دراینصورت داریم:

$$F(g) = n \to \begin{cases} 1 & n=0 \\ n \times (2n+1) & \text{otherwise} \end{cases}$$

$$F(g) = \{ (0,1) , (1,3), (2,10), (3,21) \cdots \}$$

حال اینبار g را به F میدهیم:

$$g : n \to \begin{cases} 1 & n=0 \\ n \times g(n-1) & \text{otherwise} \end{cases}$$

~~$F(g) = \{$~~ ~~$F(g) =$~~ $\{$ حال داریم: $F(g) = n \to \begin{cases} 1 & n=0 \\ n \times g(n-1) & \text{otherwise} \end{cases}$

این دربار خود g است یعنی دراینجا $Fg = g$ است در این تابع g نقطه ثابت شد.

در مورد بقیه هم همینطور است!

یعنی تابع فاکتوریل (g) Fixed point تابع F است.

The factorial function is the fixed point of F.

در مورد نوشتن برنامه‌ی بازگشتی باشی F fixed point را پیدا کنیم.

در untyped λ-calculus، Fixed point موجود است.

Subject:
Year. Month. Date. ( )

$fix = \lambda f.\ (\lambda x.\ f(x\,x))\,(\lambda x.\ f(x\,x))$ اسم آنرا بگذارید fix

حال باید $fix\ F$ را حساب کنیم.

$$fix\ F = (\lambda x.\ F(x\,x))\,(\lambda x.\ F(x\,x))$$

$$= F\,((\lambda x.\ F(x\,x))\,(\lambda x.\ F(x\,x)))$$

$$= F\,(fix\ F)$$

در نتیجه داریم: $F(fix\ F) = fix\ F$.

- اشکال این است که نمی‌توان $\lambda f.\ (\lambda x.\ f(x\,x))\,(\lambda x.\ f(x\,x))$ را تایپ‌دار کرد.

یعنی ثابت می‌شود که نمی‌توانیم تایپ‌ای برایش در نظر بگیریم.

فقط در یک صورت می‌شود که برای آن τ داشته باشیم:

$$\lambda(f:\tau)\,(\lambda(x:\tau').\ f(x\,x))\,(\lambda(x:\tau').\ f(x\,x))$$

$\tau, \tau' : \alpha \to \alpha \to \alpha \to \cdots$

e.g. $\tau, \tau' : bool \to bool \to bool \to \cdots$

$\emptyset$ تقریبی از f است اگر

$\emptyset : \mathbb{N} \to \mathbb{N} \qquad f : \mathbb{N} \to \mathbb{N}$

در اینصورت $\emptyset(m) = n \implies f(m) = n$

توجه کنید $\emptyset$ در واقع یک partial function می‌شود.

Subject:
Year. Month. Date. ( )

مثلاً تعریف برای تابع فاکتوریل این است که تابعی برا ← 5 بعد 120 و در نقطه جایا تعریف نشده

بدترین تعریف ممکن : $\emptyset(m) = \perp$ (Totally undefined)

$m \in N$

- در واقع تابع $\perp : N \to N$ ، totally undefined است.

حالا این بدترین تعریف را به عنوان یه Functional F (یا فاکتوریل جدید) :

$$F(\emptyset) = n \to \begin{cases} 1 & 0 \text{ اگر } \emptyset \text{ در نقطه } \perp \text{ (بیخودی)} \\ n \times \emptyset(n-1) & \text{otherwise} \end{cases}$$

این تابع لااقل برای $n=0$ یک مقدار ① را برمیگرداند (یه کم بهتر شد)

$$F(F(\emptyset)) = n \Rightarrow \begin{cases} 1 & n=0 \\ n \times (F(\emptyset))(n-1) & \text{otherwise} \end{cases}$$

این تابع برای $n=0$ و $n=1$ مقدار دارد (به ترتیب مقدار 0 و مقدار 1)

و به همین ترتیب ترکیب تابع $F(F(F(\emptyset)))$ برای 0 تا 2 مقدار خودش را برمیگرداند

و این تابع در واقع fixed point را برمیگرداند

نیلوالم Plotkin زبانی را پیشنهاد میدهد :

Subject:
Year. Month. Date. ( )

Sort	Abstract Form	Concrete Form
Typ	$\tau ::= nat$	nat
	$par(\tau_1;\tau_2)$	$\tau_1 \rightharpoonup \tau_2$
	Partial Arrow	partial calls function
Exp	$e ::= x$	x
	z	z
	$s(e)$	$s(e)$
	$ifz(e;e_0;x.e_1)$	$ifz\ e\{z \Rightarrow e_0 \mid s(x) \Rightarrow e_1\}$
	$lam[\tau](x.e)$	$\lambda(x:\tau)e$
	$ap(e_1;e_2)$	$e_1(e_2)$
	$fix[\tau](x.e)$	$fix\ x:\tau\ is\ e$

دینامیک کند ⟵ fixed point یک راه‌اندازی کند

$(F(F(F(\dots I))))$

در هر حال دارای type هست.

استاتیک زبان:

$$\frac{\Gamma \vdash e : nat \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x:nat \vdash e_1 : \tau}{\Gamma \vdash ifz(e;e_0;x.e_1) : \tau}$$

$$\frac{\Gamma, x:\tau_1 \vdash e : \tau_2}{\Gamma \vdash lam[\tau_1](x.e) : par(\tau_1;\tau_2)}$$

Subject:
Year. Month. Date. ( )

$$\frac{\Gamma \vdash e_1 : \mathrm{Par}(\tau_1; \tau_2) \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash \mathrm{ap}(e_1; e_2) : \tau_2}$$

$$\frac{\Gamma, x:\tau \vdash e : \tau}{\Gamma \vdash \mathrm{fix}\,[\tau]\,(x.e) : \tau}$$

دینامیک زبان:

$$\frac{}{z \ \mathrm{val}} \qquad \frac{[e \ \mathrm{val}]}{s(e) \ \mathrm{val}}$$

$$\frac{}{\mathrm{lam}\,[\tau]\,(x.e) \ \mathrm{val}}$$

$$\frac{}{\mathrm{ifz}\,(z; e_0; x.e_1) \mapsto e_0}$$

$$\frac{s(e) \ \mathrm{val}}{\mathrm{ifz}\,(s(e); e_0; x.e_1) \longmapsto [e/x]e_1}$$

اگر ifz عددی غیر از صفر بود در e_1 به جای x، e را می‌گذاریم.

- روی Ocaml خودتان تمرین کنید. ml را پیشنهاد می‌دهد اما racket

را به دانشجویان می‌گوییم تا interpreter بازند.

- نظیر ML type ، typing است.

Subject APL
Date ۹۷/۸/۲۹

$\emptyset(m) = n \Rightarrow f(m) = n$

$F(\perp) = f'$

$F(F(\perp))$

$\{nat \rightarrow\}$ (Gödel, rec) Abstract Form Concrete Form

$if_2(e; e_0; x.e_1)$ $if_2\, e\{z \Rightarrow e_0 \mid s(x) \rightarrow e_1\}$

$fix[\tau](x.e)$ $fix\ x:\tau\ is\ e$

fixed point

در این زبان تعریف partial function ها داریم.

در Dynamics:

$$\frac{}{\text{ifz}(z; e_0; x.e_1) \mapsto e_0}$$

$$\frac{s(e)\ \text{val}}{\text{ifz}(s(e); e_0; x.e_1) \mapsto [e/x]e_1}$$

$$\frac{}{\text{fix}[\tau](x.e) \mapsto [\text{fix}[\tau](x.e)/x]e}$$

خودش جای x جایگذاری می‌شود.

در این زبان safety برقرار است، Definitional Equality تعریف می‌شود.

(PCF Plotkin که در این فرم، هر چه خودمان خواستیم جمله می‌سازیم، با اصلاحات دیگری به کتاب مراجعه شود)

Definability:

fun x $(y:\tau_1):\tau_2$ is e یک notation اختصاری برای نوشتن

یعنی تابع بازگشتی که ورودی آن e است و به ترتیب τ_1 ، τ_2 است، در bound variable ها

در e هست: $x:\tau_1 \to \tau_2$, $y:\tau_1$ (هم x و هم y در e حضور دارند)

خود تابع the function itself ; چون آرگومان ورودی تابع argument

Subject APL

Date ۹۹،۸،۲۹

$$(\text{fun}\ f\ x\ (y:\tau_1):\tau_2\ \text{is}\ e)(e_1) \mapsto [\text{fun}\ x\ (y:\tau_1):\tau_2\ \text{is}\ e,\ e_1/x,y]e$$

$$\text{fix}\ x:\tau_1 \to \tau_2\ \text{is}\ \lambda(y:\tau_1)e$$

$$\text{fix}[\tau_1 \to \tau_2](x.\lambda(y:\tau_1)e)$$

$$\text{rec}\ e\{z \Rightarrow e_0 \mid s(x)\ \text{with}\ y \Rightarrow e_1\}$$

$$\equiv e'(e)$$

where

$$e' \equiv \text{fun}\ f(u:\text{nat}):\tau\ \text{is}\ \text{ifz}\ u\{z \Rightarrow e_0 \mid s(x) \Rightarrow [f(x)/y]e_1\}$$

PAPCO

Subject Simply typed, Gödel, PCF

Date

Foundations — Mitchell

Theorem — A partial function ϕ on the natural numbers is definable in PCF (that $\to$) iff it is partial recursive.

Turing Complete

PCF:

$$F(g) = n \mapsto \begin{cases} 1 & \text{if } n=0 \\ n \times g(n-1) & \text{o.w.} \end{cases}$$

fixed point

$$F(\bot) = n \mapsto \begin{cases} 1 & \text{if } n=0 \\ \times \text{ undefined} \end{cases}$$

$$F(F(\bot)) = n \mapsto \begin{cases} 1 & \text{if } n=0 \\ n \times \left(F(\bot)\right)(n-1) & \text{o.w.} \end{cases}$$

$$= n \mapsto \begin{cases} 1 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ \text{undefined} \end{cases}$$

$$F(F(F(\bot))) = n \mapsto \begin{cases} 1 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ 2 & \text{if } n=2 \\ \text{undefined} \end{cases}$$

PqPCO

Subject APL
Date ۹۴,۸,۲۹

[از این جلسه به بعد از کتاب Pierce (فصل ۹ و ۱۰) درس می‌دهیم]

Simply Typed λ-calculus:

Untyped λ : $x \mid \lambda x.t \mid tt$

Typed λ : $x \mid \lambda x:\tau.t \mid tt$

fun : $x \mid \lambda(x:\tau)\,e \mid ee$

↳ حالت خاص Typed λ

* تفاوت fun با Typed λ در این است که fun نوع پارامتر را مشخص می‌کند.

در Simply Typed Typing داریم:

$$\frac{\Gamma, x:T_1 \vdash t:T_2}{\Gamma \vdash (\lambda x:T_1.t):T_1 \to T_2}$$

در این جلسه از Concrete Form در کتاب Pierce استفاده می‌کنیم.

در این جلسه از notation استفاده می‌کنیم.

Simple Extensions on λ : در این فصل از کتاب Pierce

Subject

Date

The Curry - Howard Correspondence:

$\tau_1 \to \tau_2$

- introduction rule (abstraction)

- elimination rule (application)

$$\frac{\Gamma, x:\tau_1 \vdash e:\tau_2}{\Gamma \vdash \lambda x:\tau_1.e : \tau_1 \to \tau_2}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

- Propositions as types

- Proofs as programs

(Proof by Contradiction)

Subject APL

Date ۹۸/۹/۱۱

ریاضی‌دان معروف Hilbert ، اصل موردی‌بی‌را ارائه داد :

$\varphi \vee \neg\varphi$ (excluded middle)

که بیان می‌کرد این اصل برای هر فرمول بولی درست است. اما اثبات ارائه شده توسط او غیر سازنده (Constructive) بود.

اثبات سازنده برای $\varphi \Rightarrow \varphi$ می‌تواند به صورت زیر باشد

اثبات φ را می‌دهیم.

مثال: $\varphi \Rightarrow \varphi$ حال این را به صورت زیر نشان می‌دهیم: $\lambda x : \varphi . x$

این همان الگوریتم اثبات ما است. در واقع ثابت φ نشان می‌دهد که اثبات را دارد.

بنابراین قرار داد که جواب‌ها توابع و اثبات‌ها برنامه‌ها هستند.

مثال: $A \Rightarrow A \vee B$

Sum type

$A \Rightarrow (A \vee B)$

$e : A \Rightarrow (A \vee B)$

Subject (Martin Löf) LCF (Programs as Proofs)
Date Plotkin Logic for Computable functions (Milner)

Logic	Programming Languages
Proposition	Types
Proposition $P \Rightarrow Q$	type $P \to Q$
Proposition $P \wedge Q$	type $P \times Q$
proof of proposition P	term t of type P
Proposition P is provable	type P is inhabited (by some term)

↓
اثبات قضیه

ML ✓

proof search ✓ proof check

Proof Carrying code (PCC)

Subject APL

Date 99,9,11

Erasure and Typability:

[illegible] type [illegible]

[illegible]

[illegible] type [illegible]

[illegible] erase(x) [illegible]:

$$erase(x) = x$$

$$erase(\lambda x : T_1 . t_2) = \lambda x . erase(t_2)$$

$$erase(t_1\ t_2) = erase(t_1)\ erase(t_2)$$

[illegible] Simply Typed λ [illegible] term [illegible]

[illegible] Untyped λ [illegible] erase [illegible]

Theorem

1. If $t \longrightarrow t'$ under the typed evaluation relation, (transition like $\mapsto$)

   then $erase(t) \longrightarrow erase(t')$. (untyped)

2. If $erase(t) \longrightarrow m'$, (untyped) then there is a simply typed term t' such that $t \longrightarrow t'$ and $erase(t') = m'$

Subject
Date

"evaluation commutes with erasure"

1) Curry-style:

Syntax (to term)

Dynamics (to term)

Type System (Statics)

2) Church-style:

Syntax

Statics (well-typed terms)

Dynamics

λ calculus

base type

Function type

Turing Complete

Subject APL

Date 99/9/11 (فصل دوم پرسش)

Simple Extensions on λ:

توسیع‌هایی که به λ اضافه می‌شوند، یا نوع‌های پایه (base types) هستند یا نوع‌های تابعی (function types).

در توسیع‌های ساده، safety حفظ می‌شود.

نوع‌های پایه (base types) و نوع‌های تابعی (function types) تعریف می‌شوند.

توسیع‌ها شامل Type Computation هستند.

- A : base type — e.g. int, bool, char, ... (uninterpreted type)
- A : tobase type مجموعه

$\lambda x : A . x$

$\langle fun \rangle : A \to A$

$\lambda f : A \to A . \lambda x : A . f(f(x))$

$\langle fun \rangle : (A \to A) \to (A \to A)$

Subject
Date

The Unit type :

Base type یک

(terms) t ::= ... ← در مورد term های ساده‌ی

| unit Simple typed

سیستم‌های تایپ

(values) v ::= ...

| unit unit هم یک term است

بنابراین هم یک Unit است.

(types) T ::= ... یک Unit به تایپ‌های ساده‌ی

| Unit

اضافه می‌کنیم.

$\vdash$ unit : Unit

نتیجه‌ی evaluation می‌شود.

این یک unit در ML است. ML یک mostly functional زبان برنامه‌نویسی است.

side effect، assignment دارد. assignment

Sequencing) $t_1 ; t_2$ هم داریم که ابتدا t_1 و سپس t_2 اجرا می‌شود.

Subject APL
Date 94, 9, 11

void هم یک جا مناسب برای بعضی چیزها است. مثلا برای procedure ها که side effect دارند.

void در Java, typing, ... (مثلا در زبان ...) معادل unit است.

$t_1 ; t_2 \longrightarrow$ Sequencing

دنباله‌ی دو دستور: یعنی اول t_1 اجرا شود، سپس نتیجه دور ریخته شود، و t_2 اجرا شود.

$$t_1 ; t_2 \overset{def}{=} (\lambda x : Unit . t_2)\ t_1 \quad \text{in by-value evaluation}$$

where $x \notin FV(t_2)$

Dynamics:

$$\text{(E-SEQ)}\ \frac{t_1 \longrightarrow t_1'}{t_1 ; t_2 \longrightarrow t_1' ; t_2} \qquad \text{(E-SEQNEXT)}\ \frac{}{unit ; t_2 \longrightarrow t_2}$$

Typing:

$$\text{(T-SEQ)}\ \frac{\Gamma \vdash t_1 : Unit \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 ; t_2 : T_2}$$

Derived Forms: (Syntactic Sugar)

برای اضافه کردن یک ساختار جدید به زبان باید Syntax, Semantics, typing rule آن را مشخص کنیم.

مثلا برای Sequencing

برای این ساختار جدید، ابتدا باید آن را به یک term از قبل موجود ترجمه کنیم!

$$t_1 ; t_2 \overset{def}{=} (\lambda x . Unit . t_2)\ t_1$$

~~بعد از تعریف by-value semantics~~

Syntax, Typing Rule, Dynamics

Parse

1.9/ ترجمه‌ی این ساختار به ساختار قبلی desugaring نام دارد.

Subject

Date ۹۷/۹/۱۳

derived form ... Typing Rules ... Semantics ...

Syntactic Sugar ... derived form ... Construct ...

(desugaring) ... derived form ... derived form ...

Theorem (Sequencing is a derived form) —

Write $\lambda^{\bar{E}}$ (External) for the simply typed λ-calculus with the Unit type, the sequencing construct, and the rules $\bar{E}$-SEQ, $\bar{E}$-SEQNEXT, and T-SEQ, and λ^{I} (Internal) for the simply typed λ-calculus with the Unit only. Let $e \in \lambda^{\bar{E}} \to \lambda^{I}$ be the elaboration function that translates from the external

Subject APL
Date ۹۷/۹/۱۴

language to the internal language by replacing every occurrence of $t_1 ; t_2$ with $(\lambda x : Unit. t_2)\, t_1$, where x is chosen fresh in each case.

Now, for each term t of λ^E, we have:

- $t \xrightarrow[E]{} t'$ iff $e(t) \xrightarrow[I]{} e(t')$
- $\Gamma \vdash^E t : T$ iff $\Gamma \vdash^I e(t) : T$.

این ترجمه‌ها نشان می‌دهد که ... derived form ... (syntactic sugar) ... expressiveness ... (interm) ... λ^E ... λ^I ... safe ...

wild card : ... syntactic sugar ...

$\lambda x : \_ . t$ $\quad (x \notin FV(t))$

... x ... t ...

$\lambda x : \_ . t \overset{def}{=} \lambda x : S . t$ where x is some variable not occurring in t.

۱۱/

Subject
Date

* documentation

Ascription:

$t ::= \ldots$

$t \text{ as } T$ → term t

$v ::= \ldots$

$v \text{ as } T$

Evaluation:

$$\frac{t_1 \longrightarrow t_1'}{t_1 \text{ as } T \longrightarrow t_1' \text{ as } T}$$

Typing:

$$\frac{\Gamma \vdash t_1 : T}{\Gamma \vdash t_1 \text{ as } T : T}$$

Let Binding:

$t ::= \ldots$

$\text{let } x = t_1 \text{ in } t_2$

"evaluate the expression t_1 and bind the name x to the resulting value with evaluting t_2"

Subject APL

Date ۹۶/۸/۱۴

Evaluation:

$$\text{let } x = v_1 \text{ in } t_2 \longrightarrow [x \mapsto v_1]\, t_2$$

↳ substitution, [v1/x] t2 (slash)

$$\frac{t_1 \longrightarrow t_1'}{\text{let } x = t_1 \text{ in } t_2 \longrightarrow \text{let } x = t_1' \text{ in } t_2}$$

Typing:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 : T_2}$$

Let Binding ✓ derived term ست:

$$\text{let } x = t_1 \text{ in } t_2 \overset{\text{def}}{=} (\lambda x : T_1 . t_2)\, t_1 \quad \text{by-value}$$

elaboration function ... T_1 ...

typing derivation ... term ...

* تا اینجا فقط با نوع‌های ساده و تابعی کار کردیم: function type

Subject
Date

Pairs :

$t ::= \ldots$

$\{t, t\}$

$t.1$
$t.2$ } → projections

$v ::= \ldots$

$\{v, v\}$

$T ::= \ldots$

$T_1 \times T_2$ (product type)

↓

is a type constructor

Evaluation :

$$\frac{}{\{v_1, v_2\}.1 \longrightarrow v_1}$$ (عنصر اول)

$$\frac{}{\{v_1, v_2\}.2 \longrightarrow v_2}$$ (عنصر دوم)

$$\frac{t_1 \longrightarrow t_1'}{t_1.1 \longrightarrow t_1'.1}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.2 \longrightarrow t_1'.2}$$

Subject APL
Date 24,9,18

$$\frac{t_1 \longrightarrow t_1'}{\{t_1, t_2\} \longrightarrow \{t_1', t_2\}}$$

$$\frac{t_2 \longrightarrow t_2'}{\{v_1, t_2\} \longrightarrow \{v_1, t_2'\}}$$

Typing:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \{t_1, t_2\} : T_1 \times T_2}$$

$$\frac{\Gamma \vdash t_1 : T_1 \times T_2}{\Gamma \vdash t_1.1 : T_1}$$

$$\frac{\Gamma \vdash t_1 : T_1 \times T_2}{\Gamma \vdash t_1.2 : T_2}$$

Example — (By-value)

$(\lambda x : Nat \times Nat.\ x.2)\ \{pred\ 4, pred\ 5\}$

$\longrightarrow (\lambda x : Nat \times Nat.\ x.2)\ \{3, pred\ 5\}$

$\longrightarrow (\lambda x : Nat \times Nat.\ x.2)\ \{3, 4\}$

$\longrightarrow \{3, 4\}.2 \longrightarrow 4$

Subject APL

Date ۹۹/۹/۱۲

Tuples :

$t ::= \ldots$

$\{t_i \mid i \in 1, \ldots, n\} \rightsquigarrow \{t_1, t_2, \ldots, t_n\}$

$t.i \longrightarrow t.n, \ldots, t.1$

$V ::= \ldots$

$\{v_i \mid i \in 1, \ldots, n\} \rightsquigarrow \{v_1, \ldots, v_n\}$

$T ::= \ldots$

$\{T_i \mid i \in 1, \ldots, n\} \rightsquigarrow \{T_1, T_2, \ldots, T_n\}$

Evaluation :

$$\{v_i \mid i \in 1, \ldots, n\}.j \longrightarrow v_j$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.i \longrightarrow t_1'.i}$$

$$\frac{t_j \longrightarrow t_j'}{\{v_i^{\,i \in 1, \ldots, j-1}, t_j, t_k^{\,k \in j+1, \ldots, n}\} \longrightarrow \{v_i^{\,i \in 1, \ldots, j-1}, t_j', t_k^{\,k \in j+1, \ldots, n}\}}$$

Subject Apl

Date ۹۷/۹/۱۴

Typing:

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{t_i^{\,i\in 1,\dots,n}\} : \{T_i^{\,i\in 1,\dots,n}\}}$$

$$\frac{\Gamma \vdash t_1 : \{T_i^{\,i\in 1,\dots,n}\}}{\Gamma \vdash t_1.j : T_j}$$

Example - $\{2, \lambda x:Nat.\, x, 3\} : \{Nat, Nat \to Nat, Nat\}$

Records:

↓

Labeled tuple

Subtyping:

$t ::= \dots$

$\{l_i = t_i^{\,i\in 1,\dots,n}\}$ $\quad l_i \in Labels$

$t.l \longrightarrow$

Subject Opl

Date 94,9,14

$v ::= \ldots$

$\{l_i = v_i\}^{i \in 1,\ldots,n}$

$T ::= \ldots$

$\{l_i : T_i\}^{i \in 1,\ldots,n}$

Evaluation:

$$\frac{}{\{l_i = v_i\}^{i \in 1,\ldots,n}.\, l_j \longrightarrow v_j}$$

$$\frac{t_1 \longrightarrow t_1'}{t_1.l \longrightarrow t_1'.l}$$

$$\frac{t_j \longrightarrow t_j'}{\{l_i = v_i\}^{i \in 1,\ldots,j-1}, l_j = t_j, l_k = t_k^{k \in j+1,\ldots,n} \longrightarrow \{l_i = v_i\}^{i \in 1,\ldots,j-1}, l_j = t_j', l_k = t_k^{k \in j+1,\ldots,n}}$$

Typing:

$$\frac{\text{for each } i \quad \Gamma \vdash t_i : T_i}{\Gamma \vdash \{l_i = t_i\}^{i \in 1,\ldots,n} : \{l_i : T_i\}^{i \in 1,\ldots,n}}$$

$$\frac{\Gamma \vdash t_1 : \{l_i : T_i\}^{i \in 1,\ldots,n}}{\Gamma \vdash t_1.l_j : T_j}$$

Subject APL
Date ۹۴/۹/۱۴

Sums :

(Variant type)

$T_1 + T_2$ (Sum type)

[illegible] T_1 [illegible] T_2 [illegible]

[illegible]

[illegible] Sum type [illegible] variant type [illegible]

[illegible] Concat [illegible] (tail , head) cell [illegible]

[illegible] Sum [illegible] variant [illegible]

Disjoint Union

(اجتماع مجزا از هم)

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

union

[illegible] A [illegible] B [illegible]

$$A \uplus B = (\{A\} \times A) \cup (\{B\} \times B)$$

disjoint union

Tag

[illegible] tag [illegible]

[illegible] A [illegible]

Subject APL
Date ۹۴/۹/۱۸

sum type یا disjoint union

$T_1 + T_2$ (sum type)

Example -

physical Addr = { FirstLast : string , addr : string }

record

VirtualAddr = { name : string , email : string }

Addr = physicalAddr + VirtualAddr

(injection_left)

inl = physicalAddr ⟶ Addr

(injection_right) inr = VirtualAddr ⟶ Addr

Subject: APL

Year: ۹۷ Month: ۹ Date: ۱۸

getName = [redacted]

λa : Addr.

Case a of

inl x ⇒ x.firstlast

| inr y ⇒ y.name ;

t ::= ...

inl t

introduction ← { inr t

elimination ← case t of inl x ⇒ t | inr x ⇒ t

v ::= ...

inl v

inr v

T ::= ...

T + T

Subject:

Year: Month: Date:

Typing :

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{inl } t_1 : T_1 + T_2}$$

$$\frac{\Gamma \vdash t_2 : T_2}{\Gamma \vdash \text{inr } t_2 : T_1 + T_2}$$

$$\frac{\Gamma \vdash t_0 : T_1 + T_2 \quad \Gamma, x_1 : T_1 \vdash t_1 : T \quad \Gamma, x_2 : T_2 \vdash t_2 : T}{\Gamma \vdash \text{Case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 : T}$$

Evaluation :

$$\text{Case (inl } v_0) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_1 \mapsto v_0]\, t_1$$

$$\text{Case (inr } v_0) \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow [x_2 \mapsto v_0]\, t_2$$

$$\frac{t_1 \longrightarrow t_1'}{\text{inl } t_1 \longrightarrow \text{inl } t_1'}$$

$$\frac{t_2 \longrightarrow t_2'}{\text{inr } t_2 \longrightarrow \text{inr } t_2'}$$

$$\frac{t_0 \longrightarrow t_0'}{\text{Case } t_0 \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2 \longrightarrow \text{Case } t_0' \text{ of inl } x_1 \Rightarrow t_1 \mid \text{inr } x_2 \Rightarrow t_2}$$

BARAN

Subject: APL
Year: ۹۹ Month: ۹ Date: ۱۸

۱- مبحث Uniqueness of Type

type ها در سیستم‌های Sum مثل $T_1 + T_2$

Typing

Universal Typing) Polymorphism

quantification) parametric polymorphism

C++ Template ← زمان binding

ML Type Reconstruction

Type Inference annotation

(Unification) Typing Context

well-typed type checker

Type Reconstruction

ascription

Type Uniqueness

ascription

Subject:
Year: Month: Date:

inl t as T ... inl t

inr t as T ... inr t

inl x as T ... inl x

inr x as T ... inr x

(... Evaluation ... Typing ... Syntax ...)

... variant type ... sum ... (*)

... (right ... left) ...

Variants:

... injection ... tag ... ...

t ::= ...

(introduction) $\langle l = t \rangle$ as T

↳ ascription for uniqueness

injection ...

case t of $\langle l_i = x_i \rangle \Rightarrow t_i$, $i \in 1, \dots, n$

... variant ... component type ...

BARAN

Subject: APL
Year: 99 Month: 7 Date: 14

Example -

Addr = <physical : physicalAddr, Virtual : VirtualAddr>

a = <physical = pa>

ML

▷ a : Addr

getName = λa : Addr .

case a of <physical = x> ⇒ x.firstlast

| <virtual = y> ⇒ y.name ;

ML

▷ getName : Addr → String

Functional

(Variant Typing)

(Enumeration)

Subject:

Year: Month: Date:

$T ::= \ldots$

$\langle l_i : T_i \rangle^{i \in 1,\ldots,n}$

Typing:

$$\frac{\Gamma \vdash t_j : T_j}{\Gamma \vdash \langle l_i = t_i \rangle \text{ as } \langle l_i : T_i \rangle^{i \in 1,\ldots,n} : \langle l_i : T_i \rangle^{i \in 1,\ldots,n}}$$

$$\frac{\Gamma \vdash t_0 : \langle l_i : T_i \rangle^{i \in 1,\ldots,n} \qquad \text{for each } i \ \Gamma, x_i : T_i \vdash t_i : T}{\Gamma \vdash \text{Case } t_0 \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1,\ldots,n} : T}$$

Evaluation:

$$\frac{}{\text{Case } (\langle l_j = v_j \rangle \text{ as } T) \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1,\ldots,n} \longrightarrow [x_j \mapsto v_j] t_j}$$

$$\frac{t_0 \longrightarrow t_0'}{\text{Case } t_0 \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1,\ldots,n} \longrightarrow \text{Case } t_0' \text{ of } \langle l_i = x_i \rangle \Rightarrow t_i^{i \in 1,\ldots,n}}$$

$$\frac{t_i \longrightarrow t_i'}{\langle l_i = t_i \rangle \text{ as } T \longrightarrow \langle l_i = t_i' \rangle \text{ as } T}$$

Subject: APL

Year: 94 Month: 9 Date: 14

Enumerations: (Enumerated Types)

Example — Week day = <Monday : Unit, Tuesday : Unit, Wednesday : Unit, Thursday : Unit, Friday : Unit>

< Monday = unit > as Weekday

next Business Day : λ w : Weekday .

case w of <Monday = x> ⇒ <Tuesday = unit> as Weekday

| <Tuesday = x> ⇒ <Wednesday = unit> as Weekday

:

| <Friday = x> ⇒ <Monday = unit> as Weekday

Variants as Data types:

Type T = l_1 of T_1

| l_2 of T_2

:

| l_n of T_n

$T = \langle l_1 : T_1, \dots, l_n : T_n \rangle$

Subject:

Year: Month: Date:

Type Weekday = Monday | ... | Friday

Type BinaryTree = Leaf of Nat | Node of BinaryTree * BinaryTree

Node(Leaf(1), Leaf(2))

1 2

General Recursion:

isEven : Nat → Bool

ff = λie : Nat → Bool.
  λx : Nat.
    if iszero x then true
    else if iszero (pred x) then false
    else ie (pred (pred x))

ff : (Nat → Bool) → (Nat → Bool)

Subject: APL
Year: ۹۹ Month: ۹ Date: ۲۵

fixed point combinator

iseven = Y ff

↳ fixed point combinator

fix t ::= ...

Evaluation:

$$\frac{t_1 \longrightarrow t_1'}{fix\ t_1 \longrightarrow fix\ t_1'}$$

Subject:

Year: Month: Date:

Typing:

$$\frac{\Gamma \vdash t_1 : T_1 \to T_1}{\Gamma \vdash \text{fix } t_1 : T_1}$$

mutual recursion

ff = λieio : {iseven : Nat → Bool, isodd : Nat → Bool}.

{iseven = λx : Nat.

if iszero x then true

else ieio . isodd (pred x),

isodd = λx : Nat.

if iszero x then false

else ieio . iseven (pred x)};

ff : {iseven : Nat → Bool, isodd : Nat → Bool} →

{iseven : Nat → Bool, isodd : Nat → Bool}

r = fix ff

r : {iseven : Nat → Bool, isOdd : Nat → Bool}

Subject: APL
Year: ۹۷ Month: ۹ Date: ۲۰

iseven = ۲۰ iseven ;

iseven 7 ;

▷ false

isodd = ۲۰ isodd ;

isodd 7 ;

▷ true

Russel's Paradox :

$A = \{x \mid x \notin x\}$

a collection of sets that do not belong to themselves.

$A \in A \iff A \notin A$

type theory

Subject:
Year: Month: Date:

Classic Set Theory ⇝ Untyped λ-calculus

Set Theory ⇝ Typed λ-calculus

Typed λ ... fixed point ... Y ... Curry ... Turing ... combinator

... fix ... recursive ... infinite ...

... typed λ ... term ...

$diverg_T = \lambda\_ : Unit.\ fix\ (\lambda x : T.\ x)$

... typable ... recursive ... infinite ...

$diverg_T : Unit \to T$

$diverg_T\ unit : T$

... diverge ... unit ... T ...

... False ... False ... (Curry Howard Isomorphism)

BARAN

Subject: APL

Year: 99 Month: 9 Date: ۲۰

ی fixed point combinator (fpc) و recursive type

infiniteness

superset ←

provable ، false

diverge ، fix

safety

Lists :

List T

cons ، tail ، head

introduction

elimination

Subject:
Year: Month: Date:

t ::= ...
nil [T]
Cons [T] t t
isnil [T] t
head [T] t
tail [T] t

v ::= ...
nil [T]
Cons [T] v v ⟹ (Cons 2 (Cons 3 nil))

T ::= ...
List T

References :

BARAN

Subject: APL

Year: 94 Month: 9 Date: ۵

file input/output, mutable variables assignment ... impure entity

IPC and Sync, Continuations, jumps, network connection, display

pure functional ... impure

Monad ... effect ... Haskell

(Magic Notions of Computation)

Kleisli closure, Categorical Theory (Functor)

Scheme, ML ... mostly Functional ... side effect ... impure

statement ... Computational effect

side effect → statement

without side effect → expression

BARAN

Subject:
Year: Month: Date:

اعمال ممکن بر روی referenceها به طور کلی:

allocation, dereferencing, assignment

↓ ↓

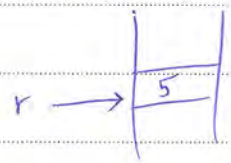
introduction elimination

ساخت یک ref جدید ref از بین بردن یک

allocation:

r = ref 5; ⟹ mutable variable

r، نوعی است که مقدار آن 5 است.



5 در یک location ذخیره می‌شود.

▷ r : Ref Nat (نوع)

Dereferencing: برای به دست آوردن مقدار ذخیره شده در reference cell

!r;

▷ 5 : Nat

(مقدار ذخیره شده در r)

Subject: APL

Year: 99 Month: 9 Date: 25

assignment: ... mutable ... r ...

r := (!r) + 1 ;

▷ () : Unit

... Unit ...

(!r)

▷ 6 : Nat

... Sequencing ... 6 mutable ...

(r := Succ (!r) ; !r) ;

▷ 7 : Nat

... Sequencing ... Nat ...

= (λ_ : Unit . !r) (r := Succ (!r))

... Unit ... Nat ...

BARAN

Subject:

Year: Month: Date:

Aliasing:

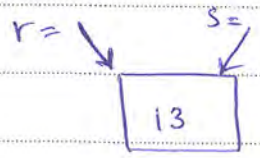
r = ref 13;

r یک متغیر است ← مقدار متغیری 13 دارد

s = r;

↳ assignment این

▷ s : Ref Nat



s := 82;

!r;

▷ 82 : Nat

r and s are aliases for the same cell.

NatArray ⊗

NatArray = Ref (Nat → Nat)

BARAN

Subject: APL
Year: 94 Month: 9 Date: 04

newarray = λ_:Unit . ref (λn:Nat.0)

▷ newarray : Unit → Ref (Nat → Nat)

or

Unit → NatArray

(initialization) توابعی که آرایه را به صفر مقداردهی اولیه می‌کند

lookup = λa:NatArray. λn:Nat. (!a) n

▷ lookup : NatArray → Nat → Nat

lookup b 5 ; ⟵ مقدار خانه پنجم b را برمی‌گرداند

update = λa:NatArray λm:Nat. λv:Nat.

let oldf = !a in

a := (λn:Nat. if equal m n then v

else oldf n);

BARAN

۱۳۹

Subject:

Year: Month: Date:

Typing: (تعیین نوع)

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \text{ref } t_1 : \text{Ref } T_1}$$

$$\frac{\Gamma \vdash t_1 : \text{Ref } T_1}{\Gamma \vdash !t_1 : T_1}$$

$$\frac{\Gamma \vdash t_1 : \text{Ref } T_1 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 := t_2 : \text{Unit}}$$

Evaluation: (ارزیابی)

What should be the values of type Ref T?

مقادیر از نوع ref چه هستند؟

Ref ها در واقع آدرس هایی هستند که در storage allocate شده اند.

مقدار آدرس

(در ML هیچ راهی برای دستکاری آدرس ref وجود ندارد.)

pointer arithmetic: (آدرس) store (محتوا) value

BARAN

Subject: APL

Year: 94 Month: 9 Date: 28

References are elements of some uninterpreted set $\mathcal{L}$ of stored locations.

Store is a partial function from locations $\mathcal{L}$ to values.

(μ)

$$\mu : \mathcal{L} \rightarrow \text{Values}$$

$$t \rightarrow t' \quad \text{یا} \quad t|\mu \rightarrow t'|\mu'$$

$$(\lambda x.T_{11}\ t_{12})\ v_2|\mu \rightarrow [x \mapsto v_2]\ t_{12}|\mu$$

Subject:

Year: Month: Date:

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1\, t_2 \mid \mu \longrightarrow t_1'\, t_2 \mid \mu'}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1\, t_2 \mid \mu \longrightarrow v_1\, t_2' \mid \mu'}$$

v ::= ...

$\lambda x : T . t$

Unit

l ⟶ locations are values.

t ::= x

$\lambda x : T . t$

tt

unit

ref t

!t

t := t

l ⟶ programmer never uses this term!

BARAN

Subject: AP2

Year: ۸۷ Month: ۹ Date: ۲۵

Evaluation strategy:

$$\frac{t_1 \mid \mu_1 \longrightarrow t_1' \mid \mu_1'}{!t_1 \mid \mu_1 \longrightarrow !t_1' \mid \mu_1'}$$

$$\frac{\mu(l) = v}{!l \mid \mu \longrightarrow v \mid \mu}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1 := t_2 \mid \mu \longrightarrow t_1' := t_2 \mid \mu'}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1 := t_2 \mid \mu \longrightarrow v_1 := t_2' \mid \mu'}$$

$$l := v_2 \mid \mu \longrightarrow unit \mid [l \mapsto v_2]\mu$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{ref\ t_1 \mid \mu \longrightarrow ref\ t_1' \mid \mu'}$$

$$\frac{l \notin dom(\mu)}{ref\ v_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto v_2)}$$

BARAN

Types 2 —

Subject: Refinement Types - Info Flow Sec - Proposed

Year: Month: Date:

۹۴/۹/۲۷

typing ، Real storage ، location ، Evaluation

what is the type of a "location"?

Typing Typing location

Safety

l_2 !

store type ، term

$l_2 : Unit$ if $\mu_1 = (l_1 \mapsto unit, l_2 \mapsto unit)$

$l_2 : Unit \to Unit$ if $\mu_2 = (l_1 \mapsto unit, l_2 \mapsto \lambda x : Unit. x)$

store ، ref ، term

$\Gamma \vdash \mu(l) : T_1$

$\Gamma \vdash l : Ref\ T_1$

BARAN

Subject: APL
Year: 94 Month: 9 Date: 27

$\Gamma | \mu \vdash \mu(l) : T_1$

$\Gamma | \mu \vdash l : \text{Ref } T_1$

$\Gamma | \mu \vdash t : T$

context store term type

$(\Gamma | \mu \vdash \Gamma \vdash$

efficient Type checking

locat term locat locat term

Example — $(l_1 \mapsto \lambda x : \text{Nat}.\ 999,$

$l_2 \mapsto \lambda x : \text{Nat}.\ (!l_1)\, x,$

$l_3 \mapsto \lambda x : \text{Nat}.\ (!l_2)\, x,$

$l_4 \mapsto \lambda x : \text{Nat}.\ (!l_3)\, x,$

$l_5 \mapsto \lambda x : \text{Nat}.\ (!l_4)\, x)$

Type checking cyclic

$(l_1 \mapsto \lambda x : \text{Nat}.\ (!l_2)\, x,\ l_2 \mapsto \lambda x : \text{Nat}.\ (!l_1)\, x)$

BARAN

Subject:
Year: Month: Date:

μ : concrete store
Σ : store typing

اصل مسئله این است که چگونه می‌توانیم location ها را در term ها پیدا کنیم.

برای این کار نیاز به Type ها داریم.

برای ref ها نیز نیاز به location ها داریم.

Store typing: a finite function mapping locations to types.
(Σ)

Typing ها نیز مانند context ها توابعی هستند که location ها را به type ها می‌نگارند.

$\Sigma : l \rightarrow Types$

store typing

تابع Σ نیز مانند context ها در ابتدا خالی است.

قواعد جدید اضافه شده:

$$\frac{\Sigma(l) = T_1}{\Gamma | \Sigma \vdash l : Ref\ T_1}$$

$$\frac{\Gamma | \Sigma \vdash t_1 : T_1}{\Gamma | \Sigma \vdash ref\ t_1 : Ref\ T_1}$$

$$\frac{\Gamma | \Sigma \vdash t_1 : Ref\ T_1}{\Gamma | \Sigma \vdash !t_1 | T_1}$$

BARAN

Subject: ApL
Year: ۹۷ Month: ۹ Date: ۲۷

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathrm{Ref}\ T_1 \qquad \Gamma \mid \Sigma \vdash t_2 : T_1}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \mathrm{Unit}}$$

Safety:

well-typed terms are not stuck.

preservation + Progress

preservation: $\Gamma \mid \Sigma \vdash t : T \ \wedge\ t \mid \mu \to t' \mid \mu' \Rightarrow \Gamma \mid \Sigma \vdash t' : T$

BARAN

Subject:
Year: Month: Date:

Definition - A store μ is said to be well-typed with respect to a typing context Γ and a store typing Σ, written $\Gamma | \Sigma \vdash \mu$, if

$dom(\mu) = dom(\Sigma)$ and $\Gamma | \Sigma \vdash \mu(l) : \Sigma(l)$ for every $l \in dom(\mu)$

(μ is consistent w.th Σ)

$$\Gamma | \Sigma \vdash t : T \;\wedge\; t | \mu \to t' | \mu' \;\wedge\; \Gamma | \Sigma \vdash \mu$$

$$\Rightarrow \Gamma | \Sigma \vdash t' : T$$

(Lem Wrong!)

Subject: ...
Year: 94 Month: 9 Date: ۲۷

Theorem (preservation) -

$\Gamma | \Sigma \vdash t : T \wedge \Gamma | \Sigma \vdash \mu \wedge t | r_1 \to t' | \mu'$

$\Longrightarrow \exists \Sigma' \supseteq \Sigma \ . \ (\Gamma | \Sigma' \vdash t' : T \wedge \Gamma | \Sigma' \vdash \mu')$

superset

location

evaluation

derivation

Lemma (substitution) -

$\Gamma, x:S \mid \Sigma \vdash t : T \wedge \Gamma | \Sigma \vdash s : S$

$\Longrightarrow \Gamma | \Sigma \vdash [x \mapsto s] t : T$

Lemma -

$\Gamma | \Sigma \vdash \mu \wedge \Sigma(l) = T \wedge \Gamma | \Sigma \vdash v : T$

$\Longrightarrow \Gamma | \Sigma \vdash [l \mapsto v] \mu$

BARAN

Subject:
Year: Month: Date:

Theorem (Progress) - Suppose t is a closed, well-typed term (i.e., $\emptyset | \Sigma \vdash t : T$ for some T and Σ). Then, either t is a value or else, for any store μ such that $\emptyset | \Sigma \vdash \mu$, there is some term t' and store μ' with $t | \mu \rightarrow t' | \mu'$.

(typing derivation)

(Reference)

(Exception)

(pointer)

Subtyping :

(Subtype Polymorphism)

orthogonal ... record

cross-cutting Subtyping

object-oriented

Subject: APL

Year: ۲۰۱۴ Month: ۹ Date: ۲۷

A Theory of objects, Abadi, 1996

(Core calculus of object-orientation)

$$\frac{\Gamma \vdash t_1 : T_1 \to T_2 \qquad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1\, t_2 : T_2}$$

Example $(\lambda r:\{x:Nat\}.\ r.x)\{x=0, y=1\}$

↓

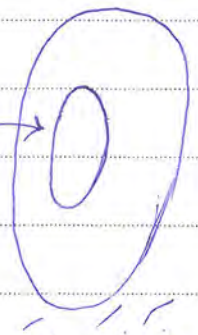
$\{x:Nat\} \to Nat$ doesn't apply to $\{x:Nat, y:Nat\}$

Subject:
Year: Month: Date:

Principle of Safe Substitution

We say that S is a subtype of T, written $S <: T$,
to mean that any form of type S can safely be used in
a context where a term of type T is expected.

e.g. $\{x:Nat, y:Nat\} <: \{x:Nat\}$



A subsumes B

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T}$$ Subsumption

BARAN

Subject: APL
Year: 24 Month: 9 Date: ۲۷

structural

Subtyping ...

$$\frac{}{S <: S} \quad \text{(Reflexivity)(S\_REFL)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \quad \text{(S\_TRANS)}$$

S_REFL (Subsumption ... Subtyping ...

S_TRANS ...

record ...

$$\{l_i : T_i \;^{i \in 1,\dots,n+k}\} <: \{l_i : T_i \;^{i \in 1,\dots,n}\} \quad \text{(S\_RCDWIDTH)}$$

width ... Subtype ...

$$\frac{\text{for each } i \quad S_i <: T_i}{\{l_i : S_i \;^{i \in 1,\dots,n}\} <: \{l_i : T_i \;^{i \in 1,\dots,n}\}} \quad \text{(S\_RCDDEPTH)}$$

Subtype ...

BARAN

۵۲

Subject:

Year: Month: Date:

(S-RCDPERM) $\{k_j : S_j \}^{j \in 1..n}$ is a permutation of $\{l_i : T_i\}^{i \in 1..m}$

$\{k_j : S_j\}^{j \in 1..n} <: \{l_i : T_i\}^{i \in 1..m}$

Example –

(S-RCDWIDTH) (S-REFL)

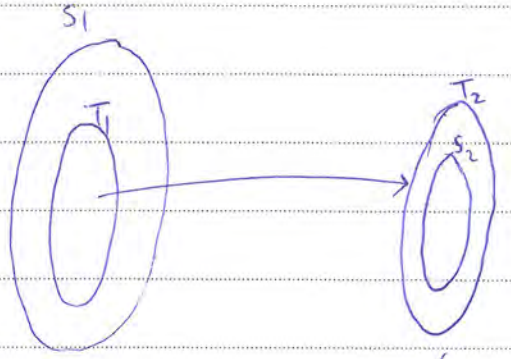
$\{a:Nat, b:Nat\} <: \{a:Nat\}$ $\{m:Nat\} <: \{m:nat\}$

(S-RCDDEPTH)

$\{x:\{a:Nat, b:Nat\}, y:\{m:Nat\}\} <: \{x:\{a:Nat\}, y:\{m:Nat\}\}$

Subtyping in Function Types:

e.g. $\lambda x:Nat \to Bool.\ x\ 0$ $: (Nat \to Bool) \to Bool$



$T_1 <: S_1$ $S_2 <: T_2$

(S-ARROW)

$S_1 \to S_2 <: T_1 \to T_2$

BARAN

statically ML ، dynamically lisp ؛ statically safe ، safe

Subject: APL

Year: 9x Month: 9 Date: ۲۷

(mostly - statically)

Subtype relation of arrow is contravariant of the argument type, and covariant of result type.

T_1 T_2

(left-side : contravariant)
(right-side : covariant)

covariant ، contravariant

antisym ، transitive ، reflexive ، subtyping

preorder ، partial order

permutation

(S-RCPPERM)

Safety:

Safety ، subtyping

subtyping

Simply typed λ

Preservation

Lemma (Inversion of Subtype Relation)

1. If $S <: T_1 \to T_2$, then S has the form $S_1 \to S_2$ with $T_1 <: S_1$ and $S_2 <: T_2$.

BARAN

۵۵

Subject:
Year: 24 Month: 10 Date: 7

2. If $S <: \{ l_i : T_i \}^{i \in 1,\dots,n}$, then S has the form $\{ k_j : S_j \}^{j \in 1,\dots,m}$ with at least the labels $\{ l_i \}^{i \in 1,\dots,n}$ i.e., $\{ l_i \}^{i \in 1,\dots,n} \subseteq \{ k_j \}^{j \in 1,\dots,m}$ and with $S_j <: T_i$ for each common label $l_i = k_j$.

Lemma 8 —

1. If $\Gamma \vdash \lambda x : S_1^{f_1}.\, s_2^{f_2} : T_1 \to T_2$, then $T_1 <: S_1$ and $\Gamma, x : S_1^{f_1} \vdash s_2^{f_2} : T_2$.

term : $s_1^{f_1}$ // $s_2^{f_2}$

2. If $\Gamma \vdash \{ k_a = s_a \}^{a \in 1,\dots,m} : \{ l_i : T_i \}^{i \in 1,\dots,n}$, then $\{ l_i \}^{i \in 1,\dots,n} \in \{ k_a \}^{a \in 1,\dots,m}$ and $\Gamma \vdash s_a : T_i$ for each common label $k_a = l_i$.

proof - Induction on Typing derivation (+ REF, TRANS, subsumption, ...)

BARAN

Subject: APL

Year: ۹۷ Month: ۱۰ Date: ۲

Lemma (Substitution) - If $\Gamma, x:S \vdash t:T$ and $\Gamma \vdash s:S$, then $\Gamma \vdash [x \mapsto s]t : T$.

proof - Induction on typing derivation (which contains variable x)

Theorem (preservation) - If $\Gamma \vdash t:T$ and $t \to t'$, then $\Gamma \vdash t':T$.

proof - Induction on typing derivation.

Lemma (Canonical Forms) -

1. If v is a closed value of type $T_1 \to T_2$, then v has the form $\lambda x:S_1.t_2$.

2. If v is a closed value of type $\{l_i : T_i^{\,i \in 1..n}\}$, then v has the form $\{k_j = v_j^{\,j \in 1..m}\}$ with $\{l_i^{\,i \in 1..n}\} \subseteq \{k_j^{\,j \in 1..m}\}$.

BARAN

۹۷/

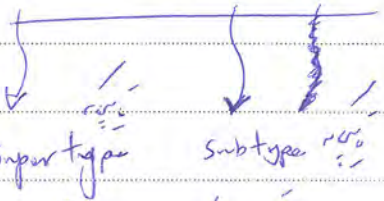
Subject:

Year: Month: Date:

Theorem (Progress) - If t is a closed well-typed term, then either t is a value or else there is some t' with $t \to t'$.

proof - Induction on typing derivations.

Top and Bottom Type:



(maximal)

$T ::= \ldots$

Top

(S-Top)

$T <: \mathrm{Top}$

BARAN

Subject: APL
Year: ۹۷ Month: ۱۰ Date: ۲

Top : نوع پایه Object مثل

parametric polymorphism ← ابزاری برای پیاده‌سازی چندریختی پارامتری

Bot ، Top

Subtype ، نوع زیرمجموعه

T ::= ...

Bot

از قواعد بروز دهی استفاده می‌کنیم:

(S-Bot)

Bot <: T

Bot نوع زیرمجموعه همه انواع

از قواعد متعدی بودن زیرنوع استفاده می‌کنیم

$$\frac{t : Bot \qquad Bot <: T_1 \to T_2}{t : T_1 \to T_2}$$

$t = \lambda x : S_1 . s$ ← ناقص!

Inversion

$$\frac{t : Bot \qquad Bot <: \{a : T\}}{t : \{a : T\}}$$

$t : \{a : T\}$ ← Inversion $t = \{a = 3, \dots\}$

BARAN

۱۵۹/

Subject:
Year: Month: Date:

Bot به term اضافه می‌کنیم که در صورت نیاز به Bot تبدیل شود.

Bot ای که نیاز به برگرداندن مقدار ندارد ← (نوع برگشتی void)

Unit به void، برای راحتی

Bot برای Continuation, Invocation و Exception

Unit برای

Example

exception raising term به error

λx : T.

if ⟨check that x is reasonable⟩ then

⟨compute the result⟩

else

error

برای ill-typed ها نیز error داریم اما این بار می‌خواهیم

branch های Bot را جدا کنیم تا بتوانیم

نوع if را به result تبدیل کنیم

BARAN

Subject: APL

Year: 44 Month: 10 Date: ۲

معنی است Type checking ، Bot کمپ دیگر.

عدم توانایی نوشتن را از این است که ...

ویژگی دیگر Bot را بیان است

ارائه‌ی جزئی Subtyping ... در مورد ... است:

Ascription

t as T

Γ ⊢ t : T

Γ ⊢ t as T : T

uniqueness : منحصر
of typing
, documentation
..., debugging

برای مجبور به ... است.

معنا ... است ... T ...

این ... Casting ... از ... است

↓

... (T) t ...

... Subtyping ... ascribe ... است

up-casting ... Supertype ...

BARAN

14 1/

Subject:
Year: Month: Date:

subtyping ... down-cast ...

(subtyping ...) ... assertion

dynamic ... down-cast ... static ... up-cast ...

(... runtime type) ... test ... safety ...

... reflection ... channel ...

... safety ... trade-off ...

... subtyping ... T ...

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T}$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \text{ as } T : T}$$

... ascription ...

(T) t

(up-cast) ... T ... supertype ...

... supertype ...

BARAN

Subject: APL
Year: 99 Month: 10 Date: 4

down-casting ... supertype ... (T)t ... T ...

... down-cast ... feature ...

... as ... :

$$\frac{\Gamma \vdash t_1 : S}{\Gamma \vdash t_1 \text{ as } T : T}$$

... ill-typed ... as ...

$$f = \lambda x : Top.\ (x \text{ as } \{a : Nat\}).\ a\ ;$$

... a ...

I know (for reasons that are too complex to tell them in typing rules), ...

BARAN ... a

Subject:

Year: Month: Date:

Pierce

Run-time test

Runtime test

Safety

Dynamic

(Runtime System)

$$\frac{\vdash V_1 : T}{V_1 \text{ as } T \longrightarrow V_1}$$

Example -

$$\frac{\vdash \{a=5, b=true\} : \{a : Nat\}}{\{a=5, b=true\} \text{ as } \{a:Nat\} \longrightarrow \{a=5, b=true\}}$$

Subject: APL
Year: 94 Month: 10 Date: 6

Example -

$$\vdash \{b=true\} : \{a:Nat\}$$

$$\{b=true\} \text{ as } \{a:Nat\} \longrightarrow \{b=true\}$$

progress و preservation

typable

raise ، Exception

Runtime (Type Test)

$$\frac{\Gamma \vdash t_1 : S \qquad \Gamma, x:T \vdash t_2 : U \qquad \Gamma \vdash t_3 : U}{\Gamma \vdash \text{if } t_1 \text{ in } T \text{ then } x \to t_2 \text{ else } t_3 : U}$$

statics

evaluation

dynamics

$$\frac{\vdash v_1 : T}{(\text{if } v_1 \text{ in } T \text{ then } x \to t_2 \text{ else } t_3) \longrightarrow [x \mapsto v_1]t_2}$$

evaluates to

$$\frac{\vdash v_1 : T}{(\text{if } v_1 \text{ in } T \text{ then } x \to t_2 \text{ else } t_3) \longrightarrow t_3}$$

except

Subject:

Year: Month: Date:

- feature خوبی است، چون down-cast
- List یعنی لیست عمومی، General است، Object
- down cast می‌کنیم
- Instantiate
- reflection با bytecode
- Top (Object) در سلسله‌مراتب
- down-cast، reflection
- down cast
- abstraction، hide، upcast
- Supertype
- Runtime، safe
- Variants، Subtyping

BARAN

Subject: APL

Year: ۲۰۲۱ Month: ۱۰ Date: ۲۴

Variants :

$\langle l_1 : T_1, l_2 : T_2 \rangle <: \langle l_1 : T_1, l_2 : T_2, l_3 : T_3 \rangle$

$t ::= \ldots$

$\langle l = t \rangle$ ⇝ injection on label l

Typing :

$$\frac{\Gamma \vdash t_1 : T_1}{\Gamma \vdash \langle l_1 = t_1 \rangle : \langle l_1 : T_1 \rangle}$$

5 - VARIANT - WIDTH)

$\langle l_i : T_i^{\,i \in 1..n} \rangle <: \langle l_i : T_i^{\,i \in 1..n+k} \rangle$

Subject:

Year: Month: Date:

Lists :

$$S <: T$$

$$List[S] <: List[T]$$

argument type ، Covariant ، List ، Type Constant

، Covariant Variant Constructor ، depth

Contravariant ، Covariant ، Constructor

Covariant Invariant Constructor

(Covariant ↓

References :

$$S <: T \quad T <: S$$

$$Ref\ S <: Ref\ T$$

Ref is an invariant type constructor for subtype relation.

Read operator ref

Write operator := → Subtyping

BARAN

Subject: APL

Year: 94 Month: 1 Date: 4

Read operator ! ⟹ S <: T

write operator := ⟹ T <: S

S <: T T <: S

Array[S] <: Array[T]

BARAN

Subject:

Year: Month: Date:

Coercion Semantics :

subtyping در حالت عادی باعث کاهش performance می‌شود.

Semantics جدیدی ارائه می‌شود که

subtyping در زمان Compile-time حل می‌شود.

typing derivation جدیدی ارائه می‌شود که

evaluation در subtyping

performance بهتری دارد.

Type Reconstruction

Subject: AP
Year: 99 Month: 10 Date: 4

Type Reconstruction :

(Type Inference)

- specify در زبان‌هایی که type discipline دارند برنامه‌نویس از برنامه‌نویسی راحت‌تری لذت می‌برد

این است usability که باعث می‌شود زبان‌هایی که استفاده بیشتری دارند این ویژگی را داشته باشند

در ML از Type Inference استفاده می‌شود یعنی نیازی به نوشتن type annotation نیست

این کار با استفاده از Refinement Types امکان‌پذیر است

در زبان‌هایی که از این نوع استنتاج نوع پشتیبانی می‌کنند برای ill-typed ها

این نوع استنتاج نوع پشتیبانی نمی‌شود

این کار باعث می‌شود برنامه‌نویس از type annotation های اضافی راحت شود

Type Inference در Haskell, ML و Polymorphism

زبان‌هایی که از Universal Typing پشتیبانی می‌کنند

Polymorphism زبان‌هایی است که

BARAN

۷۱

Subject:

Year: Month: Date:

Type Variables :

term

unInterpreted base types ← a, b, ...

placeholder ← base type

substitution

type variables → types

base type

Nat → X , Nat → Bool , Nat , X → X , X

∀ T

BARAN

Subject: APL

Year: 94 Month: 10 Date: 9

Example - $\sigma = [X \mapsto Bool]$

$\sigma(X \to X) = Bool \to Bool$

Definition - A Type Substitution σ is a finite mapping from type variables to types. $dom(\sigma)$ is the set of type variables appearing in the left-hand side and $range(\sigma)$ is the set of types appearing in the right-hand side of pairs in σ.

$dom(\sigma) \cap range(\sigma) \neq \emptyset$: یعنی اشتراک

$range(\sigma)$ و $dom(\sigma)$ یعنی اشتراک

Type Substitution و Term substitution ی "ن" ی

Example - $\sigma = [X \mapsto Bool, Y \mapsto X \to X]$

$\sigma(X \to Y) = Bool \to (X \to X)$

یعنی جایگزین ↑ X

$Bool \to (Bool \to Bool)$ یعنی

BARAN

Subject:

Year: Month: Date:

σ apply

$$\sigma(X) = \begin{cases} T & \text{if } (X,T) \in \sigma \\ X & \text{if } X \notin dom(\sigma) \end{cases}$$

$\sigma(Nat) = Nat$

$\sigma(Bool) = Bool$

primitive type

$$\sigma(T_1 \to T_2) = \sigma(T_1) \to \sigma(T_2)$$

function type

$$\sigma(\{x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n\}) = \{x_1 : \sigma T_1, x_2 : \sigma T_2, \ldots, x_n : \sigma T_n\}$$

record type

$$\sigma(x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n) = (x_1 : \sigma T_1, \ldots, x_n : \sigma T_n)$$

context

Γ Γ'

If σ and γ are two substitution,

$$\sigma \circ \gamma(X) = \begin{cases} \sigma T & \text{if } (X,T) \in \gamma \\ T & \text{if } (X,T) \in \sigma \text{ and } X \notin dom(\gamma) \end{cases}$$

(Composition)

BARAN

Subject: APL

Year: ۸۴ Month: ۱۰ Date: ۹

Theorem (Preservation under Type Substitution)

$$\Gamma \vdash t : T \Longrightarrow \sigma\Gamma \vdash \sigma t : \sigma T$$

General Judgment

$\Gamma \vdash t : T$: اثبات استواری قواعد نوع دهی

type variable : نیت (views) و در این دیدگاه

t is an expression (term) containing type variables, and

Γ is a typing context (Possibly containing type variables):

View I. $\forall \sigma . \exists T . \sigma\Gamma \vdash \sigma t : T$

View II. $\exists \sigma . \exists T . \sigma\Gamma \vdash \sigma t : T$

BARAN

۱۷۵

Subject:

Year: Month: Date:

Example -

$t_1 = \lambda f : X \to X . \lambda a : X . f(f a)$

$t_1 : (X \to X) \to (X \to X)$

t_1 is a polymorphic function

Parametric Polymorphism

Example - $t_2 = \lambda f : Y . \lambda a : X . f(f a)$

t_2 is typable

$\sigma = [Y \mapsto Nat \to Nat , X \mapsto Nat]$

$\sigma t_2 = \lambda f : Nat \to Nat . \lambda a : Nat . f(f a)$

$\sigma t_2 : (Nat \to Nat) \to (Nat \to Nat)$

BARAN

(Principal Types) t_2 is general $\sigma = [Y \mapsto X \to X]$

Subject: Apl

Year: ۹۵ Month: 6 Date: 9

Girard's System F

↑

(∀ دارد) ... Parametric Polymorphism ... ⊛

... Type Reconstruction ...

... Type Reconstruction ... (Principal Type) ...

... Parametric Polymorphism ...

Type Inference ... Parametric Polymorphism ... ML ...

... The most general ... Substitution

... Principal Type ...

Definition - Let Γ be a context and t a term. A "solution" for (Γ, t) is a pair (σ, T) such that $\sigma\Gamma \vdash \sigma t : T$.

... declarative Solution ...

↓

BARAN

۱۷

Subject:

Year: Month: Date:

BARAN

Subject: APL
Year: 94 Month: 10 Date: 11

if $t_1\ t_2$ with $\Gamma \vdash t_1 : T_1$, $\Gamma \vdash t_2 : T_2$, then

$T_1 = T_2 \rightarrow X$

$t_1\ t_2 : X$

Definition — A constraint set C is a set of equations $\{T_i' = T_i^{\,i=1\ldots n}\}$. A substitution σ is said to "unify" an equation $T' = T$ if the substitution instances $\sigma T'$ and σT are identical. We say that σ unifies (or satisfies) C if it unifies every equations in C.

Definition — A constraint typing relation $\Gamma \vdash t : T \mid_{X} C$ (X: fresh variables)

is defined by the following rules. Informally, $\Gamma \vdash t : T \mid_{X} C$ can be read "term t has type T under assumptions Γ whenever constraints C are satisfied".

BARAN

19

Subject:

Year: Month: Date:

$$\frac{}{\Gamma, x:T \vdash x:T \mid_{\emptyset}} \text{(CT-VAR)}$$

$$\frac{\Gamma, x:T_1 \vdash t_2 : T_2 \mid_{X}^{C}}{\Gamma \vdash \lambda x:T_1 . t_2 : T_1 \to T_2 \mid_{X}^{C}} \text{(CT-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid_{X_1}^{C_1} \quad \Gamma \vdash t_2 : T_2 \mid_{X_2}^{C_2}}{\Gamma \vdash t_1\, t_2 : X \mid_{X_1 \cup X_2 \cup \{X\}}^{C'}} \text{(CT-APP)}$$

$X_1 \cap X_2 = X_1 \cap FV(T_2) = X_2 \cap FV(T_1) = \emptyset$

$X \notin X_1, X_2, T_1, T_2, C_1, C_2, \Gamma, t_1, t_2$

$C' = C_1 \cup C_2 \cup \{T_1 = T_2 \to X\}$

BARAN

Subject: APL

Year: 04 Month: 6 Date: 11

$$\Gamma \vdash 0 : Nat \mid_{\emptyset} \{\}$$

$$\frac{\Gamma \vdash t_1 : T \mid_{\chi}^{C} \qquad C' = C \cup \{T = Nat\}}{\Gamma \vdash succ\ t_1 : Nat \mid_{\chi}^{C'}}$$

$$\frac{\Gamma \vdash t_1 : T \mid_{\chi}^{C} \qquad C' = C \cup \{T = Nat\}}{\Gamma \vdash pred\ t_1 : Nat \mid_{\chi}^{C'}}$$

$$\frac{\Gamma \vdash t_1 : T \mid_{\chi}^{C} \qquad C' = C \cup \{T = Nat\}}{\Gamma \vdash iszero\ t_1 : Bool \mid_{\chi}^{C'}}$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid_{\chi_1}^{C_1} \quad \Gamma \vdash t_2 : T_2 \mid_{\chi_2}^{C_2} \quad \Gamma \vdash t_3 : T_3 \mid_{\chi_3}^{C_3} \quad \chi_1, \chi_2, \chi_3 \text{ nonoverlapping} \quad C' = C_1 \cup C_2 \cup C_3 \cup \{T_1 = Bool,\ T_2 = T_3\}}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T_2 \mid_{\chi_1 \cup \chi_2 \cup \chi_3}^{C'}}$$

$$\frac{}{\Gamma \vdash true : Bool \mid_{\emptyset} \{\}} \quad \text{axiom}$$

$$\Gamma \vdash false : Bool \mid_{\emptyset} \{\}$$

BARAN

Subject:

Year: Month: Date:

Example — $t = \lambda x : X \to Y \,.\, x_0^{zero}$

$\Gamma = \emptyset$

$\emptyset \vdash \lambda x : X \to Y \,.\, x_0 : T \mid_{\chi}^{C}$: نوشتن این به صورت اثبات کلی

(CT-VAR)

$x : X \to Y \vdash x : X \to Y \mid_{\emptyset}^{\{\}}$ $\qquad$ $x : X \to Y \vdash 0 : Nat \mid_{\emptyset}^{\{\}}$

$\emptyset \cap \emptyset = \emptyset \cap FV(Nat) = \emptyset \cap FV(X \to Y) = \emptyset$

$Z \notin \emptyset, \emptyset, X \to Y, Nat, \emptyset, \emptyset, x : X \to Y, x, 0$

$C' = \emptyset \cup \{\} \cup \{\} \cup \{X \to Y = Nat \to Z\}$

(CB-APP)

$x : X \to Y \vdash x_0 : Z \mid_{\{Z\}}^{C'}$

(CB-ABS)

$\vdash \lambda x : X \to Y \,.\, x_0 : (X \to Y) \to Z \mid_{\{Z\}}^{C'}$

حال با این مثال اثبات کلی و عمومی را برای این مثال بررسی می‌کنیم

$\lambda x : X \to Y \,.\, x_0 : (Nat \to Z) \to Z$ که $\sigma = [X \mapsto Nat, Y \mapsto Z]$

(σ, T) ← نوشتن

$\Gamma, t \longrightarrow \sigma\Gamma \vdash \sigma t : T$ که نشان می‌دهد

Subject: APL
Year: 94 Month: 12 Date: 11

* : مشکل اصلی در رویکرد اول پیدا کردن Solution است و در رویکرد دوم declarative بودن

Γ, t (given)

Approach I — Declarative

σ, T s.t. $\sigma\Gamma \vdash \sigma t : T$

(از قضیه تعیین نوع اصلی)

Approach II — Algorithmic

$\Gamma \vdash t : T' \mid_{\chi} C$

Find $\sigma' \Rightarrow \sigma' T' = T''$

برای اینکه دچار مشکل نشویم باید ثابت کنیم $T = T''$, $\sigma = \sigma'$

Definition - Suppose that $\Gamma \vdash t : T \mid_{\chi} C$. A solution for (Γ, t, T, C) is a pair (σ, T') such that σ satisfies C and $\sigma T = T'$.

1. Every solution for (Γ, t, T', C) is also a solution for (Γ, t). [Soundness]
2. Every solution for (Γ, t) can be extended to a solution for (Γ, t, T', C). [Completeness]

Subject:

Year: Month: Date:

declarative (CBT) ... Soundness

declarative ... Completeness

Theorem (Soundness of constraint typing) - Suppose that $\Gamma \vdash t : S \mid_X C$. If (σ, T) is a solution for (Γ, t, S, C), then it is also a solution for (Γ, t).

Proof - By induction on the given constraint typing derivation for $\Gamma \vdash t : S \mid_X C$.

Definition - Write $\sigma \setminus X$ for the substitution that is undefined for all the variables in X and otherwise behaves like σ.

Theorem (Completeness of Constraint typing) - Suppose $\Gamma \vdash t : S \mid_X C$. If (σ, T) is a solution for (Γ, t) and $\mathrm{dom}(\sigma) \cap X = \emptyset$, then there is some solution (σ', T) for (Γ, t, S, C) such that $\sigma' \setminus X = \sigma$.

BARAN

Proof - Induction on constraint typing derivation rules.

Subject: APL
Year: 99 Month: 10 Date: 11

(Template این‌ها چیزی شبیه C++ و Parametric Polymorphism)

Complete, Sound است

Unification:

$\Gamma, t : T \mid_{\chi} C$ یعنی با استفاده از CBT ثابت کن که Γ, t می‌تواند جایگزین χ شود و C برقرار باشد.

substitution ای که σ باشد را می‌توان جایگزین χ کرد و C برقرار باشد.

unification یعنی ما یک substitution پیدا کنیم که بتواند جایگزین χ شود و C برقرار باشد. ML خیلی بزرگ است.

Definition - A substitution σ is less specific (or more general) than σ', written $\sigma \sqsubseteq \sigma'$, if $\sigma' = \gamma \circ \sigma$ for some γ.

composition

Definition - A principal unifier (or sometimes most general unifier) for a constraint set C is a substitution σ that satisfies C and such that $\sigma \sqsubseteq \sigma'$ for every substitution σ' satisfying C.

(عموما برای هر جایی هست)

BARAN

۱۵

Subject:
Year: Month: Date:

Principal type: most general type, principal unifier, ML Polymorphism

Example:

$C_1 = \{ X = Nat,\ Y = X \to X \}$

$\sigma = [X \mapsto Nat,\ Y \mapsto Nat \to Nat]$

unifier

$C_2 = \{ Y = Nat \to Y \}$

(recursive types)

$C_3 = \{ X \to Y = Y \to Z,\ Z = U \to W \}$

$\sigma_1 = [X \mapsto U \to W,\ Y \mapsto U \to W,\ Z \mapsto U \to W]$

principal unifier

$\sigma_2 = [X \mapsto Nat \to Nat,\ Y \mapsto Nat \to Nat,\ Z \mapsto Nat \to Nat]$

$\sigma_2 = [U \mapsto Nat, W \mapsto Nat] \circ \sigma_1$

Subject: APL
Year: 99 Month: 10 Date: 11

unify(C) = if C = ∅ then []

else let {T' = T} ∪ C' = C in

if T' = T then unify(C')

else if T' = X and X ∉ FV(T) then

unify([X ↦ T]C') ∘ [X ↦ T]

else if T = X and X ∉ FV(T') then

unify([X ↦ T']C') ∘ [X ↦ T']

else if T' = T1' → T2' and T = T1 → T2 then

unify(C' ∪ {T1' = T1, T2' = T2})

else FAIL

Theorem — The algorithm unify always terminates, failing when given a nonunifiable constraint set, and otherwise returning a principal unifier.

BARAN

14

Subject:
Year: Month: Date:

(یا بگوییم FV ها ، تمام اسمی که در اول این سه قدم برداشته شده .)

Example — $unify(\{X \to Y = Y \to Z,\ Z = U \to W\})$

$= unify([Z \mapsto U \to W]\{X \to Y = Y \to Z\}) \circ [Z \mapsto U \to W]$

$= unify(\{X \to Y = Y \to (U \to W)\}) \circ [Z \mapsto U \to W]$

$= unify(\emptyset \cup \{X = Y,\ Y = U \to W\}) \circ [Z \mapsto U \to W]$

$= (unify([Y \mapsto U \to W]\{X = Y\}) \circ [Y \mapsto U \to W]) \circ [Z \mapsto U \to W]$

$= (unify(\{X = U \to W\}) \circ [Y \mapsto U \to W]) \circ [Z \mapsto U \to W]$

$= [] \circ [X \mapsto U \to W] \circ [Y \mapsto U \to W] \circ [Z \mapsto U \to W]$

$= [X \mapsto U \to W, Y \mapsto U \to W, Z \mapsto U \to W]$

Γ, t

$\downarrow$ CBT

$\Gamma \vdash t : T \mid_X C$

$\downarrow$ unify

σ is the principal unifier for C

THEN σT is the principal type of t under Γ.

BARAN