Grad PL
vs.
The World

Grad PL Conclusions

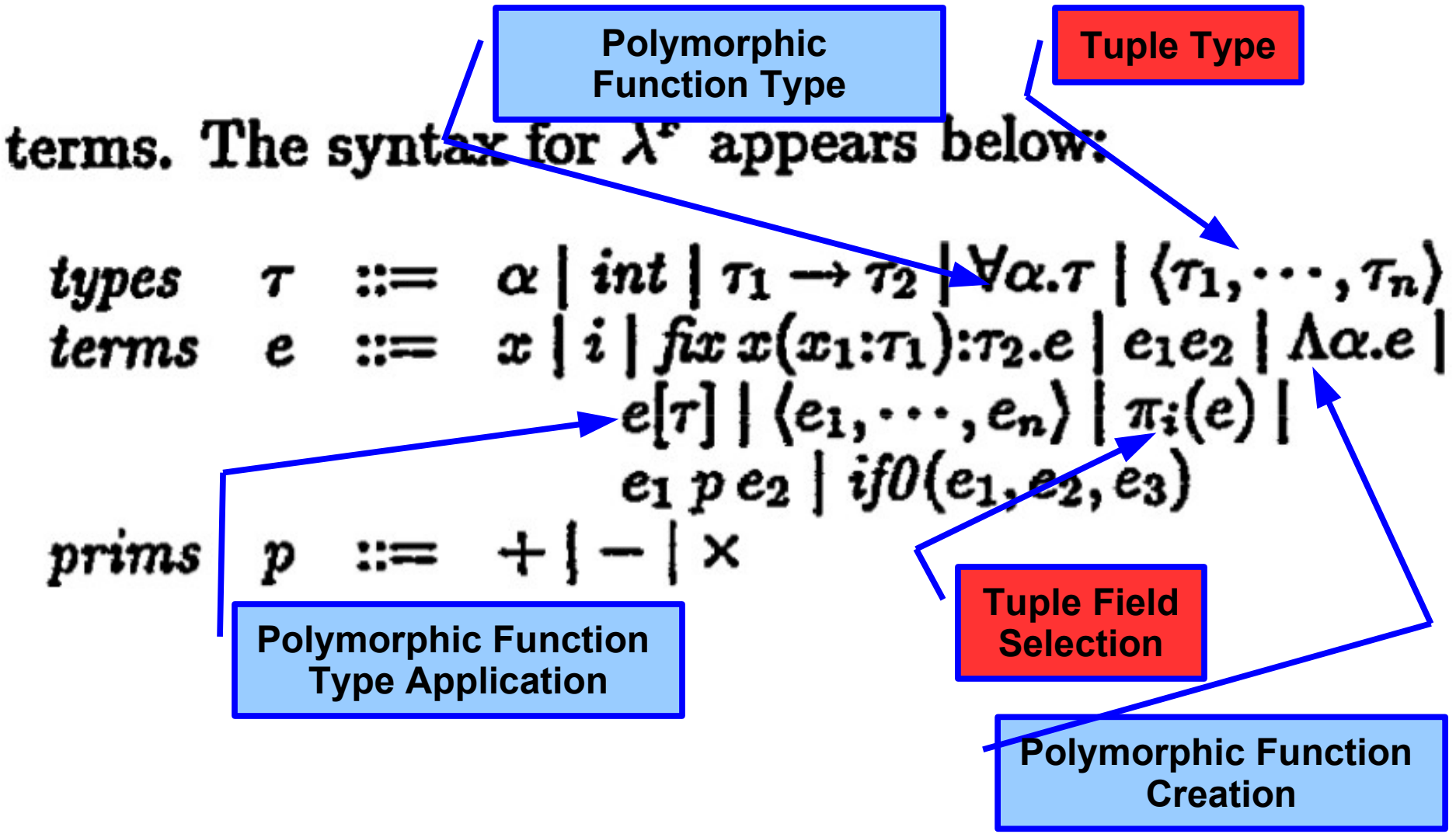
- You are now equipped to read the most influential papers in PL.
- You can also recognize PL concepts and will know what to do when they come up in your research.

ACM SIGPLAN Most Influential Paper Awards

- SIGPLAN presents these awards to the author(s) of a paper presented at ICFP, OOPSLA, PLDI, and POPL held 10 years prior to the award year. The award includes a prize of $1,000 to be split among the authors of the winning paper. The papers are judged by their influence over the past decade. Each award is presented at the respective conference.

- 2008 (POPL 1998): **From System F to Typed Assembly Language**, Greg Morrisett, David Walker, Karl Crary, and Neal Glew.
- ... began a major development in the application of type system ideas to low level programming. The paper shows how to compile a high-level, statically typed language into TAL, a typed assembly language defined by the authors. The type system for the assembly language ensures that source-level abstractions like closures and polymorphic functions are enforced at the machine-code level while permitting aggressive, low-level optimizations such as register allocation and instruction scheduling. This infrastructure provides the basis for ensuring the safety of untrusted low-level code artifacts, regardless of their source. A large body of subsequent work has drawn on the ideas in this paper, including work on proof-carrying code and certifying compilers.

From System F to Typed Assembly Language



From System F to Typed Assembly Language

... but you know it.

We interpret λ^F with a conventional call-by-value operational semantics (not presented here). The static semantics is specified as a set of inference rules that allow us to conclude judgments of the form $\Delta;\Gamma \vdash_F e : \tau$ where Δ is a set containing the free type variables of Γ, e, and τ; Γ assigns types to the free variables of e; and τ is the type of e.

Typing Judgment

As a running example, we will be considering compilation and evaluation of 6 factorial:

What is "fix" like?

$$(fix\ f(n{:}int){:}int.\ if0(n, 1, n \times f(n-1)))\ 6.$$

The operational semantics of TAL is presented in Figure 7 as a deterministic rewriting system $P \longmapsto P'$ that maps programs to programs. Althoug

Operational Semantics, Forward Symex

$(H, R, S) \longmapsto P$ where	
if $S =$	then $P =$
add $r_d, r_s, v; S'$	$(H, R\{r_d \mapsto R(r_s) + \hat{R}(v)\}, S')$ and similarly for mul and sub
bnz $r, v; S'$ when $R(r) = 0$	(H, R, S')
bnz $r, v; S'$ when $R(r) = i$ and $i \neq 0$	$(H, R, S''[\vec{\tau}/\vec{\alpha}])$ where $\hat{R}(v) = \ell[\vec{\tau}]$ and $H(\ell) = \text{code}[\vec{\alpha}]\Gamma.S''$
jmp v	$(H, R, S'[\vec{\tau}/\vec{\alpha}])$ where $\hat{R}(v) = \ell[\vec{\tau}]$ and $H(\ell) = \text{code}[\vec{\alpha}]\Gamma.S'$
ld $r_d, r_s[i]; S'$	$(H, R\{r_d \mapsto w_i\}, S')$ where $R(r_s) = \ell$ and $H(\ell) = \langle w_0, \ldots, w_{n-1}\rangle$ with $0 \leq i < n$
malloc $r_d[\tau_1, \ldots, \tau_n]; S'$	$(H\{\ell \mapsto \langle ?\tau_1, \ldots, ?\tau_n\rangle\}, R\{r_d \mapsto \ell\}, S')$ where $\ell \notin H$

" if false ... ; S' "

"L" is fresh. Small-step opsem For allocation.

From System F to Typed Assembly Language

Lemma 5.1 (Subject Reduction) *If* $\vdash_{\text{TAL}} P$ *and* $P \longmapsto P'$ *then* $\vdash_{\text{TAL}} P'$.

Type Preservation

Lemma 5.2 (Progress) *If* $\vdash_{\text{TAL}} P$ *then either:*

cf. Decomposition

1. *there exists* P' *such that* $P \longmapsto P'$, *or*
2. P *is of the form* $(H, R\{\mathtt{r1} \mapsto w\}, \mathtt{halt}[\tau])$ *where there exists* Ψ *such that* $\vdash_{\text{TAL}} H : \Psi$ *and* $\Psi; \emptyset \vdash_{\text{TAL}} w : \tau$.

Corollary 5.3 (Type Soundness) *If* $\vdash_{\text{TAL}} P$, *then there is no stuck* P' *such that* $P \longmapsto^* P'$.

- 2011 (POPL 2001): **Anytime, Anywhere: Modal Logics for Mobile Ambients**, Luca Cardelli and Andrew D. Gordon.
- ... helped spur a flowering of work in the area of process calculi that continues today. The paper focused on modal logics for reasoning about both temporal and spacial modalities for ambient behaviours, demonstrating techniques that also apply to other process calculi (even those without an explicit notion of location), so contributing to excitement in an area that was growing at that time and continues. The work has led to application of concurrency theory in fields as diverse as security, safety critical applications, query languages for semistructured data, and systems biology.

Anytime, Anywhere: Modal Logics for Mobile Ambients

Processes

$P,Q,R ::=$	processes	
$\mathbf{0}$	void	spatial
$P \mid Q$	composition	
$!P$	replication	
$M[P]$	ambient	
$M.P$	capability action	temporal
$(n).P$	input action	
$\langle M \rangle$	output action	
$M ::=$	messages	
n	name	names
in M	can enter into M	capabilities
out M	can exit out of M	
open M	can open M	
ε	null	paths
$M.M'$	composite	

"Pi Calculus"-esque

cf. pi send and receive

Anytime, Anywhere: Modal Logics for Mobile Ambients

Structural Congruence

$P \equiv P$	(Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(Struct Amb)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	
$P \mid \mathbf{0} \equiv P$	
$!(P \mid Q) \equiv !P \mid !Q$	
$!\mathbf{0} \equiv \mathbf{0}$	
$!P \equiv P \mid !P$	
$!P \equiv !!P$	

Pi Calculus

Synchronous Rendezvous

Reduction

$n[in\ m.\ P \mid Q] \mid m[R] \longrightarrow m[n[P \mid Q] \mid R]$	(Red In)
$m[n[out\ m.\ P \mid Q] \mid R] \longrightarrow n[P \mid Q] \mid m[R]$	(Red Out)
$open\ n.\ P \mid n[Q] \longrightarrow P \mid Q$	(Red Open)
$(n).P \mid \langle M \rangle \longrightarrow P\{n \leftarrow M\}$	(Red Comm)
$P \longrightarrow Q \Rightarrow n[P] \longrightarrow n[Q]$	(Red Amb)
$P \longrightarrow Q \Rightarrow P \mid R \longrightarrow Q \mid R$	(Red Par)
$P' \equiv P, P \longrightarrow Q, Q \equiv Q' \Rightarrow P' \longrightarrow Q'$	(Red $\equiv$)

$\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$

- 2011 (PLDI 2001): **Automatic predicate abstraction of C programs**, Thomas Ball, Rupak Majumdar, Todd Millstein, Sriram K. Rajamani.
- ... presented the underlying predicate abstraction technology of the SLAM project for checking that software satisfies critical behavioral properties of the interfaces it uses and to aid software engineers in designing interfaces and software that ensure reliable and correct execution. The technology is now part of Microsoft's Static Driver Verifier in the Windows Driver Development Kit. This is one of the earliest examples of automation of software verification on a large scale and the basis for numerous efforts to expand the domains that can be verified.

Automatic predicate abstraction of C programs

Axiomatic Semantics

4.1 Weakest Preconditions and

For a statement s and a predicate φ, let $WP(s, \varphi)$ denote the *weakest liberal precondition* [16, 20] of φ with respect to statement s. $WP(s, \varphi)$ is defined as the weakest predicate whose truth before s entails the truth of φ after s terminates (if it terminates). Let "`x = e`" be an assignment, where x is a scalar variable and e is an expression of the appropriate type. Let φ be a predicate. By definition $WP(\texttt{x = e}, \varphi)$ is φ with all occurrences of x replaced with e, denoted $\varphi[e/x]$. For example:

$$WP(\texttt{x=x+1}, x < 5) \;=\; (x + 1) < 5 \;=\; (x < 4)$$

- 2009 (PLDI 1999): **A Fast Fourier Transform Compiler**, Matteo Frigo
- ... describes the implementation of genfft, a special-purpose compiler that produces the performance critical code for a library, called FFTW (the “Fastest Fourier Transform in the West”), that computes the discrete Fourier transform. FFTW is the predominant open fast Fourier transform package available today, as it has been since its introduction a decade ago. genfft demonstrated the power of domain-specific compilation—FFTW achieves the best or close to best performance on most machines, which is remarkable for a single package. By encapsulating expert knowledge from the FFT algorithm domain and the compiler domain, genfft and FFTW provide a tremendous service to the scientific and technical community by making highly efficient FFTs available to everyone on any machine. As well as being the fastest FFT in the West, FFTW may be the last FFT in the West as the quality of this package and the maturity of the field may mean that it will never be superseded, at least for computer architectures similar to past and current ones.

A Fast Fourier Transform Compiler

```
type node =
    | Num of Number.number
    | Load of Variable.variable
    | Store of Variable.variable * node
    | Plus of node list
    | Times of node * node
    | Uminus of node
```

Figure 3: Objective Caml code that defines the `node` data type, which encodes an expression dag.

No joke. cf. HW5

Your Questions

Grad PL Conclusions

- You are now equipped to read the most influential papers in PL.
- You can also recognize PL concepts and will know what to do when they come up in your research.