

# Typing $\lambda$ -terms

The uptyped  $\lambda$ -calculus allows "strange" terms to be formed:

- $D D =_{\beta} not(D D)$
- succ (pair ff ff)

Solution: Rule out ill-formed terms using types (Church 1940)

Ill-formed term: Computation can "go wrong"

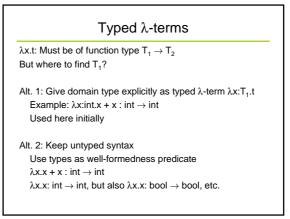
- succ (pair ff ff): Cannot complete computation to
- produce a sensible value = normal form
- Type unsafety runtime error

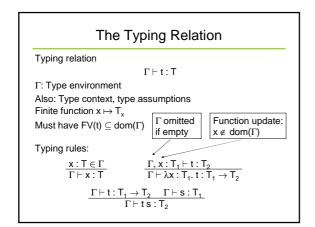
Types

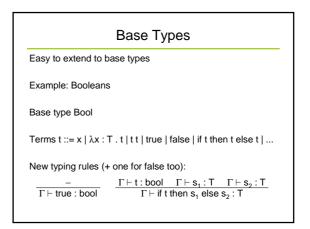
Simply typed  $\lambda$ -calculus,  $\lambda_{\rightarrow}$ : Only two types, base types and function types Syntax:

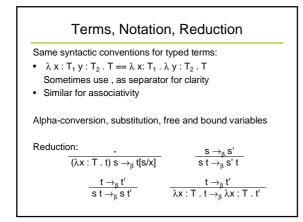
- $T ::= A | T \rightarrow T$
- A: Base type, e.g. bool, int, float, array[int], ...
- $T_1 \rightarrow T_2$ : Type of functions from  $T_1$  to  $T_2$

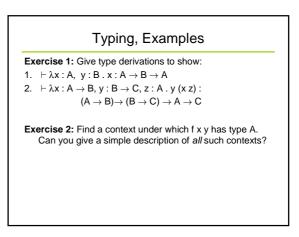
 $\begin{array}{l} \mbox{Type constructor} \rightarrow \mbox{right-associative:} \\ \mbox{T}_1 \rightarrow \mbox{T}_2 \rightarrow \mbox{T}_3 = = \mbox{T}_1 \rightarrow (\mbox{T}_2 \rightarrow \mbox{T}_3) \end{array}$ 











### Properties of the Typing Relation

Lemma 1:

1. If  $\Gamma \vdash x : T$  then  $x : T \in \Gamma$ 

- 2. If  $\Gamma\vdash\lambda$  x : T\_1. t : S then S = T\_1  $\to$  T\_2 for some S such that  $\Gamma,$  x : T\_1  $\vdash$  t : T\_2
- 3. If  $\Gamma \vdash t \, s : T_2$  then there is some  $T_1$  such that

 $\Gamma \vdash t: \mathsf{T_1} \to \mathsf{T_2} \text{ and } \Gamma \vdash s: \mathsf{T_1}$ 

Exercise 3: Prove this statement

**Exercise 4:** Is there any context  $\Gamma$  and type T such that  $\Gamma \vdash x x : T$ ? If so, give a type derivation. If not, prove it.

### Unique Typing and Normal Forms

**Lemma 2**: If  $\Gamma \vdash t$ :  $T_1$  and  $\Gamma \vdash t$ :  $T_2$  then  $T_1 = T_2$ **Exercise 5**: Prove this statement.

Unique typing fails for many richer languages

 $\begin{array}{l} \text{Values:} \\ v \in \text{Val} ::= x \mid x \; v \; ... \; v \mid \lambda x : T \; . \; v \end{array}$ 

 $\label{eq:lemma} \begin{array}{l} \mbox{Lemma 3: } t \nrightarrow_{\beta} \mbox{iff } t \in Val \\ \mbox{Exercise 6: Prove (or disprove) this statement.} \end{array}$ 

### Substitution Lemma

 $\Gamma \leq \Delta :$  For all x,  $\Gamma (x)$  is defined implied  $\Delta (x)$  is defined and then  $\Gamma (x) = \Delta (x)$ 

**Proposition 1**: If  $\Gamma \vdash t$  : T and  $\Gamma \leq \Delta$  then  $\Delta \vdash t$  : T

**Lemma 4 [Substitution]:** If  $\Gamma$ ,  $x : S \vdash t$ : T and  $\Gamma \vdash s : S$ then  $\Gamma \vdash t[s/x] : T$ 

We'll prove this statement in class.

**Theorem 1 [Subject Reduction]:** If  $\Gamma \vdash t$  : T and  $t \rightarrow_{\beta} t'$  then  $\Gamma \vdash t'$  : T

**Exercise 7:** Prove this statement (hint: Use induction on the derivation of  $\Gamma \vdash t:T$ )

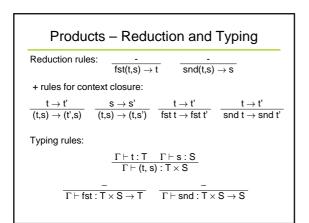
## Extensions - Products

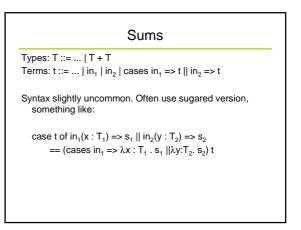
Many extensions possible, see TAPL for more First: Product types

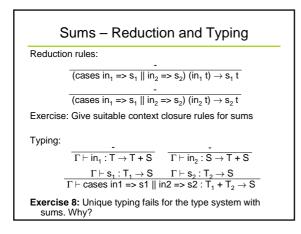
Types: T ::= ... | T × T Terms: t ::= ... | (t, t) | fst | snd

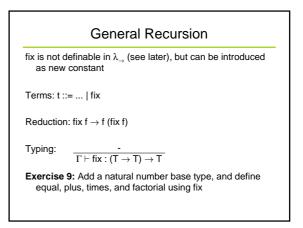
Reduction: Use generic  $\rightarrow$  instead of  $\rightarrow_{\beta}$ 

Can support different evaluation orders





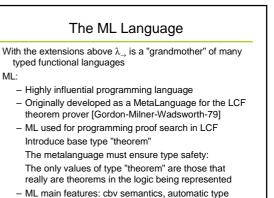




### More Exercises

 $\begin{array}{l} \label{eq:constructs} \textbf{Exercise 10:} \ \mbox{Add the following constructs to simply typed} \\ \mbox{lambda calculus, with reduction and typing rules:} \\ t ::= ... | let x : T = t_1 in t_2 | letrec x : T = t_1 in t_2 \\ \mbox{The intention (of course) is that "let" is used for non-recursive definitions, and "letrec" for recursive ones. \\ \mbox{Give reduction and typing rules for "let" and "letrec".} \\ \mbox{Show how "let" and "letrec" can be coded in $\lambda_{\rightarrow}$. Do the same for mutually recursive definitions: \\ t ::= ... | letrec x_1 : T_1 = t_1, \ldots, x_n : T_n = t_n in t \\ \end{array}$ 

Note: In more realistic languages one will generally want type annotations T,  $T_1,...$  to be inferred automatically by the type checker



inference, polymorphic types

### ML, Haskell, PCF

ML and other languages:

- ML was influenced by Landin's ISWIM
- SML Standard ML of 1997
- Comprehensive formal transition semantics and type system by [Milner-Tofte-Harper, 1990]
- Check out: SML of New Jersey, OCAML
- SML used in descendants of LCF: HOL, Isabelle
- Haskell is a descendant with cbn (lazy) semantics (and other twists)
- PCF [Plotkin-77]
- $\lambda_{\rightarrow}$  + naturals + more types + recursion
- Popular in theoretical studies

### Strong Normalization

We are now addressing the base calculus  $\lambda_{\rightarrow}$  with a single base type A Strong normalization:

 $\begin{array}{l} t\in SN_n \text{ iff any } \rightarrow_\beta \text{derivation } t=t_0 \rightarrow_\beta t_1 \rightarrow_\beta \cdots \rightarrow_\beta t_n \rightarrow_\beta \\ \cdots \text{ has length at most } n \end{array}$  $SN = \{t \mid \exists n. t \in SN_n\}$ 

**Theorem 2 [Strong Normalization]**: If  $\vdash$  t : T then t  $\in$  SN

This immediately shows that all terms of functional type must express total functions on closed terms

Thus, general recursion cannot be encoded in  $\lambda$ .

### Logical Relations

Exercise 11: Why is normalization tricky to prove?

As always, the trick is to find the right inductive argument

Proof here follows Tait [JSL-67] and Girard-Lafont-Taylor, Proofs and Types, CUP'89

Define predicate  $R_T$  on closed terms by:  $- R_A = \{t \mid t \in SN\}$ 

-  $R_{S \, \rightarrow \, T}$  = {t | whenever  $s \in R_S$  then t  $s \in R_T$ } Note: Do not require  $t \in R_T$  implies  $\vdash t : T$ .

## **Proof of Normalization**

**Lemma 6:** If  $t \rightarrow_{\beta} t'$  and  $t \in R_T$  then  $t' \in R_T$ Proof: By structural induction on the structure of T

Exercise 12: Prove lemma 6.

Neutral term: Either a variable or an application

Lemma 7:

1. If  $t \in R_{\tau}$  then  $t \in SN$ 

2. If t is neutral and for all t',  $t \rightarrow_{\beta} t'$  implies  $t' \in R_T$ , then  $t \in$ R<sub>T</sub>

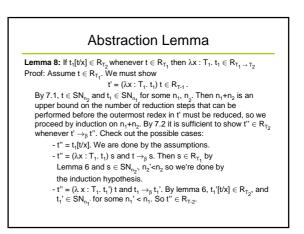
### Proof of Lemma 7

Proof by simultaneous induction on T

T = A. Both 1 and 2 are immediate

```
\mathsf{T}=\mathsf{T}_1\to\mathsf{T}_2.
```

1: Let  $t\in R_T$ . By the induction hypothesis (2),  $x\in R_{T_1}$ , so t $x\in R_{T_2}$ . Then t $x\in SN$ , so  $t\in SN$  as well. 2: Suppose t is neutral and whenever  $t \rightarrow_{\beta} t'$  then  $t' \in$  $R_{T}.$  Let  $t_{1} \in R_{T_{1}}.$  We show t  $t_{1} \in R_{T_{2}}.$  By the induction hypothesis (1),  $t_{1} \in SN_{n}$  for some n. We proceed by nested induction on n. It is sufficient to show  $t_2 \in R_{T_2}$  whenever  $t \ t_1 \rightarrow_\beta t_2$ , by the induction hypothesis (2), and since t  $t_1$  is neutral. Since t is neutral, either  $t \rightarrow_\beta t'$  and  $t_2 = t' \ t_1$ , or else  $t_1 \rightarrow_\beta t_1'$ , and  $t_2 = t \ t_1'$ . In the first case,  $t_2$  $\in \mathsf{R}_{\mathsf{T}_2}$  by the assumptions, and in the second,  $t_1{'} \in \mathsf{R}_{\mathsf{T}_1}.$ Then  $t_1' \in SN_{n'}$ , n' < n. So by the inner i.h. T  $t_1' \in R_{T_2}$ .



### **Fundamental Lemma**

 $\begin{array}{l} \textbf{Lemma 9: Suppose } x_1:T_1,...,x_n:T_n\vdash t:T. \mbox{ If } t_i\in R_{T_i} \mbox{ for all } i:1\leq i\leq n, \mbox{ then } t[t_1/x_1,...,t_n/x_n]\in R_T. \end{array}$ 

**Note:** This proves theorem 2, for n = 0.

Proof: By induction on size of the type derivation. Let  $\Gamma = x_1$ :  $T_1,...,x_n : T_n$  and  $\underline{t/x}$  abbreviate  $t_1/x_1,...,t_n/x_n$ .

- $t=x_i$  . Then  $t[\underline{t/x}]=t_i,\ T=T_i,$  and  $t_i\in R_{T_i}$  by the assumptions.
- $t = t^{*} t^{"}$ : By the induction hypothesis,  $t^{'}[\underline{t}'\underline{x}] \in \mathsf{R}_{T \to T}$  and  $t^{"}[\underline{t}'\underline{x}] \in \mathsf{R}_{T}^{'}$ . Then  $t[\underline{t}'\underline{x}] = (t^{*}t^{"})[\underline{t}'\underline{x}] = (t^{*}[\underline{t}'\underline{x}]) \in \mathsf{R}_{T}^{'}$ .
- $t = \lambda x^{"}$ : T". t'. Then  $T = T^{"} \rightarrow T'$ . Let  $t^{"} \in R_{T'}$  be arbitrary. By the induction hypothesis,  $t'[\underline{t}/\underline{x}, t''/x''] \in R_{T'}$ . But then  $\lambda x$ : T".  $t'[\underline{t}/\underline{x}] = t[\underline{t}/\underline{x}] \in R_{T}$  as desired.

### Exercise

**Exercise 13:** We did not require that  $t \in R_T$  only if  $\vdash t : T$ . Why was that?