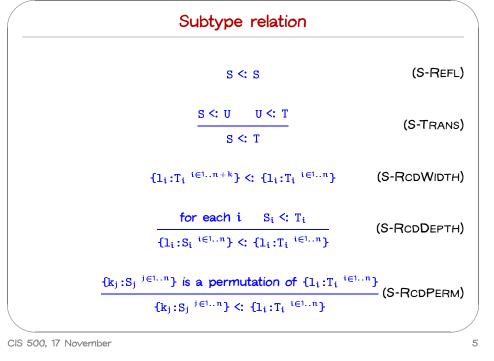
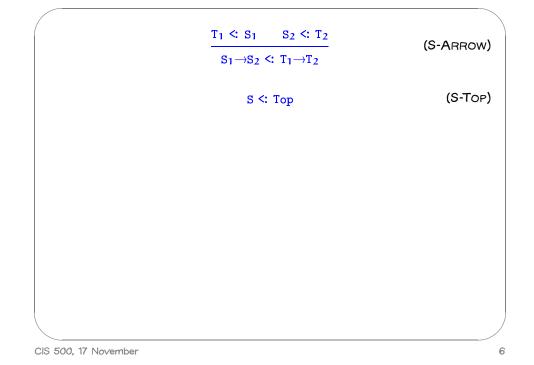


Comments on Exam

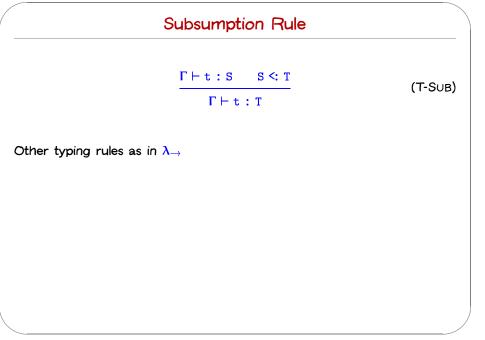
- Performance on "proof" parts was very bimodal (and strongly correlated with ability to draw derivation trees!)
- A number of people did poorly on the final question, which was taken verbatim from a recent homework.
 - There will be a homework problem on the final exam.
- In general, MT2 is a good indication of the difficulty of the final exam

Subtyping (Review)









Metatheory of Subtyping (Preview)

Syntax-directed rules

In the simply typed lambda-calculus (without subtyping), each rule can be "read from bottom to top" in a straightforward way.

> $\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}$ (T-APP) $\Gamma \vdash t_1 t_2 : T_{12}$

If we are given some Γ and some t of the form t_1 t_2 , we can try to find a type for t by

Syntax-directed sets of rules

that the set of typing rules is syntax-directed, in the sense that, for every

E.g., if t is an application, then we must proceed by trying to use T-APP. If we succeed, then we have found a type (indeed, the unique type) for

The second important point about the simply typed lambda-calculus is

"input" Γ and t, there one rule that can be used to derive typing

t. If it fails, then we know that t is not typable.

- 1. finding (recursively) a type for t_1
- 2. checking that it has the form $T_{11} \rightarrow T_{12}$
- 3. finding (recursively) a type for t_2
- 4. checking that it is the same as T_{11}

CIS 500, 17 November

statements involving t.

a

Technically, the reason this works is that We can divide the "positions" of the typing relation into input positions (Γ and t) and output positions (T).

- For the input positions, all metavariables appearing in the premises also appear in the conclusion (so we can calculate inputs to the "subgoals" from the subexpressions of inputs to the main goal)
- For the output positions, all metavariables appearing in the conclusions also appear in the premises (so we can calculate outputs from the main goal from the outputs of the subgoals)

 $\Gamma \vdash t_1 : T_{11} \rightarrow T_{12} \qquad \Gamma \vdash t_2 : T_{11}$ $\Gamma \vdash t_1 t_2 : T_{12}$

(T-APP)

10

CIS 500, 17 November

Non-syntax-directedness of typing

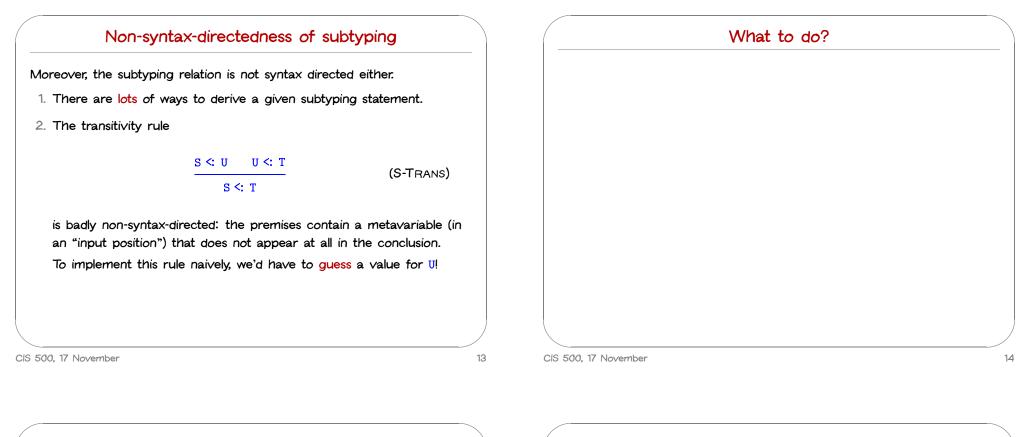
When we extend the system with subtyping, both aspects of syntax-directedness get broken.

1. The set of typing rules now includes two rules that can be used to give a type to terms of a given shape (the old one plus $T-S\cup B$)

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T}$$
(T-SUB)

2. Worse yet, the new rule T-SUB itself is not syntax directed: the inputs to the left-hand subgoal are exactly the same as the inputs to the main goal!

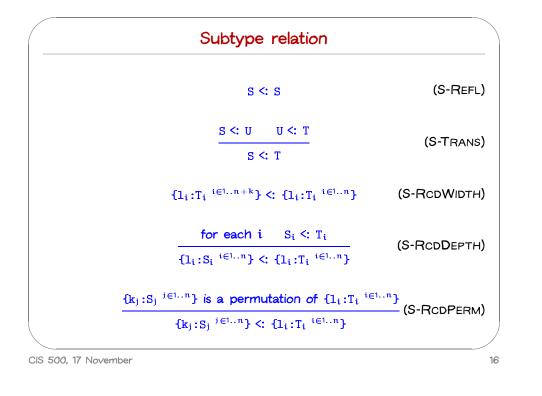
(Hence, if we translated the typing rules naively into a typechecking function, the case corresponding to T-SUB would cause divergence.)

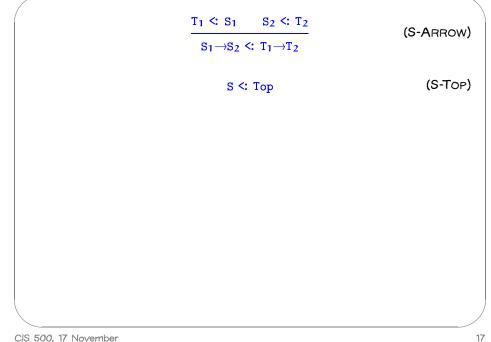


What to do?

- 1. Observation: We don't need 1000 ways to prove a given typing or subtyping statement one is enough.
 - \longrightarrow Think more carefully about the typing and subtyping systems to see where we can get rid of excess flexibility
- 2. Use the resulting intuitions to formulate new "algorithmic" (i.e., syntax-directed) typing and subtyping relations
- 3. Prove that the algorithmic relations are "the same as" the original ones in an appropriate sense.

Metatheory of Subtyping





lssues

For a given subtyping statement, there are multiple rules that could be used last in a derivation.

- 1. S-RCD-WIDTH, S-RCD-DEPTH, and S-RCD-PERM overlap with each other
- 2. S-REFL and S-TRANS overlap with everything

Step 1: simplify record subtyping

Idea: combine all three record subtyping rules into one "macro rule" that captures all of their effects

$$\frac{\mathbf{l}_{i} \ ^{i \in 1..n}\} \subseteq \{k_{j} \ ^{j \in 1..m}\} \qquad k_{j} = \mathbf{l}_{i} \text{ implies } s_{j} <: T_{i}}{\{k_{j} : s_{j} \ ^{j \in 1..m}\} <: \{\mathbf{l}_{i} : T_{i} \ ^{i \in 1..n}\}}$$
(S-RcD)

CIS 500, 17 November

Simpler subtype relation	
S <: S	(S-Refl)
<u>s <: u u <: t</u>	(S-Trans)
S <: T	
$\{\mathbf{l}_i^{-i\in 1.,n}\} \subseteq \{k_j^{-j\in 1.,m}\} \qquad k_j = \mathbf{l}_i \text{ implies } S_j <: 2$	[i (S-Rcd)
$\{k_j:S_j \in 1m\} \leq \{l_i:T_i \in 1n\}$	
$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \qquad \mathtt{S}_2 <: \mathtt{T}_2}{}$	(S-Arrow)
$S_1 \rightarrow S_2 <: T_1 \rightarrow T_2$	
S <: Top	(S-TOP)

17 November	20	CIS 500, 17
Even simpler subtype relation		
S <: U U <: T S <: T	(S-Trans)	Observ Lemma use S-
$\frac{\{\mathbf{l}_i^{-i\in 1n}\} \subseteq \{k_j^{-j\in 1m}\} \qquad k_j = \mathbf{l}_i \text{ implies } S_j <: T_j \\ \{k_j : S_j^{-j\in 1m}\} <: \{\mathbf{l}_i : T_i^{-i\in 1n}\}$	(S-Rcd)	use of
$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \rightarrow S_2 <: T_1 \rightarrow T_2}$	(S-Arrow)	
S <: Top	(S-Top)	
	/	

 Step 2: Get rid of reflexivity

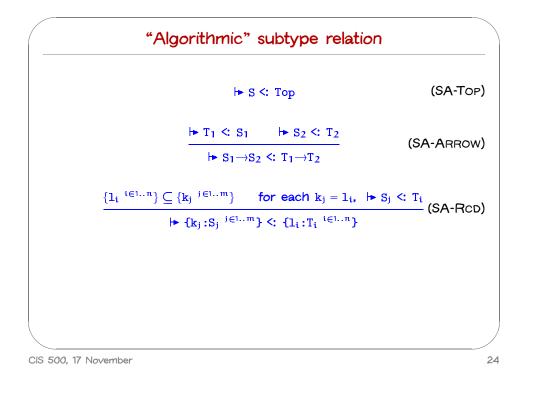
 Observation: S-REFL is unnecessary.

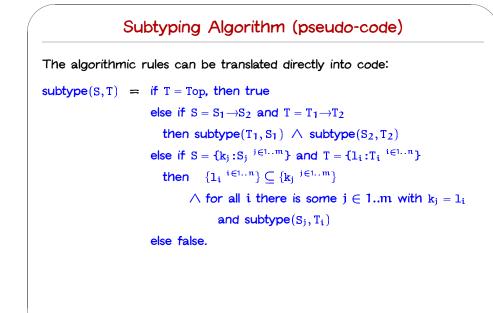
 Lemma: S <: S can be derived for every type without using S-REFL.</td>

CIS 500, 17 November

Step 3: Get rid of transitivity bservation: S-TRANS is unnecessary. mma: If S <: T can be derived, then there is a derivation that does not be S-TRANS.

21





Soundness and completeness

Theorem: $S \leq T$ iff $\vdash S \leq T$.

Proof: ...

Terminology:

♦ The algorithmic presentation of subtyping is sound with respect to the original if ► S <: T implies S <: T.</p>

(Everything validated by the algorithm is actually true.)

♦ The algorithmic presentation of subtyping is complete with respect to the original if S <: T implies I S <: T.</p>

(Everything true is validated by the algorithm.)

CIS 500, 17 November

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to {true, false} such that p(u) = true iff $u \in R$.

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that p(u) = true iff $u \in R$.

Is our subtype function a decision procedure?

CIS 500, 17 November

27-a

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that p(u) = true iff $u \in R$.

Is our subtype function a decision procedure?

Since subtype is just an implementation of the algorithmic subtyping rules, we have

1. if subtype(S, T) = true, then $\vdash S \leq T$

(hence, by soundness of the algorithmic rules, S <: T)

2. if subtype(S,T) = false, then not \models S <: T

(hence, by completeness of the algorithmic rules, not s <: T)

CIS 500, 17 November

27-b

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that p(u) = true iff $u \in R$.

Is our subtype function a decision procedure?

Since subtype is just an implementation of the algorithmic subtyping rules, we have

1. if subtype(S,T) = true, then \vdash S <: T

(hence, by soundness of the algorithmic rules, S <: T)

2. if subtype(S,T) = false, then not $\models S \leq T$

(hence, by completeness of the algorithmic rules, not s <: T)

Q: What's missing?

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that p(u) = true iff $u \in R$.

Is our subtype function a decision procedure?

Since subtype is just an implementation of the algorithmic subtyping rules, we have

1. if subtype(S, T) = true, then $\vdash S \leq T$

(hence, by soundness of the algorithmic rules, S <: T)

2. if subtype(S, T) = false, then not $\vdash S \leq T$

(hence, by completeness of the algorithmic rules, not s <: T)

Q: What's missing?

A: How do we know that subtype is a total function?

Decision Procedures

A decision procedure for a relation $R \subseteq U$ is a total function p from U to $\{true, false\}$ such that p(u) = true iff $u \in R$.

Is our subtype function a decision procedure?

Since subtype is just an implementation of the algorithmic subtyping rules, we have

1. if subtype(S,T) = true, then $\vdash S \leq T$

(hence, by soundness of the algorithmic rules, S <: T)

2. if subtype(S,T) = false, then not \models S <: T

(hence, by completeness of the algorithmic rules, not $S \leq T$)

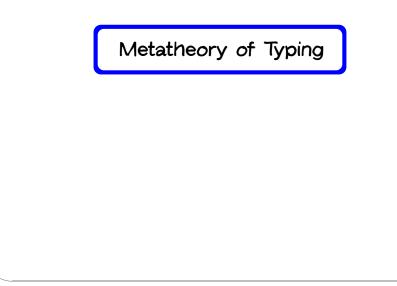
Q: What's missing?

A: How do we know that subtype is a total function?

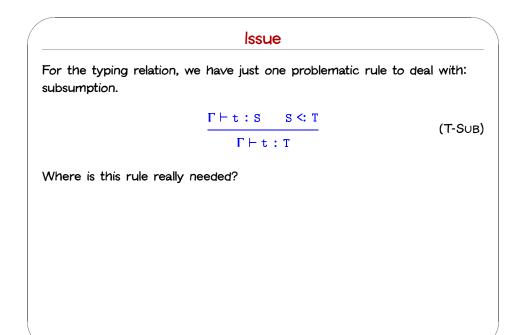
Prove it!

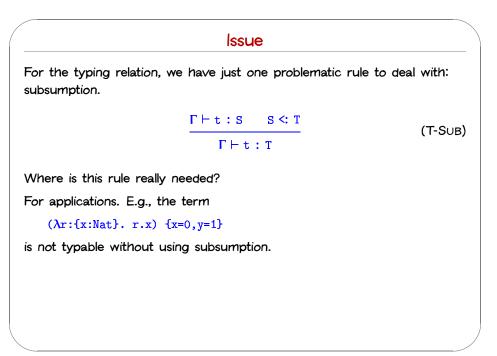
CIS 500, 17 November

27-е



CIS 500, 17 November





28

IssueFor the typing relation, we have just one problematic rule to deal with:
subsumption. $\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T}$ (T-SUB)Where is this rule really needed?For applications. E.g., the term
 $(\lambda r: \{x: Nat\}, r.x) \ \{x=0, y=1\}$
is not typable without using subsumption.Where else??

lssue

For the typing relation, we have just one problematic rule to deal with: subsumption.

 $\frac{\Gamma \vdash t: s \quad s \lt: T}{\Gamma \vdash t: T}$ (T-SUB)

Where is this rule really needed?

For applications. E.g., the term

 $(\lambda r: \{x:Nat\}, r.x) \{x=0, y=1\}$

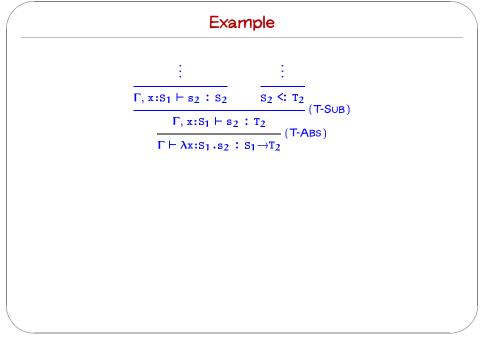
is not typable without using subsumption.

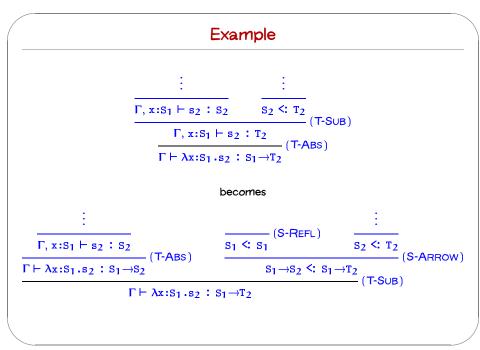
Where else??

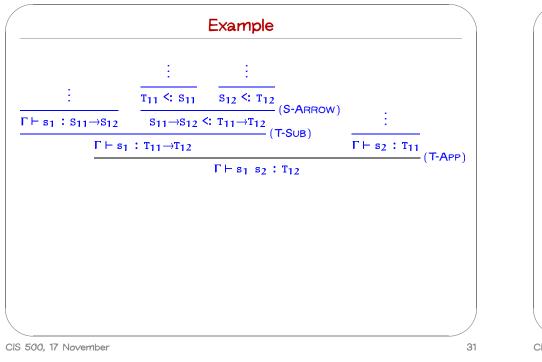
Nowhere else! Uses of subsumption to help typecheck applications are the only interesting ones.

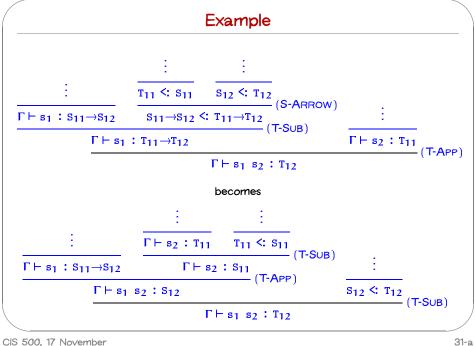
CIS 500, 17 November

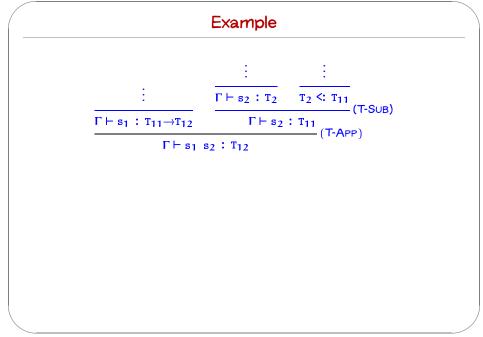
29-с

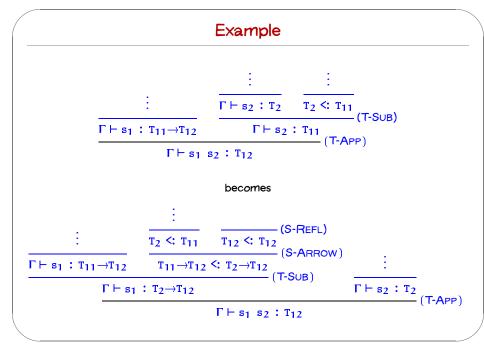


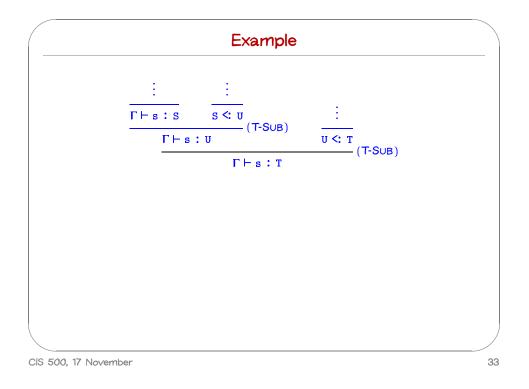


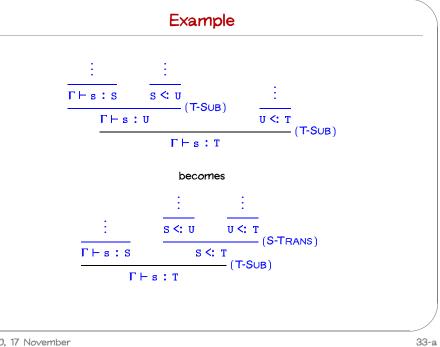












CIS 500, 17 November