Type-Driven Gradual Security Typing

Matías Toro, Ronald Garcia, Éric Tanter

Scenario

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Disney and Flanagan. “Gradual Information Flow Typing”

Scenario

Low Security Data

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Scenario

Low Security Data

High Security Data

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Scenario

Low Security Data

High Security Data

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Low Security Channel

Scenario

Low Security Data

High Security Data

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Low Security Channel

Security Leak!!

Unchecked Semantic Error

More Types!

$$\Gamma \vdash \heartsuit$$

Information-Flow Security Typing

Operating Systems

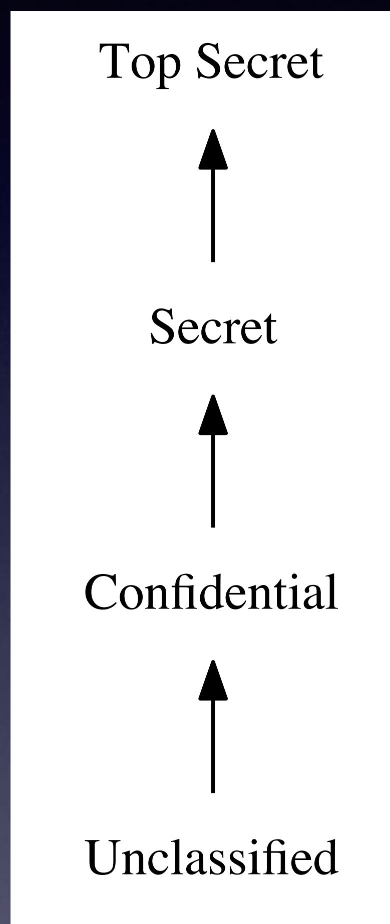
R.S. Gaines
Editor

A Lattice Model of Secure Information Flow

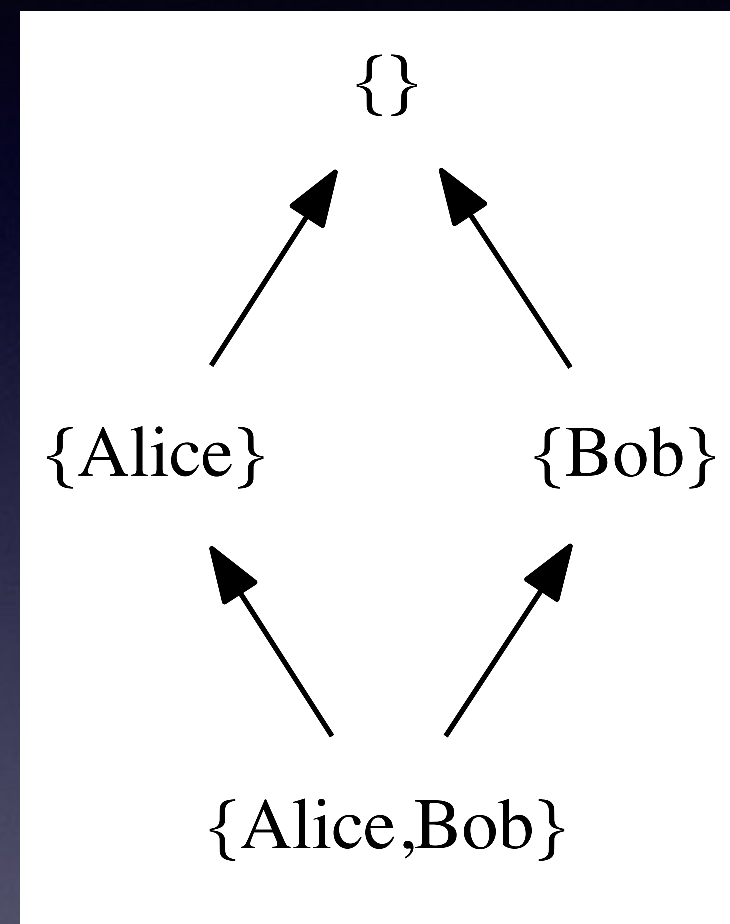
Dorothy E. Denning
Purdue University

[CACM 1976]

Security as a Lattice



Classification

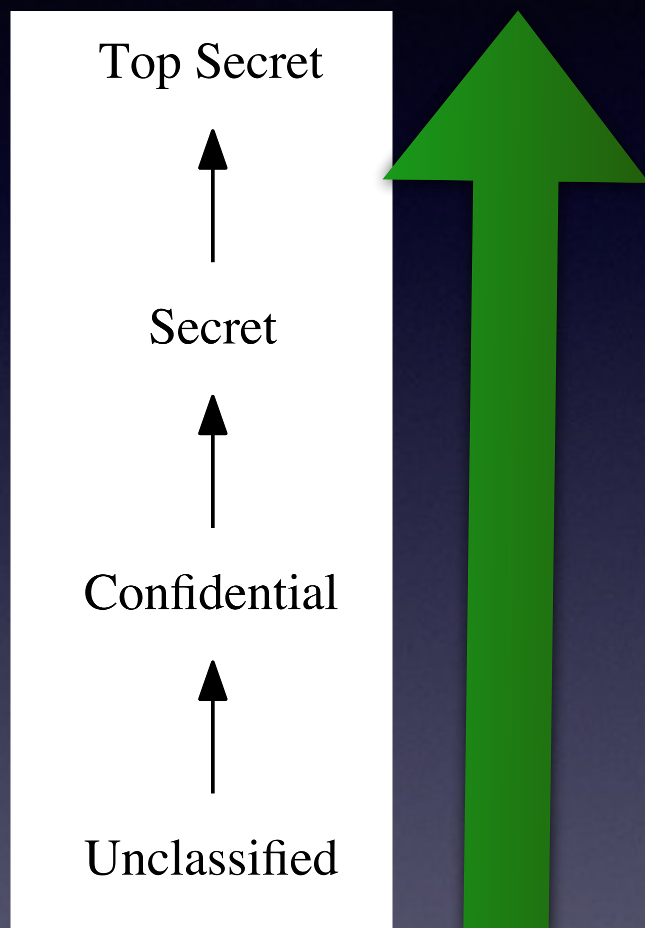


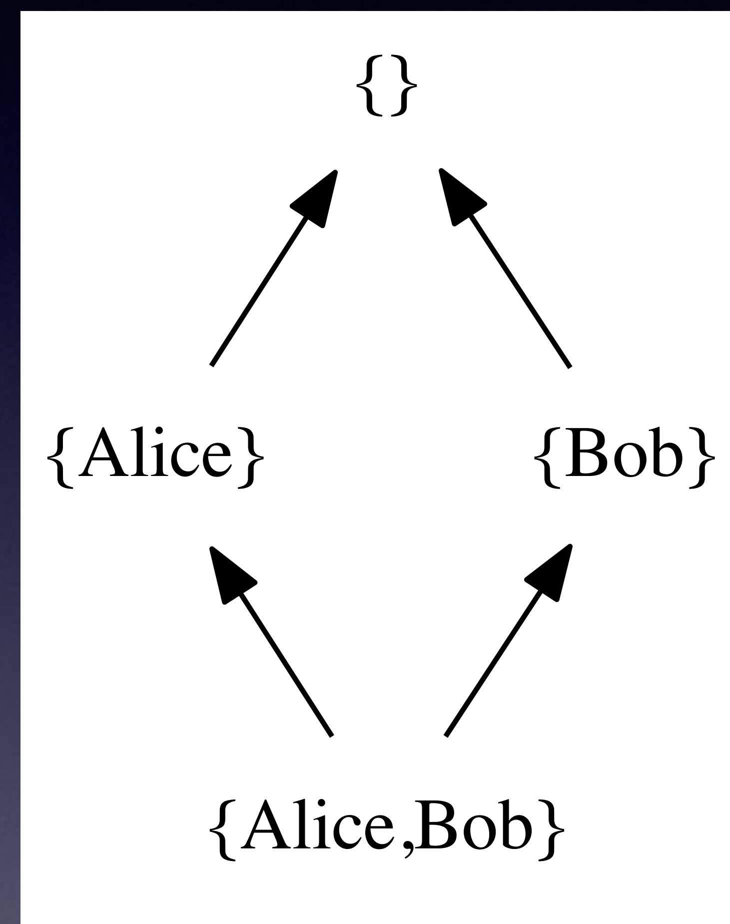
Permitted Readers

$$\boxed{\ell_1 \preccurlyeq \ell_2}$$

Zdancewic. “Programming Languages for Information Security”

Security as a Lattice

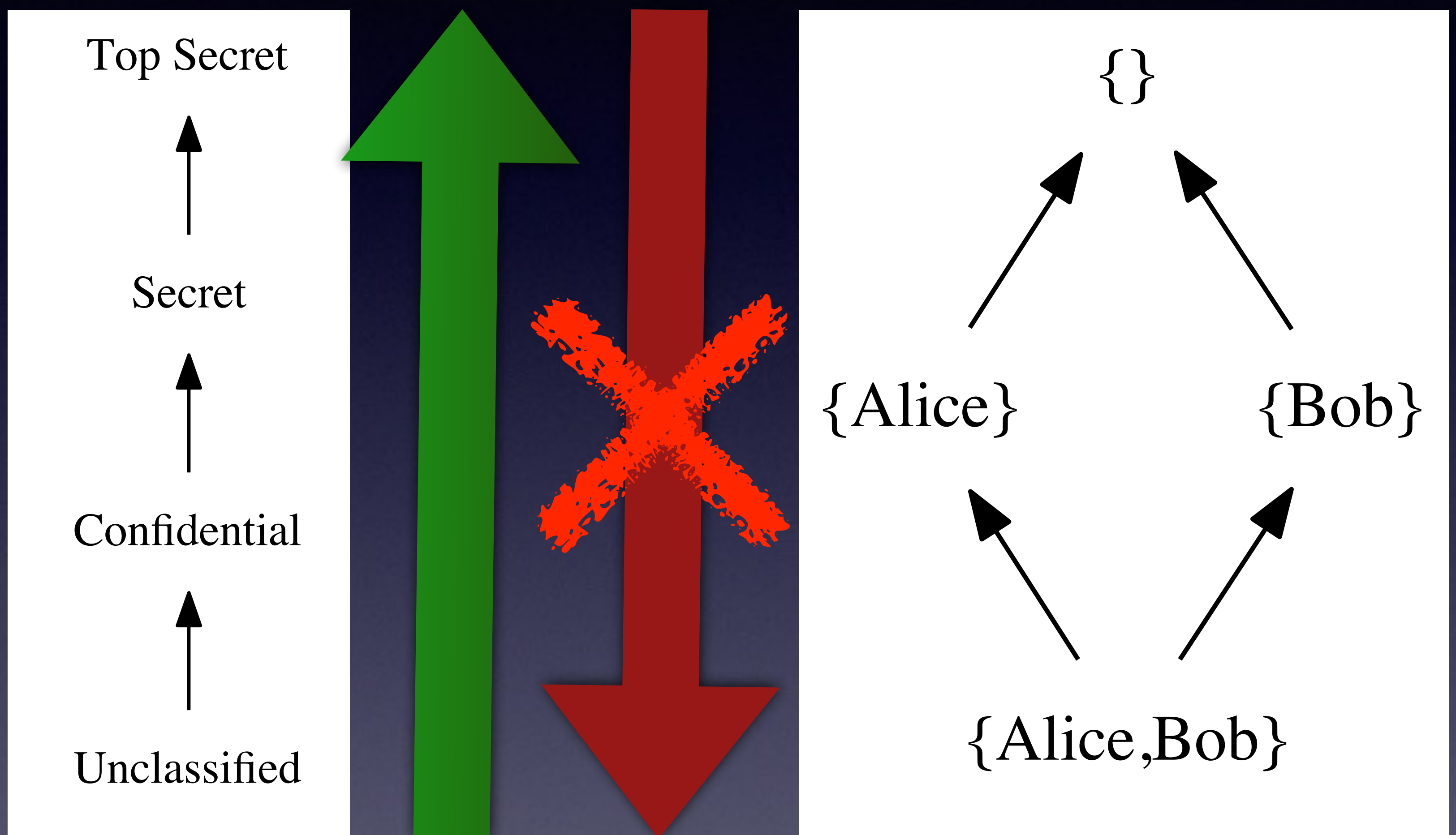




Low-security information may flow
to high-security contexts

Zdancewic. “Programming Languages for Information Security”

Security as a Lattice



High-security information may not flow to low-security contexts

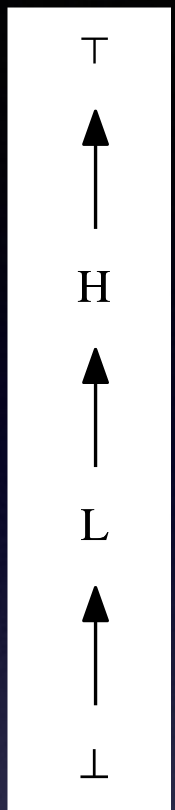
Zdancewic. “Programming Languages for Information Security”

Security Typing

$$\mathsf{Int} \longrightarrow \mathsf{Int}$$

Simple Types

Security Typing



Higher Security

Lower Security

$$\mathrm{Int}_{H} \longrightarrow_{L} \mathrm{Int}_{H}$$

Security-Indexed Types

Security Typing

$$\mathsf{Int}_L <: \mathsf{Int}_H$$

$$\mathsf{Int}_H \longrightarrow_L \mathsf{Int}_L \;<:\; \mathsf{Int}_L \longrightarrow_H \mathsf{Int}_H$$

Natural Subtyping Structure

Back to Scenario

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

Silent Leak

Simple Typing

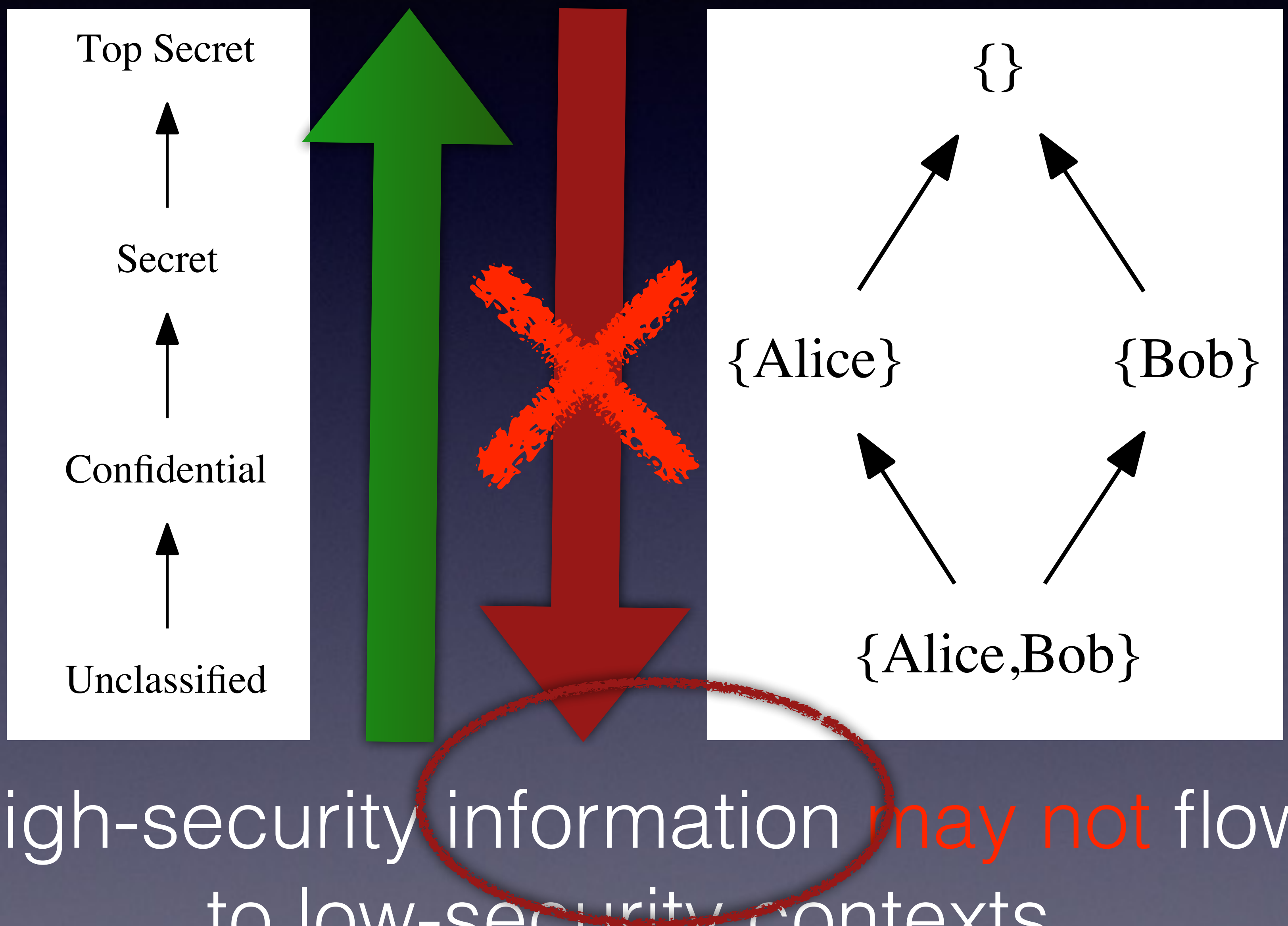
Scenario

```
let age : IntL = 31L
let salary : IntH = 58000H
let intToString : IntL →L StringL = ...
let print : StringL →L UnitL = ...
print(intToString(salary))
```

Type Error!

Security Typing

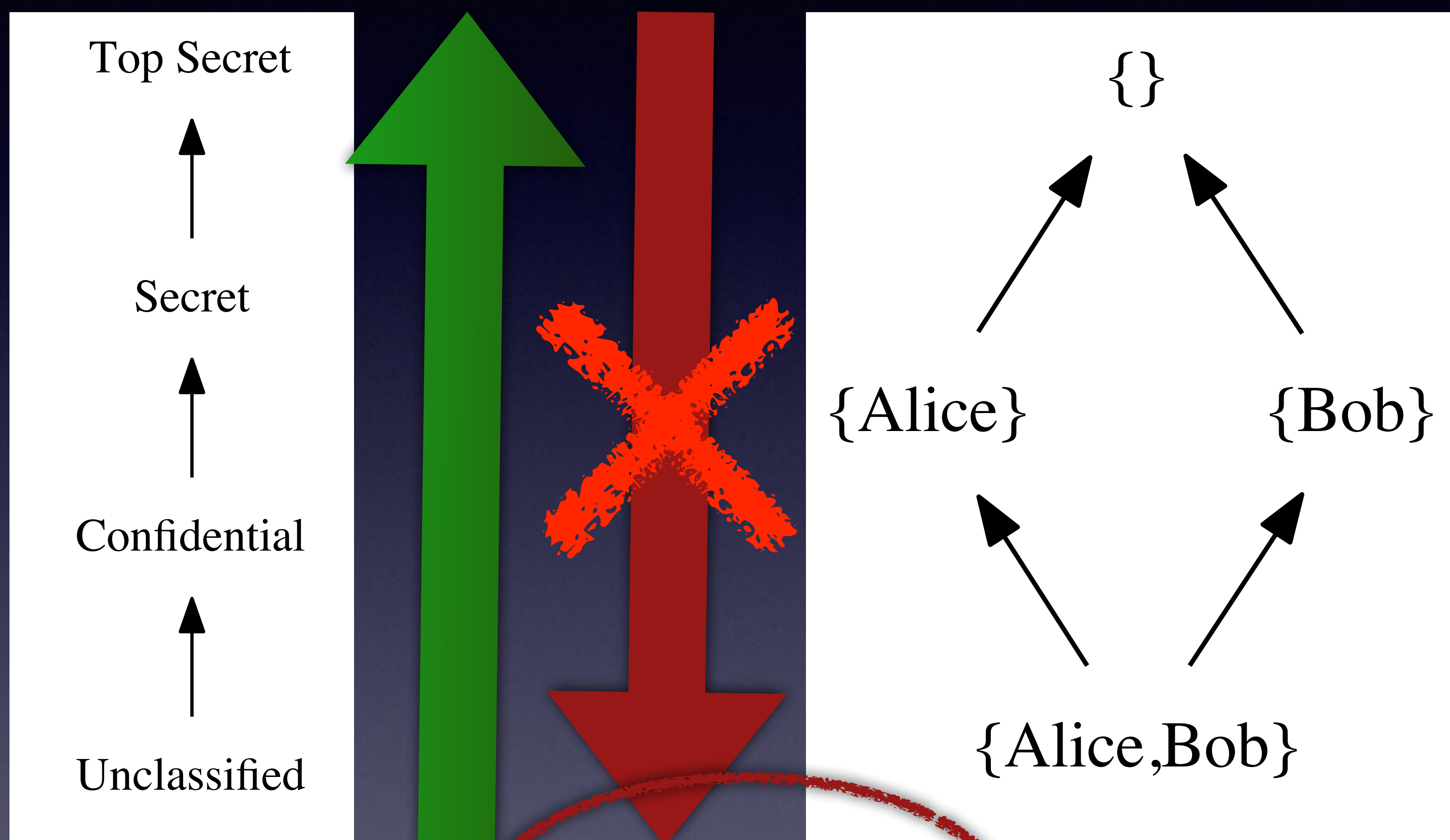
More Subtleties



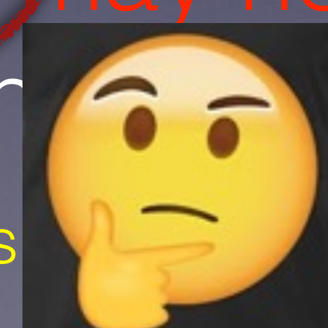
High-security information may not flow to low-security contexts

Zdancewic. “Programming Languages for Information Security”

More Subtleties



High-security information may not flow to low-security con



Zdancewic. “Programming Languages tion Security”

Implicit Information Flows

```
fun b : BoolH =>
  let tt : BoolL = true
  let ff : BoolL = false
  if b then tt else ff
```

High-Security **data** can affect **control** flow of a program

Implicit Information Flows

```
fun b : BoolH =>
  let tt : BoolL = true
  let ff : BoolL = false
  if b then tt else ff
```

High-Security **data** can affect **control** flow of a program

What's it's Type?

Implicit Information Flows

```
fun b : BoolH =>
  let tt : BoolL = true
  let ff : BoolL = false
  if b then tt else ff
```

High-Security **data** can affect **control** flow of a program

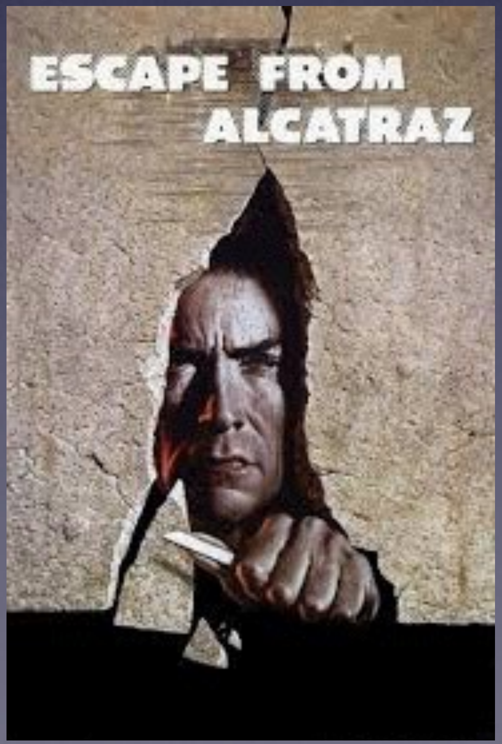
~~What's it's Type?~~

Where is it safe to use?

Assignment Can Leak Info!

```
let r : BoolL ref = ref tt
fun b : BoolH =>
  if b then ()L else (r := ff; ()L)
```

High-Security information can escape
via mutable state



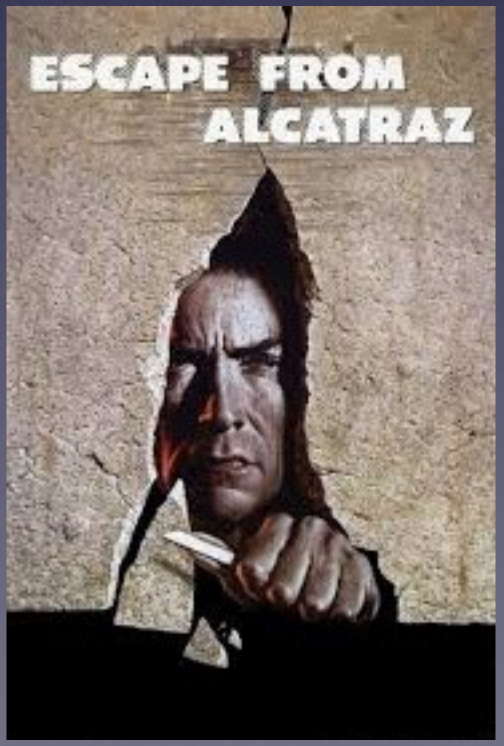
Assignment Can Leak Info!

```
let r : BoolL ref = ref tt
fun b : BoolH =>
  if b then ()L else (r := ff; ()L)
```

High-Security information can escape
via mutable state

~~What's it's Type?~~

Where is it safe to use?



Security Typing Judgment

$$\Gamma; \Sigma; \ell \vdash t : T$$

Security Typing Judgment

$$\Gamma; \Sigma; \ell \vdash t : T$$

How Can My Local Variables Behave?

Security Typing Judgment

$$\Gamma; \Sigma; \ell \vdash t : T$$

How Can My Local Variables Behave?

How Can Mutable References Behave?

Security Typing Judgment

$$\Gamma; \Sigma; \ell \vdash t : T$$

How Can My Local Variables Behave?

How Can Mutable References Behave?

What Security Information can
leak through Assignment

Security Typing Judgment

$$\Gamma; \Sigma; \ell \vdash t : T$$

How Does t Behave?

Carpal Typing Syndrome

```
let age : IntL = 31L
let salary : IntH = 58000H
let intToString : IntL →L StringL = ...
let print : StringL →L UnitL = ...
print(intToString(salary))
```

Type Error!

Security Typing

Secure All the Things!

```
let age
let sal                        H
let int                         StringL = ...
let pri                        itL = ...
print(i                        )
```

Type Error!

Security Typing

Gradual Typing

Dynamic Typing ⟷ Simple Typing

Gradual Typing!

Security

Simple Typing

Security Typing

Disney and Flanagan. "Gradual Information Flow Typing"

Gradual Typing!

Security

Simple Typing

Security Typing

Fennell and Thiemann, Gradual Security Typing with References

Simple Typing

```
let age : Int = 31
let salary : Int = 58000
let intToString : Int → String = ...
let print : String → Unit = ...
print(intToString(salary))
```

“Gradually Secure” Program

High Security Data

```
let age : Int = 31
let salary : IntH = 58000
let intToString : Int → String = ...
let print : StringL → Unit = ...
print(intToString(salary))
```

Low Security Channel

Runtime Error!

“Gradually Secure” Program

```
let age : Int = 31
let salary : IntH = 58000
let intToString : IntL → String = ...
let print : StringL → Unit = ...
print(intToString(salary))
```

Type Error!

“Gradually Secure” Program

```
let age : Int = 31
let salary : IntH = 58000
let intToString : IntL → String = ...
let print : StringL → Unit = ...
print(intToString(age))
```

All Good!

What do types tell us?

```
let mix : IntL -> IntH -> IntL =
  fun pub priv =>
    ...
```

Local Reasoning Principles???

What do types tell us?

```
let mix : IntL -> IntH -> IntL =
  fun pub priv =>
    ...
```

Take 1: Upper-bounds on security tags

Constrains any individual run of the code

Weak security guarantee

Proof Technique: Wright-Felleisen Type Safety

Disney and Flanagan. “Gradual Information Flow Typing”

Fennell and Thiemann, Gradual Security Typing with References

What do types tell us?

```
let mix : IntL -> IntH -> IntL =
  fun pub priv =>
    ...
```

Take 2: Non-interference

Constrains relationship among runs of the code

Strong security guarantee

Proof Technique: Logical relations

Modular, compositional, static reasoning about security

Heintze and Riecke. The Slam Calculus: Programming with Secrecy and Integrity

What do types tell us?

```
let mix : IntL -> IntH -> IntL =
```

Milner Award Lecture: The Type Soundness Theorem That You Really Want to Prove (and Now You Can) - POPL 2018

Type systems—and the associated concept of "type…

POPL18.SIGPLAN.ORG

Co … de

https://popl18.sigplan.org/event/popl-2018-papers-keynote-milner-lecture

Proof Technique: Logical relations

Modular, compositional, static reasoning about security

Heintze and Riecke. The Slam Calculus: Programming with Secrecy and Integrity

What do types tell us?

```
let mix : IntL -> IntH -> IntL =
  fun pub priv =>
    ...
```

Take 2: Non-interference

MISSION: achieve this richer meaning

Strong security guarantee

Modular, compositional, gradual reasoning about security

Gradual Security

$$\ell \in \textsc{Label}$$

$$g \in \textsc{GLabel} ::= \ell \mid ?$$

$$\textsc{Label} \subseteq \textsc{GLabel}$$

Unknown Label

“Gradually Secure” Program

```
let age : Int = 31
let salary : IntH = 58000
let intToString : IntL → String = ...
let print : StringL → Unit = ...
print(intToString(age))
```

“Gradually Secure” Program

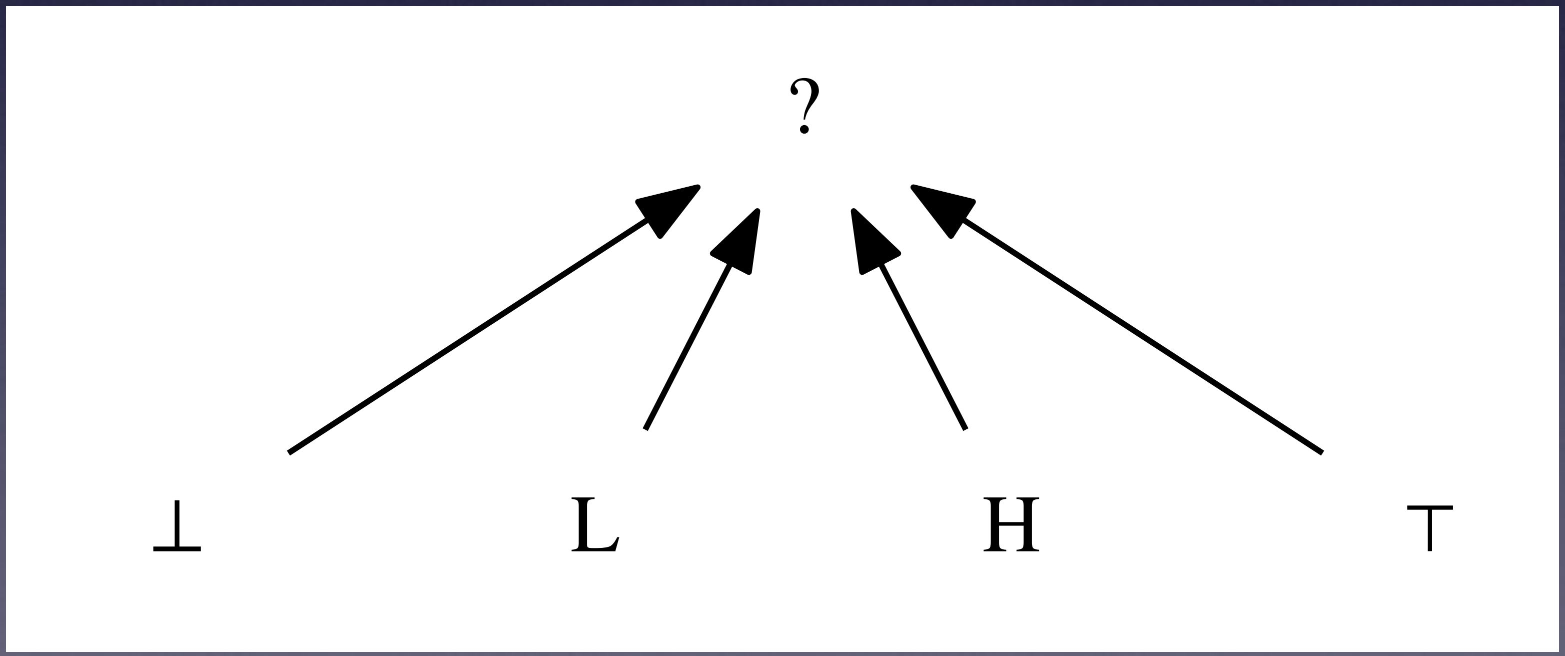
Desugared

```
let age : Int? = 31?
let salary : IntH = 58000?
let intToString : IntL →? String? = ...
let print : StringL →? Unit? = ...
print(intToString(age))
```

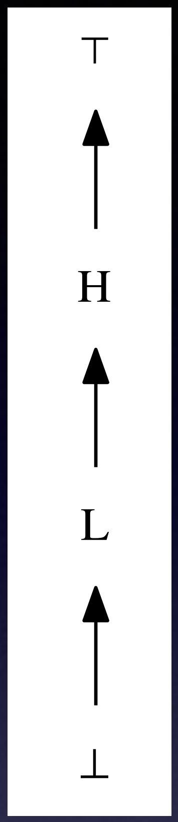
Gradual Language Embeds
Simply Typed and Security Typed Languages

Gradual Label Precision

$$g_1 \sqsubseteq g_2$$



Label Ordering



Higher Security

Lower Security

$$\ell_1 \preccurlyeq \ell_2$$

Consistent Label Ordering

$$\begin{array}{ccc} g_1 & \widetilde{\preccurlyeq} & g_2 \\ \rotatebox{90}{$\sqsubseteq$} & & \rotatebox{90}{$\sqsubseteq$} \\ \ell_1 & \preccurlyeq & \ell_2 \end{array}$$

for some ℓ_1, ℓ_2

Consistent Ordering

$$L \precsim H$$

$$H \not\precsim L$$

$$L \precsim L$$

$$? \precsim L$$

$$L \precsim ?$$

Conservatively Extends
Label Ordering

Consistent "Ordering"

$$L \precsim H$$

$$H \not\precsim L$$

$$L \precsim L$$

$$? \precsim L$$

$$L \precsim ?$$

Not really an order

Gradual Types

$U \in \textsc{GType}$

Just add gradual labels!

$\texttt{Bool}_L$

$\texttt{Int}_H$

$\texttt{Bool}_?$

$\textsc{Type} \subseteq \textsc{GType}$

Gradual Types

Type Precision $U_1 \sqsubseteq U_2$

Covariant on function types!

$$\begin{tikzcd}[column sep=small]
 & & \mathtt{Int}_? \to_? \mathtt{Bool}_? & & \\
\mathtt{Int}_L \to_? \mathtt{Bool}_? \arrow[urr] & & & & \mathtt{Int}_? \to_? \mathtt{Bool}_H \arrow[ull] \\
 & & \mathtt{Int}_? \to_\perp \mathtt{Bool}_? \arrow[uu] & & \\
 & \mathtt{Int}_L \to_\perp \mathtt{Bool}_? \arrow[uul] \arrow[ur] & & \mathtt{Int}_? \to_\perp \mathtt{Bool}_H \arrow[ul] \arrow[uur] & \\
 & & \mathtt{Int}_L \to_\perp \mathtt{Bool}_H \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Gradual Types

Consistent Subtyping

Conservatively extends subtyping (but not really a subtyping relation)

$$U_1 \lesssim U_2$$

if and only if $\sqsupseteq \qquad \sqsupseteq$

$$T_1 <: T_2$$

for some T_1, T_2

Consistent Subtyping

$\texttt{Int}_L \lesssim \texttt{Int}_H$

$\texttt{Int}_L \not\lesssim \texttt{Bool}_H$

$\texttt{Int}_H \not\lesssim \texttt{Int}_L$

$\texttt{Int}_H \lesssim \texttt{Int}_?$

$\texttt{Int}_? \lesssim \texttt{Int}_L$

$\texttt{Int}_? \not\lesssim \texttt{Bool}_H$

Conservatively Extends Subtyping

Consistent Subtyping

$$\texttt{Int}_L \lesssim \texttt{Int}_H$$

$$\texttt{Int}_L \not\lesssim \texttt{Bool}_H$$

$$\texttt{Int}_H \not\lesssim \texttt{Int}_L$$

$$\texttt{Int}_H \lesssim \texttt{Int}_?$$

$$\texttt{Int}_? \lesssim \texttt{Int}_L$$

$$\texttt{Int}_? \not\lesssim \texttt{Bool}_H$$

Not Transitive!

Consistent Subtyping

$\texttt{Int}_L \lesssim \texttt{Int}_H$

$\texttt{Int}_L \not\lesssim \texttt{Bool}_H$

$\texttt{Int}_H \not\lesssim \texttt{Int}_L$

$\texttt{Int}_H \lesssim \texttt{Int}_?$

$\texttt{Int}_? \lesssim \texttt{Int}_L$

$\texttt{Int}_? \not\lesssim \texttt{Bool}_H$

Not Transitive!

Does NOT denote safe substitutibility

Consistent Subtyping

$$\mathtt{Int}_L \lesssim \mathtt{Int}_H$$

$$\mathtt{Int}_L \not\lesssim \mathtt{Bool}_H$$

$$\mathtt{Int}_H \not\lesssim \mathtt{Int}_L$$

$$\mathtt{Int}_? \lesssim \mathtt{Int}_?$$

$$\mathtt{Int}_? \lesssim \mathtt{Int}_L$$

$$\mathtt{Int}_? \not\lesssim \mathtt{Bool}_H$$

Not Transitive!

Does NOT denote safe substitutibility

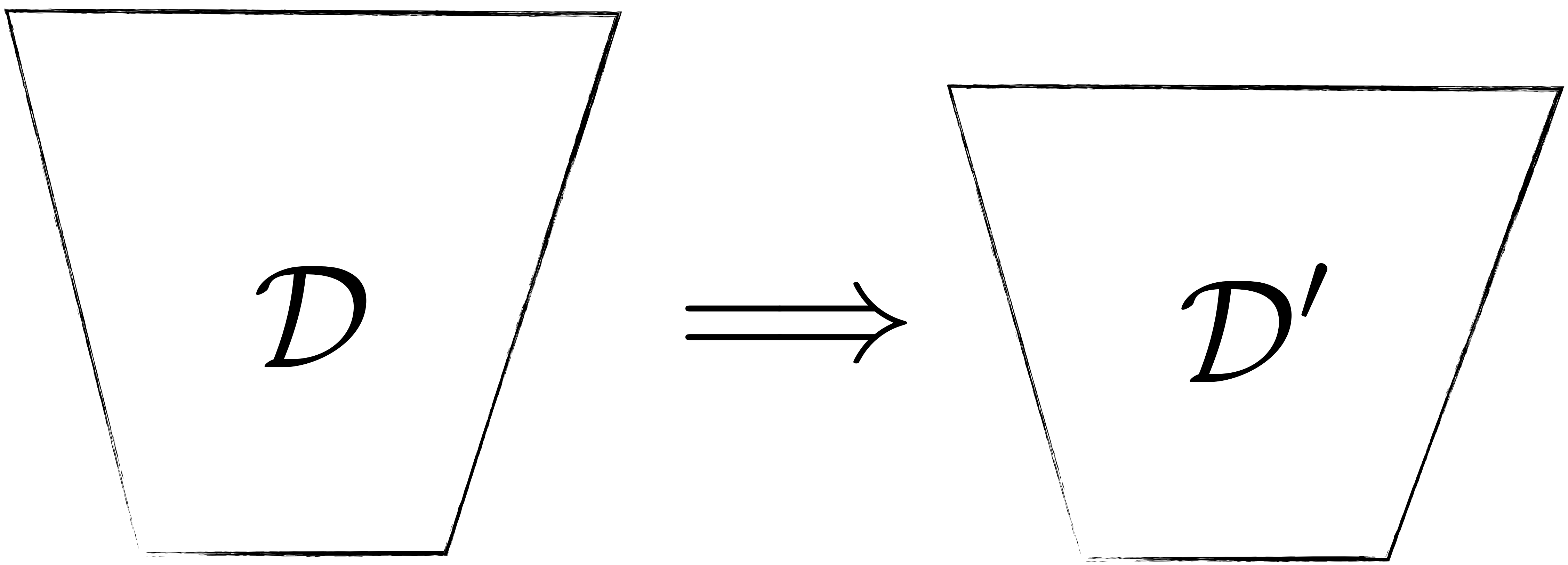
Not a Subtyping Relation!

Lifting Typing Rules

$$\frac{\Gamma;\Sigma;\ell_c \vdash t_1 : T_{11} \xrightarrow{\ell'}_{\ell_x} T_{12} \qquad \Gamma;\Sigma;\ell_c \vdash t_2 : T_2 \qquad T_2 <: T_{11} \qquad \ell_c \curlyvee \ell_x \preccurlyeq \ell'}{\Gamma;\Sigma;\ell_c \vdash t_1\ t_2 : T_{12} \curlyvee \ell_x}$$

$$\frac{\Gamma;\Sigma;g_c \vdash \widetilde{t}_1 : U_{11} \xrightarrow{g'}_{g_x} U_{12} \qquad \Gamma;\Sigma;g_c \vdash \widetilde{t}_2 : U_2 \qquad U_2 \lesssim U_{11} \qquad g_c \curlyvee g_x \preccurlyeq g'}{\Gamma;\Sigma;g_c \vdash \widetilde{t}_1\ \widetilde{t}_2 : U_{12} \mathbin{\widetilde{\curlyvee}} g_x}$$

Dynamic Semantics: Runtime Type Safety Argument



$\vdash t : T$ $\qquad$ $t \longmapsto t'$ $\qquad$ $\vdash t' : T$

Some extra complications

Garcia et al. Abstracting Gradual Typing (POPL 2016)

Noninterference (roughly)

$$\Gamma; \Sigma; g \vdash t : U \implies \Gamma; \Sigma; g \models t : U$$

Syntactic Type Judgment

Semantic Type Judgment

Semantic Type Soundness

Theses

- Types let you reason about program *fragments*
- Type Systems are not their Type Checkers
  - Type Systems are for *reasoning*
  - Type Checkers are for *enforcement*
  - Dynamic Checks are for *enforcement too*!

Conclusion

- Gradual typing is relative: not just for “scripting”
- Gradual typing conservatively extends two related languages
    - Syntax
    - Dynamic Semantics
    - Semantics *of types*

Image Credits

- “see no evil…” by ucumari (flickr)
- “concentration” by cdell (flickr)
- “Can you say ‘SAWHEEET’?” by locomotion (flickr)