Concurrent Program Logics
Lecture 1. Overview

Viktor Vafeiadis

What is a program logic?

- Formal system for reasoning about programs
- Assign specifications to programs
- Prove that program adheres to its spec.
- Adhere to correctness spec => no bugs
- *Concurrent program logic* =>
  program logic for concurrent programs

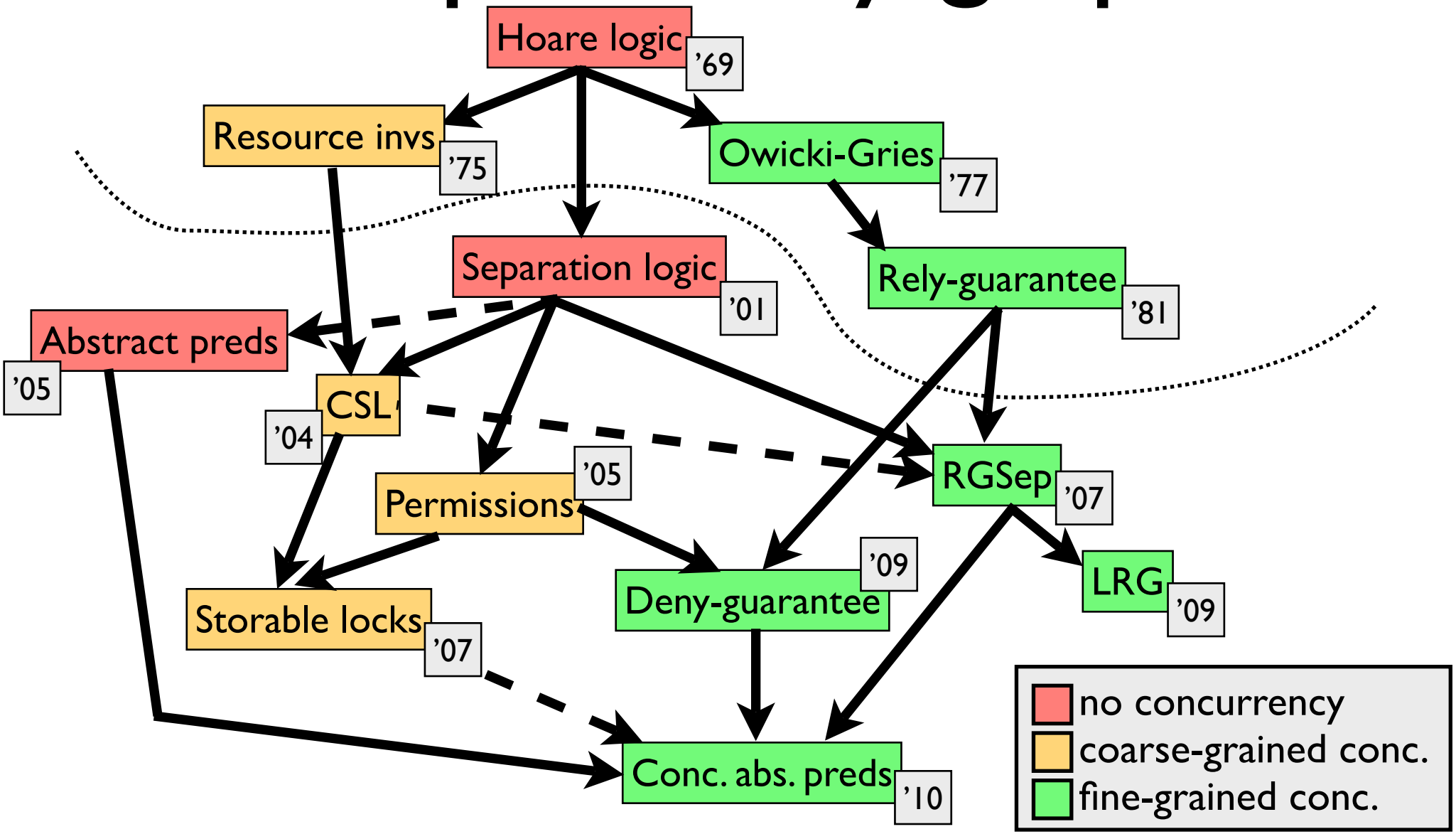
What is in this course?

- Study several concurrent program logics
- Prove programs correct using program logics
- Prove soundness of the program logics
- Mostly manual proofs
  (Automation/mechanisation later on)

Syllabus

- Hoare logic
- Owicki-Gries
- Separation logic
- Abstract predicates
- Concurrent separation logic (CSL)
- Permissions
- Storable locks
- Rely/guarantee
- RGSep
- Local rely-guarantee (LRG)
- Deny-guarantee
- Concurrent abstract predicates

Dependency graph



Programming language

Expressions:

E ::= x | n | E + E | E - E | ...

Boolean expressions:

B ::= T | F | E = E | E < E | ...

Commands:

C ::= skip | x := E | C; C | if B then C else C
| while B do C

Expression semantics

Store, s : var -> Z

Expressions, [[E]] s : Z

Boolean expressions, [[B]]s : bool

[[x]]s = s(x)
[[n]]s = n
$[[E_1 + E_2]]s = [[E_1]]s + [[E_2]]s$
...etc...

Command semantics

$x := E, s \rightarrow \text{skip}, s(x := [[E]]s)$

$\text{skip}; C, s \rightarrow C, s$

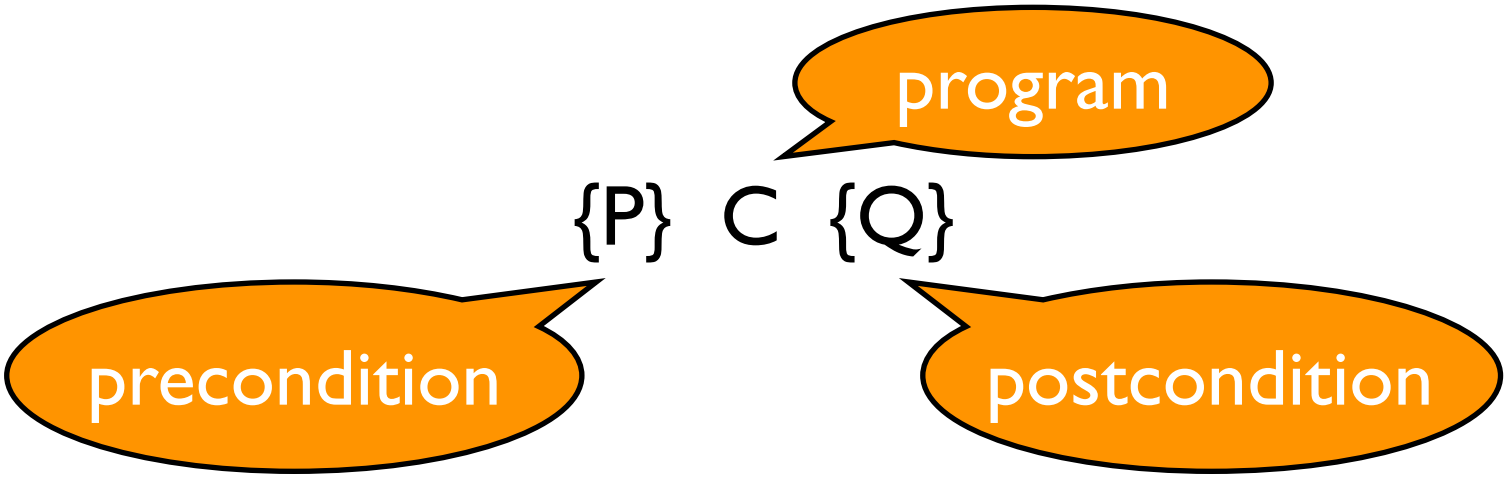
$C_1; C_2, s \rightarrow C'; C_2, s' \quad \text{if } C_1, s \rightarrow C', s'$

$\text{if } B \text{ then } C_1 \text{ else } C_2, s \rightarrow C_1, s \qquad \text{if } [[B]]s$

$\text{if } B \text{ then } C_1 \text{ else } C_2, s \rightarrow C_2, s \qquad \text{if not } [[B]]s$

$\text{while } B \text{ do } C, s$
$\rightarrow \text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip}, s$

Hoare logic



If $P(\sigma)$ and $C,\sigma \rightarrow^* \textbf{skip},\sigma'$ then $Q(\sigma')$

CAR Hoare. An axiomatic basis for computer programming. CACM 1969.

Assignment axiom

Forward (Floyd):

$$\{P\}\ \ x := E\ \ \{\exists x'.\ P[x'/x] \wedge x = E[x'/x]\}$$

Backward (Hoare):

$$\{Q[E/x]\}\ \ x := E\ \ \{Q\}$$

Skip:

$$\{P\}\ \ \textbf{skip}\ \ \{P\}$$

Structural rules

$$\frac{P \Rightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ C\ \{Q\}}$$

$$\frac{\{P\}\ C\ \{Q_1\} \qquad \{P\}\ C\ \{Q_2\}}{\{P\}\ C\ \{Q_1 \wedge Q_2\}}$$

$$\frac{\{P_1\}\ C\ \{Q\} \qquad \{P_2\}\ C\ \{Q\}}{\{P_1 \vee P_2\}\ C\ \{Q\}}$$

$$\frac{\{P\}\ C\ \{Q\} \qquad x \notin fv(C, Q)}{\{\exists x.\, P\}\ C\ \{Q\}}$$

Other proof rules

$$\frac{\{P\}\ C_1\ \{Q\} \qquad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

$$\frac{\{P \wedge B\}\ C_1\ \{Q\} \qquad \{P \wedge \neg B\}\ C_2\ \{Q\}}{\{P\}\ \textbf{if}\ B\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ C\ \{P\}}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{P \wedge \neg B\}}$$

Derived while rule

$$\frac{P \Rightarrow R \qquad \dfrac{\{R \wedge B\}\ C\ \{R\}}{\{R\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{R \wedge \neg B\}} \qquad R \wedge \neg B \Rightarrow Q}{\{P\}\ \textbf{while}\ B\ \textbf{do}\ C\ \{Q\}}$$

Heap commands

C ::= ...

| x := [E]

| [E] := E

| x := alloc E

| dispose E

Concurrency

Static threads & CCRs:

C ::= ... | C||C | with r when B do C
| let r in C

First class threads & locks:

C ::= ... | x := fork C | join E
| x := create_lock | free_lock E
| lock E | unlock E