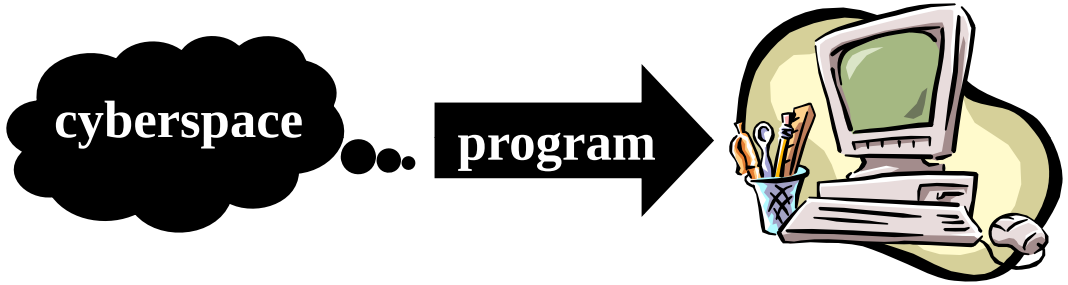
A Compositional Logic for Control Flow

Gang Tan, Boston College
Andrew W. Appel, Princeton University
Jan 8, 2006

Mobile Code Security

- Protect trusted system against untrusted code

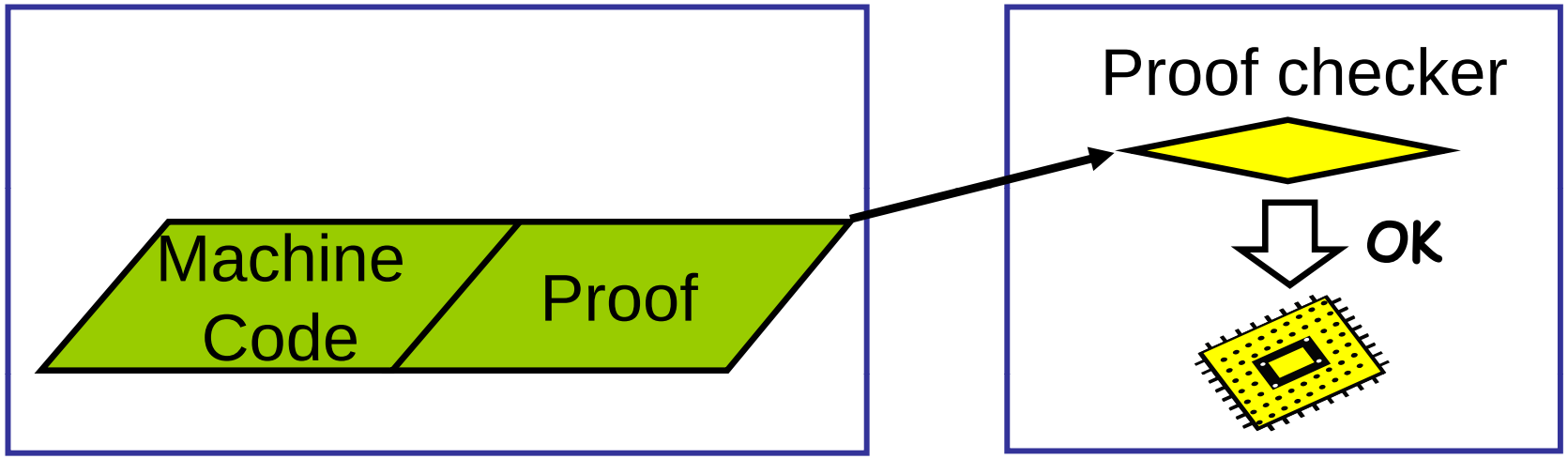


- Everybody loves extensibility
  - Extensible OS kernel
  - Web browsers, routers, switches, …

How to give foreign code direct access without compromising host integrity?

Foundational Proof-Carrying Code (FPCC)

- Code + a safety proof.
  - PCC [Necula & Lee 97], TAL[Morrisett et al. 98].



- FPCC [Appel & Felty 00]: The proof is w.r.t. raw machine semantics + HOL.

The proof is about machine code.

Require a Logic for Machine Code

Requirements for the Logic:

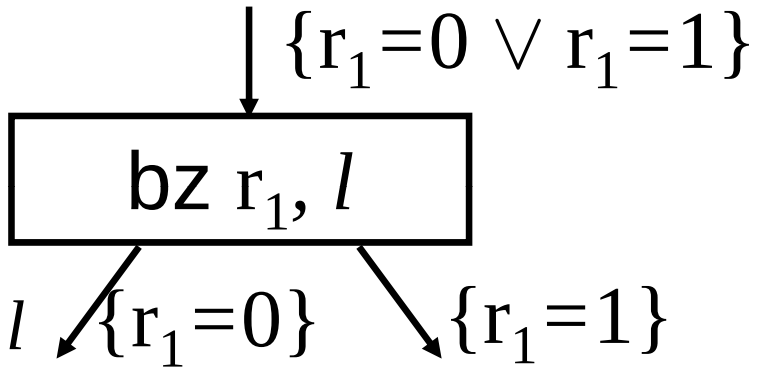
- Modularly reason about properties of machine code:
  - Unstructured control flow: direct jumps, indirect jumps, and pc-relative jumps.
  - Express properties about low-level abstractions (e.g., memory) and intermediate states.
- Satisfy the foundational requirement in FPCC:
  - Have a way to turn a derivation in the logic to a foundational proof, which is purely based on raw machine semantics.

What About Hoare Logic?

- Specification using Hoare triple: $\{p\}\ S\ \{q\}$
- For structured programs: no gotos
  - Written using constructs such as “if-then-else”, “repeat-until”, “while-do”, …
  - Each program fragment has exactly **one** entry and **one** exit.

Hoare Logic: Not Suitable for Machine-Language Programs

- Unstructured programs
  - Goto statements with unrestricted destinations.
  - Each program fragment has possibly **multiple** entries and **multiple** exits.



… [in Hoare Logic], it is not surprising that trouble arises in considering program segments with more than one mode of entry and/or exit. -- Michael J. O'Donnell, 1982

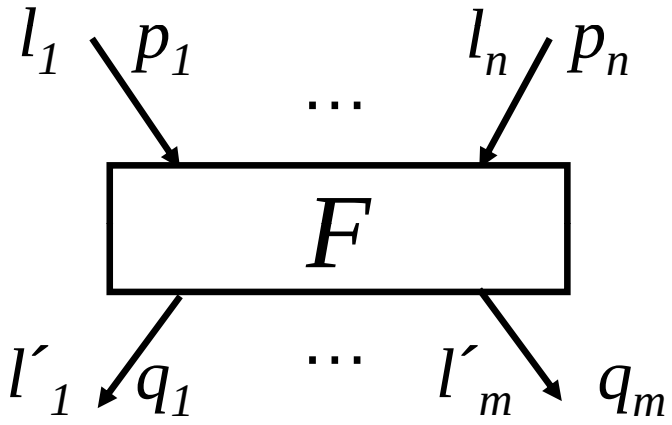
Talk Outline

- Motivation
- ➡ $\mathcal{L}_C$: A Logic for Machine-Language Programs
- Denotational Semantics of $\mathcal{L}_C$
- Implementation in FPCC and Related work

Multiple Entries and Multiple Exits

- Reasoning units: Multiple-entry and multiple-exit program fragments.

Informal syntax:



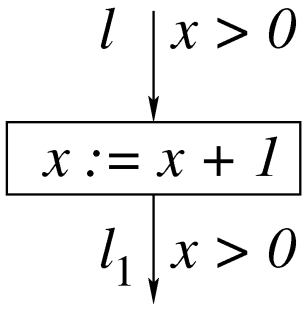
Formal syntax:

$$F\,;\ \underbrace{\{l'_1 \triangleright q_1, \ldots, l'_m \triangleright q_m\}}_{\text{Exits, } \Phi} \vdash \underbrace{\{l_1 \triangleright p_1, \ldots, l_n \triangleright p_n\}}_{\text{Entries, } \Psi}$$

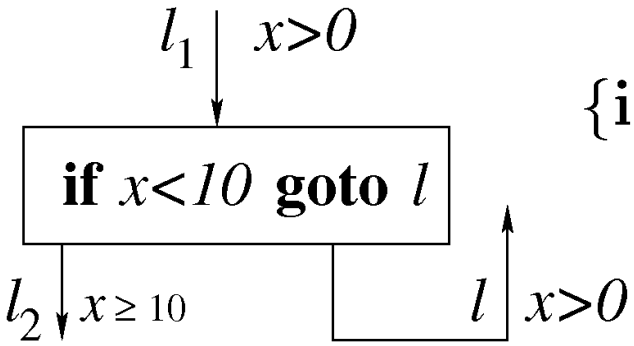
Rules for Individual Statements

- Rules for individual statements

Examples:



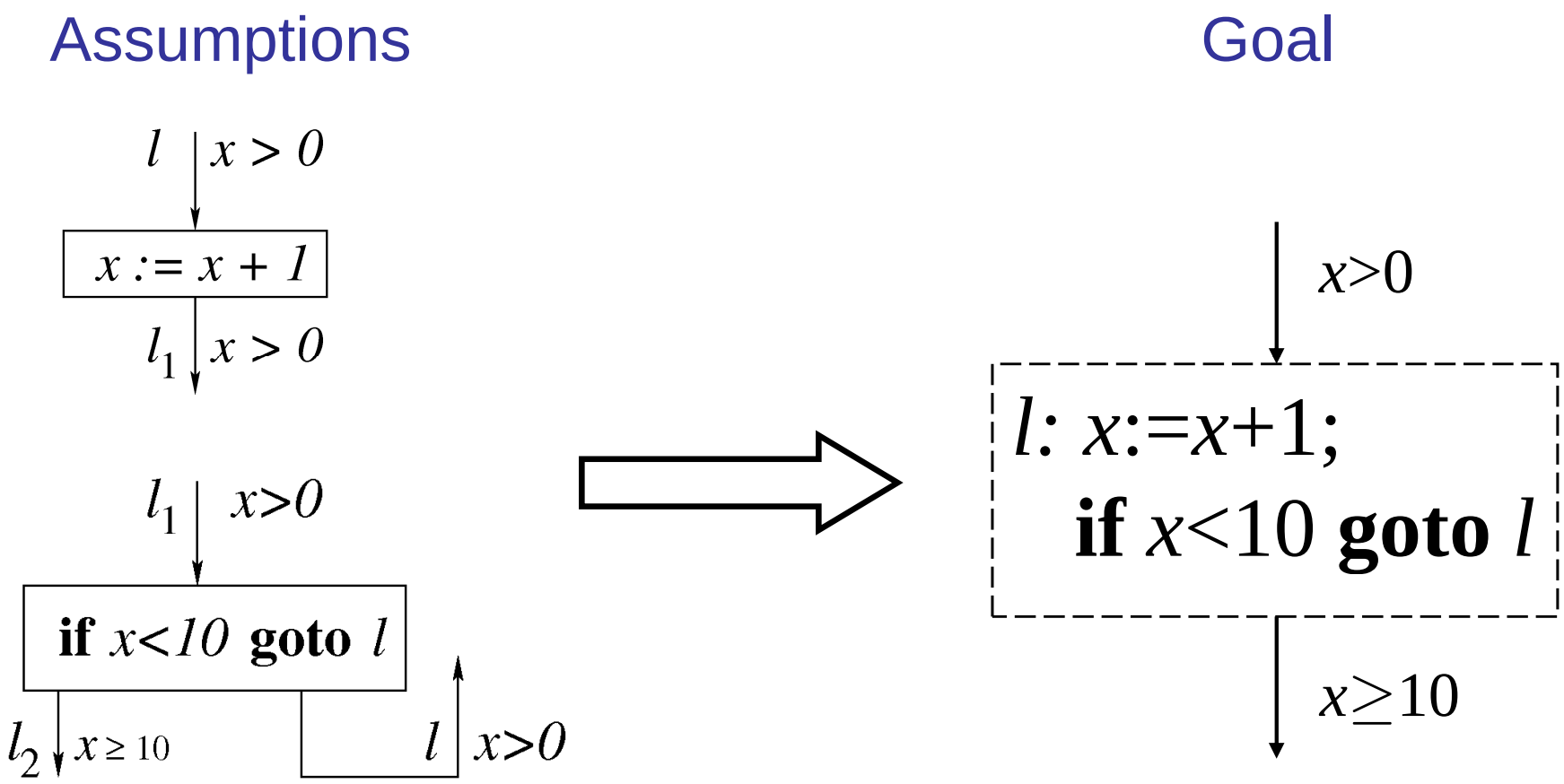
$$\{l : x := x + 1 : l_1\}; \{l_1 \triangleright (x > 0)\} \vdash \{l \triangleright (x > 0)\}$$



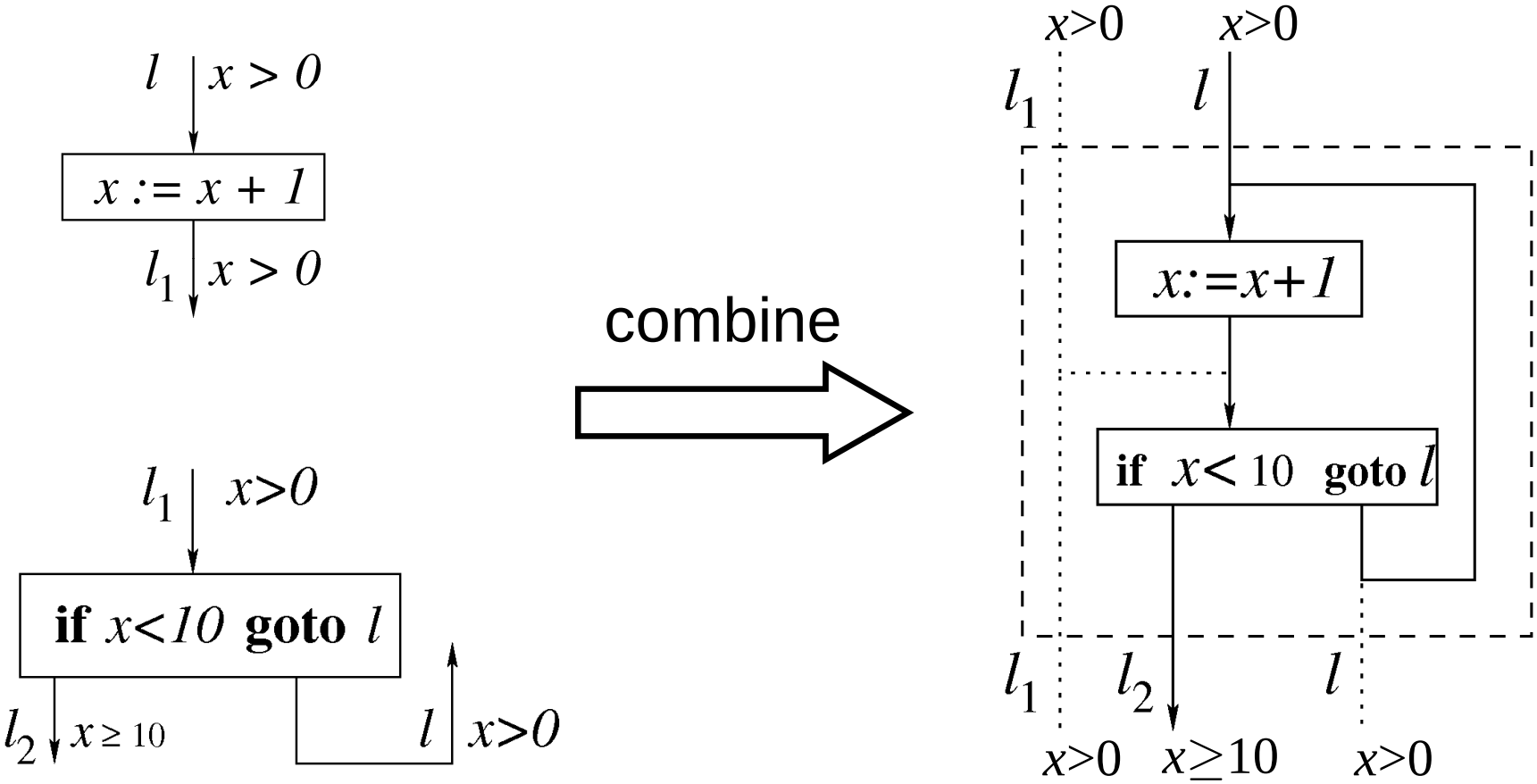
$$\{\textbf{if } x < 10 \textbf{ goto } l\}; \{l_2 \triangleright x \geq 10, l \triangleright (x > 0)\} \vdash \{l_1 \triangleright (x > 0)\}$$

Composition Rules

- Compose fragments together to form properties on the combined fragment.

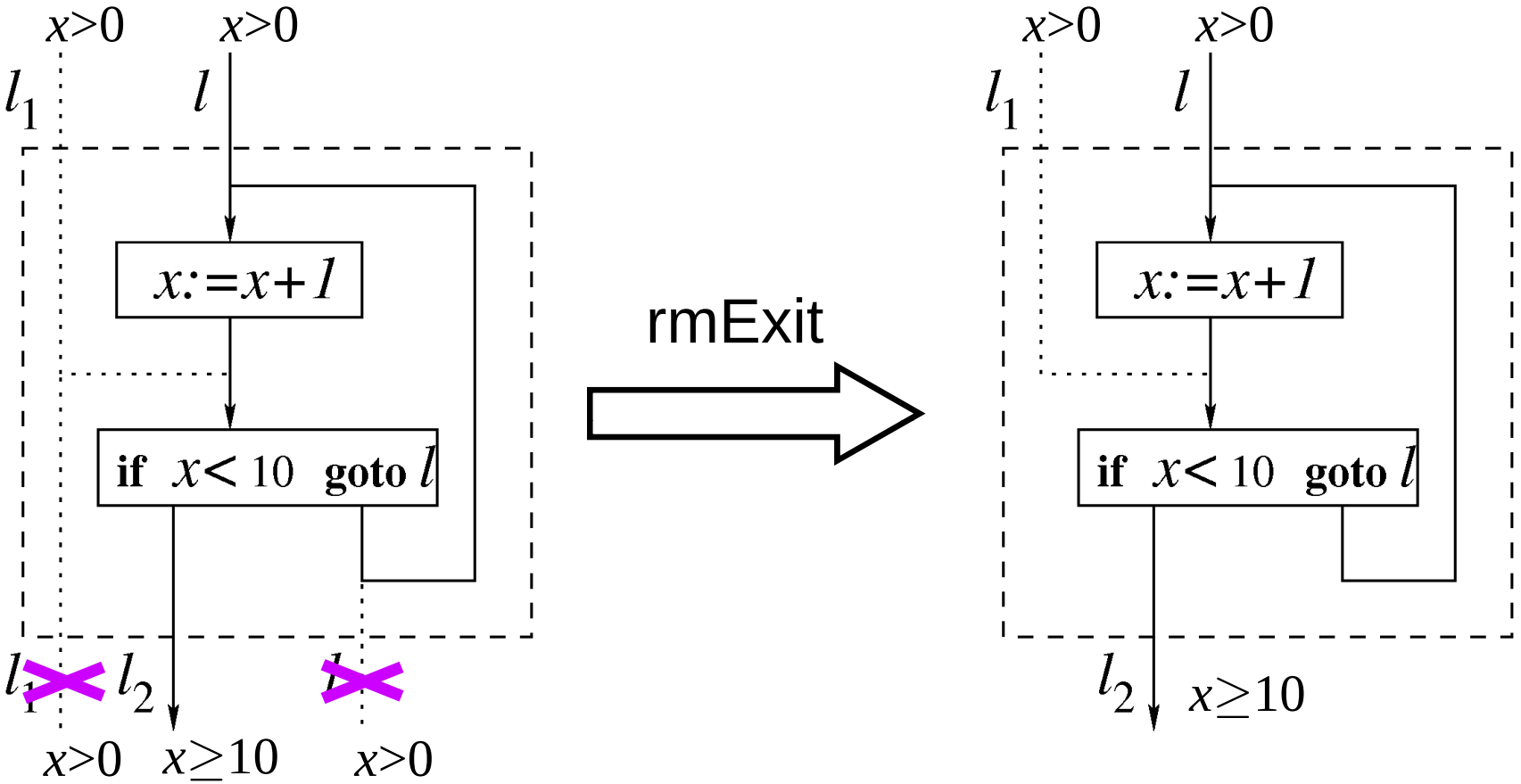


Step 1: Combining Fragments



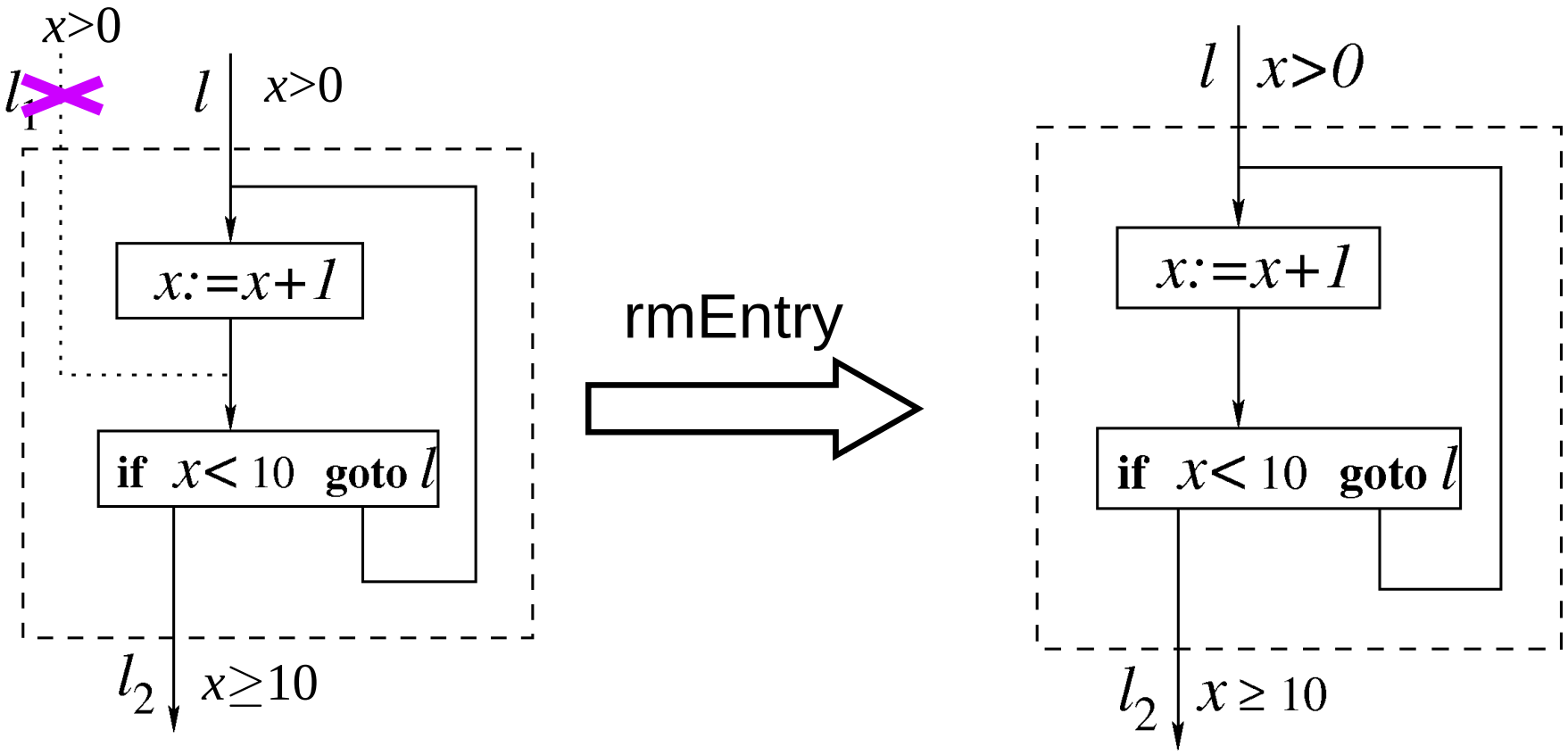
$$\frac{F_1 ; \Phi_1 \vdash \Psi_1 \quad F_2 ; \Phi_2 \vdash \Psi_2}{F_1 \cup F_2 ; \Phi_1 \cup \Phi_2 \vdash \Psi_1 \cup \Psi_2}\ \text{combine}$$

Step 2: Removing Exits



$$\frac{F\,;\ \Phi \cup \{l \triangleright p\} \ \vdash\ \Psi \cup \{l \triangleright p\}}{F\,;\ \Phi \ \vdash\ \Psi \cup \{l \triangleright p\}}\ \text{rmExit}$$

Step 3: Removing Entries



$$\frac{F\,;\,\Phi \vdash \Psi_1 \cup \Psi_2}{F\,;\,\Phi \vdash \Psi_1}\ \text{rmEntry}$$

$\mathcal{L}_C$'s Composition Rules

$$\frac{F_1 ; \Phi_1 \vdash \Psi_1 \quad F_2 ; \Phi_2 \vdash \Psi_2}{F_1 \cup F_2 ; \Phi_1 \cup \Phi_2 \vdash \Psi_1 \cup \Psi_2}\ \text{combine}$$

$$\frac{F ; \Phi \cup \{l \triangleright p\} \vdash \Psi \cup \{l \triangleright p\}}{F ; \Phi \vdash \Psi \cup \{l \triangleright p\}}\ \text{rmExit}$$

$$\frac{F ; \Phi \vdash \Psi_1 \cup \Psi_2}{F ; \Phi \vdash \Psi_1}\ \text{rmEntry}$$

- Fine-grained composition rules
  - Support reasoning about unstructured control flow
  - Support derivation of rules for common control-flow structures

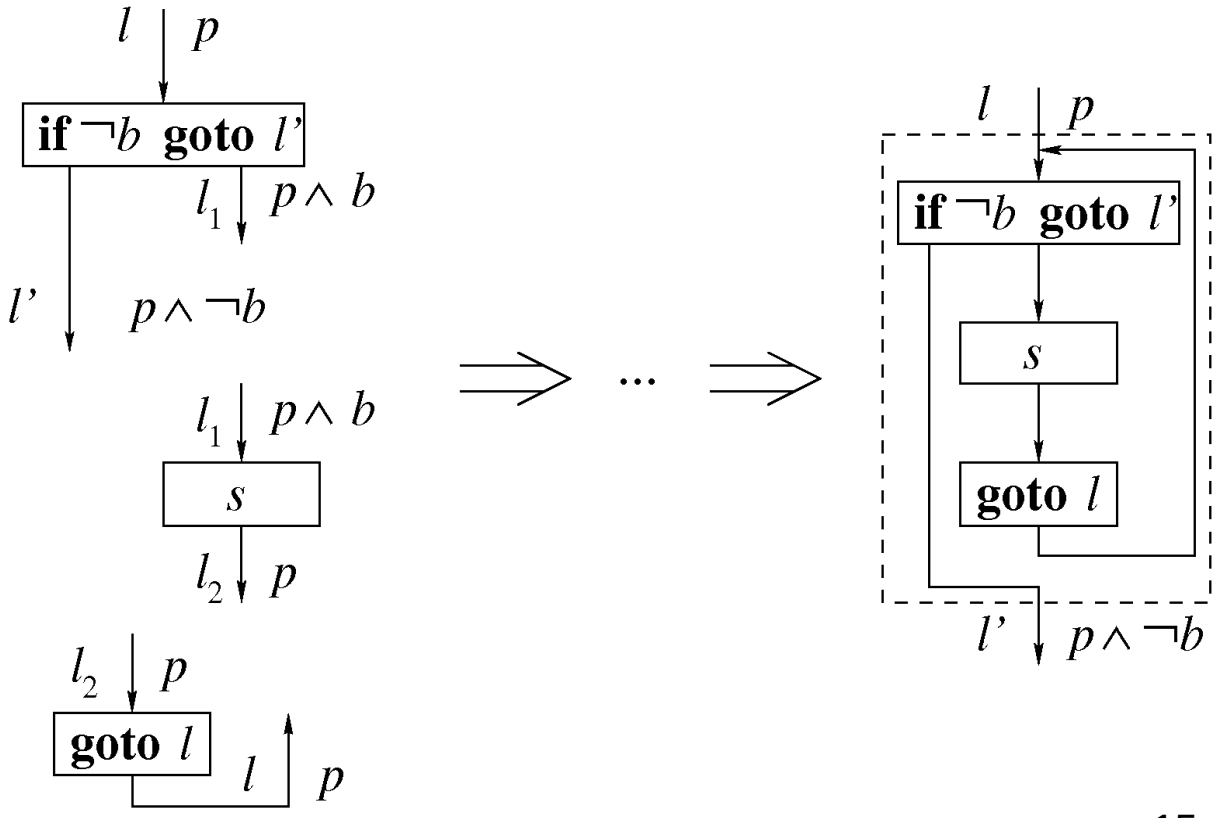
Deriving Hoare Logic Rules

In Hoare Logic:

$$\frac{\{p \wedge b\}s\{p\}}{\{p\}\textbf{while } b \textbf{ do } s\{p \wedge \neg b\}}$$

while b **do** s :

l : **if** Éb **goto** l'
l_1 : s
l_2 : **goto** l
l' :



Talk Outline

- Motivation
- $\mathcal{L}_C$: A Logic for Machine-Language Programs
- ➡ Denotational Semantics of $\mathcal{L}_C$
  - Give $\mathcal{L}_C$ a denotational semantics based on HOL and machine semantics.
  - **Convert a derivation in $\mathcal{L}_C$ to a proof w.r.t. HOL and machine semantics.**
  - A naïve semantics won't work.
- Implementation in FPCC and Related work

Machine States and Step relation

- Machine states: σ
- Small step operational semantics: $\sigma \mapsto \sigma'$
- A stuck state σ: no σ' to step to.

Semantics of $l \triangleright p$: Continuations

- $l \triangleright p$ being true in a state σ

  Safe to continue from the label l
  provided that the assertion p is met.

$$\begin{aligned} & \sigma \models l \triangleright p \\ \triangleq\ & (\mathsf{control}(\sigma) = l) \ \wedge \ (p \text{ true in } \sigma) \\ & \Rightarrow \ (\sigma \mapsto^{\omega} \cdots) \end{aligned}$$

- $l \triangleright p$ being **approximately** true:

Indexed Model: Appel & McAllester

$$\begin{aligned} & \sigma \models_k l \triangleright p \\ \triangleq\ & (\mathsf{control}(\sigma) = l) \ \wedge \ (p \text{ true in } \sigma) \\ & \Rightarrow \ (\sigma \mapsto^{k} \cdots) \end{aligned}$$

Semantics of $F; \Phi \vdash \Psi$

- A set of continuations being approx. true.

$$\begin{aligned} & \sigma \models_k \Psi \\ \triangleq\ & \forall (l \triangleright p) \in \Psi.\ \sigma \models_k l \triangleright p \end{aligned}$$

- Semantics:

$$\begin{aligned} & F\,;\ \Phi \models \Psi \\ \triangleq\ & \forall \sigma, k.\ \mathsf{loaded}(F, \sigma) \ \wedge\ \sigma \models_k \Phi \\ & \qquad \Rightarrow\ \sigma \models_{k+1} \Psi \end{aligned}$$

Requires at least one computation step from an entry to an exit.

Why Is the One-Step Requirement?

- Because of the rmExit rule:

$$\frac{F\,;\,\Phi \cup \{l \triangleright p\} \vdash \Psi \cup \{l \triangleright p\}}{F\,;\,\Phi \vdash \Psi \cup \{l \triangleright p\}}\ \text{rmExit}$$

Special case of rmExit:

$$\frac{F\,;\,\{l \triangleright p\} \vdash \{l \triangleright p\}}{F\,;\,\{\} \vdash \{l \triangleright p\}}$$

Without the one-step requirement, the rule would be like:
From “A imply A”, derive A.

Our semantics assume the left to approximation k, and prove the right to approximation k+1.

Soundness and Completeness

- **Soundness**: If $F; \Phi \vdash \Psi$, then $F; \Phi \models \Psi$.
- **Relative Completeness**:
  If $F; \Phi \models \Psi$, then $F; \Phi \vdash \Psi$.
  - With some assumptions:
    - Assume a complete derivation system for the assertion language.
    - Assume the assertion language is expressive enough.
  - Adaptation of Cook's completeness proof for Hoare Logic

$\mathcal{L}_C$ in Princeton's FPCC Project

- $\mathcal{L}_C$ is implemented as an intermediate logic in FPCC.
  - With machine-checked soundness proofs.
  - Utilized to derive memory-safety proofs of SPARC machine programs.
  - Around 30k lines of Twelf proofs.
  - Handle **indirect jumps** and pc-relative jumps.
- Assertion language is a rich typed language:
  - **Continuation types**, polymorphic and existential types, mutable references, …

Related Work: Program Logics for Unstructured Programs

- Early work
  - Clint & Hoare 69; Kowaltowski 77; Arbib & Alagic 79; de Bruin 81; TAL: Morrisett et al. 98
- de Bruin's system
  - Separate rules for different control-flow constructs
  - Not modular: Need global invariants

de Bruin's System: Need Global Invariants

$$\langle L_1 : p_1, \ldots, L_n : p_n | \{p\} s \{q\} \rangle$$

Global invariant:
all label invariants in a program

- Composition requires matching of global labels
  - $\langle \quad | \ \{x{>}0\}\ x :@ x + 1 \quad \{x{>}0\} \rangle$
  - $\langle l{:}(x{>}0) \mid \{x{>}0\}\ \textbf{if}\ x{<}10\ \textbf{goto}\ l\ \{x{\geq}10\} \rangle$

☹: Cannot Compose!

Related Work

- Floyd’s flowchart verification.
- Cardelli 97: Linking logic.
- Glew and Morrisett 99: Modular typed assembly language.
- Benton 05, Saabas & Usstalu 05.
  - Labels are associated with pre and post conditions.

The End