This Week

- Lecture on relational semantics
- Exercises on logic and relations
- Labs on using Isabelle to do proofs

Synthesis, Analysis, and Verification

Lecture 02a

Relational Semantics

Lectures:

Viktor Kuncak

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

More Relations and Functions

$r \subseteq A \times B$

functional on A: $\forall \underset{}{x}^{\in A}, y_1^{\in A}, y_2^{\in A}.\ (x, y_1) \in r \wedge (x, y_2) \in r \rightarrow y_1 = y_2$

total on A×B $\quad \forall x \in A\ \exists y \in B.\ (x, y) \in r$

$r : A \rightarrow B \quad$ r is functional $\wedge$ total on A×B

injective: def 1: r^{-1} is functional

def2: $\forall x, y.\ f(x) = f(y) \rightarrow x = y$ $\quad$ | def 2 $\rightarrow$ def 1:

$\quad\quad\quad\quad\quad\quad\quad\quad z$

surjective: r^{-1} is total

bijective: $\quad$ injective $\wedge$ surjective

Function Updates

$\mathrm{dom}(r) = \{x \mid \exists y.\ (x,y) \in r\}$ domain

$\mathrm{ran}(r) = \{y \mid \exists x.\ (x,y) \in r\}$ range

Partial function $f : A \hookrightarrow B$ is functional relation $f \subseteq A \times B$

$f : A \hookrightarrow B$, $g : A \hookrightarrow B$

$$f \oplus g = \{ (x,y) \mid [(x,y) \in f \land x \notin \mathrm{dom}(g)] \lor (x,y) \in g \}$$

$f(x := v)$ means $f \oplus \{(x,v)\}$

Oberve:

$$(f(x:=v))(y) = \begin{cases} v, & \text{if } y = x \\ f(y), & \text{if } y \neq x \end{cases}$$

A Simple Property

remember:

$S \bullet r = \{ y \mid \exists x \in S.\ (x,y) \in r \}$

$t \circ r = \{ (x,z) \mid \exists y.\ (x,y) \in t \wedge (y,z) \in r \}$

$\Delta_A = \{ (x,x) \mid x \in A \}$

Theorem: $S \bullet r = \mathrm{ran}(\Delta_S \circ r)$ for $r \subseteq A \times A$, $S \subseteq A$

$e \in S \bullet r$

$\exists x \in S \quad (x,e) \in r$

$e \in \mathrm{ran}(\Delta_S \circ r)$

$\mathrm{ran}(\{ (u,u) \mid u \in S \} \circ r)$

$e \in \mathrm{ran} \{ (p,q) \mid \exists w.\ (p,w) \in \Delta_S$
$(w,q) \in r \}$

$\exists p.\ \boxed{\exists w.\ (p,w) \in \Delta_S \quad (w,e) \in r}$ $\quad p \in S \wedge (p,e) \in r$

Transitive Closure

$r \subseteq A^2$

$r^0 = \Delta_A$ $\qquad$ $r^1 = r \circ \Delta_A = r$

$r^{n+1} = r \circ r^n = r^n \circ r$

$r^* = \bigcup_{i \geq 0} r^i = \Delta_A \cup \underbrace{r \cup r^2 \cup \ldots}$

Theorem: $\bigcap \{ s \mid \Delta_A \cup s \circ r \subseteq s \} = r^*$

(r^* is the least s satisfying the recursive condition)

Proof:

$H = \{ s \mid \Delta_A \cup s \circ r \subseteq s \}$ $\qquad$ should prove: $\bigcap H = r^*$

(1) $r^* \in H$

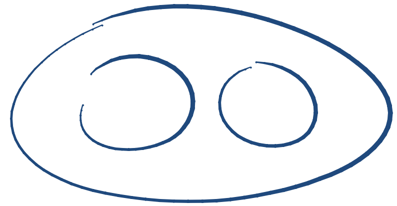
goal: $\Delta_A \cup \underbrace{r^* \circ r} \subseteq r^*$

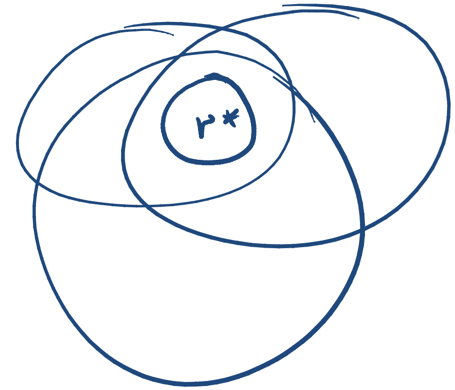
$\Delta_A \cup \left(\bigcup_{i \geq 0} r^i \right) \circ r = \Delta_A \cup \bigcup_{i \geq 0} r^{i+1} = r^*$

$r^* \in H$

so: $\bigcap H = \ldots \cap r^* \cap \ldots \subseteq r^*$

proof





(2) $s \in H \to r^* \subseteq s$

$$\boxed{\Delta_A \cup s \circ r \subseteq s}$$

$\Delta_A \subseteq s \quad / \circ r$

$r = \Delta_A \circ r \subseteq s \circ r \subseteq s$

$r \subseteq s \quad / \cdot r$

$r \circ r \subseteq s \circ r \subseteq s$

$r^2 \subseteq s$

$\vdots$

$r^n \subseteq s \quad / \circ r$

$r^n \circ r \subseteq s \circ r \subseteq s$

$\forall n \geq 0. \; r^n \subseteq s$ $\qquad r^* = \bigcup_{n \geq 0} r^n \subseteq s$

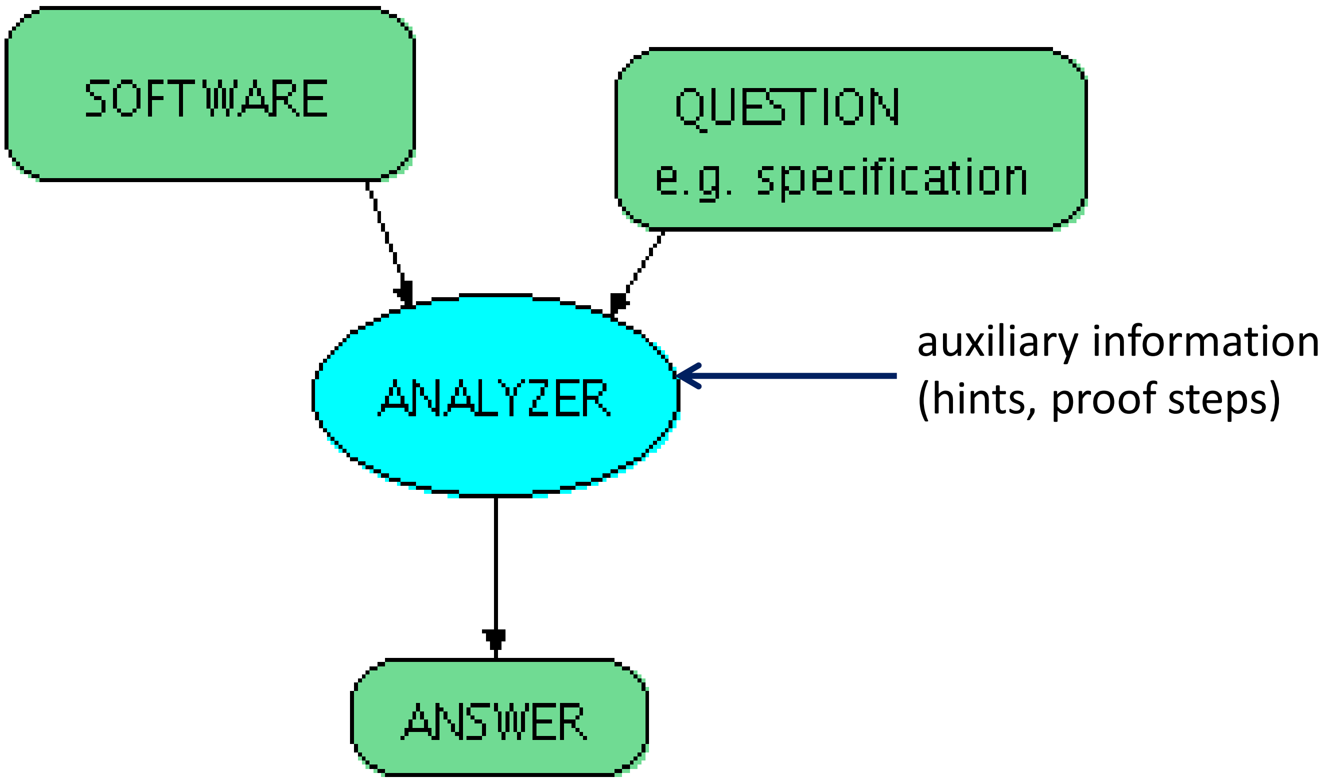
(1) $\bigcap H \subseteq r^*$

(2) $r^* \subseteq \bigcap H$

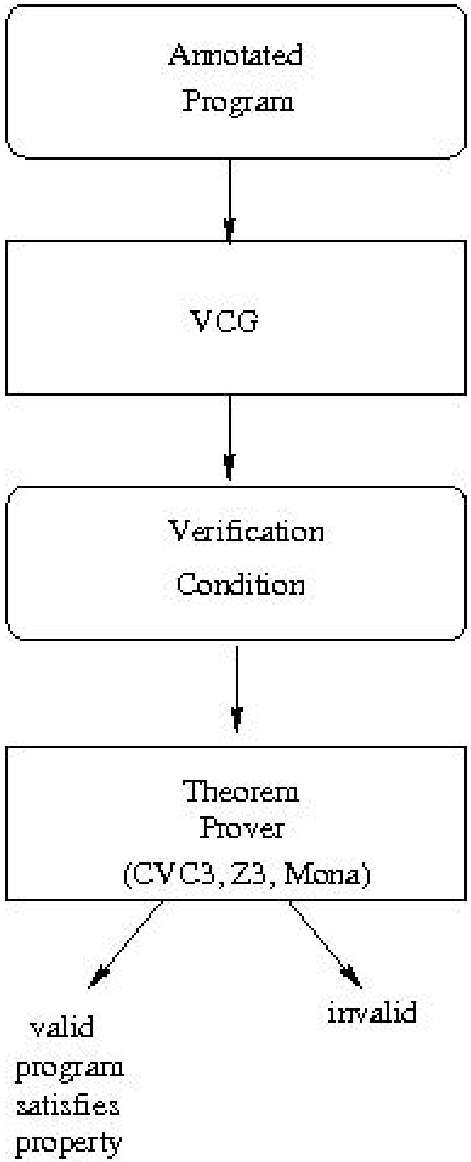
$$\frac{}{\bigcap H = r^*}$$

so, $\bigcap H = \dots \cap s_1 \cap s_2 \cap \dots \supseteq \dots \cap r^* \cap r^* \cap \dots$

Analysis and Verification



Verification-Condition Generation



Steps in Verification

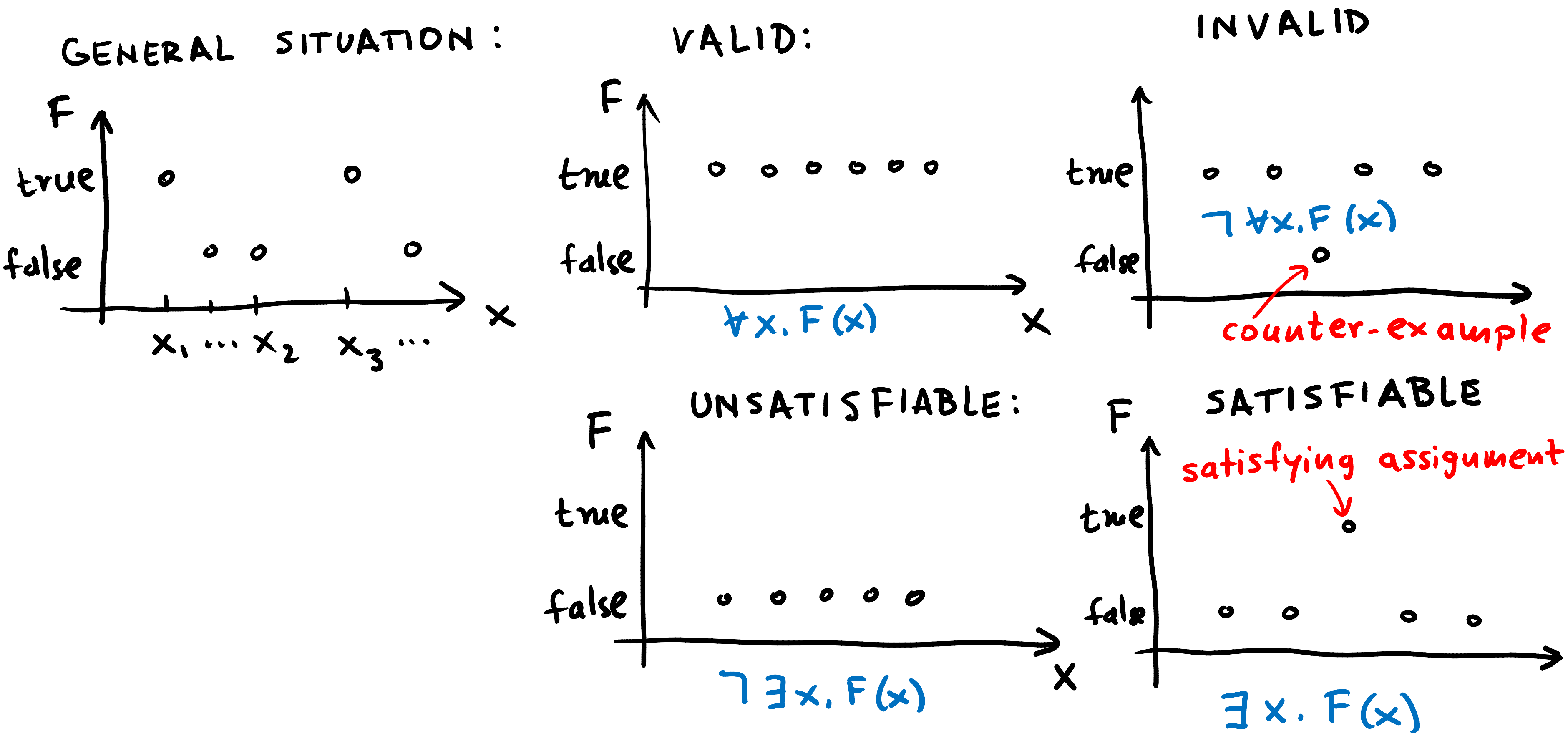
- generate formulas implying program correctness
- attempt to prove formulas
  - if formula is **valid**, program is correct
  - if formula has a **counterexample**, it indicates one of these:
    - error in the program
    - error in the property
    - error in auxiliary statements (e.g. loop invariants)

Terminology

- generated formulas:
  verification conditions
- generation process:
  verification-condition generation
- program that generates formulas:
  verification-condition generator (VCG)

Validity and Satisfiability

$F(x)$ - formula with free variable(s) x



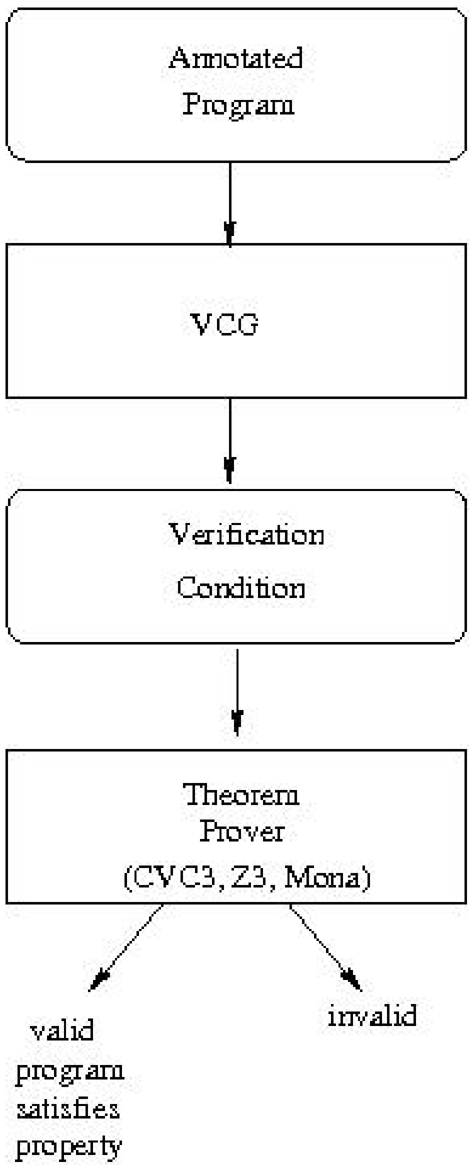
F is valid ⇔ ¬F is unsatisfiable

F is invalid ⇔ ¬F is satisfiable

F is invalid ⇔ not the case that F is valid

F is unsatisfiable ⇔ not the case that F is satisfiable

Verification-Condition Generation



Steps in Verification

- generate formulas implying program correctness
- attempt to prove formulas
  - if formula is valid, program is correct
  - if formula has a counterexample, it indicates one of these:
    - error in the program
    - error in the property
    - error in auxiliary statements (e.g. loop invariants)

Terminology

- generated formulas:
  verification conditions
- generation process:
  verification-condition generation
- program that generates formulas:
  verification-condition generator (VCG)

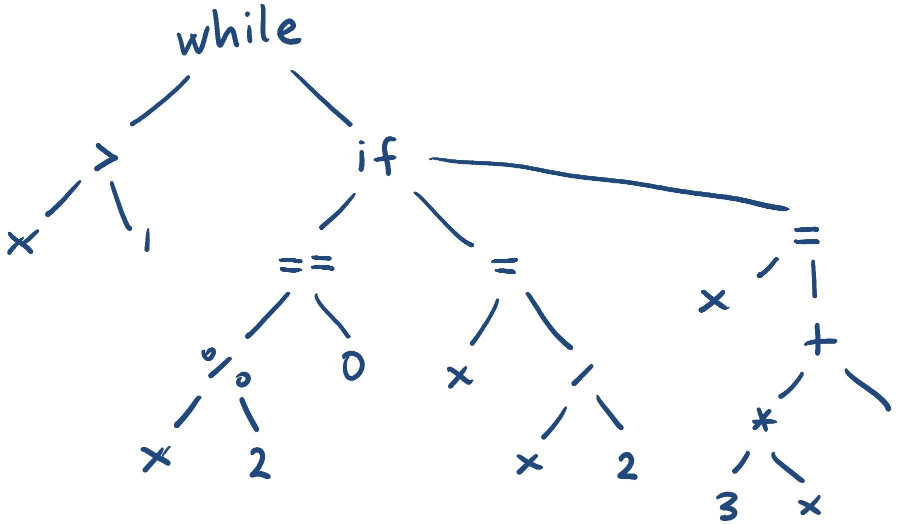
Simple Programming Language

x = T
if (F) c1 else c2
c1 ; c2
while (F) c1

c ::= x=T | (if (F) c else c) | c ; c | (while (F) c)
T ::= K | V | (T + T) | (T - T) | (K * T) | (T / K) | (T % K)
F ::= (T==T) | (T < T) | (T > T) | (~F) | (F && F) | (F || F)
V ::= x | y | z | ...
K ::= 0 | 1 | 2 | ...

Simple Program and its Syntax Tree

```
while (x > 1) {
  if (x % 2 = 0)
    x = x / 2
  else
    x = 3 * x + 1
}
```



Remark: Turing-Completeness

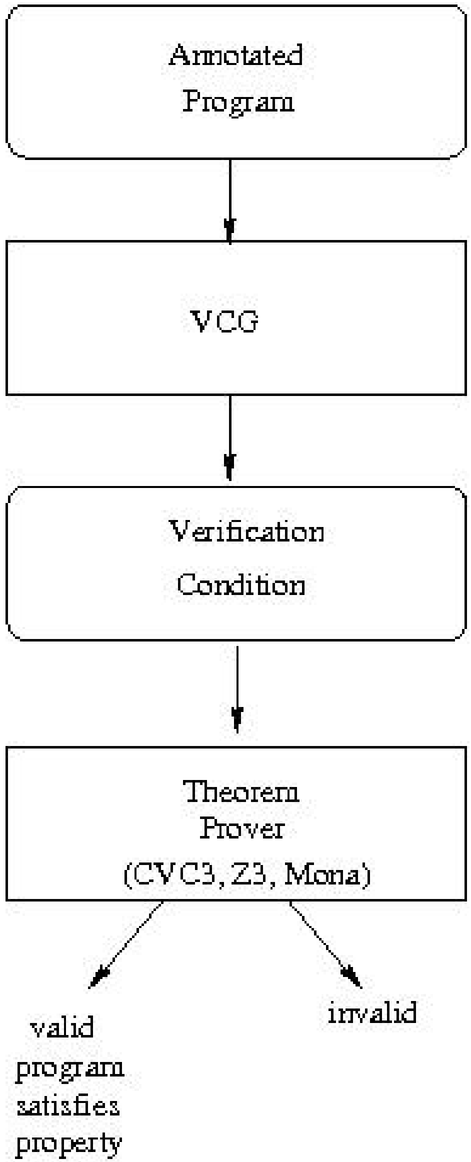
This language is Turing-complete

- it subsumes counter machines, which are known to be Turing-complete
- every possible program (Turing machine) can be encoded into computation on integers (computed integers can become very large)
- the problem of taking a program and checking whether it terminates is undecidable
- Rice's theorem: all properties of programs that are expressed in terms of the results that the programs compute
(and not in terms of the structure of programs) are undecidable

In real programming languages we have bounded integers, but we have other sources of unboundedness, e.g.

- bignums
- example: sizes of linked lists and other containers
- program syntax trees for an interpreter or compiler
(would like to handle programs of any size!)

Relational Semantics

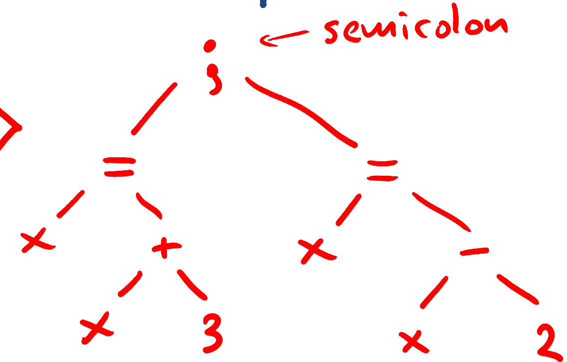


relation
(infinite mathematical object)

given by set comprehension
(formula, with finite syntax tree)

$x = x+3;$
$x = x-2$

$\Rightarrow$



$\Downarrow$

$\{(x,x') \mid x' = x+1\} \Leftarrow \{(0,1), (1,2), (2,3), (3,4), \dots\}$

Examples

`x = x+3;` `x = x+2`	$\{(x,x') \mid x' = x+5\}$
`x = x+x`	$\{(x,x') \mid x' = 2x\}$
`while (x != 10) x = x+1 }`	$\{(x,x') \mid x \leq 10 \wedge x' = 10\}$
`while (5==5){ x=x }`	$\emptyset$

Relation between initial and all possible final states.

Why Relations

The meaning is, in general, an arbitrary *relation*. Therefore:

• For certain states there will be no results.
In particular, if a computation starting at a state does not terminate

• For certain states there will be multiple results.
This means command execution starting in state will sometimes compute one and sometimes other result.
Verification of such program must account for both possibilities.

• Multiple results are important for modeling e.g. concurrency, as well as approximating behavior that we do not know
(e.g. what the operating system or environment will do,
or what the result of complex computation is)

Guarded Command Language

assume(F) - stop execution if F does not hold
pretend execution never happened

s1 [] s2 - do either s1 or s2

s* - execute s zero, once, or more times

Guarded Commands and Relations - Idea

$x = T$	$\{(x,T) \mid true\}$ gets more complex for more variables
assume(F)	Δ_S S is set of values for which F is true (satisfying assignments of F)
s^*	r^*
s_1 [] s_2	$r_1 \cup r_2$

Assignment for More Variables

var x,y

…

y = x + 1

$\{ ((x,y),(x',y')) \mid y' = x+1 \wedge x' = x \}$

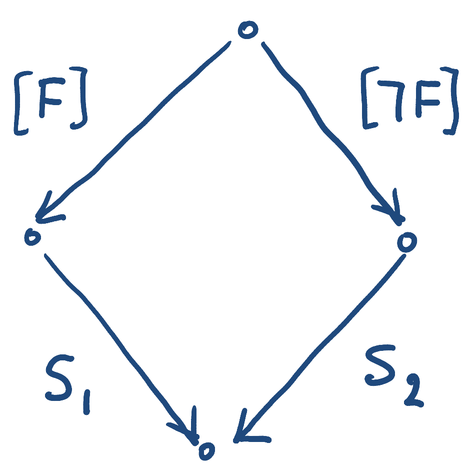
frame condition

‘if’ condition using assume and []

if (F)
  s1
else
  s2

(assume(F); s1)
  []
(assume($\neg$F); s2)

CFG:



$$(\Delta_{\text{"}F\text{"}} \circ S_1) \cup (\Delta_{\text{"}\neg F\text{"}} \circ S_2)$$

Example: y is absolute value of x

```
if (x>0)
  y = x
else
  y = -x
```

(assume(x>0); y=x)

[]

(assume(¬(x>0)); y=-x)

$x \le 0$

$\Delta_{\text{"}x>0\text{"}} \circ r_{\text{"}y=x\text{"}}$

$\cup$

$\Delta_{\text{"}x\le 0\text{"}} \circ r_{\text{"}y=-x\text{"}}$

$\Delta_{\text{"}x>0\text{"}} = \{((x,y),(x',y')) \mid x>0 \wedge x'=x \wedge y'=y\}$

$\Delta_{\text{"}x\le 0\text{"}} = \{((x,y),(x',y')) \mid x\le 0 \wedge x'=x \wedge y'=y\}$

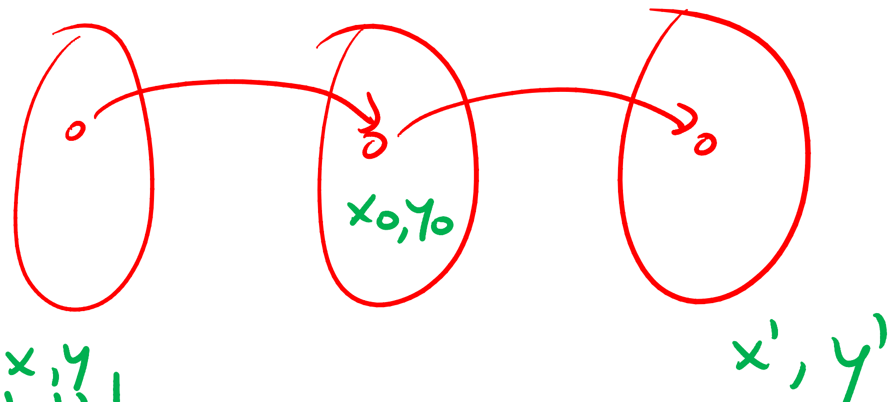
$r_{y=-x} = \{((x,y),(x',y')) \mid x'=x \wedge y'=-x\}$

(calculating absolute value)

$$x \le 0 \wedge \overset{x_0}{x'} = x \wedge \overset{y_0}{y'} = y$$

$$\circ$$

$$x' = \underset{x_0}{x} \wedge y' = -\underset{x_0}{x}$$



$$\{((x,y),(x',y')) \mid \exists x_0, y_0.\ x \le 0 \wedge x_0 = x \wedge y_0 = y \wedge x' = x_0 \wedge y' = -x_0\} = \Delta_{\text{"}x \le 0\text{"}} \circ r_{y=-x}$$

$$= \{((x,y),(x',y')) \mid x \le 0 \wedge x' = x \wedge y' = -x\}$$

$$\cup\ \{(\ldots) \mid x > 0 \wedge x' = x \wedge y' = x\} \qquad \Delta_{\text{"}x > 0\text{"}} \circ r_{y=x}$$

guards

$\exists x.\ x = t \wedge H(x)$

$\Updownarrow$

$H(t)$

$F \rightarrow c$

$\mathrm{assume}(F);\ c$

$\downarrow$

Lemma:

$\{(\ \ )\mid F \wedge x' = x \wedge y' = y\} \circ r = \{(\ \ )\mid N\}$

$\exists x_0, y_0.\ F \wedge x_0 = x \wedge y_0 = y \wedge N[x := x_0, y := y_0]$

$[\![\mathrm{assume}(F); c]\!] = \{((x,y),(x',y')) \mid F \wedge N\}$

$[\![c]\!] = \{((x,y),(x',y')) \mid N\}$

$\emptyset = \{((x,y),(x',y')) \mid \mathrm{false}\}$

$(F \wedge N) \vee (\neg F \wedge \mathrm{false}) \equiv F \wedge N$

‘while’ using assume and *

while (F)
  s

(assume(F); s)*
    []
assume($\neg$F)

CFG: