This Week

- Finish relational semantics
- Hoare logic
- Interlude on one-point rule
- Building formulas from programs

Synthesis, Analysis, and Verification

Lecture 03a

Relational Semantics and Consequences

Hoare Logic

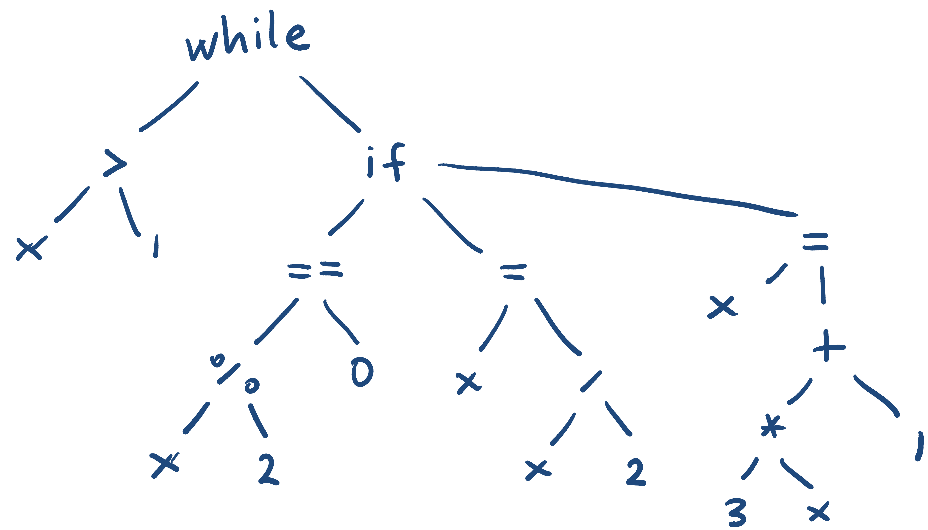
Lectures:

Viktor Kuncak

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Mapping Programs

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



into relations

$$\{((x_{1,\dots,}x_n), (x'_{1,\dots,}x'_n)) \mid F(x_{1,\dots,}x_n\; x'_{1,\dots,}x'_n)\}$$

Guarded Command Language

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

s1 [] s2 - non-deterministic execution of
both s1 and s2

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

Guarded Commands and Relations - Idea

x = T	{(x,T) | 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

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

frame condition

State as a Map from Variables to Values

V - variables

$s : V \to \mathbb{Z}$ states

statement

$x=3;$

becomes

$\{ (s, s') \mid s' = s(\text{"x"} := 3) \} \subseteq (V \to \mathbb{Z})^2$

and $x = t$

becomes

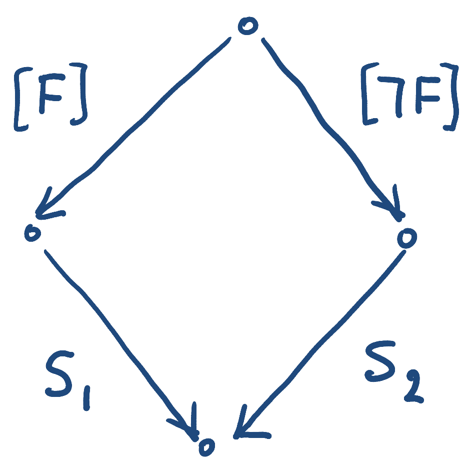
$\{ (s, s') \mid s' = s(\text{"x"} := \underbrace{[\![t]\!]_s}_{\text{meaning of } t \text{ is state } s}) \}$

‘if’ condition using assume and []

if (F)
  s1
else
  s2

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

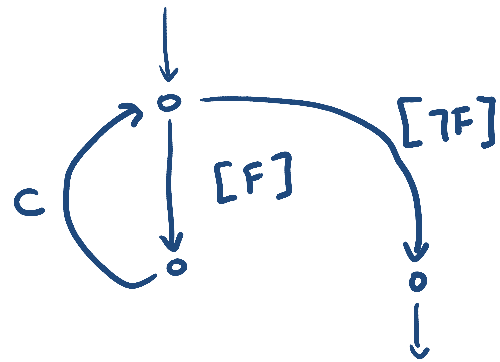
CFG:



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

‘while’ using assume and *

while (F)
  c



(assume(F); c)*;
assume(¬F)

$$[\![\text{while } (F)\ c]\!] = (\Delta_{[\![F]\!]} \circ [\![c]\!])^* \circ \Delta_{[\![\neg F]\!]}$$

Compute Relation for this Program

```
r = 0;
while (x > 0) {
  r = r + 3;
  x = x - 1
}
```

...assume (...) ...

$r = \{((r,x),(r',x')) \mid \ldots\}$?

As the program state use the pair of integer variables (r,x)

1) compute guarded command language for this program (express 'while' and 'if' using 'assume').

Compute Relation for this Program

```
r = 0;
while (x > 0) {
  r = r + 3;
  x = x - 1
}
```

$\longrightarrow$

```
r = 0;
( assume(x>0);
  r = r + 3;
  x = x – 1 )* ;
assume(x <= 0)
```

⟦ assume(x>0); r = r + 3; x = x – 1 ⟧ $= [\![B]\!] = \{ \ldots \mid x' = x-1 \land r' = r+3 \land x > 0 \}$

⟦ B* ⟧ $= [\![B]\!]^* = \bigcup_{k \geq 0} [\![B]\!]^k = \{ \ldots \mid r' = r + 3(x - x') \land x' \geq 0 \land x - x' \geq 0 \} \cup \ldots$

⟦ r = 0; B*; assume(x <= 0) ⟧ $= \{ \ldots \mid r' = 3(x - x') \land x' \geq 0 \land x - x' \geq 0 \land x' \leq 0 \} \cup \ldots$

$= \{ \ldots \mid r' = 3x \land x' = 0 \land x \geq 0 \} \cup \{ x < 0 \land x' = x \land r' = 0 \}$

2) compute meaning of program pieces, from smaller to bigger

B

$r' = r + 3$ &&

$x' = x - k$ &&

$x > 0$

B^k

$r' = r + 3k$ &&

$x' = x - k$ &&

$x > 0$ && $x - 1 > 0$ && … && $x - (k-1) > 0$

i.e.

$r' = r + 3k$ &&

$x' = x - k$ &&

$x - (k - 1) > 0$ i.e. $x - k >= 0$

B*

(s,s') in B* ⇔ exists k. (s,s') in B^k

B* :

exists k. k >= 0 &&

  r' = r + 3k &&

  x' = x – k &&

  x – k >= 0

i.e. by one-point rule:

  r' = r + 3(x-x') && x' >= 0 && x – x' >= 0

and we must also add the diagonal

Havoc Statement

Havoc Statement

- Havoc statement is another useful declarative statement. It changes a given variable entirely arbitrarily: there will be one possible state for each possible value of integer variable.

havoc (x)

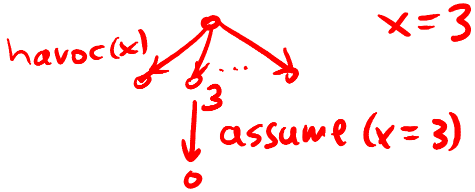
$\{((r,x),(r',x')) \mid r' = r\}$

$\{(s,s') \mid \exists v.\ s' = s(\text{"x"} := v)\}$

Expressing Assignment with Havoc+Assume

- We can prove that the following equality holds under certain conditions:

x = 3



x = E   is   havoc(x); assume(x==E)

In other words, assigning a variable is the same as changing it arbitrarily and then assuming that it has the right value.

Under what condition does this equality hold?

Correctness as Relation Inclusion

program $\rightarrow$ relation p
specification $\rightarrow$ relation s

program meets specification:

$$p \subseteq s$$

example: $p = \{((r,x),(r',x')).\ r'=2x \ \&\&\ x'=0\ \}$
$s = \{((r,x),(r',x')).\ x > 0 \rightarrow r' > x'\ \}$

then the above program p meets the specification s

because implication holds:

$$r'=2x \ \&\&\ x'=0 \quad \rightarrow (\ x > 0 \rightarrow r' > x'\ )$$

Normal form for Loop-Free Programs

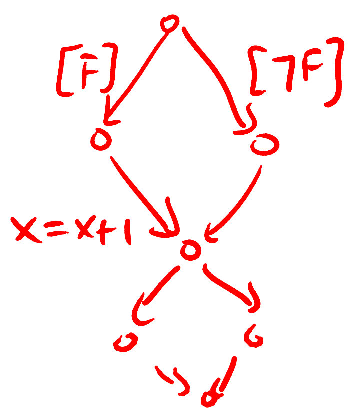
Lemma: Let P be a program without loops. Then for some natural number n,

$$[\![P]\!] = \bigcup_{i=1}^{n} p_i$$

where each p_i is relation composition of

- relations for assignments
- diagonal relations $\Delta_{\text{"F"}}$

Prove this.



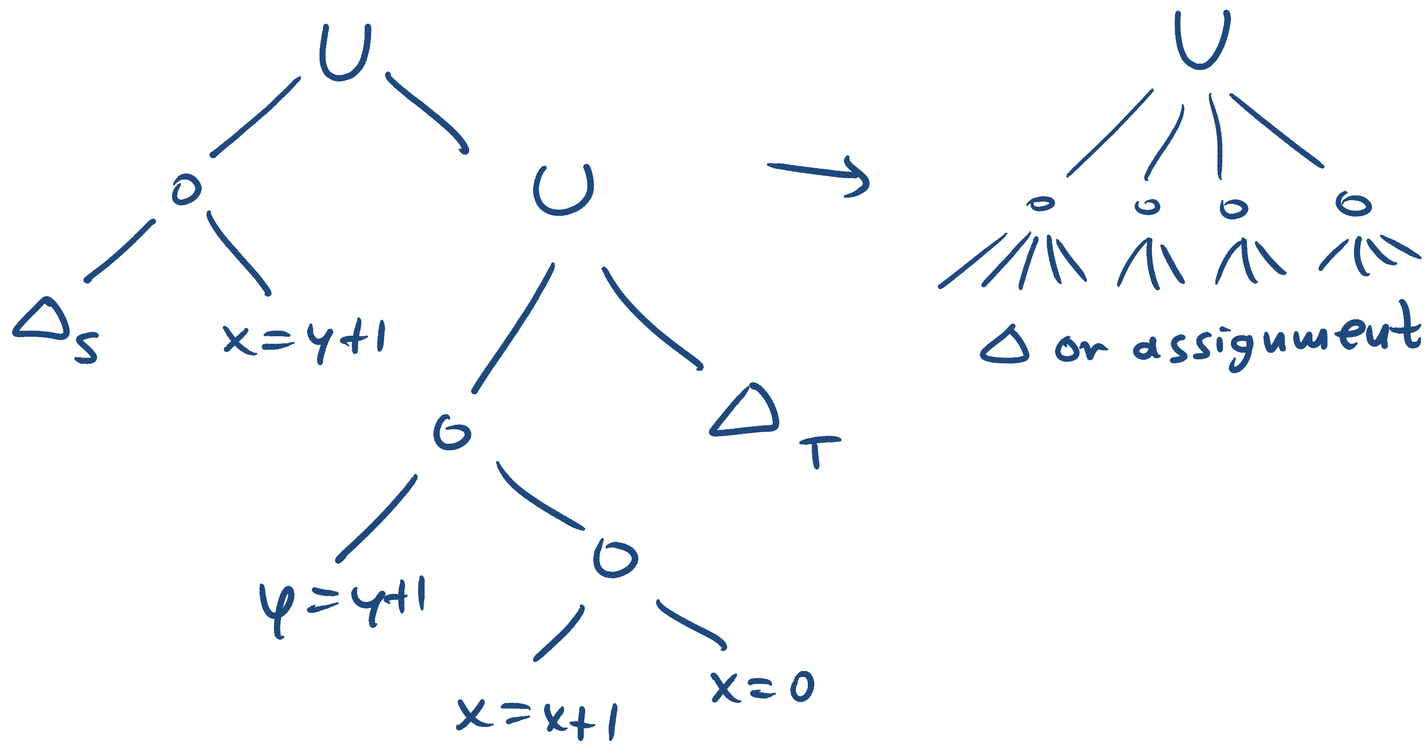
$$\Delta_S \circ [\![x=x+1]\!] \circ \Delta_T \circ [\![x=\;]\!] \circ [\![y=\dots]\!]$$

Δ_S, $[\![x=E]\!]$, $\circ$, $\cup$

move $\cup$ to top level

Proof

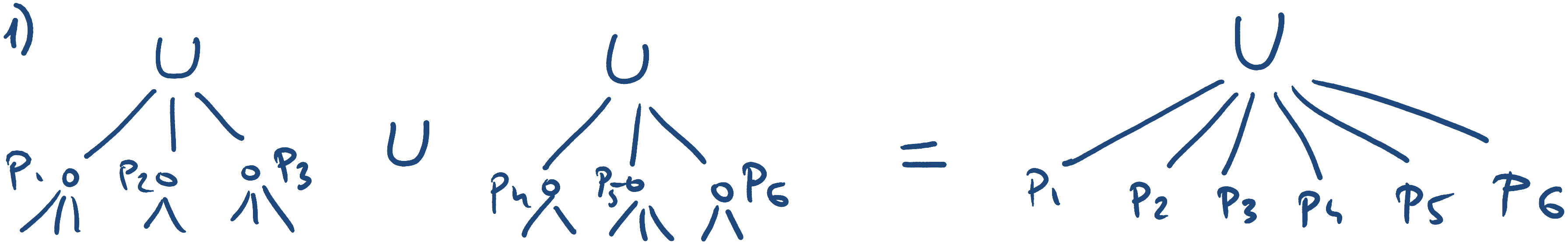
structural induction on program tree



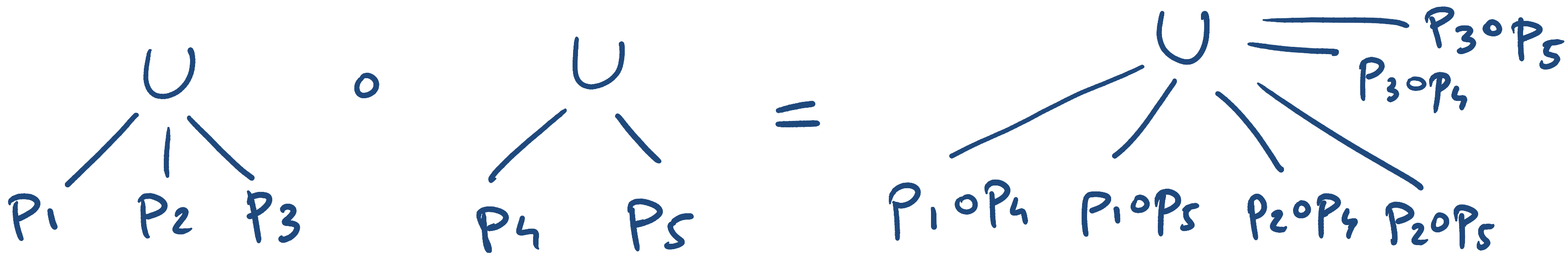
base case: trivial

Inductive Case

1)



2)



$$(r \overset{\cap ?}{\cup} s) \circ t = r \circ t \ \overset{\cap ?}{\cup} \ s \circ t$$

$$r \circ (p \cup q) = r \circ p \ \cup \ r \circ q$$

A Hoare Logic Proof

```
//{0 <= y}
i = y;
//{0 <= y & i = y}
r = 0;
//{0 <= y & i = y & r = 0}
while //{r = (y-i)*x & 0 <= i}
 (i > 0) (
  //{r = (y-i)*x & 0 < i}
  r = r + x;
  //{r = (y-i+1)*x & 0 < i}
  i = i - 1
  //{r = (y-i)*x & 0 <= i}
)
//{r = x * y}
```

$\{r = (y-i)*x \wedge 0 < i\}$

$r = r + x$

$\{r = (y-i+1)*x \wedge 0 < i\}$

Hoare Logic

- see wiki