

COURSE STRUCTURE

CONSTRAINT SOLVERS

- Propositional Logic, SAT solving, DPLL
- First-Order Logic, SMT
- First-Order Theories

DEDUCTIVE VERIFICATION

- Operational Semantics
- Strongest Post-condition, Weakest Pre-condition
- Hoare Logic

MODEL CHECKING AND OTHER VERIFICATION TECHNIQUES

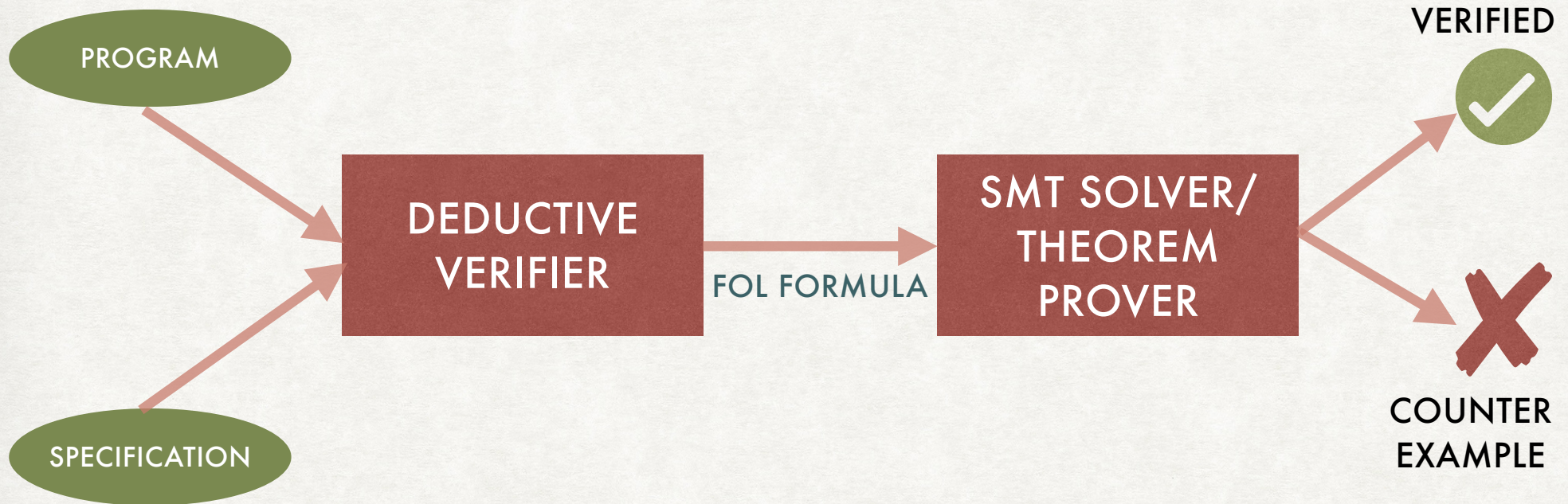
- Predicate Abstraction, CEGAR
- Abstract Interpretation
- Property-directed Reachability

FORMAL SPECIFICATION AND VERIFICATION OF PROGRAMS

INTRODUCTION

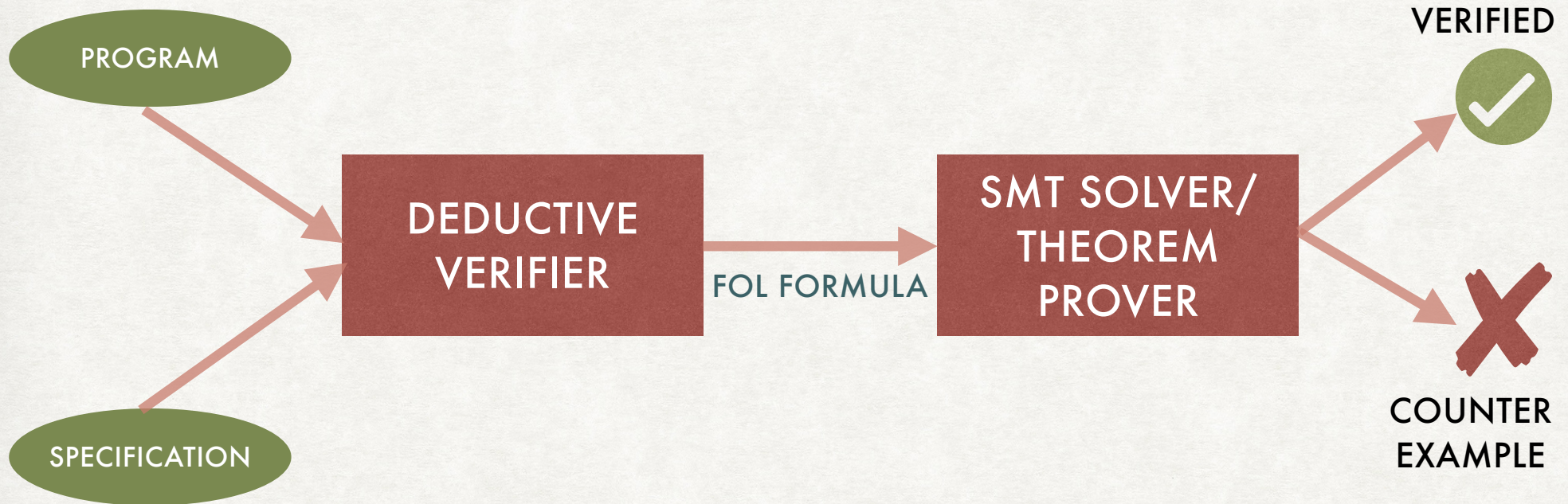
- So far we have seen...
 - Syntax, Semantics for Propositional Logic and First-Order Logic and (some examples of) Decision Procedures for Validity/Satisfiability
 - Underlying engine for **Deductive Verification** of programs
- Now we will how to reduce the program verification problem to the satisfiability problem in first-order logic.

AUTOMATED VERIFICATION OVERVIEW



AUTOMATED VERIFICATION

OVERVIEW



- Assertions
- Pre-conditions/Post-conditions
- Invariants
- ...

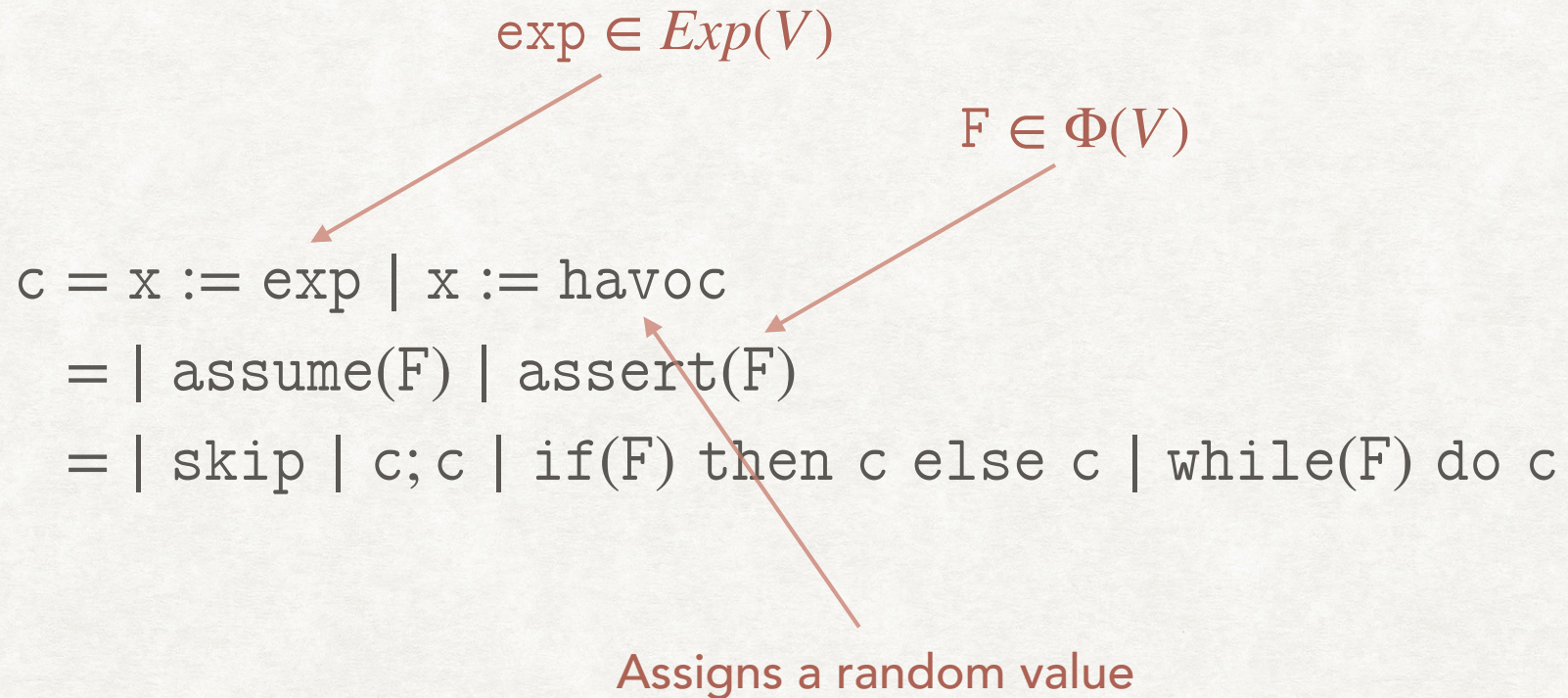
IMP

A SMALL IMPERATIVE PROGRAMMING LANGUAGE

- Let V be a set of program variables
- Let $Exp(V)$ be the set of linear expressions, and $\Phi(V)$ be the set of linear formulae over V
 - $Exp(V)$ are terms in LRA or LIA
 - $\Phi(V)$ are formulae in LRA or LIA
- Examples
 - $3x + 2y \in Exp(\{x, y\})$
 - $x \leq y + z \wedge z = w \in \Sigma(\{x, y, z, w\})$

IMP

A SMALL IMPERATIVE PROGRAMMING LANGUAGE



EXAMPLES

PRE-CONDITION

`assume(i = 0 ∧ n ≥ 0);`

`while(i < n) do`

`i := i + 1;`

`assert(i = n);`

POST-CONDITION

EXAMPLES

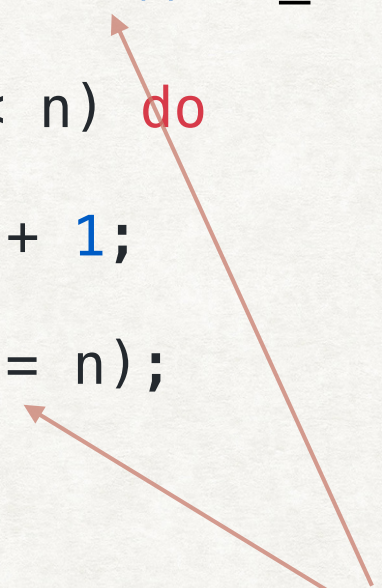
```
assume(i = 0 ∧ n ≥ 0);
```

```
while(i < n) do
```

```
    i := i + 1;
```

```
assert(i = n);
```

FOL formula in LIA whose
free variables are program variables



EXAMPLES

$\{i = 0 \wedge n \geq 0\}$

while($i < n$) **do**

$i := i + 1;$

$\{i = n\}$

EXAMPLES

$\{i = 0 \wedge n \geq 0\}$

while($i < n$) **do**

$i := i + 1;$

$\{i = n\}$

{Pre-condition}

Program

{Post-condition}

EXAMPLES

Linear Search

Input: Array a , Lower limit l , Upper limit u , Element to be searched e

Output: true if element is present, false otherwise

```
i := l;  
present := false;  
while(i <= u && !present)  
{  
    if (a[i] == e) then  
        present := true;  
    else  
        i := i + 1;  
}
```


EXAMPLES

Linear Search

Input: Array a , Lower limit l , Upper limit u , Element to be searched e

Output: true if element is present, false otherwise

```
assume(?);  
i := l;  
present := false;  
while(i <= u && !present)  
{  
    if (a[i] == e) then  
        present := true;  
    else  
        i := i + 1;  
}  
assert(?);
```


EXAMPLES

Linear Search

Input: Array a , Lower limit l , Upper limit u , Element to be searched e

Output: true if element is present, false otherwise

```
assume( $l \geq 0 \wedge u \leq |a|$ );  
 $i := l$ ;  
present := false;  
while( $i \leq u \ \&\& \ !\text{present}$ )  
{  
    if ( $a[i] == e$ ) then  
        present := true;  
    else  
         $i := i + 1$ ;  
}  
assert(?);
```


EXAMPLES

Linear Search

Input: Array a , Lower limit l , Upper limit u , Element to be searched e

Output: true if element is present, false otherwise

```
assume( $l \geq 0 \wedge u \leq |a|$ );  
 $i := l$ ;  
present := false;  
while( $i \leq u \ \&\& \ !\text{present}$ )  
{  
    if ( $a[i] == e$ ) then  
        present := true;  
    else  
         $i := i + 1$ ;  
}  
assert( $\text{present} \leftrightarrow l \leq i \leq u \wedge a[i] = e$ );
```


EXAMPLES

Linear Search

Input: Array a , Lower limit l , Upper limit u , Element to be searched e

Output: true if element is present, false otherwise

```
assume( $l \geq 0 \wedge u \leq |a|$ );  
 $i := l$ ;  
present := false;  
while( $i \leq u \ \&\& \ !\text{present}$ )  
{  
    if ( $a[i] == e$ ) then  
        present := true;  
    else  
         $i := i + 1$ ;  
}  
assert( $\text{present} \leftrightarrow \exists x. l \leq x \leq u \wedge a[x] = e$ );
```

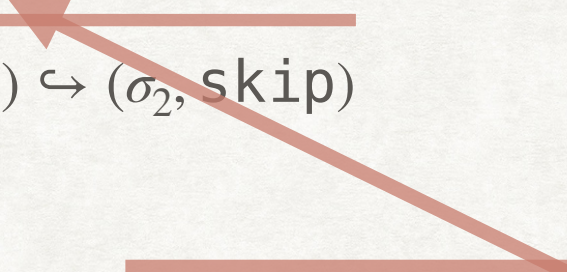

OPERATIONAL SEMANTICS OF IMP

- In order to formally define the verification problem, i.e. 'the program satisfies its specification', we will first define **Operational Semantics** of Imp.
- The operational semantics formally define how the program state evolves during execution.
- A program state (σ, c) consists of two components:
 - $\sigma : V \rightarrow \mathbb{R}$ is a valuation of program variables
 - c is the rest of the program to be executed
- Let $\Sigma = (\mathbb{R}^{|V|} \cup \{Error\}) \times \mathcal{P}$ be the set of all states
 - \mathcal{P} is the set of all Imp programs.
- A transition $(\sigma_1, c_1) \hookrightarrow (\sigma_2, c_2)$ denotes a step taken by the program

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})}$$

TRANSITIONS OF IMP

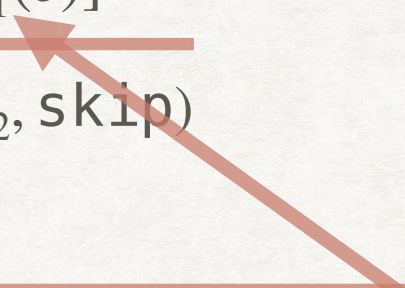
$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})}$$


NOTATION ALERT:

$f = g[a \mapsto b]$ means:

- $f(a) = b$
- $\forall x \in \text{dom}(g) . x \neq a \rightarrow f(x) = g(x)$

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})}$$


NOTATION ALERT:

For $e \in \text{Exp}(V)$ and $\sigma \in \mathbb{R}^{|V|}$, $\sigma(e)$ denotes the evaluation of e at σ using the standard interpretations of Arithmetic operators.

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-ASSIGN]}$$

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-ASSIGN]}$$

$$\frac{\sigma_2 = \sigma_1[x \mapsto n] \quad n \in \mathbb{R}}{(\sigma_1, x := \text{havoc}) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-HAVOC]}$$

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-ASSIGN]}$$

$$\frac{\sigma_2 = \sigma_1[x \mapsto n] \quad n \in \mathbb{R}}{(\sigma_1, x := \text{havoc}) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-HAVOC]}$$

$$\frac{???}{(\sigma_1, \text{assume}(F)) \hookrightarrow (\sigma_1, \text{skip})} \quad \text{[T-ASSUME]}$$

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-ASSIGN]}$$

$$\frac{\sigma_2 = \sigma_1[x \mapsto n] \quad n \in \mathbb{R}}{(\sigma_1, x := \text{havoc}) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-HAVOC]}$$

$$\frac{\sigma_1 \models F}{(\sigma_1, \text{assume}(F)) \hookrightarrow (\sigma_1, \text{skip})} \quad \text{[T-ASSUME]}$$

TRANSITIONS OF IMP

$$\frac{\sigma_2 = \sigma_1[x \mapsto \sigma_1(e)]}{(\sigma_1, x := e) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-ASSIGN]}$$

$$\frac{\sigma_2 = \sigma_1[x \mapsto n] \quad n \in \mathbb{R}}{(\sigma_1, x := \text{havoc}) \hookrightarrow (\sigma_2, \text{skip})} \quad \text{[T-HAVOC]}$$

$$\frac{\sigma_1 \models F}{(\sigma_1, \text{assume}(F)) \hookrightarrow (\sigma_1, \text{skip})} \quad \text{[T-ASSUME]}$$

$$\frac{\sigma_1 \models F}{(\sigma_1, \text{assert}(F)) \hookrightarrow (\sigma_1, \text{skip})} \quad \text{[T-ASSERT-TRUE]}$$

$$\frac{\sigma_1 \not\models F}{(\sigma_1, \text{assert}(F)) \hookrightarrow (\text{Error}, \text{skip})} \quad \text{[T-ASSERT-FALSE]}$$

TRANSITIONS OF IMP

[T-SEQ-1]

$$(\sigma_1, c_1) \hookrightarrow (\sigma_2, c'_1)$$

$$(\sigma_1, c_1; c_2) \hookrightarrow (\sigma_2, c'_1; c_2)$$

[T-SEQ-2]

$$(\sigma_1, \text{skip}; c_2) \hookrightarrow (\sigma_1, c_2)$$

TRANSITIONS OF IMP

[T-SEQ-1]

$$(\sigma_1, c_1) \hookrightarrow (\sigma_2, c'_1)$$

$$(\sigma_1, c_1; c_2) \hookrightarrow (\sigma_2, c'_1; c_2)$$

[T-SEQ-2]

$$(\sigma_1, \text{skip}; c_2) \hookrightarrow (\sigma_1, c_2)$$

[T-IF-TRUE]

$$\sigma_1 \models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_1)$$

[T-IF-FALSE]

$$\sigma_1 \not\models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_2)$$

TRANSITIONS OF IMP

[T-SEQ-1]

$$(\sigma_1, C_1) \hookrightarrow (\sigma_2, C'_1)$$

$$(\sigma_1, C_1; C_2) \hookrightarrow (\sigma_2, C'_1; C_2)$$

[T-SEQ-2]

$$(\sigma_1, \text{skip}; C_2) \hookrightarrow (\sigma_1, C_2)$$

[T-IF-TRUE]

$$\sigma_1 \models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_1)$$

[T-IF-FALSE]

$$\sigma_1 \not\models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_2)$$

$$\sigma_1 \models F$$

$$(\sigma_1, \text{while}(F) \text{ do } c) \hookrightarrow (\sigma_1, c; \text{while}(F) \text{ do } c)$$

[T-WHILE-TRUE]

TRANSITIONS OF IMP

[T-SEQ-1]

$$(\sigma_1, C_1) \hookrightarrow (\sigma_2, C'_1)$$

$$(\sigma_1, C_1; C_2) \hookrightarrow (\sigma_2, C'_1; C_2)$$

[T-SEQ-2]

$$(\sigma_1, \text{skip}; C_2) \hookrightarrow (\sigma_1, C_2)$$

[T-IF-TRUE]

$$\sigma_1 \models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_1)$$

[T-IF-FALSE]

$$\sigma_1 \not\models F$$

$$(\sigma_1, \text{if}(F) \text{ then } c_1 \text{ else } c_2) \hookrightarrow (\sigma_1, c_2)$$

$$\sigma_1 \models F$$

$$(\sigma_1, \text{while}(F) \text{ do } c) \hookrightarrow (\sigma_1, c; \text{while}(F) \text{ do } c)$$

[T-WHILE-TRUE]

$$\sigma_1 \not\models F$$

$$(\sigma_1, \text{while}(F) \text{ do } c) \hookrightarrow (\sigma_1, \text{skip})$$

[T-WHILE-FALSE]

EXAMPLE

```
assume(i = 0 ∧ n ≥ 0);  
while(i < n) do  
    i := i + 1;  
assert(i = n);
```

$(\{i \mapsto 0, n \mapsto 2\}, \text{assume}(i=0 \wedge n \geq 0); \dots)$

$\hookrightarrow (\{i \mapsto 0, n \mapsto 2\}, \text{skip}; \dots)$

[T-SEQ-1, T-ASSUME]

$\hookrightarrow (\{i \mapsto 0, n \mapsto 2\}, \text{while}(i < n) \text{ do } i := i + 1; \dots)$

[T-SEQ-2]

$\hookrightarrow (\{i \mapsto 0, n \mapsto 2\}, i := i + 1; \text{while}(i < n) \text{ do } i := i + 1; \dots)$

[T-WHILE-TRUE]

$\hookrightarrow (\{i \mapsto 1, n \mapsto 2\}, \text{while}(i < n) \text{ do } i := i + 1; \dots)$

[T-SEQ-1, T-ASSIGN, T-SEQ-2]

$\hookrightarrow (\{i \mapsto 1, n \mapsto 2\}, i := i + 1; \text{while}(i < n) \text{ do } i := i + 1; \dots)$

[T-WHILE-TRUE]

$\hookrightarrow (\{i \mapsto 2, n \mapsto 2\}, \text{while}(i < n) \text{ do } i := i + 1; \dots)$

[T-SEQ-1, T-ASSIGN, T-SEQ-2]

$\hookrightarrow (\{i \mapsto 2, n \mapsto 2\}, \text{assert}(i=n);)$

[T-WHILE-FALSE, T-SEQ-2]

$\hookrightarrow (\{i \mapsto 2, n \mapsto 2\}, \text{skip};)$

[T-ASSERT-TRUE]

REACHABILITY AND VERIFICATION

- Let $\Delta \subseteq \Sigma \times \Sigma$ be the set of transitions (\hookrightarrow) defined in the previous slides.
 - Is Δ finite?
 - Is Δ defined for a specific program c or for any program?
 - Is Δ finite if restricted to a specific program c ?
- Given a program c , a sequence of transitions $(\sigma_0, c) \hookrightarrow (\sigma_1, c_1) \dots \hookrightarrow (\sigma_n, c_n)$ is called an **execution** of c .
 - A program state σ is called **reachable** if there exists an execution $(\sigma_0, c) \hookrightarrow \dots \hookrightarrow (\sigma, c_n)$ which ends in the state σ .
- Verification Problem: Is $(Error, c')$ reachable for some c' ?
 - Program c is called **safe** if the error state is not reachable.
 - What about the initial state?

EXAMPLE

```
assume(i = 0 ∧ n ≥ 0);
```

```
while(i < n) do
```

```
    i := i + 1;
```

```
assert(i = n);
```

- Is $(Error, c')$ reachable?

PRE/POST-CONDITIONS AND VERIFICATION

- Alternatively, we can express the Verification problem in terms of pre-conditions and post-conditions.
- A program c satisfies the specification $\{P\}c\{Q\}$ if:
 - $\forall \sigma, \sigma'. \sigma \models P \wedge (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \models Q$
- $\{P\}c\{Q\}$ is also called a 'Hoare Triple'.
- If c satisfies the specification $\{P\}c\{Q\}$, then we also say that the Hoare Triple $\{P\}c\{Q\}$ is valid.

TOTAL CORRECTNESS

- Both ways of specifying the verification problem deal with **Partial Correctness**.
 - They only consider terminating executions. Non-terminating executions trivially satisfy both definitions.
- **Total Correctness** also requires all program executions to be of finite length.
- A program c satisfies the specification $[P]c[Q]$ if every execution which begins in a state satisfying P should terminate in a state satisfying Q .

TOTAL CORRECTNESS

- A program c satisfies the specification $[P]c[Q]$ if every execution which begins in a state satisfying P should terminate in a state satisfying Q .
 - $\forall \sigma . \sigma \models P \rightarrow \exists n, \sigma' . (\sigma, c) \hookrightarrow^n (\sigma', \text{skip}) \wedge \sigma' \models Q$
- Is this correct?
 - This only says that for every state satisfying P , there is some execution which terminates in a state obeying Q .
 - However, IMP is non-deterministic (due to havoc).
 - Hence, we need to say that every execution beginning from a state satisfying P should terminate.

$$\begin{aligned} & \forall \sigma . \exists n . \sigma \models P \rightarrow \neg(\exists m, \sigma' . m > n \wedge (\sigma, c) \hookrightarrow^m (\sigma', c')) \\ & \wedge \forall \sigma, \sigma' . \sigma \models P \wedge (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \models Q \end{aligned}$$

TOTAL CORRECTNESS

```
[T]
i := havoc;
if (i = 5)
    while(i > 0) do
        i := i+1;
else
    skip;
[i ≠ 5]
```

- The above program satisfies the definition of total correctness as per $\forall \sigma. \sigma \models P \rightarrow \exists n, \sigma'. (\sigma, c) \hookrightarrow^n (\sigma', \text{skip}) \wedge \sigma' \models Q$.
- However, there are clearly non-terminating executions.
- It does not satisfy the following formula

$$\forall \sigma. \exists n. \sigma \models P \rightarrow \neg(\exists m, \sigma'. m > n \wedge (\sigma, c) \hookrightarrow^m (\sigma', c'))$$

- $\wedge \forall \sigma, \sigma'. \sigma \models P \wedge (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \models Q$

EXAMPLES OF HOARE TRIPLES

- What can be said about the following triples?
 - $\{ \top \} \text{ c } \{ Q \}$
 - $\{ \perp \} \text{ c } \{ Q \}$
 - $\{ P \} \text{ c } \{ \top \}$
 - $\{ \top \} \text{ c } \{ \perp \}$
- Partial and total correctness
 - Is $\{ x = 0 \} \text{ while}(x \geq 0) \text{ do } x := x + 1 \{ x = 1 \}$ valid?
 - What about $[x = 0] \text{ while}(x \geq 0) \text{ do } x := x + 1 [x = 1]$?

AUTOMATED VERIFICATION

- We will reduce the verification problem to the satisfiability problem (modulo theories) in FOL.
- As a first step, we encode the operational semantics in FOL.
- If V is the set of variables used in a program c , then an FOL formula $F[V]$ encodes a set of states of the program.
 - E.g. If $V = \{x, y, z\}$, then the formula $x + y > 0$ encodes the set of states $\{(x \mapsto m, y \mapsto n, z \mapsto o) \mid m + n > 0\}$

AUTOMATED VERIFICATION

If $(\sigma, c) \hookrightarrow (\sigma', \text{skip})$, then we will use the FOL formula $\rho(\mathbf{c})[\mathbf{V}, \mathbf{V}']$ to encode the states σ and σ' .

- If $(\sigma, c) \hookrightarrow (\sigma', \text{skip})$ then σ, σ' are models of $\rho(c)$.
- Example: $\rho(x:=y+1) \triangleq x' = y + 1 \wedge y' = y$
- We will use a special variable $\text{error} \in V$ to indicate the *Error* state (obtained after assertion failure). $\text{error} = 0$ indicates a non-error state.

SEMANTICS IN FOL

For $U \subseteq V$, we define $frame(U)$ to be the formula $\bigwedge_{v \in V \setminus U} v' = v$

- E.g. $V = \{x, y, z\}$, $frame(x) \triangleq (y' = y) \wedge (z' = z)$

Now, the semantics of commands in FOL can be defined as follows:

- $\rho(x := e) \triangleq$
- $\rho(x := \text{havoc}) \triangleq$
- $\rho(\text{assume}(F)) \triangleq$
- $\rho(\text{assert}(F)) \triangleq$

SEMANTICS IN FOL

For $U \subseteq V$, we define $frame(U)$ to be the formula $\bigwedge_{v \in V \setminus U} v' = v$

- E.g. $V = \{x, y, z\}$, $frame(x) \triangleq (y' = y) \wedge (z' = z)$

Now, the semantics of commands in FOL can be defined as follows:

- $\rho(x := e) \triangleq x' = e \wedge frame(x)$
- $\rho(x := \text{havoc}) \triangleq frame(x)$
- $\rho(\text{assume}(F)) \triangleq F \wedge frame(\emptyset)$
- $\rho(\text{assert}(F)) \triangleq F \rightarrow frame(\emptyset)$