# 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

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

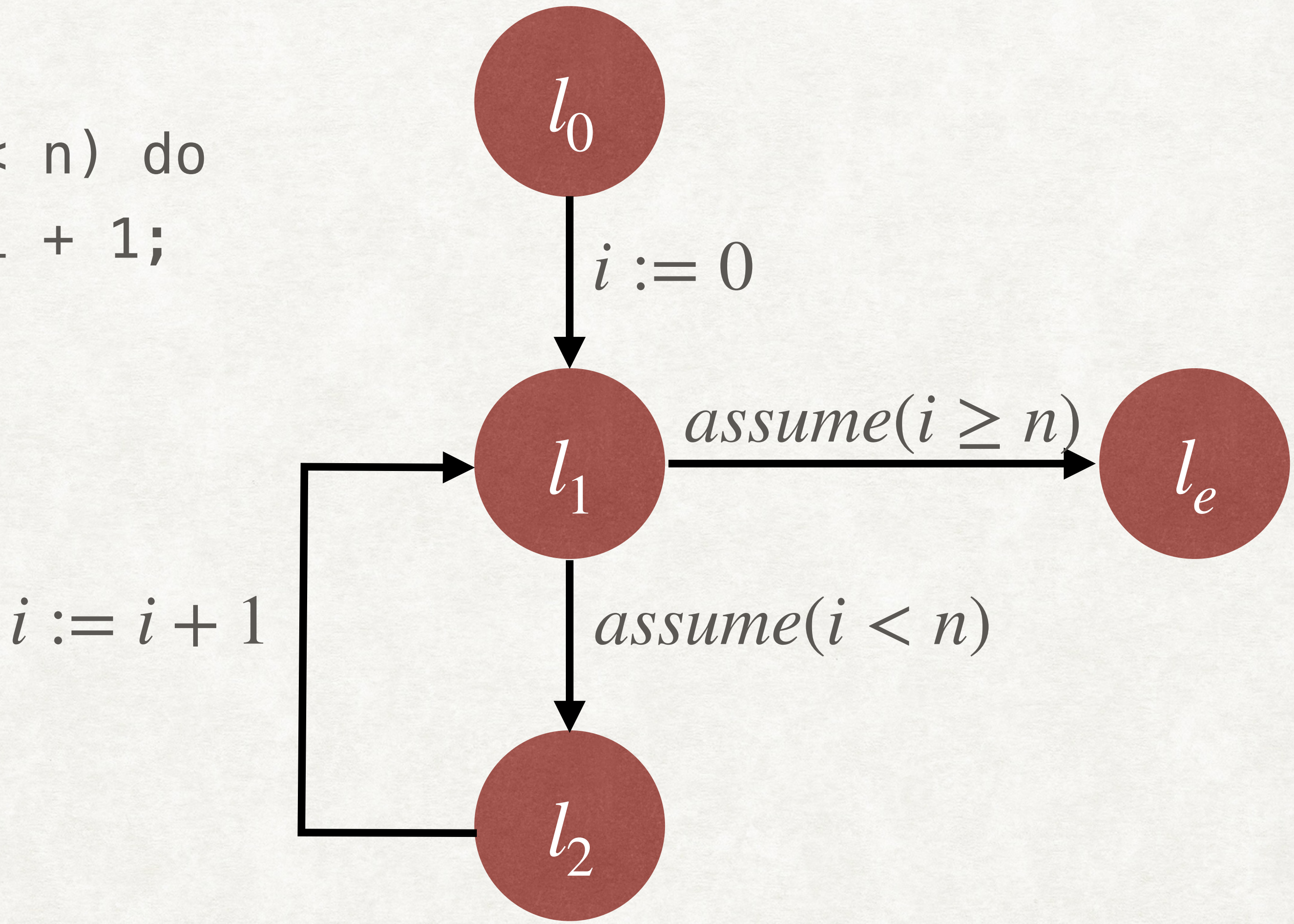
# ABSTRACT INTERPRETATION

# LABELLED TRANSITION SYSTEM

- We express the program $c$ as a labelled transition system $\Gamma_c \equiv (V, L, l_0, l_e, T)$
  - $V$ is the set of program variables
  - $L$ is the set of program locations
  - $l_0$ is the start location
  - $l_e$ is the end location
  - $T \subseteq L \times c \times L$ is the set of labelled transitions between locations.

# EXAMPLE

```
i := 0;
while(i < n) do
    i := i + 1;
```



# PROGRAMS AS LTS

- There are various ways to construct the LTS of a program
  - We can use control flow graph
  - We can use basic paths as defined by the book (BM Chapter 5). A basic path is a sequence of instructions that begins at the start of the program or a loop head, and ends at a loop head or the end of the program.
- Program State $(\sigma, l)$ consists of the values of the variables $(\sigma : V \to \mathbb{R})$ and the location.
- An execution is a sequence of program states, $(\sigma_0, l_0), (\sigma_1, l_1), \ldots, (\sigma_n, l_n)$, such that for all $i$, $0 \leq i \leq n-1$, $(l_i, c, l_{i+1}) \in T$ and $(\sigma_i, c) \hookrightarrow^* (\sigma_{i+1}, skip)$.
- A program satisfies its specification $\{P\}c\{Q\}$ if $\forall \sigma \in P$, for all executions $(\sigma, l_0), (\sigma_1, l_1), \ldots, (\sigma', l_e)$ of $\Gamma_c$, $\sigma' \in Q$.

# INDUCTIVE ASSERTION MAP

- With each location, we associate a set of states which are reachable at that location in any execution.
  - $\mu : L \to \Sigma(V)$
- To express that such a map is an inductive assertion map, we will use Strongest Post-condition.
  - $\forall (l, c, l') \in T \,.\, sp(\mu(l), c) \to \mu(l')$
- Then, if $\mu$ is an inductive assertion map on $\Gamma_c$, the Hoare triple $\{P\}c\{Q\}$ is valid if $P \to \mu(l_0)$ and $\mu(l_e) \to Q$.

# GENERATING THE INDUCTIVE ASSERTION MAP

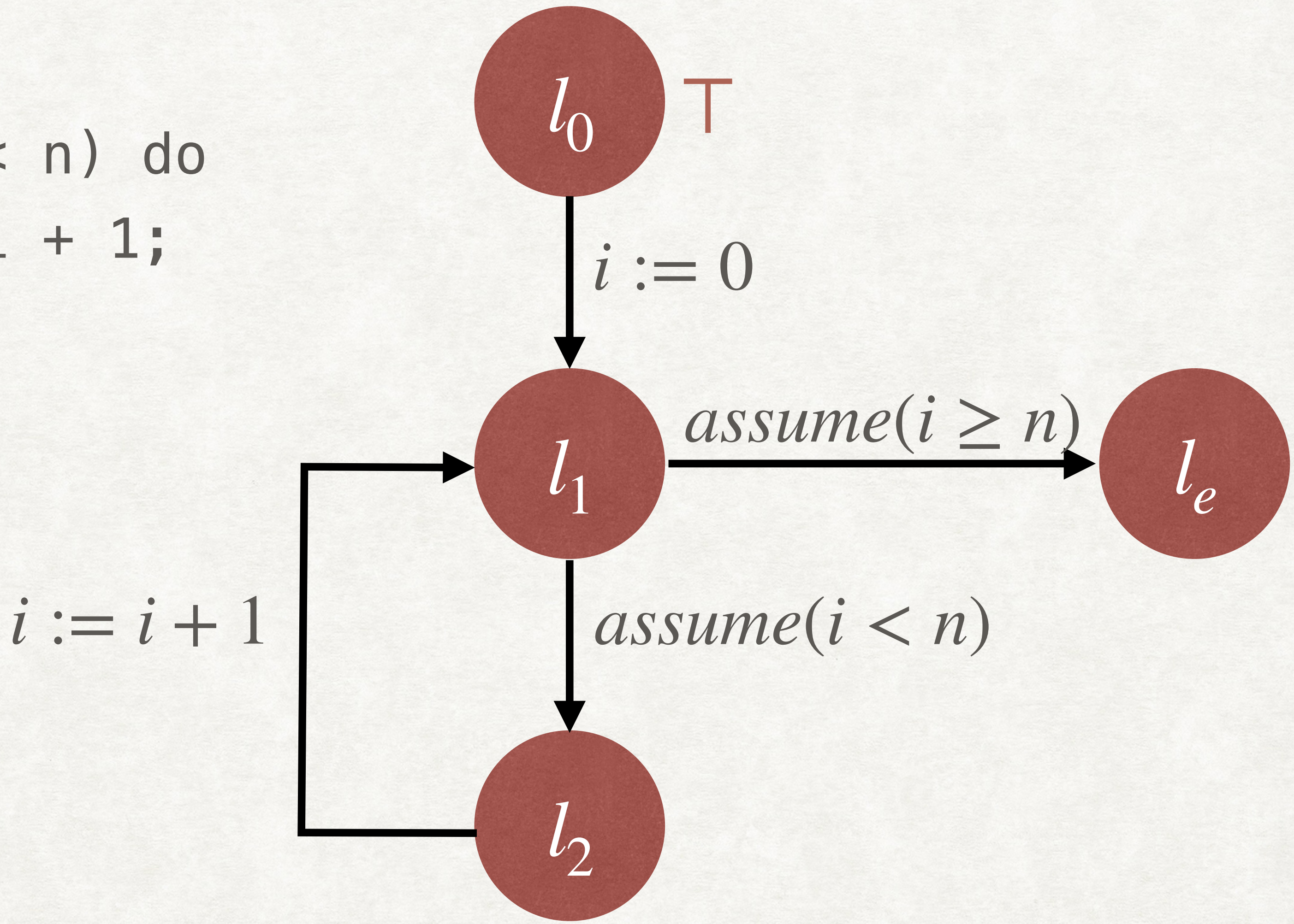
- We can express the inductive assertion map as a solution of a system of equations:
  - $X_{l_0} = P$
  - For all other locations $l \in L \backslash \{l_0\},\ X_l = \bigvee_{(l',c,l) \in T} sp(X_{l'}, c)$

# GENERATING THE INDUCTIVE ASSERTION MAP

```
ForwardPropagate(Γ_c,P)
  S := {l_0};
  μ(l_0) := P;
  μ(l) := ⊥, for l ∈ L\{l_0};
  while S ≠ ∅ do{
     l := Choose S;
     S := S \ {l};
     foreach (l,c,l') ∈ T do{
        F := sp(μ(l),c);
        if ¬(F → μ(l')) then{
           μ(l') := μ(l') ∨ F;
           S := S ∪ {l'};
        }
     }
  }
```

# EXAMPLE

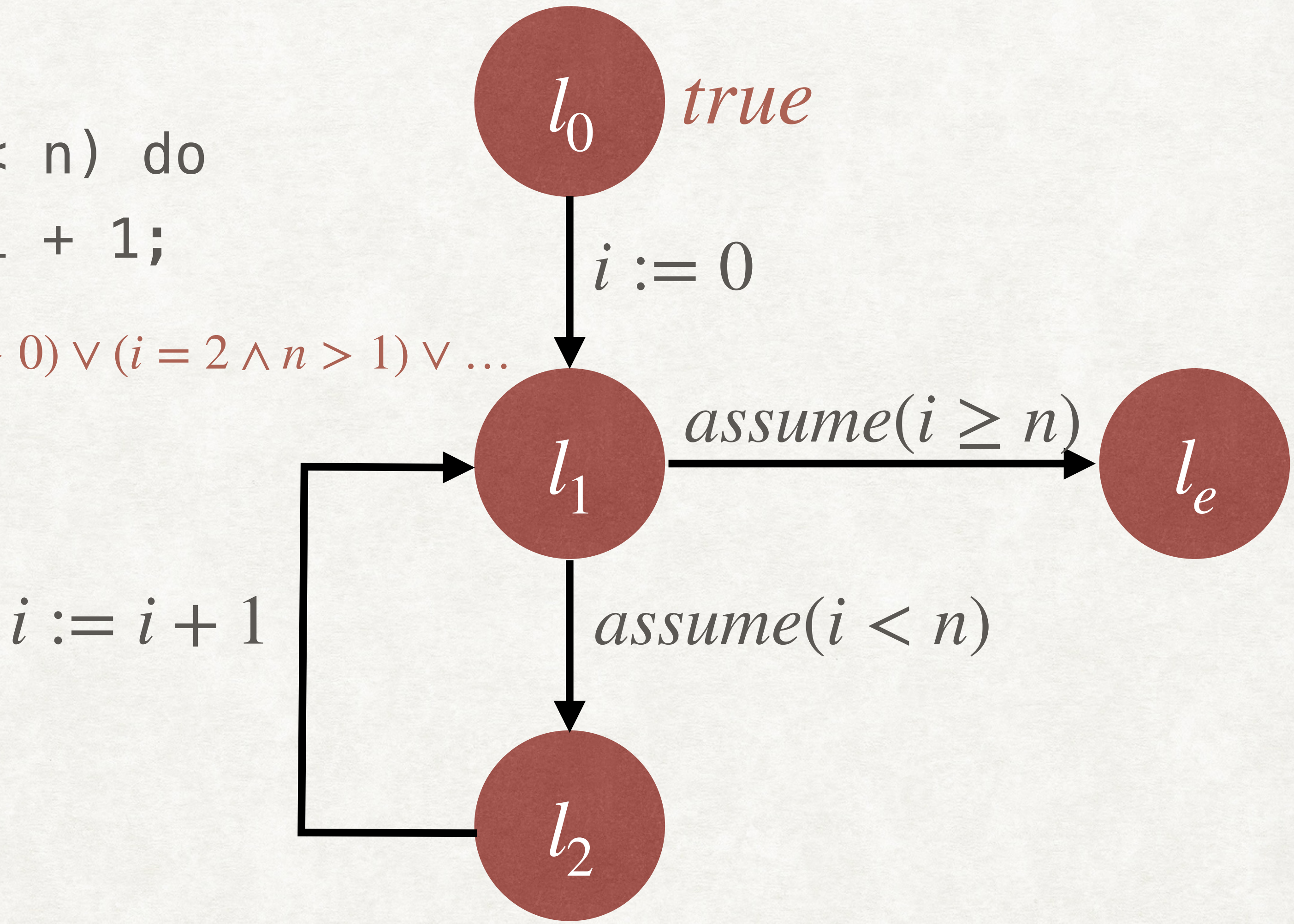
```
i := 0;
while(i < n) do
    i := i + 1;
```



# EXAMPLE

```
i := 0;
while(i < n) do
    i := i + 1;
```

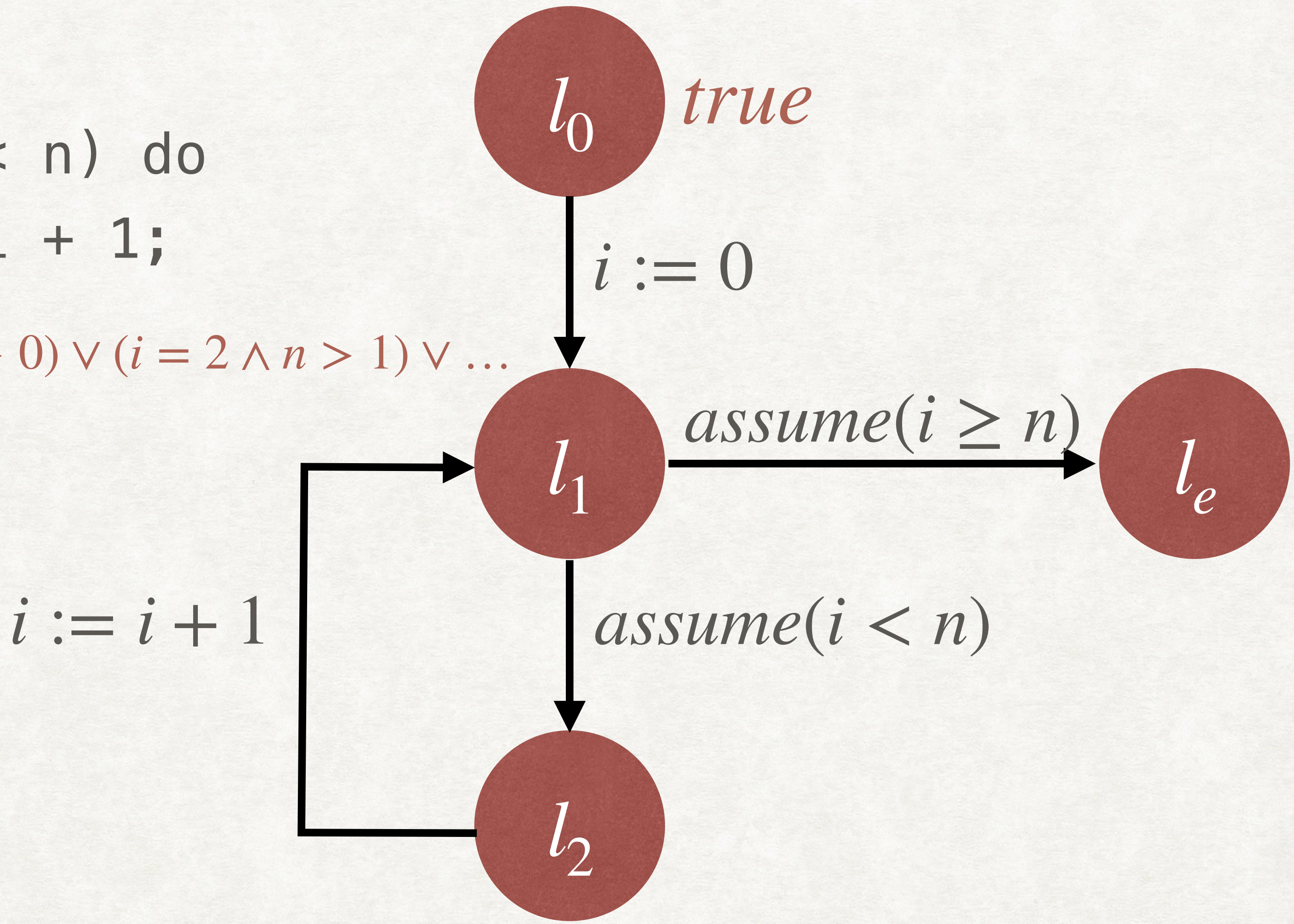
$(i = 0) \vee (i = 1 \wedge n > 0) \vee (i = 2 \wedge n > 1) \vee \dots$



# EXAMPLE

```
i := 0;
while(i < n) do
    i := i + 1;
```

$(i = 0) \vee (i = 1 \wedge n > 0) \vee (i = 2 \wedge n > 1) \vee \ldots$




FORWARDPROPAGATE WILL NOT TERMINATE

# ABSTRACT INTERPRETATION: OVERVIEW

- Instead of maintaining an arbitrary set of states at each location, maintain an artificially constrained set of states, coming from an abstract domain $D$.
  - $\hat{\mu} : L \to D$
- Let $States \triangleq V \to \mathbb{R}$ be the set of all possible concrete states.
  - Abstraction function, $\alpha : \mathbb{P}(States) \to D$
  - Concretization function, $\gamma : D \to \mathbb{P}(States)$
- $\hat{\mu}$ over approximates the set of states at every location.
  - For all locations $l$, $\gamma(\hat{\mu}(l)) \supseteq \mu(l)$
- Use abstract strongest post-condition operator $\hat{sp} : D \times c \to D$
  - $\gamma(\hat{sp}(d, c)) \supseteq sp(\gamma(d), c)$

# GENERATING THE INDUCTIVE ASSERTION MAP

```
ForwardPropagate(Γ_c,P)
  S := {l_0};
  μ(l_0) := P;
  μ(l) := ⊥, for l ∈ L\{l_0};
  while S ≠ ∅ do{
     l := Choose S;
     S := S \ {l};
     foreach (l,c,l') ∈ T do{
        F := sp(μ(l),c);
        if ¬(F → μ(l')) then{
           μ(l') := μ(l') ∨ F;
           S := S ∪ {l'};
        }
     }
  }
```

# ABSTRACT FORWARD PROPAGATE

```
AbstractForwardPropagate(Γ_c,P)
  S := {l_0};
  μ̂(l_0) := α(P);
  μ̂(l) := ⊥, for l ∈ L\{l_0};
  while S ≠ ∅ do{
      l := Choose S;
      S := S \ {l};
      foreach (l, c, l′) ∈ T do{
          F := ŝp(μ̂(l), c);
          if ¬(F ≤ μ̂(l′)) then{
              μ̂(l′) := μ̂(l′) ⊔ F;
              S := S ∪ {l′};
          }
      }
  }
```