Natural Semantics

- 実行過程の、最初と最後の状態（state）の関係を考える。
- transition（推移）は、

  ＜S, s＞ → s’

  と書く。
- →の定義　が、Table2.1（p20）に書いてある。

derivation tree（導出木）

- root ‥ 証明したい、＜S, s＞ → s’
  （一番下にある）
- leaves ‥ 公理の例（一番上に来る）
- internal node ‥ 規則（rule）の例の、結論。自分のすぐ上に、対応する仮定（premise）が来る。（真ん中にある）
- 使った公理と規則の条件（condition）は、すべて満たされているべき。
- 公理を木にしたものはsimple，規則を木にしたものはcompositeと呼ぶ

導出木の組み立て方

- rootから始めて、下から上へ築いてゆく。
- まず、成り立つことを示したい、＜S, s＞ → s'の左側＜S, s＞と一致する、公理や規則を見つける。
  （１）もし、公理と一致して、条件を満たすなら最終状態を決定して、導出木は完成。
  （２）もし、規則と一致するなら、まず規則の前提部分を作る。次に、条件がみたされているか調べて、OKだったら、最終状態を決定する。

terminate と loop（p25）

- もし、＜S, s＞ → s’ となる s’が存在すれば、状態 s におけるステートメント S の実行はterminateする。
- もし存在しなければ、loopする。

Properties of the semantics

- transition systemは、ステートメントとその性質について、論じる方法を提供してくれる。例えば、S1とS2が意味的に同じものであるか考える。
  すべての s と s’ において、
  ＜S1, s＞ → s’ かつ ＜S2, s＞ → s’であるなら、S1 と S2 は、意味的に同じであるといえる。

補題２．５

- 次のステートメント while b do S は、
  if b then (S; while b do S) else skip と
  意味的に同じである。
- （証明）２段階で行う。まず、(*)から(**)を示す。次に(**)から(*)を示す。

Induction on the Shape of Derivation Tree (導出木の構造における帰納法)

1. transition system の公理(Table2.1)で成り立つことを示して、証明したい性質がすべてのsimpleな導出木で成り立つことを証明する。
2. 証明したい性質が、すべてのcompositeな導出木で成り立つことを示す。(以下のように)。各規則に対して、その性質が規則の仮定(これをinduction hypothesisとよぶ)で成り立つとする。そして、条件が満たされているゆえ、その性質は規則の結果で成り立つことを証明する

定理 2.9

- Table2.1で表される natural semantics は deterministic である。
- deterministic ‥ どのステートメントSや初期状態 s においても、S の実行が終了するならば、最終状態 s’ を一意に決めることができる。
- （証明） ＜S, s＞ → s’ を仮定して、
  ＜S, s＞ → s’’ならば s’ = s” を証明する。
  導出木の構造における帰納法を使う。

つまり‥Induction on the Shape of Derivation Treeの言いたいことは‥

- まず公理で成り立つことを証明。 次に、規則で成り立つことを証明。それが言えれば、証明したかったものが transition system において、成り立つと言える。
- leaf（公理）が成立して、internal node（規則）が成立すれば、root（証明したいもの）が成立することが言える。（どんな形の木になるかは判らない。しかし公理と規則が成立することが言えたので、証明できることがわかる）

注意 (○ _ ○)

- ステートメントSについて、構造的帰納法を用いて、定理２．９を証明するのは不可能なぜなら[While tt]で While b do S の semantics をそのままの自分を使って定義しているから。
- 構造的帰納法は、証明するもののsemanticsが、compositionally に定義されている時に有効。
- Compositionally ‥ $\mathcal{A}$や$\mathcal{B}$の定義のようなもの。$\mathcal{A}$や$\mathcal{B}$自体は出てくるけれど、小さくなっている。

The semantic function Sns

- ステートメントの意味‥状態から状態への部分関数
- 次のように定義する。
  Sns : Stm → (State ↪ State)

  ステートメントを受け取って、状態を受け取って、状態を返すかもしれないし、返さないかもしれない関数を返す、関数。
- Sns [[s]] s = s' if ＜S, s＞ → s'
  | undef otherwise