ソフトウェアモデル論（2010年度）
第12回・2010/12/11

桑原 寛明
情報理工学部 情報システム学科

自然演繹の健全性

- 論理式集合 $P_1, \ldots, P_n$ から論理式 Q が導出できるならば Q は $P_1, \ldots, P_n$ の論理的帰結である
  - $P_1, \ldots, P_n \vdash Q$ ならば $P_1, \ldots, P_n \models Q$
- 証明は $P_1, \ldots, P_n \vdash Q$ の証明木の高さに関する帰納法による
  - 高さが 1 の場合を示す
  - 高さが n 未満の場合に成り立つと仮定して n の場合を示す

自然演繹の完全性

- 論理式 Q が論理式集合 $P_1, \ldots, P_n$ の論理的帰結ならば $P_1, \ldots, P_n$ から Q が導出できる
  - $P_1, \ldots, P_n \models Q$ ならば $P_1, \ldots, P_n \vdash Q$
- 証明は以下の順

  $P_1, \ldots, P_n \models Q$
  $\Rightarrow \models P_1 \rightarrow (P_2 \rightarrow (\ldots(P_n \rightarrow Q)\ldots))$
  $\Rightarrow \vdash P_1 \rightarrow (P_2 \rightarrow (\ldots(P_n \rightarrow Q)\ldots))$
  $\Rightarrow P_1, \ldots, P_n \vdash Q$

モデル検査

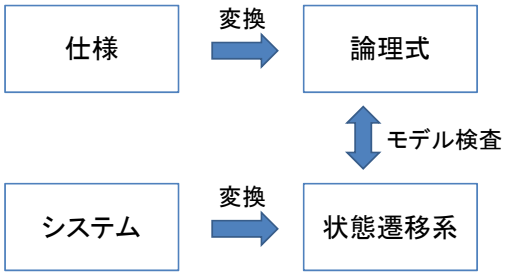
モデル検査

- 状態遷移系として記述されたシステムが、論理式として記述された性質を満たすか否か、網羅的かつ機械的に検証する手法
- 利点
  - 網羅的、機械的、反例
- 例えば、プログラムが必ず停止すること、デッドロックしないこと、などを検証できる

モデル検査の手順

- 検査対象のシステムを状態遷移系を用いて記述する
  - 対象はプログラムや設計など
    - 「動作する」ものであれば何でも対象になる
  - 状態遷移系としてはKripke構造やオートマトンなど
- 検査したい性質を論理式を用いて記述する
  - 時相論理や様相論理を用いる
- 検査アルゴリズムを実行する
  - アルゴリズムを実装した様々なツールがある

モデル検査の手順



Kripke構造

- 状態遷移系の一種
- 直観的には
  - オートマトンから記号を除去
  - オートマトンの各状態にその状態で真になる命題を追加
    - 「x の値は 1 である」など
    - 命題は命題論理の範囲内で

Kripke構造の定義

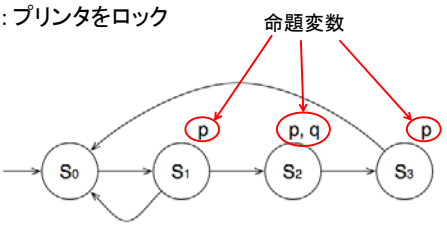
- Kripke構造 M = (S, R, L)
  - S : 状態の有限集合
  - R : 遷移関係
    - $R \subseteq S \times S$
    - R(s, s') : 状態 s から s' への遷移がある
  - L : ラベル付け関数
    - $L : S \rightarrow 2^{PV}$
    - PV は命題変数の集合
    - 各状態にその状態で真となる命題を表す命題変数を割り当てる関数

経路

- 遷移関係に従って移り変わる状態の列
- $\sigma = s_0 s_1 s_2 \cdots$
  - すべての $i \geq 0$ に対して $R(s_i, s_{i+1})$
- 関係 R(s, s') は、システムの状態が s の場合、次の時刻で状態が s' に変わると理解する

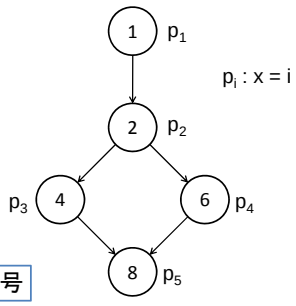
Kripke構造の例

- COPY = (S, R, L)
  - $L(s_0) = \emptyset$, $L(s_1) = L(s_3) = \{p\}$, $L(s_2) = \{p, q\}$
  - p : スキャナをロック
  - q : プリンタをロック



Kripke構造の例

```
1:  x = 1;
2:  x = 2;
3:  if (...) {
4:    x = 3;
5:  } else {
6:    x = 4;
7:  }
8:  x = 5;
```

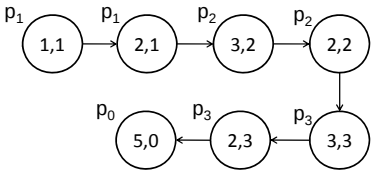


状態は行番号

Kripke構造の例

```
1:  x = 1;
2:  while (x < 3) {
3:    x = x + 1;
4:  }
5:  x = 0;
```

状態は行番号と
x の値の組



$p_i : x = i$

時相論理

- 命題論理は、ある瞬間の状況に関する命題のみ扱える
  - 前後（過去、未来）の状況との関連は扱えない
- 扱えるように拡張したものが時相論理
  - 「いつか停止する」
  - 「x の値はずっと正である」
  - など

時間のとらえ方

- 離散 vs 連続
  - 離散：単位時間を仮定
  - 連続：任意の2時刻間に別の時刻の存在を仮定
- 点 vs 区間
- 未来 vs 過去
- 分岐 vs 線形
  - 分岐：時間の流れによる状態変化のすべての可能性を同時に考慮
  - 線形：一つの可能性のみを選択

CTL（計算木論理）

- 離散、点、未来、分岐
- 計算木に対する論理
  - 計算木は、ある状態を開始状態とするすべての経路を1つにまとめた木構造
  - 開始状態が木構造の根
- 経路の選択
  - すべての経路において、ある経路において
- タイミングの選択
  - 次、将来のいつか、今後ずっと、ある時点まで

CTLの論理式

1. 命題変数は状態式
2. P, Q が状態式ならば ¬P, P∧Q, P∨Q, P→Q は状態式
3. P が経路式ならば **A** P, **E** P は状態式
4. P, Q が状態式ならば **X** P, **F** P, **G** P, P **U** Q は経路式
5. 以上が状態式と経路式のすべてであり、状態式がCTLの論理式のすべて

CTLの意味論

- M, s ⊨ P
  - Kripke構造 M = (S, R, L) における状態 s∈S を開始状態とする計算木においてCTL式 P が成り立つ
- PV は命題変数の集合であり p∈PV
- M, s ⊨ p
  - p∈L(s)
- M, s ⊨ ¬P
  - M, s ⊨ P でない

CTLの意味論

- M, s ⊨ P∧Q
  - M, s ⊨ P かつ M, s ⊨ Q
- M, s ⊨ P∨Q
  - M, s ⊨ P または M, s ⊨ Q
- M, s ⊨ P→Q
  - M, s ⊨ P ならば M, s ⊨ Q
- M, s ⊨ **AX** P
  - σ(0)=s なるすべての経路σにおいて M, σ(1) ⊨ P
- M, s ⊨ **EX** P
  - σ(0)=s かつ M, σ(1) ⊨ P なる経路σが存在する

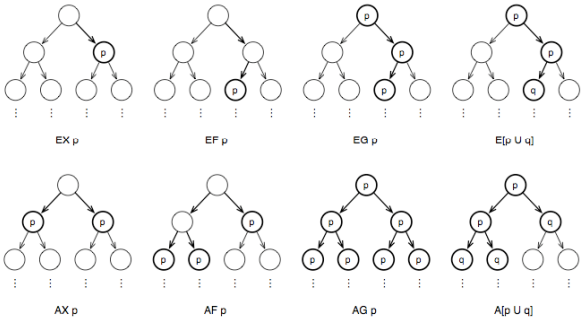
CTLの意味論

- M, s ⊨ **AF** P
  - σ(0)=s なるすべての経路σにおいてある i≥0 が存在して M, σ(i) ⊨ P
- M, s ⊨ **EF** P
  - σ(0)=s かつ M, σ(i) ⊨ P なる i≥0 が存在する経路σが存在する
- M, s ⊨ **AG** P
  - σ(0)=s なるすべての経路σにおいて任意の i≥0 に対して M, σ(i) ⊨ P
- M, s ⊨ **EG** P
  - σ(0)=s かつ任意の i≥0 に対して M, σ(i) ⊨ P なる経路σが存在する

CTLの意味論

- M, s ⊨ **A** [P **U** Q]
  - σ(0)=s なるすべての経路σにおいてある j≥0 が存在して M, σ(j) ⊨ Q かつ 0≤i<j に対して M, σ(i) ⊨ P
- M, s ⊨ **E** [P **U** Q]
  - σ(0)=s かつ、ある j≥0 が存在して M, σ(j) ⊨ Q かつ 0≤i<j に対して M, σ(i) ⊨ P なる経路σが存在する

CTLの意味論



経路演算子、時相演算子

- ¬**AX** P = **EX** ¬P
- ¬**AF** P = **EG** ¬P
- ¬**AG** P = **EF** ¬P

- EX, EG, EU があれば他の演算子は表現可能
  - EX, EG, EU の組み合わせに限らない

例

- COPY, s_0 ⊨ **EF**(p∧q)
  - p, q がともに成り立つ状態に到達できる経路がある
- COPY, s_0 ⊨ ¬**AF**(p∧q)
  - すべての経路で p, q がともに成り立つ状態に到達できるわけではない