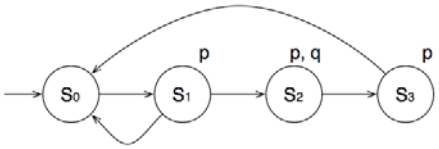
ソフトウェアモデル論（2013年度）
第15回・2014/01/16

桑原 寛明
情報理工学部 情報システム学科

例 （復習）

- COPY, $s_0 \models$ **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, $s_0 \models \neg$**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



モデル検査アルゴリズム （復習）

- Kripke構造 M、状態 s、CTL式 P
- CTL式の演算子は¬、∨、EX、EG、EU

- Check(M, P)
  - M の各状態に対して P の各部分式の中で成り立つものを求める

- 最終的に s で成り立つ式の中に P が含まれていれば M, s ⊨ P

モデル検査アルゴリズム （復習）

case: $P \in PV$
  for all $s \in S$ **do**
    if $P \in L(s)$ **then** $label(s) := label(s) \cup \{P\}$
case: $P \equiv \neg Q$
  $Check(M, Q)$
  for all $s \in S$ **do**
    if $Q \notin label(s)$ **then** $label(s) := label(s) \cup \{\neg Q\}$
case: $P \equiv Q_1 \vee Q_2$
  $Check(M, Q_1)$
  $Check(M, Q_2)$
  for all $s \in S$ **do**
    if $Q_1 \in label(s)$ **or** $Q_2 \in label(s)$ **then** $label(s) := label(s) \cup \{Q_1 \vee Q_2\}$

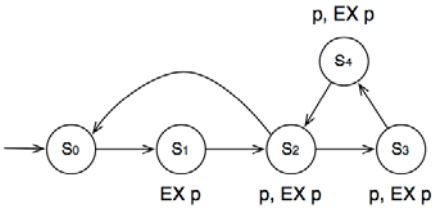
モデル検査アルゴリズム （復習）

- **EX** Q の場合
  - 一つ遷移すると Q が成り立つ状態に到達できる状態では **EX** Q が成り立つ

case: $P \equiv \mathbf{EX}\, Q$
  $Check(M, Q)$
  for all $s \in \{s \mid Q \in label(s)\}$ **do**
    for all s' **such that** $R(s', s)$ **do**
      $label(s') := label(s') \cup \{\mathbf{EX}\, Q\}$

モデル検査アルゴリズム （復習）

- **EX** Q の場合（続き）



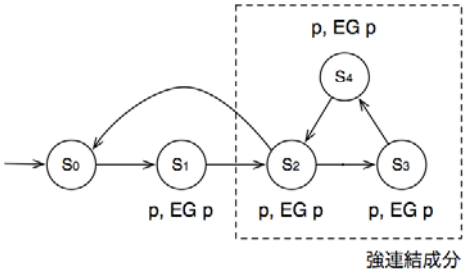
モデル検査アルゴリズム （復習）

- **EG** Q の場合
  - 常に Q が成り立っている経路を見つける
- Q が成り立つ状態のみに着目
- 強連結成分
  - 任意の2状態間に経路が存在
  - 極大
- 強連結成分中のいずれかの状態に到達可能

```
Check(M, Q)
S' := {s | Q ∈ label(s)}
SCC := {C | C は S' の強連結成分 }
T := ∪_{C∈SCC} {s | s ∈ C}
for all s ∈ T do label(s) := label(s) ∪ {EG Q}
while T ≠ ∅ do
  choose s ∈ T
  T := T − {s}
  for all s' ∈ S' such that R(s', s) do
    if EG Q ∉ label(s') then
      label(s') := label(s') ∪ {EG Q}
      T := T ∪ {s'}
```


モデル検査アルゴリズム （復習）

- **EG** Q の場合（続き）



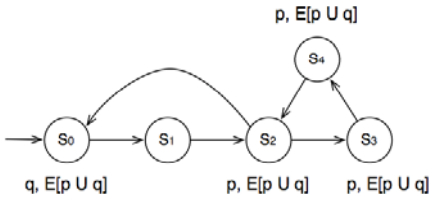
モデル検査アルゴリズム （復習）

- **E**[Q_1 **U** Q_2] の場合
  - Q_2 が成り立っている状態から遷移をさかのぼって Q_1 が成り立っている間 **E**[Q_1 **U** Q_2] が成り立つ

```
Check(M, Q1)
Check(M, Q2)
T := {s | Q2 ∈ label(s)}
for all s ∈ T do label(s) := label(s) ∪ {E[Q1 U Q2]}
while T ≠ ∅ do
  choose s ∈ T
  T := T − {s}
  for all s' such that R(s', s) do
    if E[Q1 U Q2] ∉ label(s') and Q1 ∈ label(s') then
      label(s') := label(s') ∪ {E[Q1 U Q2]}
      T := T ∪ {s'}
```

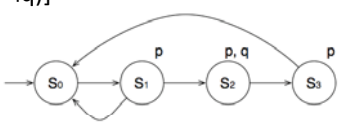

モデル検査アルゴリズム （復習）

- **E**[Q_1 **U** Q_2] の場合（続き）



練習問題6.12（レポートその13）

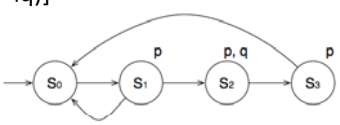
- COPY, s_0 ⊨ **EG**(¬pV¬q)]
  - ¬**AF**(p∧q) か？



- label(s_0) = { }
- label(s_1) = { }
- label(s_2) = { }
- label(s_3) = { }

練習問題6.12（レポートその13）

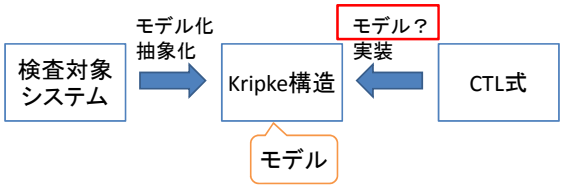
- COPY, s_0 ⊨ **EG**(¬pV¬q)]
  - ¬**AF**(p∧q) か？



- label(s_0) = {¬p, ¬q, ¬pV¬q, **EG**(¬pV¬q)}
- label(s_1) = {¬q, ¬pV¬q, **EG**(¬pV¬q)}
- label(s_2) = {}
- label(s_3) = {¬q, ¬pV¬q, **EG**(¬pV¬q)}

なぜ「モデル検査」と呼ぶか

- モデル検査は、Kripke構造がCTL式のモデルになっているか検査すること
- 一般的には、状態遷移系が（時相）論理式のモデルになっているか検査



並行プログラム

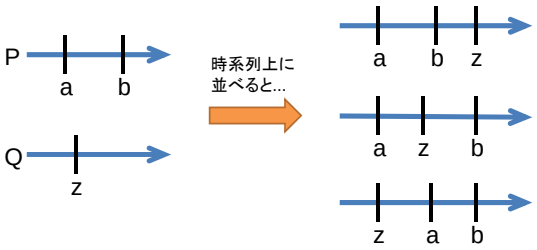
- 複数の計算を（見かけ上）同時に実行するプログラム
  - 並行：論理的に複数の計算を同時に実行
  - 並列：物理的に複数の計算を同時に実行
- 分散システム、クラスタ
- マルチプロセッサ、マルチコア
- マルチタスク、マルチプロセス、マルチスレッド

並行プログラムに固有の難しさ

- 同時に実行される各計算における命令実行のタイミング
  - 非決定性
- 同時に実行される計算同士の相互作用
  - ファイルなどの資源の共有
  - メッセージ通信
- 並行プログラムの動作が正しさを確認することは逐次プログラムに比べはるかに難しい

非決定性

- 並行に実行される各計算の動作を時系列上に並べる方法は一通りではない



非決定性

- 並行プログラムの実行系列は一通りでない
  - 同じ入力に対して同じ実行を行うとは限らない
- どの実行系列が実行されたかは実行が完了して初めてわかる
- 例えば
  - 初めに実行される a または z を非決定的に選択
  - a の次に実行される b または z を非決定的に選択
- すべての可能性を尽くしてテストすることは非常に困難

資源共有と相互排除

- 並行に実行される計算同士でファイルやネットワークなどを共有する
  - 変数の共有もありえる
- 同時使用はできないので排他制御が必要
  - セマフォ、モニタ
  - デッドロック、ライブロック

デッドロック

- P : スキャナ、プリンタの順にロックしてコピー
- Q : プリンタ、スキャナの順にロックしてコピー

1. P がスキャナをロック
2. Q がプリンタをロック
3. ??

メッセージ通信

- 並行実行される複数の計算の間でデータをやり取りする
- 同期通信
  - 送信側と受信側の準備が整ったら通信する
- 非同期通信
  - 送信側は受信側を気にせず送信
  - 受信側はメッセージが来ていれば受信、来ていなければ来るまで待機
- 通信プロトコル

並行プログラムのモデル化

- 並行プログラムにはモデル検査が有効
  - 非決定性によるたくさんの可能性を網羅できる
- どのようにKripke構造でモデル化するか？
  1. 並行動作する個々の計算（プロセス）をモデル化する
  2. 合成して全体のモデルを得る

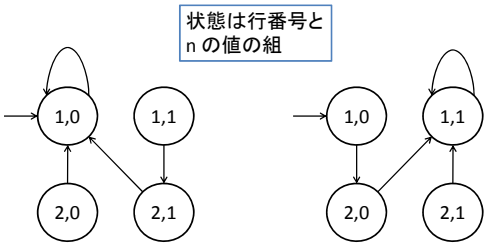
並行プログラムの例

プロセスP

```
1:  if (n == 0) goto 1;
2:  n = 0; goto 1;
```

プロセスQ

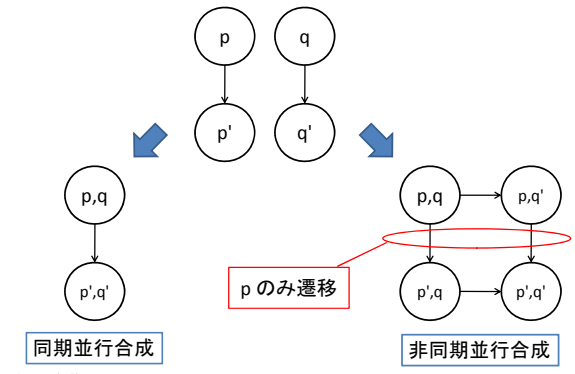
```
1:  if (n == 1) goto 1;
2:  n = 1; goto 1;
```



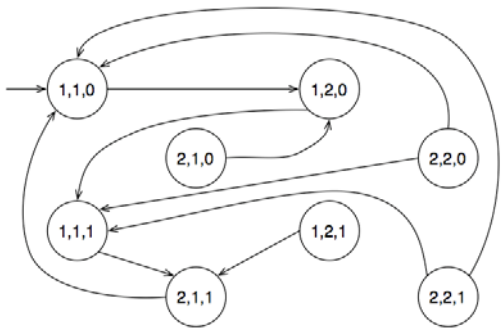
並行合成

- 並行動作するプロセスを一つにまとめること
- 同期並行合成
  - 各プロセスにおける状態遷移が同期して発生する
- 非同期並行合成
  - 各プロセスにおける状態遷移は互いに無関係に発生する
  - 通常は1回の遷移で1つのプロセスのみが状態遷移する

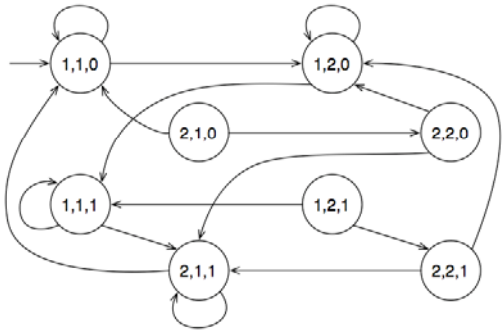
並行合成



同期並行合成の例



非同期並行合成の例



命題の割り当て

- 各状態に対してその状態で成り立つ命題の集合を割り当てる
- 例えば
  - 命題変数
    - P_i : プロセス P の i 行目を実行
    - Q_i : プロセス Q の i 行目を実行
    - N_i : 変数 n の値が i
  - 状態 (p, q, n) に命題集合 { P_p, Q_q, N_n } を割り当てる

調べたい性質の例

- プロセス P とプロセス Q がともに 2 行目を実行することはない
  - 同時に変数 n に代入は危険
- CTLで表現すると **AG**$\neg(P_2 \wedge Q_2)$
  - この例の場合では、状態 (2,2,0) および (2,2,1) に到達しないことと同義

モデル検査の実行

- モデル検査アルゴリズムを実行
- M_s, (1,1,0) ⊨ **AG** $\neg(P_2 \wedge Q_2)$
  - 同期並行合成と命題の割り当てによって得られる Kripke構造 M_s
- M_a, (1,1,0) ⊨ **AG** $\neg(P_2 \wedge Q_2)$
  - 非同期並行合成と命題の割り当てによって得られるKripke構造 M_a

NuSMVを使って検査

```
MODULE main
VAR
p : {1, 2};   ← Pの行番号
q : {1, 2};   ← Qの行番号
n : {0, 1};   ← nの値
ASSIGN
init(p) := 1;
next(p) :=
  case
    p = 1 & n = 0 : 1;
    p = 1 & n = 1 : 2;
    p = 2 : 1;
  esac;
init(q) := 1;
next(q) :=
  case
    q = 1 & n = 1 : 1;
    q = 1 & n = 0 : 2;
    q = 2 : 1;
  esac;
init(n) := 0;
next(n) :=
  case
    p = 2 & q = 2 : {0, 1};
    p = 2 & q = 1 : 0;
    p = 1 & q = 2 : 1;
    p = 1 & q = 1 : n;
  esac;
```

論理式 → `SPEC AG !(p = 2 & q = 2);`

NuSMVを使って検査

```
% NuSMV concurrent_loop.smv
...Copyrightなどの表示...
-- specification AG !(p = 2 & q = 2)  is true
```


定期試験

- 2013/01/23 (Thu)
- 2限（11:00 - 12:00）
- F201
- 持ち込みなし
  - 自然演繹の推論規則は用紙に記載
- 質問は早めに
  - 電子メールでも可

定期試験

- 以下の範囲から大問 3 題を出題
  - 有限オートマトンと正規表現
  - チューリング機械
  - 命題論理
  - モデル検査（6.4節まで）
- 概念とアルゴリズムをよく理解しておくこと
- 難易度は練習問題、演習問題と同程度
- 解答の経過も採点対象