ソフトウェアモデル論（2010年度）
第15回・2011/01/17

桑原 寛明
情報理工学部 情報システム学科

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

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
  - **EF** P = **E**[⊤ **U** P]　　⊤は恒真（任意の状態で真）
  - **A**[P **U** Q] = ¬(**EG** ¬Q ∨ **E**[¬Q **U** (¬P∧¬Q)])
  - **EX**, **EG**, **EU** の組み合わせに限らない

練習問題 6.4（レポートその12 ）

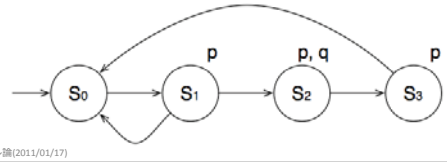
- **EF** P = **E**[⊤ **U** P]
- **EG** P = ¬**AF**(¬P)
- **AX** P = ¬**EX**(¬P)
- **AF** P = **A**[⊤ **U** P]
- **AG** P = ¬**EF**(¬P)

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

例

- COPY, $s_0 \models$ **EF**(p∧q)
  - s_0から開始しp, q がともに成り立つ状態に到達できる経路がある
- COPY, $s_0 \models$ ¬**AF**(p∧q)
  - s_0から開始するすべての経路で 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

- 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 の各部分式のうち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

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** $label(s) := label(s) \cup \{Q_1 \vee Q_2\}$

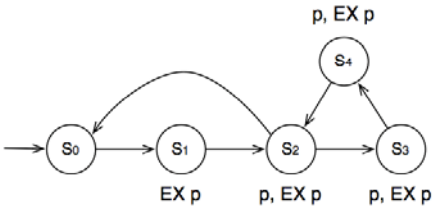
モデル検査アルゴリズム

- **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

- **EX** p



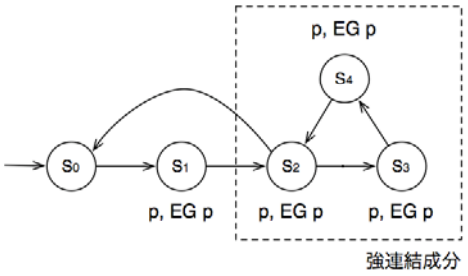
モデル検査アルゴリズム

- **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点が到達可能
  - 極大
- 強連結成分中のいずれかの状態に到達可能

$Check(M, P)$
$S' := \{s \mid P \in label(s)\}$
$SCC := \{C \mid C$ は S'の強連結成分 $\}$
$T := \bigcup_{C \in SCC} \{s \mid s \in C\}$
for all $s \in T$ **do** $label(s) := label(s) \cup \{\mathbf{EG}\, P\}$
while $T \neq \emptyset$ **do**
  choose $s \in T$
  $T := T - \{s\}$
  for all $s' \in S'$ **such that** $R(s', s)$ **do**
    if $\mathbf{EG}\, P \notin label(s')$ **then**
      $label(s') := label(s') \cup \{\mathbf{EG}\, P\}$
      $T := T \cup \{s'\}$

モデル検査アルゴリズム

- **EG** p



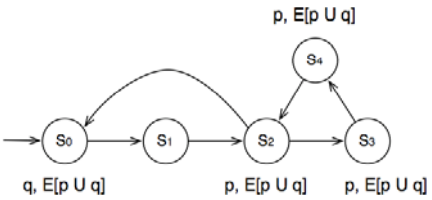
モデル検査アルゴリズム

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

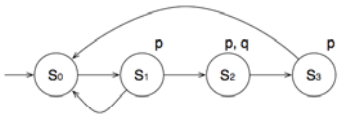

モデル検査アルゴリズム

- **E**[p **U** q]



例

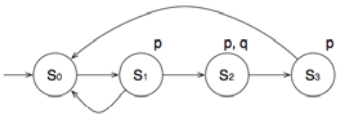
- COPY, $s_0 \models$ **E**[⊤**U**¬(¬p∨¬q)]
  - **EF**(p∧q) か？



- label(s_0) = {⊤, ¬p, ¬q, ¬p∨¬q, **E**[⊤**U**¬(¬p∨¬q)}
- label(s_1) = {⊤, ¬q, ¬p∨¬q, **E**[⊤**U**¬(¬p∨¬q)}
- label(s_2) = {⊤, ¬(¬p∨¬q), **E**[⊤**U**¬(¬p∨¬q)}
- label(s_3) = {⊤, ¬q, ¬p∨¬q, **E**[⊤**U**¬(¬p∨¬q)}

練習問題 6.11（レポートその12）

- COPY, $s_0 \models$ **EG**(¬p∨¬q)
  - ¬**AF**(p∧q)



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

並行プログラム

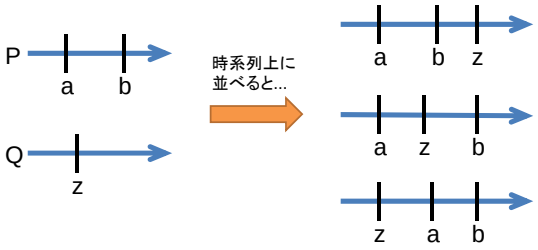
- 複数の計算を（見かけ上）同時に実行するプログラム
  - 並行：論理的に複数の計算を同時に実行
  - 並列：物理的に複数の計算を同時に実行
- 分散システム、クラスタ
- マルチプロセッサ、マルチコア
- マルチタスク、マルチプロセス、マルチスレッド

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. ??

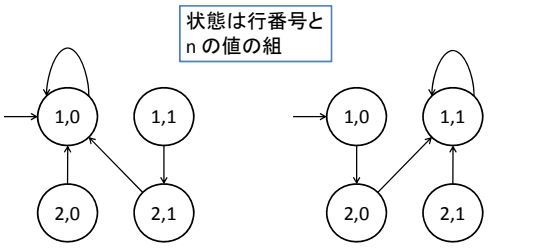
並行プログラムのモデル化

- 並行プログラムにはモデル検査が有効
  - 非決定性による可能性を網羅することができる
- 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;
```

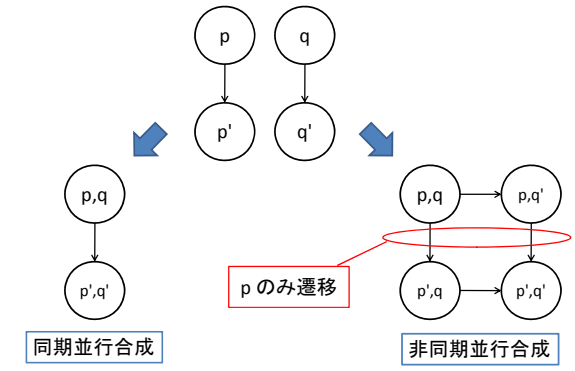
プロセスQ
```
1:  if (n == 1) goto 1;
2:  n = 1; goto 1;
```



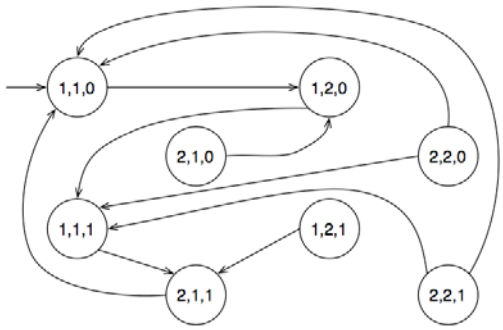
並行合成

- 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

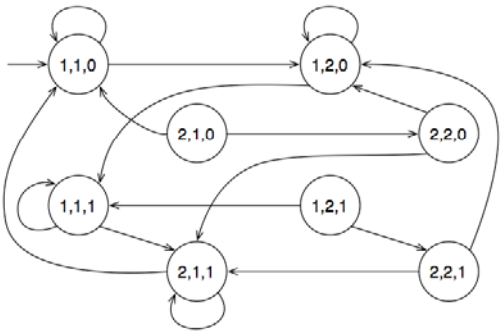
並行合成



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**¬($P_2 \wedge Q_2$)
  - 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** ¬(P_2 ∧ 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** ¬(P_2 ∧ 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

CTLによる性質の記述

- モデル検査では、検証したいシステムの性質を時相論理式（例えばCTL）で記述する
  - 慣れないと難しい
- 典型的な性質
- 記述パターン

システムに対する要求

- 機能要求
  - 入力に対する出力
- 非機能要求
  - 性能、使いやすさ
  - 危険な動作をしない
  - 停止しない

  など

典型的な性質

- 活性
- 到達可能性
- 安全性
- 公平性
  - 並行に実行される各プロセスに実行の機会が与えられること
  - 特定のプロセスのみが実行されないこと

到達可能性

- システムが初期状態からある特定の状態に到達する可能性があること
- 関数 f を呼び出す可能性がある
  - **EF** (call_f)
- x > n が成り立つ可能性はない
  - ¬**EF** (x > n)
- スキャナとプリンタをロックする可能性がある
  - **EF** (lock_scanner ∧ lock_printer)

安全性

- システムがある望ましくない状況に決して陥らないこと
- 2つのプロセスが同時に関数 f を実行しない
  - **AG** ¬(1_call_f ∧ 2_call_f)
- メモリのオーバーフローが発生しない
  - **AG** ¬overflow
- ログインしなければデータにアクセスできない
  - ¬**E**[¬login **U** (access ∧ ¬login)]

活性

- 将来いつか必ず、ある望ましい状況が発生すること
- スイッチを押せば電源が入る
  - **AG** (push_switch → **AF** power_on)
- データを入力してOKボタンをクリックすれば送信される
  - **AG** ((input_data ∧ click_ok) → **AF** data_sent)
- 信号が青に変わる
  - **AGAF** green

記述パターン

- システムに要求される性質を、自然言語で表現することは難しくない
- システムに要求される性質を、時相論理式で表現することは（慣れないと）難しい
- 典型的な表現のパターン
  - 自然言語と時相論理式のマッピング
  - http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml

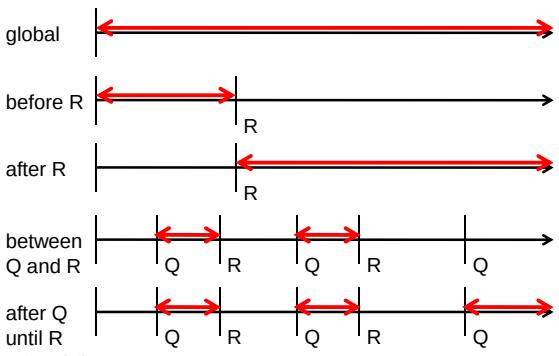
スコープ

- システムの実行系列において、性質の成否に着目する範囲
- global
  - 実行系列のすべてが対象
- before
  - 指定されたイベントや状態に到達するまでが対象
- after
  - 指定されたイベントや状態に到達してからが対象

スコープ

- between
  - 指定された2つのイベントや状態の間が対象
  - 1つ目のイベントや状態に到達しても2つ目のイベントや状態が発生しなければ対象外
- after-until
  - 指定された2つのイベントの状態の間が対象
  - さらに、1つ目のイベントや状態に到達してから2つ目のイベントや状態に到達しない場合も対象

スコープ



Occurrence パターン

- 指定されたスコープ内で、指定されたイベントが発生すること、指定された状態に到達することを表す
- Absence
  - 決して発生・到達しない
- Universality
  - 常に発生・到達する
- Existence
  - 発生・到達することがある
- Bounded Existence
  - k回発生・到達することがある

Order パターン

- 指定されたスコープ内で、特定の順序でイベントが発生・状態に到達することを表す
- Precedence
  - p が必ず q に先行する
- Response
  - q は必ず p の後で発生・到達する
- Precedence Chains
  - $p_1, \ldots, p_n$ が必ず $q_1, \ldots, q_m$ に先行する
- Response Chains
  - $q_1, \ldots, q_m$ は必ず $p_1, \ldots, p_n$ の後で発生・到達する

P の Absence を CTL で表現

- global
  - **AG** (¬P)
- before R
  - ¬**E**[¬R **U** (P ∧ **EF** R ∧ ¬R)]
- after R
  - **AG** (R → **AG** (¬P))
- between Q and R
  - **AG** (Q ∧ ¬R → ¬**E**[¬R **U** (P ∧ **EF** R ∧ ¬R)])
- after Q until R
  - **AG** (Q ∧ ¬R → ¬**E**[¬R **U** (P ∧ ¬R)])

定期試験

- 2011/01/24 (Mon)
- 2限（11:00 - 12:00）
- F302
- 持ち込みなし
  - 自然演繹の推論規則は問題用紙に記載
- 解答の方法

- 1/20, 21 は出張で不在のため質問は早めに
  - 電子メールでも可

定期試験

- 以下の7題から4題選択して解答
  - 1. 用語説明
  - 2. 有限オートマトンと正規表現
  - 3. 有限オートマトンと正規表現
  - 4. チューリング機械
  - 5. 命題論理
  - 6. 命題論理
  - 7. モデル検査
    - 6.4 節は範囲外