プログラミング代数特論 II
(2006/1/13, 2006/1/20)

佐藤秀明

概要

- 7.9 と 7.10
  - A Case Study: Peterson's Mutual Exclusion Algorithm
    - Coalgebra による仕様記述と検証の例
  - Refinements between Coalgebraic Specifications
    - 仕様記述の具体化 / 抽象化

Peterson アルゴリズム

- 共有資源に対する相互排他アクセスを保証
  - 同時に critical section に入れるプロセスは 1 つのみ
    - とりあえずプロセス数は 2 とする
- 2 種類のフラグを用意
  - A_interest/B_interest
    - プロセス A/B が critical section に入りたがっているか
    - 相手の interest フラグは read-only
  - turn
    - 実際に critical section に入る権利を持つのは A/B のどちらか
    - 2 つのプロセス間で共有される変数

Peterson アルゴリズムの疑似コード

```
init:
  A_interest = false;
  B_interest = false;
  turn = A;        // or turn = B;
```

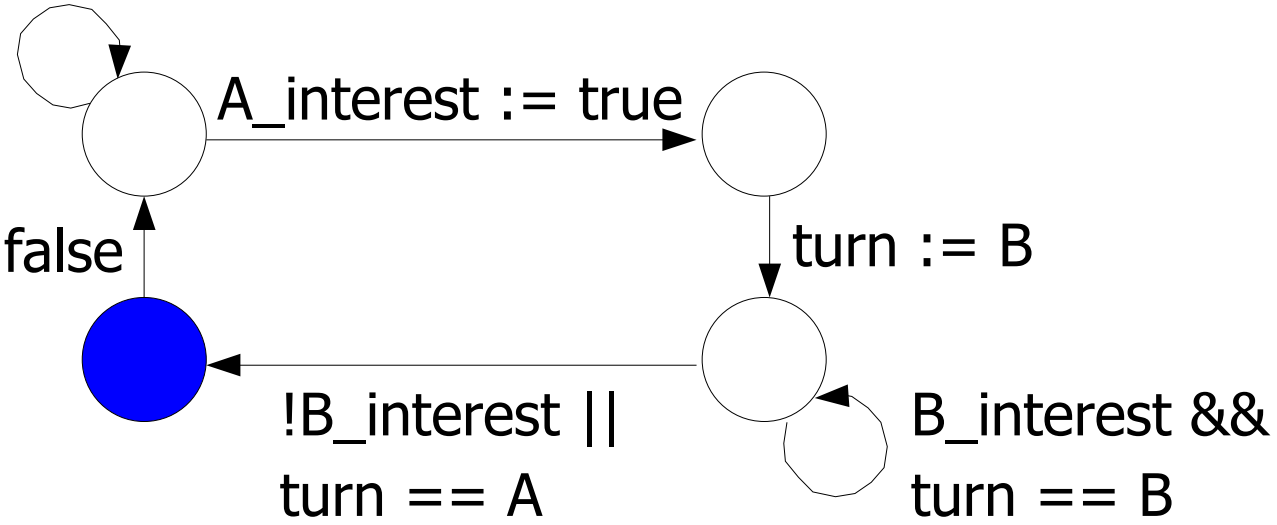
```
process A:
  A_interest = true;
  turn = B;
  while(B_interest && turn != A){}
      // critical section
  A_interest = false;
```

```
process B:
  B_interest = true;
  turn = A;
  while(A_interest && turn != B){}
      // critical section
  A_interest = false;
```

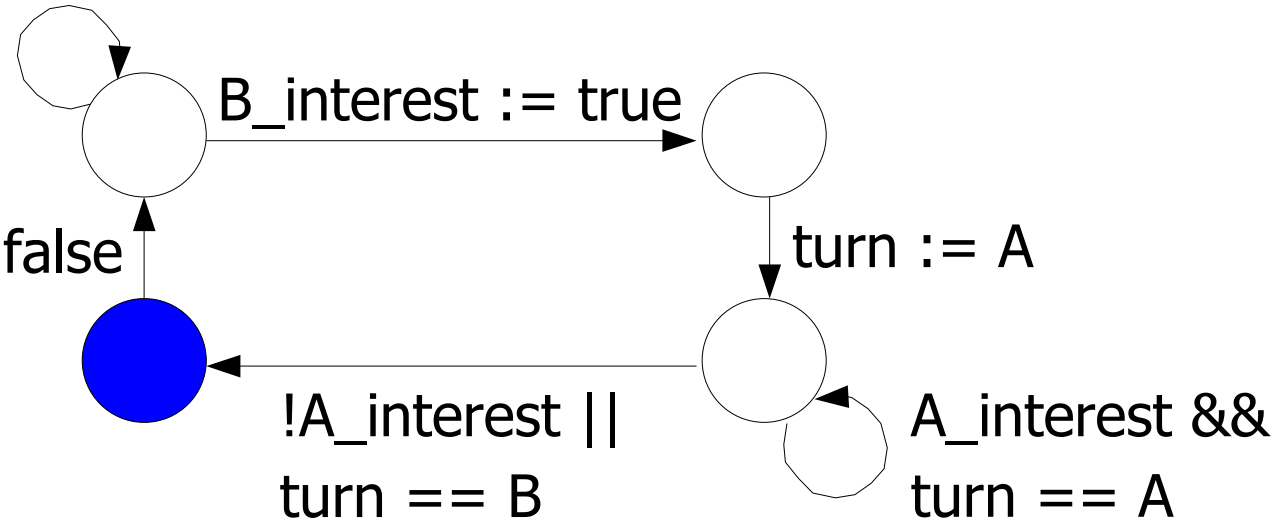
疑似コードは検証には不向き
→ オートマトンモデルで表現

アルゴリズムのオートマトン表現

プロセス A:



プロセス B:



オートマトン表現の問題

- 仕様記述が曖昧
  - 時刻に関する記述が明示的でない
    - 2 つのプロセスが「同時に」turn を書き換えない
    - critical section に入る「際に」turn の値は不変
    - など
- Coalgebraic specification で解決
  - CPU 時刻単位ですべての状態を明示的に扱う

時刻の扱い

- DiscreteTime を表現 (Figure 7)
  - tick で 1 単位後の状態を返す
  - n 単位時間後の状態を $tick^n$ で表現することにする

$$tick^0(x) = x \quad \text{and} \quad tick^{n+1}(x) = tick(tick^n(x))$$

Discrete Time の応用例

- Timer(Figure 8)
  - set してから N 単位時間後に status が off になる
    - N はパラメタとして与えられる
  - Timer は DiscreteTime のメンバを継承
    - polynomial functor の包含関係として表面化

$$T(X) = DT(X) \times X \times \{on, off\} = X \times X \times \{on, off\}$$

  - X のモデルとして自然数を採用可能

$$tick(x) = \begin{cases} 0 & \text{if } x = 0 \\ x - 1 & \text{otherwise} \end{cases}$$

$$set(x) = N$$

$$status(x) = \begin{cases} off & \text{if } x = 0 \\ on & \text{otherwise} \end{cases}$$

Temporal Logic of Actions からの変換

- TLA: 条件を action として記述する temporal logic
  - action: 事前条件と事後条件を and で繋いだ predicate

$$(a(x) > 0) \land (a'(x) = a(x) - 1)$$

    - a'(x) は a(x) の 1 単位時間後の値
- action の変換法
  - 「1 単位時間後」→ tick で表現
  - □ と◇→ tick と量化子で表現

$$\Box(P)(x) \Leftrightarrow \forall n.\ P(tick^n(x))$$
$$\Diamond(P)(x) \Leftrightarrow \exists n.\ P(tick^n(x))$$

    - $\{y \in X \mid \forall n.\ P(tick^n(y))\}$ が DiscreteTime について P に含まれる greatest invariant

Continuous Time の表現

- tick の拡張
  - 経過時間を第 2 引数にとる

    $ticks(x, 0) = x \quad and \quad ticks(x, n+m) = ticks(ticks(x, m), n)$
  - さらに実数の経過時間をとれるようにする (Figure 9)

    $flow: X \times R_{\geqq 0} \rightarrow X$

    $flow(x, 0) = x \quad and \quad ticks(x, s+t) = ticks(ticks(x, t), s)$

- 以降、Continuous Time は扱わない

Class-Valued Methods

- クラスのオブジェクトを返す関数を考える

  tree: X → BinaryTreeSpace[N]

  - どのようなモデルを返すべきか？
- terminal(or final) model を用いるとよい
  - 任意のモデル M に対してユニークな homomorphism M → F が定まる
  - 親クラスを表す変数への cast が必要なときに役立つ
    - これ以上は beyond scope

Peterson アルゴリズムの仕様記述

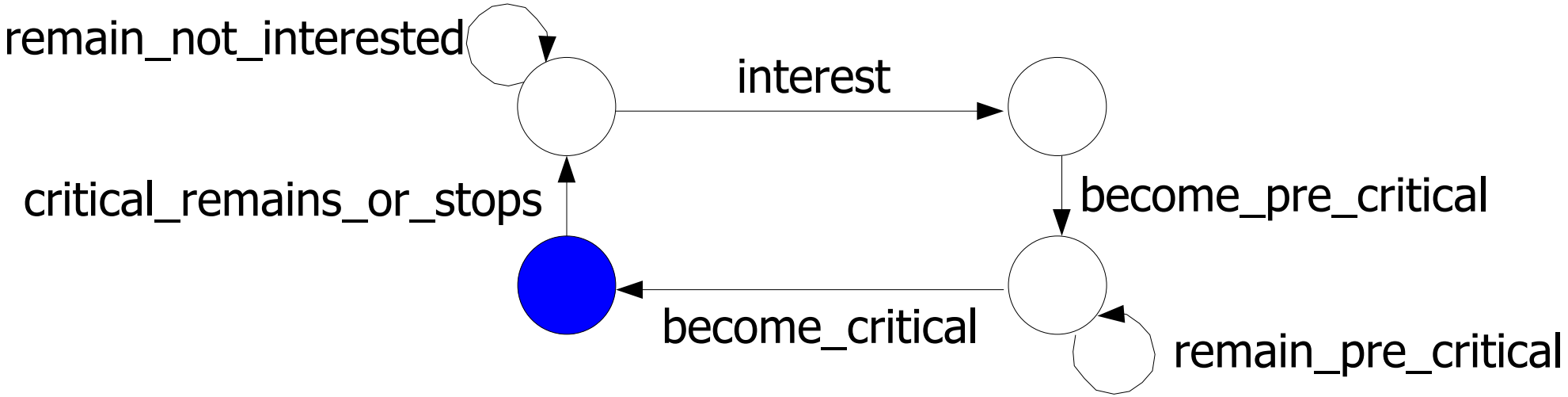
- 複数の specification を組み合わせる (Figure 10)
  - Peterson が 2 つの PetersonProcess を協調させる
  - PetersonProcess は turn フラグの扱いについて Parametrize される

PetersonProcess の Methods

- フラグの内容取得
  - own_interest/other_interest:　自他の interest フラグ
  - turn: turn フラグ
- フラグの更新
  - interest:　自分の interest フラグを立てる
- 現在の状態を取得
  - pre_critical: critical section 直前のループにいる状態
  - critical: critical section にいる状態

PetersonProcess の Assertions

- 状態遷移に関する制約を設定
  - 共有変数 turn に対するアクセス競合を慎重に回避



- critical_stops: 必ず critical section から抜ける
  - ◇(super): 親クラス (DiscreteTime) について◇を適用
    - tick についての◇

Peterson の Methods/Assertions

- Methods
  - 各プロセスの実体を取得
    - ppT, ppF
  - 各プロセスの interest フラグを立てる
    - interestT, interestF
- Assertions
  - 各フラグの値をプロセス間で協調
    - share_turn, exchange_interest
  - 各 methods の性質を bisimilarity などで規定
    - interestT, interestF, synchronise

証明のための下準備

- いくつかの predicates を導入
  - critical_exclusion など (別紙参照)
- Lemma 11
  - critical_exclusion は Peterson の invariant
  - 証明 :

$$\forall x \in X,\ \text{critical\_exclusion}(x) \Rightarrow \begin{cases} \text{critical\_exclusion}(\text{tick}(x)) \\ \text{critical\_exclusion}(\text{interestT}(x)) \\ \text{critical\_exclusion}(\text{interestF}(x)) \end{cases}$$

    を示せばよい。□
    - 簡単だが場合分けが非常に面倒→証明器を使うとよい

相互排他性の証明

- Theorem 12.1
  - すべての到達可能な状態において相互排他が成立：

    $\Box(\{\ y \in X \mid \neg(\mathrm{critical}(\mathrm{ppT}(y)) \wedge \mathrm{critical}(\mathrm{ppF}(y)))\ \})(\mathrm{new})$

  - 証明：

    $$\begin{cases} Q(\mathrm{new}) \\ Q(y) \Rightarrow \neg(\mathrm{critical}(\mathrm{ppT}(y)) \wedge \mathrm{critical}(\mathrm{ppF}(y))) \end{cases}$$

    をみたす invariant Q が存在することを示せばよい。
    Lemma 11 より Q=critical_exclusion ととれる。□

公平性の証明

- Theorem 12.2
  - critical section への興味はいつか必ず満たされる

    $\Diamond(\{ y \in X \mid critical(ppT(y)) \})(interestT(x))$

  - 証明：

$own\_interest(ppT(x)) \wedge pre\_critical(ppT(x)) \wedge \neg critical(ppT(x))$
$\Rightarrow \exists n \in N.\ \neg own\_interest(tick^n(ppF(y))) \vee turn(tick^n(ppT(x)))$

が成り立つ。

[1] 相手が pre_critical なら、turn は自分にある
[2] 相手が critical なら、critical_stops により相手の interest はいつか消滅する

あとは become_critical により題意が導かれる。□

仕様記述の抽象度

- 初期の仕様は曖昧
  - アルゴリズムの概要のみ決定
  - 実装の詳細は後で詰める
- 作業の進み具合によって仕様の抽象度が変化
  - 段階的な仕様策定 / 検証が可能だとうれしい
    - refinement の概念を導入

各種記法の説明

- 記号を整理した表

	Coalgebraic Specification	Polynomial Functor	Coalgebra
具体的なモデル	*C*	*IC*	c:X→*IC*(X)
抽象的なモデル	*A*	*IA*	c':X→*IA*(X)

- Predicates の導入
  - *C*-Create(c): *C* の creation 条件を∧で連結したもの
  - *C*-Assert(c): *C* の assertions を∧で連結したもの
    - *A* についても同様に *A*-Create(c') 、*A*-Assert(c') を定義

Simple Refinement

- 「A の C による simple refinement 」を定義
  - 両者の間に map を構成

    $(c:X \to IC(X),\ new \in X) \to (c':X \to IA(X),\ new' \in X)$
  - ただしこの map は以下の条件を充足

    $(C\text{-Create}(c)(new) \wedge \forall x \in X.\ C\text{-Assert}(c)(x))$
    $\Rightarrow (A\text{-Create}(c')(new') \wedge \forall x \in X.\ A\text{-Assert}(c')(x))$
- 具体から抽象への straightforward な帰着
  - 実際には straightforward にはいかない場合が多い
    - 任意の $x \in X$ について　A-Assert(c')(x)
      を証明することができない

(Non-Simple) Refinement

- 改善策：invariant の考え方を利用
  - 状態空間を X の部分集合 P に制限して証明

    $\forall x \in P \subseteq X.\ A\text{-Assert}(c')(x)$
  - さらに、new' から c' で到達可能な全状態は P に含まれることを示す
    - P(new') が成立
    - P が c' についての invariant
- つまり、以下の条件を満たすような map を構成

  $(C\text{-Create}(c)(new) \wedge \forall x \in X.\ C\text{-Assert}(c)(x))$
  $\Rightarrow (A\text{-Create}(c')(new') \wedge \Box(c')(A\text{-Assert}(c'))(new'))$

Refinement の例

- Proposition 13
  - Peterson は MutualExclusion(Figure 14) の refinement
    - MutualExclusion は相互排他の本質を簡潔に表現したもの
  - 証明：

    (c=(tick, ppT, ppF, interestT, interestF), new)

    → (c'=(tick, criticalT, criticalF, interestT, interestF), new)

    ただし $\begin{cases} \text{criticalT}(x) = \text{critical}(\text{ppT}(x)) \\ \text{criticalF}(x) = \text{critical}(\text{ppF}(x)) \end{cases}$

    のように map を構成。これは条件を満たす。□
    - *MutualExclusion*-Create(c')(new) は明らかに成立
    - □(c')(*MutualExclusion*-Assert(c'))(new) を示すための invariant として critical_exclusion(Lemma 11) を利用可能

まとめ

- Coalgebra を用いた仕様記述の例
  - 相互排他性、公平性の証明
  - 仕様の段階的な refinement