2023年度
ソフトウェア工学
ソフトウェアテスト (続き)

2023年12月25日

渥美 紀寿 (情報環境機構)

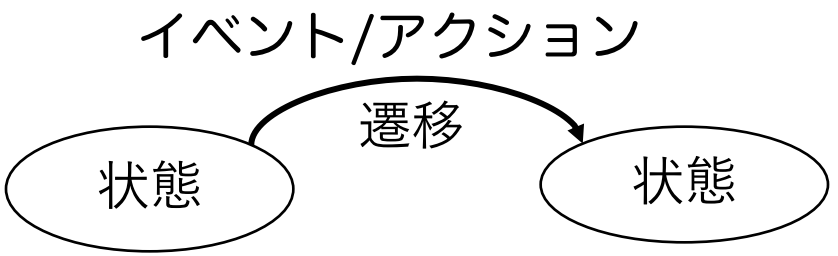
京都大学

参考文献

- 鯵坂 恒夫著: ソフトウェア工学入門, サイエンス社
- 岸 知二，野田 夏子著：ソフトウェア工学, 近代科学者

状態遷移モデルに基づく技法

- システムの状態遷移仕様に基づく
  - 状態遷移図
  - 状態遷移表 (状態とイベント，アクションの組合わせ表)
- 状態遷移仕様
  - 状態
  - 遷移
  - イベント
  - アクション



現在の状態	イベント	アクション	次の状態
S1	E1	A1	S2
S2	E2	A2	S1
…	…	…	…

状態遷移テストの網羅性

- すべての状態を少なくとも1回訪れる
- すべてのイベントが少なくとも1回起動される
- すべてのパスが少なくとも1回実行される
- すべての遷移が少なくとも1回行われる

状態遷移テストの適用範囲と制約

- 単体テスト，結合テスト，システムテストのどのテスト粒度でも利用可能
- システム要求に状態遷移が仕様化されている場合に有効
- システム外部からのイベントにシステムが応答する必要のない場合には適用できない

オブジェクト指向プログラムのテスト

- コンストラクタ・デスクトラスタによるオブジェクトの生成・廃棄
- 継承によるオーバライドやスーパクラスの参照
- クラスフィールドやパブリックなデータフィールドの定義・参照関係

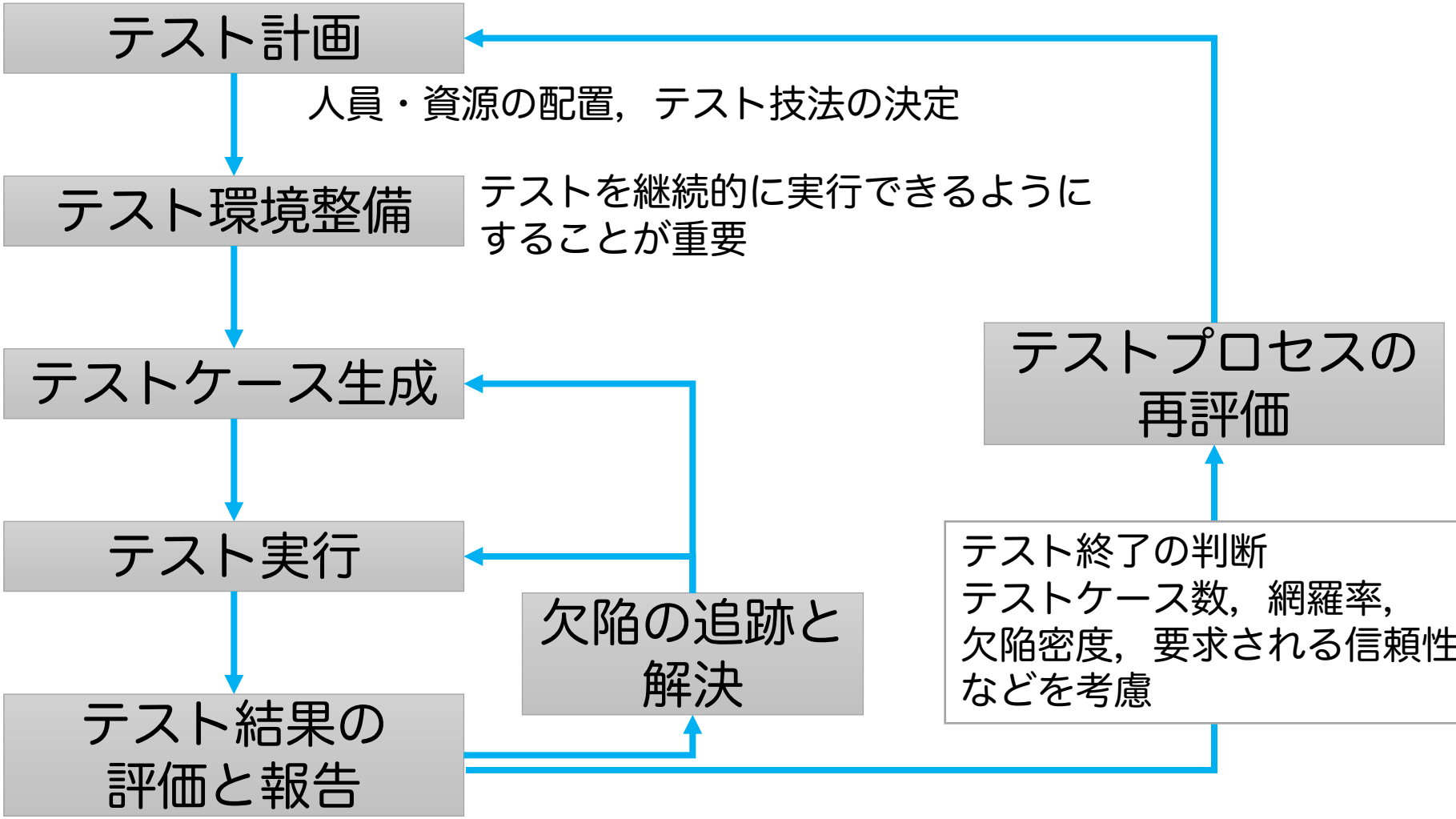
Webシステムのテスト

- ページとその遷移
- セッション
- 動的にページ (URLを含む) が生成される
- クライアントサイドスクリプトがサーバと独立して動作

その他のテスト

- GUIのテスト
  - 目視が重要な要素
- 並行処理，リアルタイム処理
  - 実行のタイミング
- 信頼性保証
  - 多数のランダムに生成されたデータの投入
  - 実際の運用に近いデータ生起確率に基づく統計的テスト

テストのプロセス


テスト計画
人員・資源の配置，テスト技法の決定
テスト環境整備
テストを継続的に実行できるようにすることが重要
テストケース生成
テスト実行
テスト結果の評価と報告
欠陥の追跡と解決
テストプロセスの再評価
テスト終了の判断
テストケース数，網羅率，欠陥密度，要求される信頼性などを考慮

まとめ

- ソフトウェアの正しさを確認するための手法
- 様々なテストの種類
- テストの網羅性基準
  - 制御フローに基づく基準
  - 命令網羅，分岐網羅，パス網羅
- テストデータの決め方
  - 同値分割，境界値分析など

課題3

整数の四則演算をする計算機アプリをオブジェクト指向言語で作る

1. どのようにモジュール化すると良いか，理由とともに説明しなさい
2. モジュール間の関係を図示し，個々のモジュールの役割とモジュール間の関係について説明しなさい

締切：2024/01/15 15:00

2023年度
ソフトウェア工学
形式手法

2023年12月25日

渥美 紀寿 (情報環境機構)

京都大学

静的検査

- レビュー
  - 仕様 (要求・設計)やプログラムを読み直すことで検査
  - 顧客の要求(特に非機能要求)の充足を確認することは困難
- 静的解析
  - 型検査
  - 制御フロー解析，データフロー解析
  - コードクローン解析
- 形式手法
  - 仕様を形式言語で記述し，検査したい性質を形式的に表明
  - 自動で網羅的な検証が可能

ソフトウェア検査の目的

- ソフトウェアの「正しさ」を確認すること
- 「正しさ」を妨げる要因があれば，その箇所と原因を指摘

ソフトウェアの「正しさ」とは

- ソフトウェアが求められる機能を実行できる
  - 要求される目的・使命を果たすことができる
- ソフトウェアが求められる品質を満たす
  - 実行の性能や精度
  - ソフトウェアの信頼性

形式手法

- 数学や論理学における形式性を備えた記述形式を用いる
- モデル記述に厳密性を与える
- 使用する論理体系
  - 命題論理
    - 命題変数を論理演算子 (¬，∧，　∨，→，　⇔) で関係
  - 述語論理
    - 論理式に含まれる変数を量化 (∀，∃)
  - 時相論理
    - 時間経過に伴なって変化する論理を表現

形式手法の目的

- ソフトウェア開発において定義される対象を厳密に記述する
  - 要求仕様，設計，プログラム
- 何らかの論理体系に基づいてその正しさについて議論する

対象の厳密な定義

- 自然言語や図による表現は曖昧性や記述漏れが入りやすい
  - 自然言語や図表現は人間が読んで理解しやすいが誤解を招くこともある
  - 形式表現は誰でも同じ意味を共有できる
- 自然言語や図による表現の正しさの検査が難しい
  - 何が正しいかを厳密に定義できない
  - 形式表現では満たすべき性質を厳密に表現可能

形式手法による不具合の検査

- 論理体系に基づいて対象を表現し，満たすべき性質を記述することで，機械的に確認可能
  - 開発工程の上流で形式手法を適用することで，不具合の早期発見が可能
- 対象が不具合を含んでいない，正しいものであることを確認可能
  - 求められる性質を列挙し，それらすべてを満たしていることを論理体系で証明

不具合がないことを常に証明できるとは限らない

形式検証

- 論理体系に基づいた表現方法で仕様が記述されるとその論理体系の中でその正しさについて議論可能
- 定理証明
  - 数学の証明と同様に推論規則を適用して性質が成立ことを証明する技術
  - 自動定理証明ツールを利用可能
  - 複雑な証明は自動証明できない
- モデル検査
  - 形式記述から導出される状態空間を全数探索することで，性質を確認
  - モデル検査ツールを利用可能
  - 状態爆発を起こす場合には検査できない

機能の形式化 (入力条件と出力条件)

- 対象システムの機能を入力時に成立すべき条件と出力時に成立すべき条件のペアで表わす
- 条件の記述は述語論理などの論理的な表現が用いられる

例：正の整数 x と y の最大公約数 z を求める
入力条件: integer(x) ∩ integer(y) ∩ x > 0 ∩ y > 0
出力条件: integer(z) ∩ divide(z, x) ∩ divide(z, y)
∩ ∀w. (integer(w)
∩ (divide(w,x) ∩ divide(w, y) ⊃ z ≧ w))
integer(x): x が整数であることを意味する述語
divide(a, b): b が a によって割り切れることを意味する述語

機能の形式化 (関数として機能を捉える)

- 機能を，入力を変換して出力を返す関数として捉える
- 関数の定義はその関数が満たすべき性質を等式などの公理として与える

例：最大公約数を与える関数 gcd(x, y)
[公理]
$\gcd(x, y) = \gcd(x, y \bmod x) = \gcd(x \bmod y, y)$
$\gcd(x, y) = \gcd(y, x)$
$\gcd(x, 0) = x$

データの形式化 (抽象データ型)

- 構造的プログラミングの中心概念
- データ構造を，それに対する演算の組により定義する
- 演算には，データ生成のための演算と参照のための演算がある
- 1つの型に属するデータには，外部に公開されているいくつかの演算を通してのみアクセス可能
- 外部に公開する演算の仕様と内部の実装を分離し，内部構造を隠す
- データ型の実装は，データ型間の写像として定式化

データの形式化 (代数的仕様記述)

- 演算の意味を与える方法

例：スタックの代数的仕様

```
module STACK {
  [Stack Elt]
  signature {
    op nilstack : -> Stack
    op push : Elt Stack -> Stack
    op empty : Stack -> Bool
    op pop : Stack -> Stack
    op top : Stack -> Elt
  }
  axioms {
    var s : Stack
    var v : Elt
    eq empty(nilstack) = true .
    eq empty(push(v, s)) = false .
    eq pop(push(v, s)) = s .
    eq top(push(v, s)) = v .
  }
}
```

ホーア論理

- プログラムについて論じるための形式的理論
  - 2種類の表明 (assertions) をプログラムに付加
    - 実行前に成り立つ事前条件 (precondition)
    - 実行後に成り立つ事後条件 (postcondition)
- 様々な仕様技術言語や検証・解析ツールなどの基礎
- Tony Hoare により提案 (1969)
  - Robert Floyd のフローチャートに関する議論を発展させており，Floyd-Hoare Logic とも呼ばれる

命題論理

- 命題：真偽が決まる言明
- 命題論理：命題に対する正しい推論の形式
- 原子命題：要素的な命題
- 論理式：原子命題あるいは論理式を論理記号 (¬，∧，　∨，　→，　⇔) で結んだ新たな命題

真理値表

A	B	A∧B	A∨B	A→B	A⇔B
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

述語論理

- 述語：対象の性質に関する言明
- 述語論理：述語に対する正しい推論形式
- 量化記号

  全称記号 $\forall x\ P(x)$：

  すべての x についてP(x) が真

  存在記号 $\exists x\ P(x)$：

  P(x) を真にする x が存在

表明付きプログラム

- 実行前に成り立つべき条件 (命題) と，
  プログラム実行後に成り立つべき条件 (命題)
  を明記する
  - これを基に正当性 (正しさ) を議論
- 実行が完了するかどうかも考える
  - 無限ループしないことを保証

表明付きプログラムと正当性

- {A}P{B}：部分正当性 (partial correctness)
  - プログラム P を事前条件 A が成り立つ状態で実行したとき，停止するならば，事後条件 B が成り立つ
- <A>P<B>：完全正当性 (total correctness)
  - プログラム P を事前条件 A が成り立つ状態で実行したとき，停止し，かつ事後条件 B が成り立つ

表明記述のための語彙

- プログラムを記述する語彙と異なっても良い
- 表明の方が強力な語彙を用いることが多い
  - プログラム内のブール式では，評価・実行が単純でない
  - 述語論理の限量子 (∀，∃) は一般に用いないが，表明では必要・有用
- 仕様を表現するため，表明にのみ read-only で現れる変数 (仕様変数) が必要となる
  - swap 処理の場合，実行前の値を仕様変数として表現しておかないと事後条件を表現できない

表明付きプログラムにおける証明

- 公理
  - 基本的な表明付きプログラムについては真と判断する
- 推論規則
  - 1つまたは複数の表明付きプログラムが真と判断されたとき，別の表明付きプログラムを真と判断する方法を決める

スキップ文の公理

{A} skip {A}

代入文の公理

$\{A[t/x]\}\ x = t\ \{A\}$

x を t に変えたら A が成り立つという状況において x に t を代入すると，A が成り立つようになる

$\{a > 0 \land b > 0\}$
$x = a;$
$\{x > 0 \land b > 0\}$

複合文の推論規則

$$\frac{\{A\}P1\{S1\} \quad \{S1\}P2\{S2\} \cdots \quad \{Sn\text{-}1\}Pn\{B\}}{\{A\}\ P1;\ P2;\ \cdots\ ;\ Pn\ \{B\}}$$

「ある文の事後条件を次の文の事前条件にしたときに…」という議論を繰り返し，順次つないでいけば複数の文全体に対する正当性を示すことができる

条件文の推論規則

$$\frac{\{C \land A\}\ P\ \{B\} \qquad \{\neg C \land A\}\ Q\ \{B\}}{\{A\}\ \text{if } C \text{ then } P \text{ else } Q \text{ fi } \{B\}}$$

分岐いずれに進んだときも正当性が示せれば,
if 文全体についての正当性を示すことができる

$$\frac{\{x < 0 \land \text{int}(x) \land \text{int}(y)\}\ y = -1 * x\ \{x^2 = y^2 \land y > 0\} \qquad \{x >= 0 \land \text{int}(x) \land \text{int}(y)\}\ y = x\ \{x^2 = y^2 \land y > 0\}}{\begin{array}{c}\{\text{int}(x) \land \text{int}(y)\} \\ \text{if } (x < 0) \text{ then } y = -1 * x; \text{ else } y = x; \text{ fi} \\ \{x^2 = y^2 \land y > 0\}\end{array}}$$

while文の推論規則

$$\frac{\{C \wedge A\}\ P\ \{A\}}{\{A\}\ \text{while } C \text{ do } P \text{ od } \{\neg C \wedge A\}}$$

- ループ内部を実行するときには C が成り立つが，このとき A が成り立つならば，その後にも A が引き続き成り立つことを示す (帰納法の一部)
- これにより，ループ回数によらず，while 文の実行前後で A が維持されることが示せる．また，実行後にはループを抜けているので ¬C である
- この A のことをループ不変条件という

帰結規則

$$\frac{\{B\}\ P\ \{C\}}{\{A\}\ P\ \{D\}}$$ ただし，$A \rightarrow B$，$C \rightarrow D$

- ある表明付きプログラムが真なら，その事前条件をより強く，または同値に変換したものも，真である
- ある表明付きプログラムが真なら，その事後条件をより弱く，または同値に変換したものも真である

証明の方針

- 帰結規則以外は，用いる公理・推論規則がプログラムの構造から一意に決まる
  - プログラムの構文木の構造通りに証明木の基本構造が決まる
- 表明内の具体的な論理式は，目的に合わせて適切なものを定める必要がある
  - 特に while 文におけるループ不変条件
- 帰結規則においては，表明の語彙に応じた公理・定理，推論規則を利用することになる
  - 整数に関するもの，命題論理・述語論理

表明付きプログラムの証明例

a を 5 で割った商を求める下記プログラムを証明せよ

{a >= 0}

q := 0;

while a – 5 * q >= 5 do

    q := q + 1;

od

{5 * q <= a ∧ a < 5 * (q + 1)}

完全正当性に関する証明

- 完全正当性を示すためには，部分正当性と停止性を示せば良い
- 停止しない可能性があるのは while 文
- 停止性を示す方針
  - ループの進み具合を判断する数式を定める
  - その数式の値は常に 0 以上の整数であり，かつループごとに必ず減少することを示す

while文の推論規則: 停止性

$$\frac{\langle measure = n \wedge C \wedge A \rangle\ P\ \langle measure < n \wedge A \rangle}{\langle A \rangle\ \text{while } C \text{ do } P \text{ od } \langle \neg C \wedge A \rangle}$$

- 部分正当性の while 文の推論規則に加え,
  - ループ内部の文 P の実行前において，ある数式 measure の値が n であるとき，実行後には measure の値が n より小さい
  - ただし，数式 measure の計算結果は 0 以上の整数とし，n は C，A，P に現れない新しい仕様変数

完全正当性の証明の例

- measure を a – 5 * q として，仮定内の事前条件，事後条件に加えれば良い

現実のプログラム

- 考え方は様々な形で活用できる
- 公理・推論規則の拡張を行えば，より複雑なプログラムも扱うことが可能
- 限界はある
  - 完全性を示すことができないプログラムは存在する

モデル検査

- 仕様を有限状態モデルで定義
  - クリプキ構造
  - 有限オートマトン
- 検査したい性質を時相論理で表現
- モデル上で性質が成立することを全数探索により自動検証
- 成立しない場合にはその状況を出力

クリプキ構造

- $M = \langle S, S_0, R, L \rangle$
- AP: 原子命題の集合
- S: 状態の有限集合
- S_0: 初期状態の集合 ($S_0 \subseteq S$)
- R: 遷移の集合 ($R \subseteq S \times S$)
- L: ラベル付け関数 ($S \rightarrow 2^{AP}$)
- 遷移
  - 決定的: ある状態から遷移可能な状態が1つのみ
  - 非決定的: ある状態から遷移可能な状態が複数

クリプキ構造の例

センサーで開くゲート

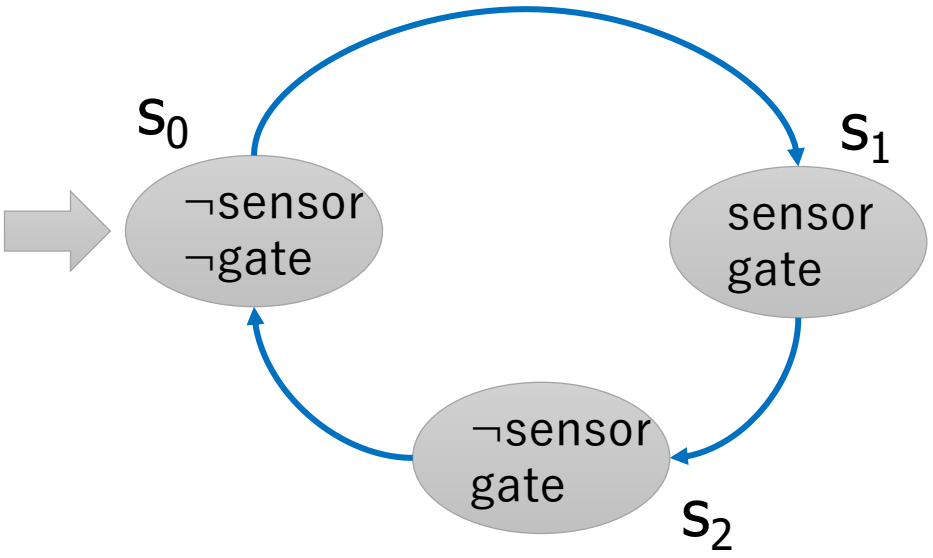
AP = {sensor, gate}

$S = \{s_0, s_1, s_2\}$

$S_0 = \{s_0\}$

$R = \{(s_0, s_1),$
$(s_1, s_2),$
$(s_2, s_0)\}$

$L = \{(s_0, (\neg sensor, \neg gate)),$
$(s_1, (sensor, gate)),$
$(s_2, (\neg sensor, gate))\}$



有限オートマトン

- $M = \langle S, S_0, F, \Sigma, \delta \rangle$
- S: 状態の有限集合
- S_0: 初期状態の集合 ($S_0 \subseteq S$)
- F: 受理状態の集合 ($F \subseteq S$)
- Σ: イベントの有限集合
- δ: 遷移の集合 ($\delta \subseteq S \times \Sigma \times S$)
- 決定性有限オートマトン
  - 遷移先が1つのみ
- 非決定性有限オートマトン
  - 遷移先が複数

有限オートマトンの受理言語

- 遷移列
  - $(s_0, e_0, s_1), (s_1, e_1, s_2), \cdots, (s_{n-1}, e_{n-1}, s_n)$
    $(s_{i-1}, e_{i-1}, s_i) \subseteq \Sigma \ (i=1, \cdots n)$
- 語
  - イベント列 $(e_0, e_1, \cdots, e_{n-1})$
- 受理
  - 実行系列の $s_n \subseteq F$ のとき，語 $e_0, e_1, \cdots, e_{n-1}$ は受理されるという
- 受理言語
  - 受理される語の集合

Büchiオートマトン

- 有限オートマトンは最終的に受理状態に遷移するため，停止する
- 停止することなくイベントを待ち受けるシステムでも扱えるようにする
- 有限オートマトンと同じ5つ組で表現
- 受理の扱いが異なる

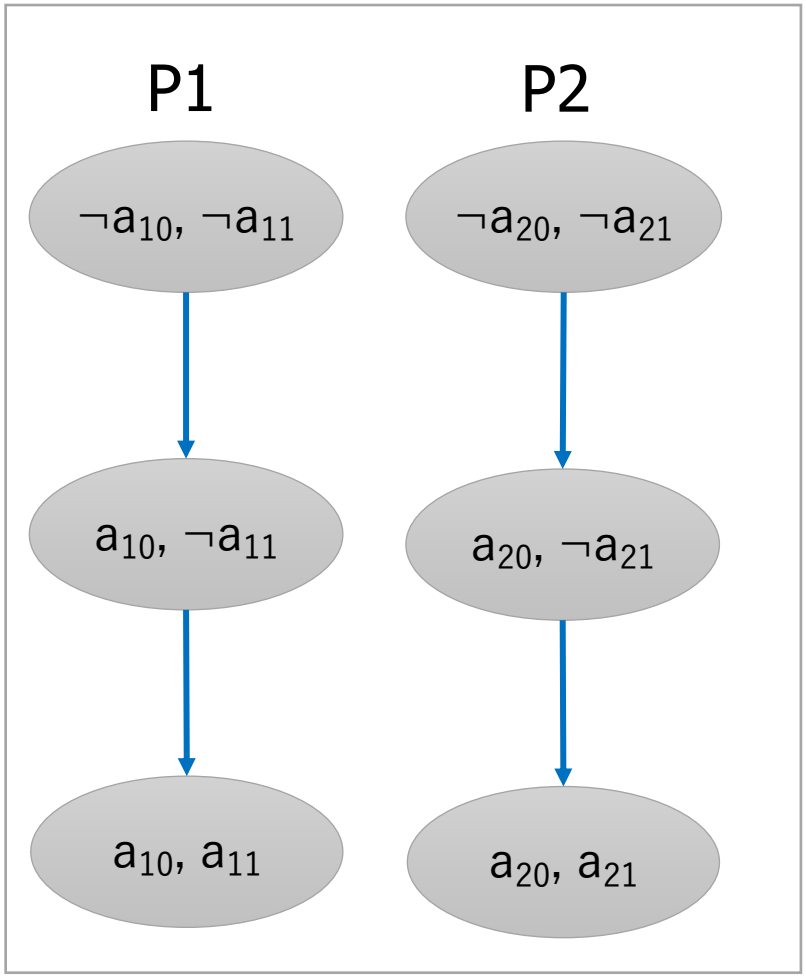
Büchiオートマトンの受理言語

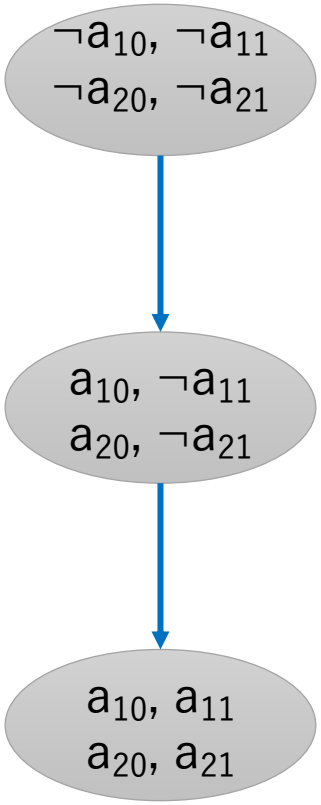
- 遷移列
  - $(s_0, e_0, s_1), (s_1, e_1, s_2), \cdots, (s_{n-1}, e_{n-1}, s_n)$
    $(s_{i-1}, e_{i-1}, s_i) \subseteq \Sigma \ (i=1, \cdots n)$
- 遷移列中に受理状態が無限個含まれるならば受理される

並行処理

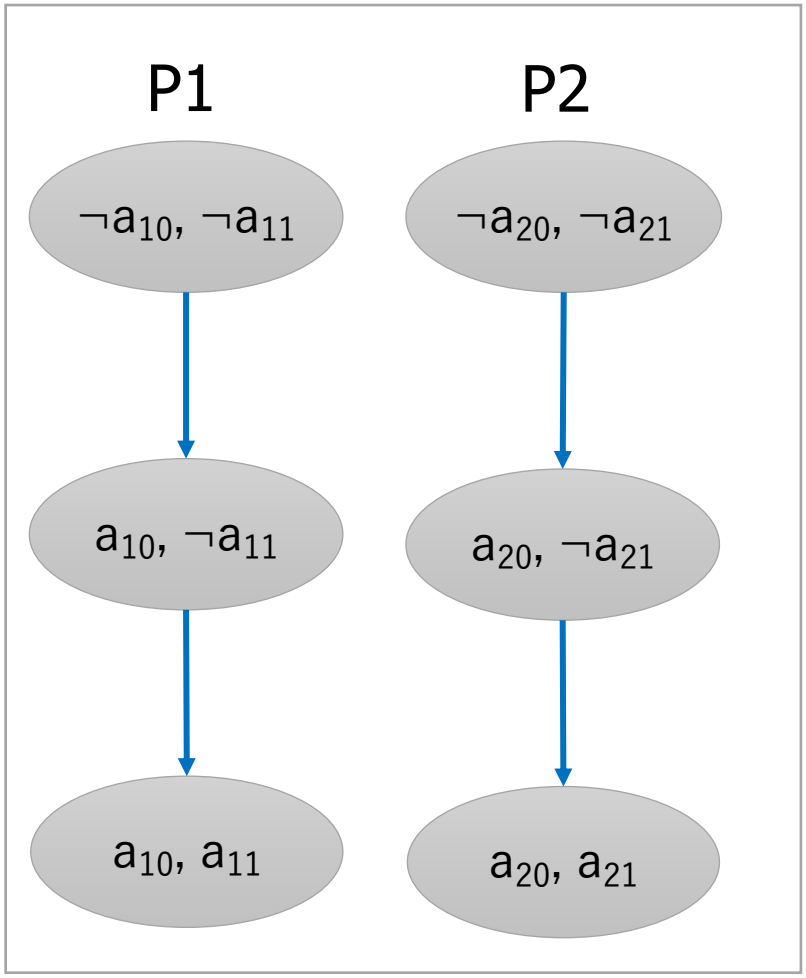
- 複数の処理を同時に実行
- 並行処理の典型的な実行
  - 同期実行
    - 例：遷移は複数の処理で同時に起こる
  - 非同期実行
    - 個々の遷移が任意のタイミングで起こる

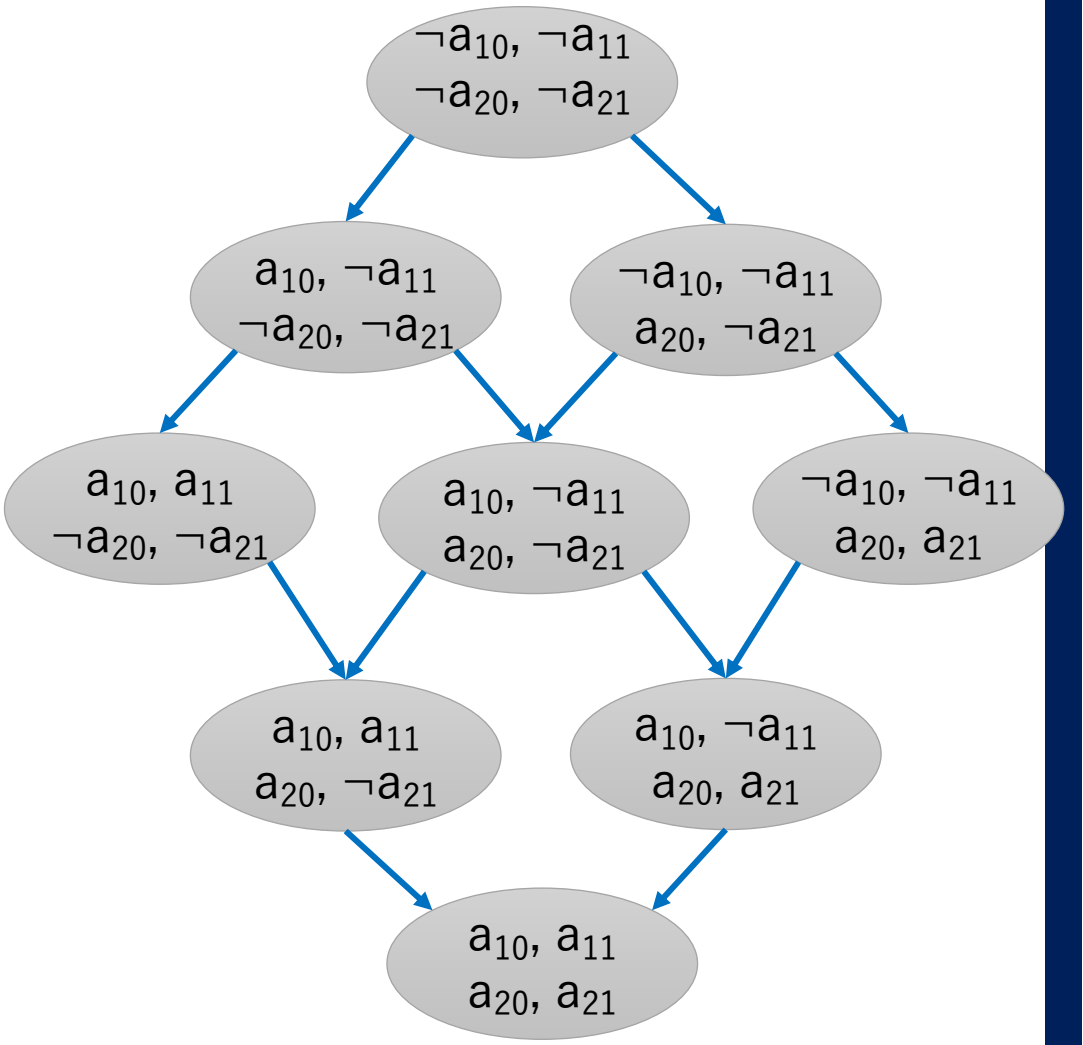
クリプキ構造での同期並行合成



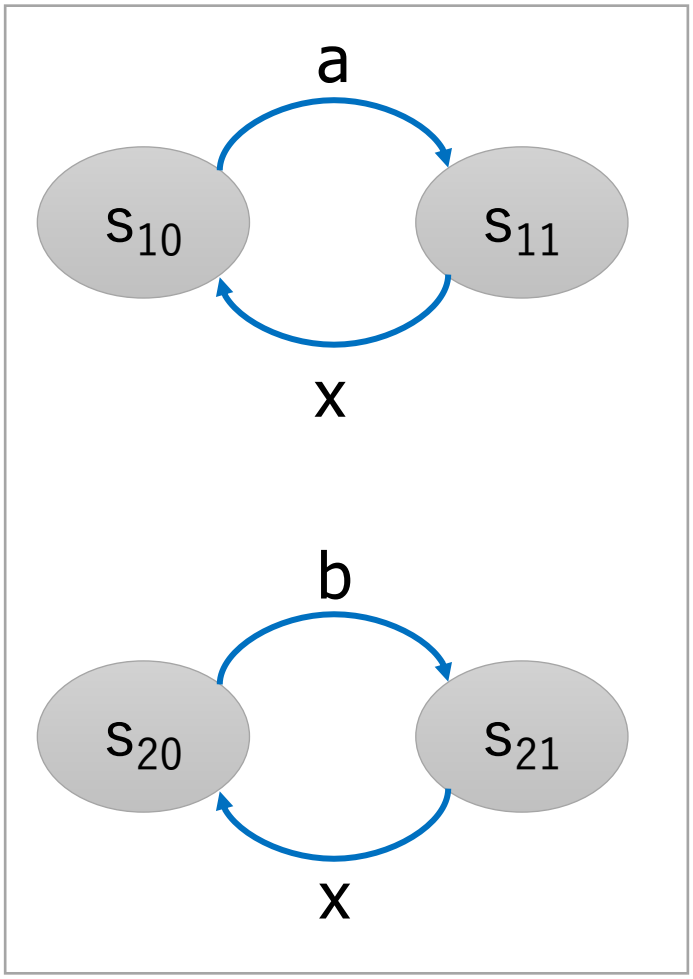


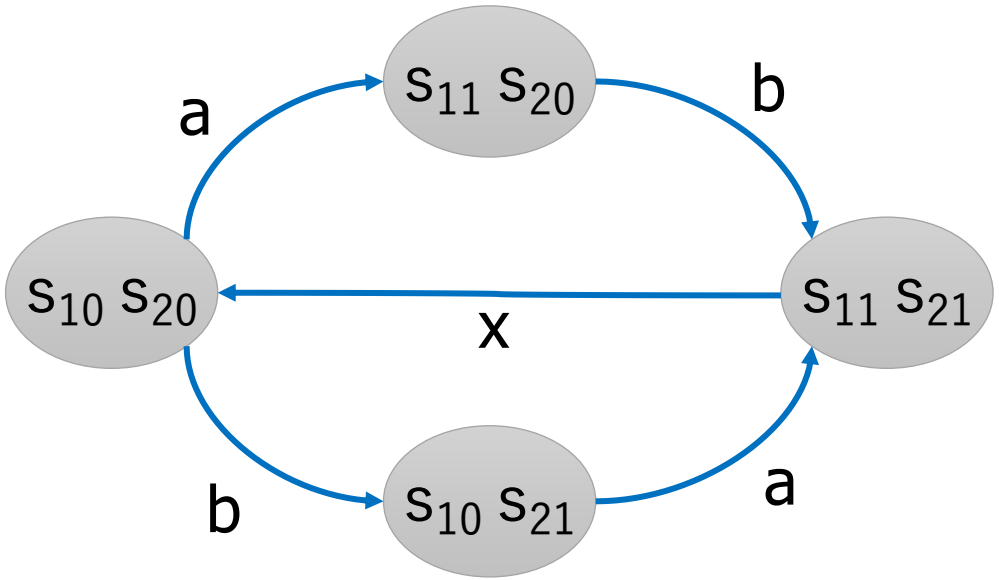
クリプキ構造での非同期並行合成





オートマトンでの同期合成





時相論理

- 命題の真偽が時間に依存する論理
- 線形時相論理 (LTL)
  - ある時点の次の時点が決定的に決まる
- 計算木論理 (CTL, CTL*)
  - ある時点の次の時点が非決定的に決まる

時間に関する論理記号

- M, π |= ψ : クリプキ構造M の実行経路π 上で論理式ψが成り立つ
- π^i : 実行経路 $s_0, s_1, s_2, \cdots$ における s_i から始まる実行経路
- M, π |= Xψ ⇔ M, π^1 |= ψ
- M, π |= Fψ ⇔ M, π^k |= ψとなる k (k≧0) が存在
- M, π |= Gψ ⇔ M, π^i |= ψがすべての i (i≧0) で成立
- M, π |= ψUϕ ⇔ M, π^k |= ϕ となる k (k≧0) が存在し, かつ M, π^i |= ψ がすべての i (k>i≧0) で成立
- M, π |= ψRϕ ⇔ すべての j ≧ 0 に対し, すべてのi < j で M, π^i |≠ ψならば M, π^j |= ϕが成立

経路限量子

- Eψ: ψが成立する実行経路が存在
- Aψ: すべての実行経路でψが成立

モデル検査における性質記述

- 安全性
  - ある状況に決して陥いらない
  - 例：電車通過中に踏み切りが上がらない
- 活性
  - ある条件が成立すれば必ずある状況に至る
  - 例：電車が来たら踏み切りが降りる
- 到達可能性
  - 初期状態から特定の状態に到達しうる
  - 例：電車が来る

状態爆発

- 検査する状態数が大きくなり，現実的な時間やメモリ量で検査が終了しない状況
- モデル検査ツールの進歩やコンピュータの性能向上で扱える状態は増えている
- 検査する状態数は組合わせ的に増えるため，状態爆発は用意に起こる
- 状態爆発への対応
  - 抽象化
  - 部分的探索

まとめ

- 数学や論理学における記述形式を用いることによってソフトウェアのモデル記述に厳密性を与える
- 表明付きプログラム
- モデル検査
- 利用する論理体系
  - 命題論理
  - 述語論理
  - 時相論理など