情報実験I
プログラムの仕様とテスト

第1回
4月19日

予定

- 第1回（4月19日）
  - 実験の目的の説明
  - プログラムの仕様の説明
  - プログラムのテストに関する講義
- 第2回（4月26日）
  - 2人1組になり、テストケースを交換し、自分のプログラムをテストする。
  - 機能拡張のためのリファクタリングについて
  - 拡張要求：色付長方形
  - テストケースの追加
- 第3回（5月10日）
  - 2人1組になり、プログラムを交換し、自分のテストケースで相手のプログラムをテストする。
  - GUIの講義（1）
  - 仕様変更要求：出力の仕様
- 第4回（5月17日）
  - GUIの講義（2）
  - 仕様変更要求：入力の仕様

プログラムのテストとは？

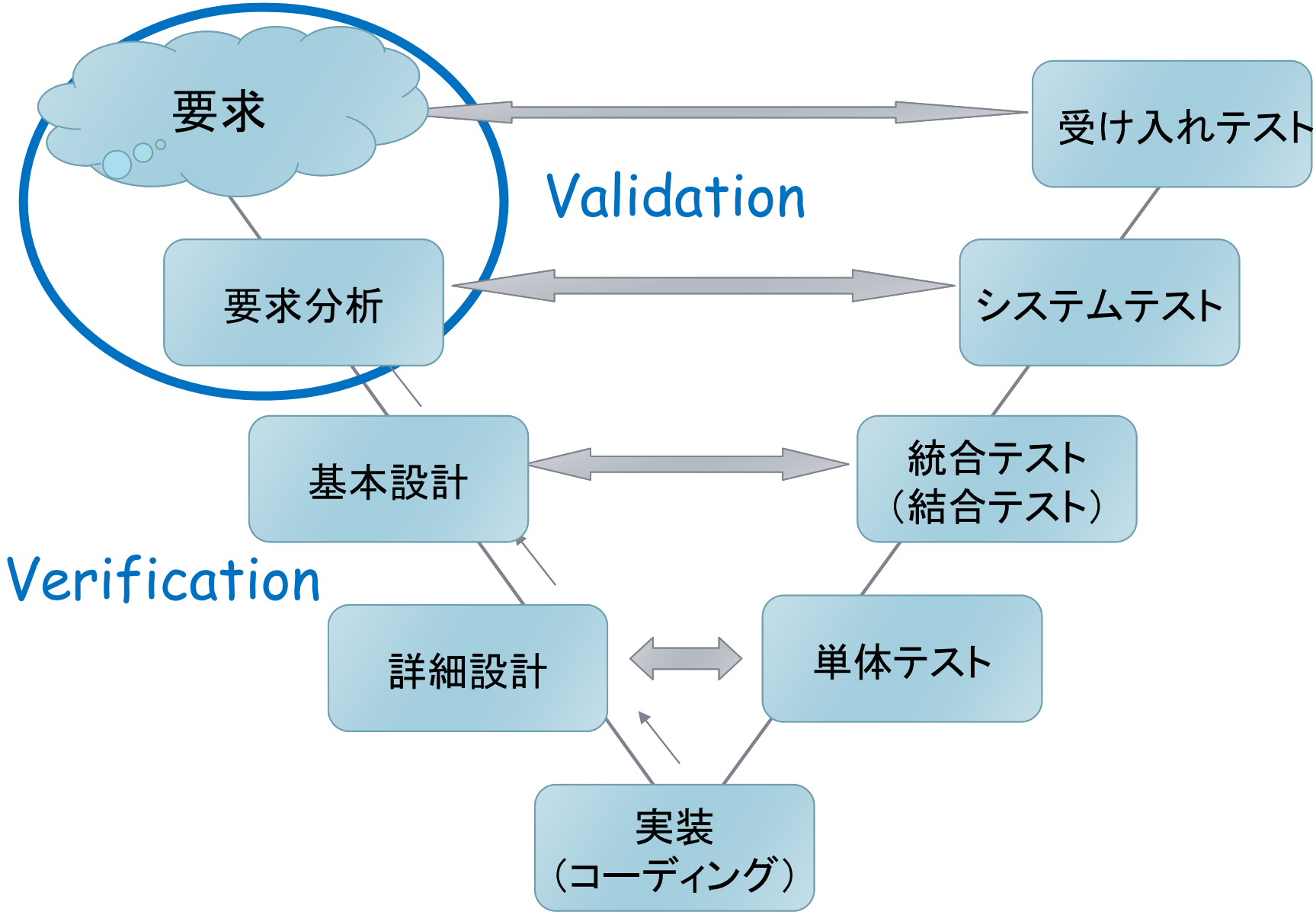
- プログラムが正しいことを確認する。

⇒ ✕

- テストとはエラーをみつけるつもりでプログラムを実行する過程である。
  - 満足できない場所を精力的に検出することが目的

  - プログラムに欠陥がまったくないことを証明することはできない。
  - テストでは欠陥の存在しか示すことはできない。

- **今回は、プログラムに要求されていることは何かを考える！**
  ⇒　要求を満たしているかをテストしながら開発する


要求
受け入れテスト
Validation
要求分析
システムテスト
基本設計
統合テスト
（結合テスト）
Verification
詳細設計
単体テスト
実装
（コーディング）

ブラックボックステスト

- プログラムを1つのブラックボックス（暗箱）とみなす
- テストをする人
  - プログラムの内部構造と動作には関知しない
  - この箱が仕様書とおりの振る舞いをしないことを見つける
- 別名
  - データ依存テスト
  - 入出力依存テスト

ブラックボックステスト

- プログラムの全エラーを見つけるには

⇒ **徹底的な入力テスト**

  - テストケースとしてすべての可能な入力条件を示す
- すべての入力を列挙してテストする？

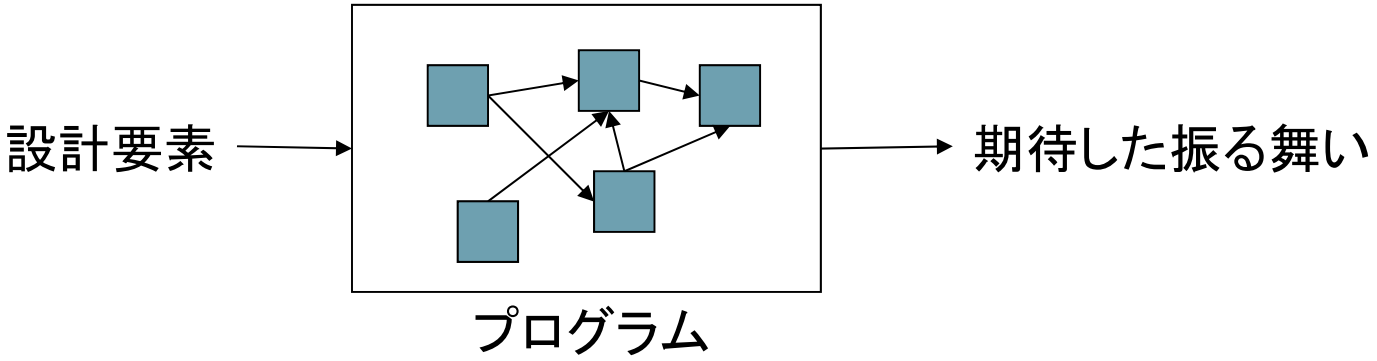
<u>長方形エディタプログラム</u>

- すべての可能な長方形に対する操作のテストケースをつくる必要がある
  - 一体いくつのテストケース？
  - 「可能な」は「正しい」だけではない

ホワイトボックステスト

- プログラムを1つのホワイトボックス（透明な箱）とみなす
- テストをする人
  - プログラムの内部構造の論理を調べて、テストデータを作成する
  - 設計を詳細に分析し、パス・制御を走査するテストを考える

- 別名
  - 論理依存テスト

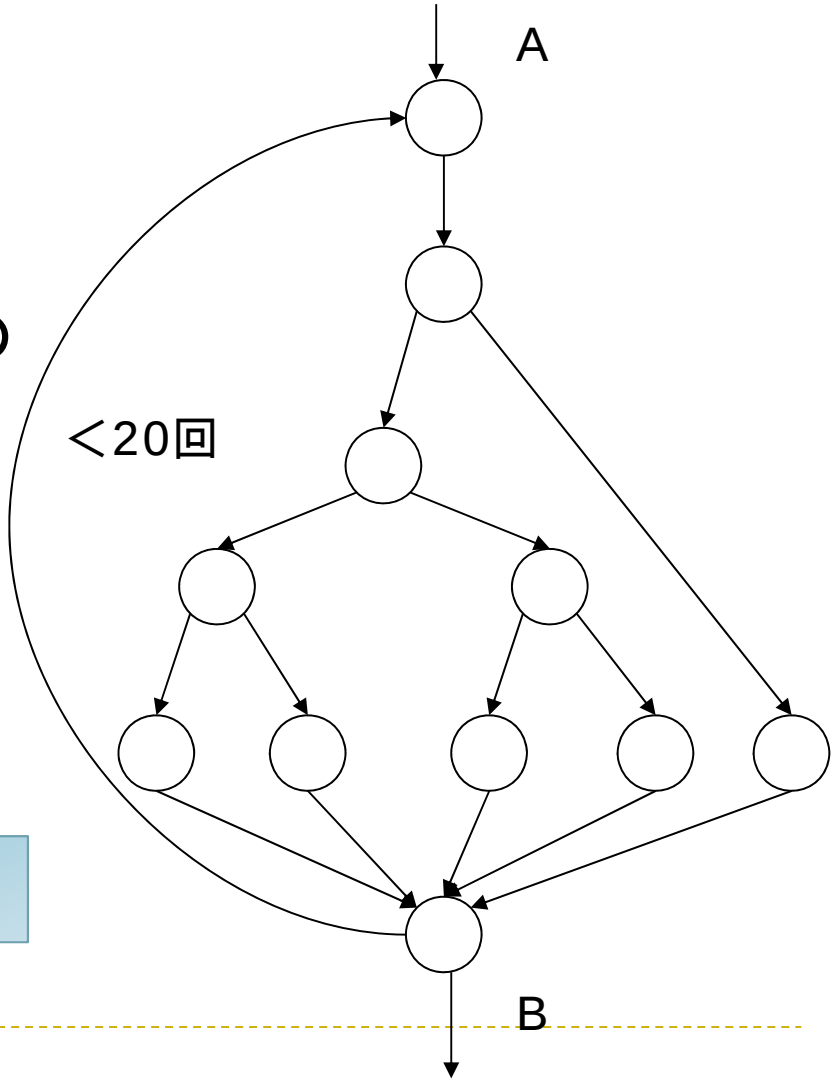


ホワイトボックステスト

- **徹底的経路テスト**
  - プログラムのすべての可能な制御経路を実行する
- プログラムの異なる論理経路の数は？
  - 最高20回繰り返すループと分岐
  - 分岐が独立しているならば

AからBへは

$5^{20}+5^{19}+\cdots+5^{1}$　通りの経路



徹底的経路テストは非現実的！

徹底的経路テストは完全か？

- プログラムのすべての経路がテストできたとしてもプログラムはまだエラーをもっているかもしれない

- 徹底的経路テストではプログラムが仕様と一致することを保証できない
- 「経路を抜かす」という誤りを発見できない
- データに依存するエラーを発見できない
  - 2つの数の差がある規定値より小さいかどうかを判定する

    IF((A-B)＜EPSILON)…

    エラーの発見はAおよびBの値に依存する

テストケースの設計

- **可能なテスト・ケースのどのようなサブセットが最もエラーを発見できる確率が高いか**

- ブラックボックス・テストの手法
  - 同値分類
  - 限界値分析
  - 原因 - 結果グラフ
- ホワイトボックス・テストの手法
  - 命令網羅
  - 判定条件網羅または分岐網羅
  - 条件網羅
  - 判定条件網羅／条件網羅
  - 複数条件網羅

テストケース

- テスト対象　どの項目か？
- 正常ケース？例外ケース？
- テストデータ（どのような場合に何を与える？）
  - テスト開始時の状態
  - 入力データの値
- テスト手順（どのような順序で実行する？）
- 期待されるテスト結果
  （どうなれば成功か？失敗か？）

同値分割(equivalence partitioning)

テストデータの選定方法

- 入力データを同値類に分割し、代表元を選択する
- **可能性のある無限の組み合わせを有限の組み合わせで最もうまく表現する**
  - 同値類を決めると、この集合の1つの要素をテストしてエラーが見つからなければ、同じ集合の他の要素でテストしてもエラーが見つからない。
  - 入力条件から有効な入力の同値類と無効な入力の同値類を選別する。

例

- 入力条件:1 ≤ 項目数 ≤ 99999
- 有効な入力の同値類: 1 ≤ 項目数 ≤ 99999
- 無効な入力の同値類:

項目数 ≤ 0 , 項目数 ≥100000

例

- 入力条件：名前のはじめの1文字はアルファベットでなければならない

この時

- 有効な入力の同値類：
  - はじめの1文字がアルファベットである名前
- 無効な入力の同値類
  - はじめの1文字がアルファベットではない名前

限界値分析

- 限界値分析
  - 同値分割との違いは代表元としてどの要素を選んでも良いのではなく、**同値分割の端とその付近を調べること**
  - **結果**（出力値の同値類）も考える

<u>限界値分析によるテストケースの選び方</u>

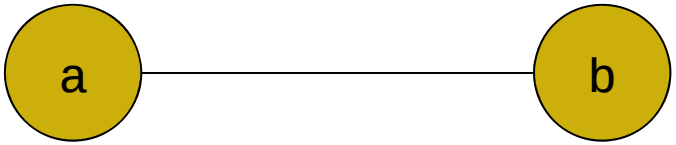
- 入力条件が値の範囲を指定する場合：
  - 領域の両端
  - それらを超えた状態の無効入力
    - 入力値の範囲：-1.0～＋1.0
    - テストケース：-1.0,+1.0,-1.001,+1.001

原因－結果グラフ

- 同値分割・限界値分析の弱点
  - 入力状況の組み合わせを明確にできない
  - 同値分割では結果の観点がない

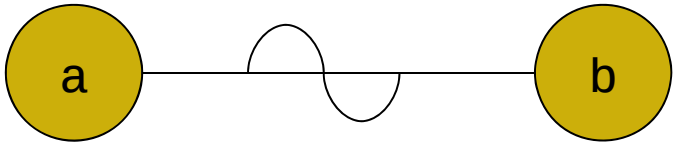
- 入力条件（原因）と対応する所定動作（結果）とこれらの論理関係をグラフで表記したもの
- 自然言語で書かれた仕様が形式言語表現に翻訳されたもの（ブール論理を用いたグラフ）
- 系統的な方法で誤りの発見効果の高いテストケースのセットを選択するのに有効

基本的な原因－結果グラフの記号



IDENTITY（同値）

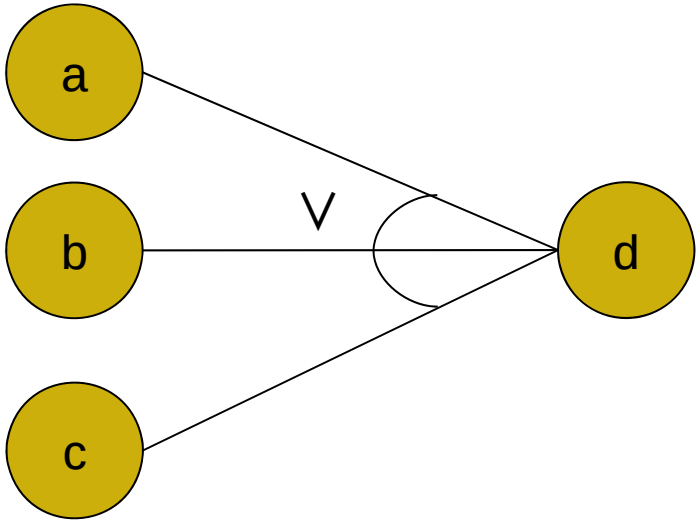
aが1ならばbは1
aが1でないならばbは0



NOT（否定）

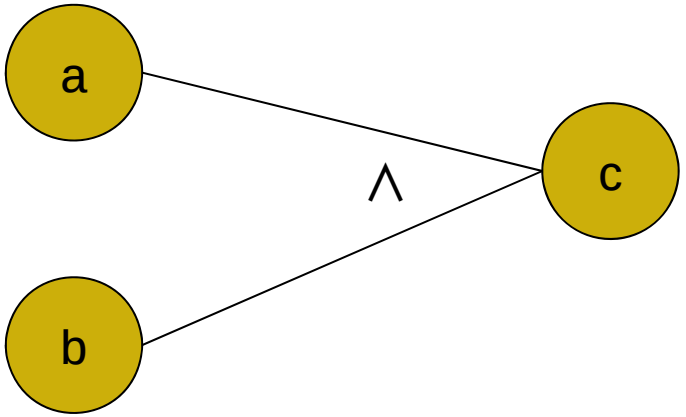
aが1ならばbは0
aが1でないならばbは1

基本的な原因－結果グラフの記号



OR（和）

aが1またはbが1またはcが1ならばdは1
その他ならばdは0



AND（積）

aが1かつbが1ならばcは1
その他ならばcは0

例題

- 第1列目の文字は”A”か”B”でなければならない。第2列目の文字は数字でなければならない。この状況においてファイルの更新が行われる。最初の文字が正しくなければX12のメッセージを印字する。2番目の文字が正しくなければX13のメッセージを印字する。

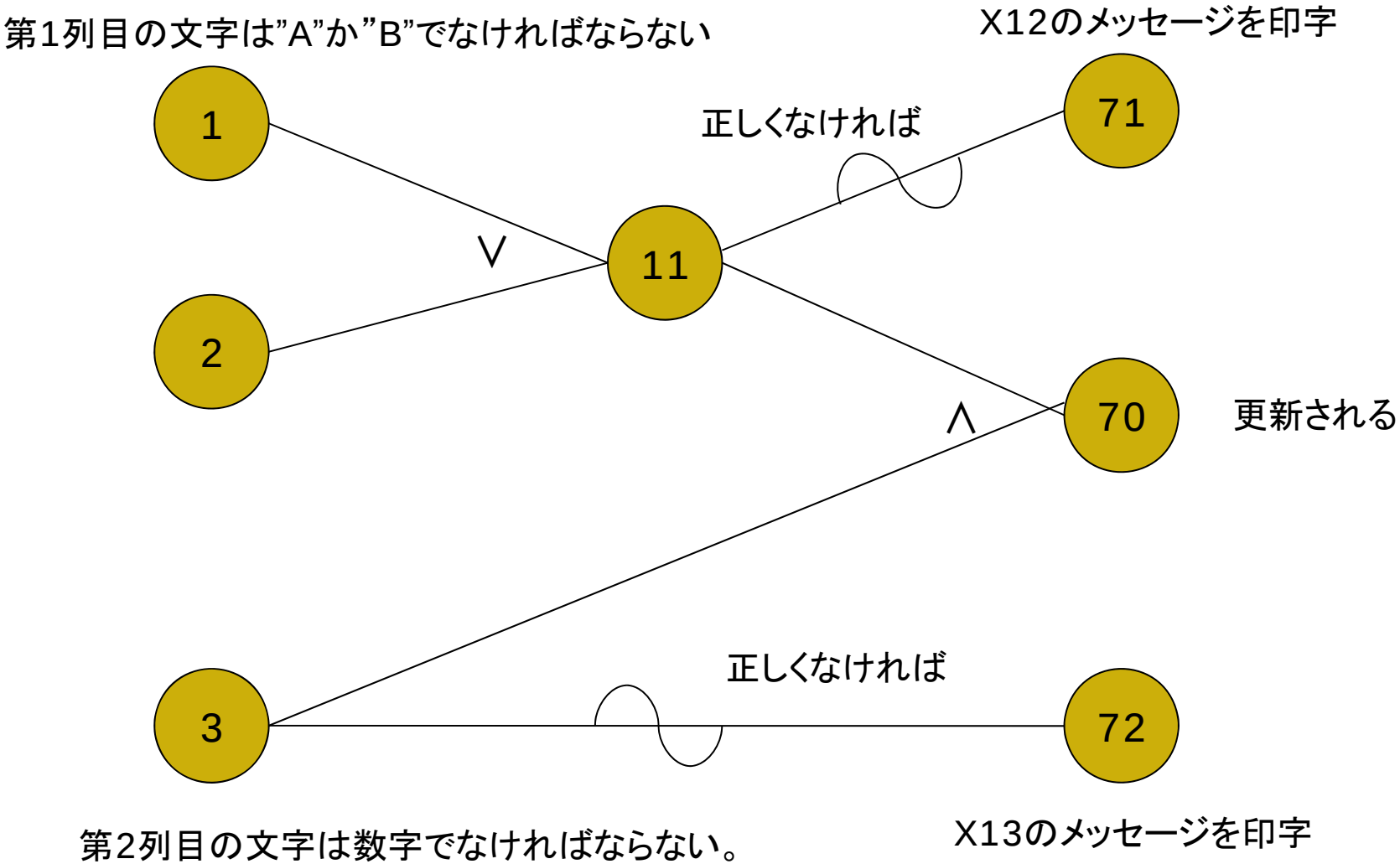
- 原因

  1：第1列の文字が”A”
  2：第１列の文字が”B”
  3：第2列の文字が数字

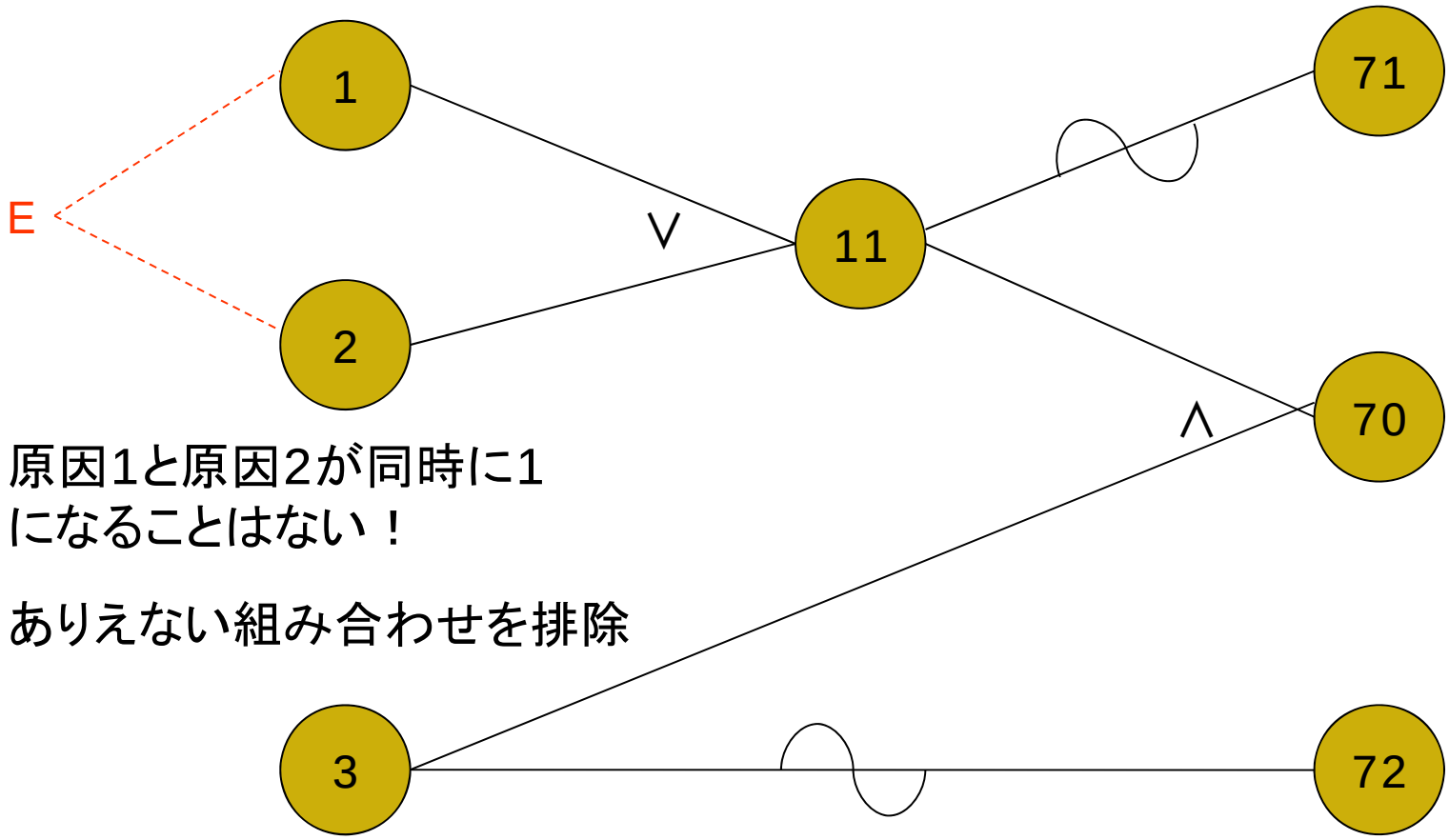
- 結果

  70：更新される
  71：X12のメッセージが印字される
  72：X13のメッセージが印字される

例題からつくられる原因－結果グラフ



仕様の修正（制約記号の付加）



原因1と原因2が同時に1
になることはない！

ありえない組み合わせを排除