形式手法とテスト
そして、その先について

ライフロボティクス株式会社
川口　順央

自己紹介

川口 順央（かわぐち　まさてる）
@masateruk
ライフロボティクス株式会社
CORO開発プロジェクト プロジェクトリーダー

2004年から形式手法に興味を持ち、勉強と試行の繰り返し

個人レベルでは何度か現場に導入したことあり（Spin, Alloy, CSP など）

インタビュー

» 2016年08月24日 06時00分 更新

水曜インタビュー劇場（ロボット公演）：

“手先が伸びて縮むだけ”のロボットが、「在庫ゼロ」になるほど売れている理由（1/7）

「ロボット」と聞けば、複雑な動きをするモノ――。といったイメージをしている人も多いと思うが、手先が伸縮するだけのロボットが売れている。トヨタ自動車やオムロンといった大企業が導入していて、現在の在庫は「ゼロ」。なぜ多くの企業が、単純な動きをするロボットを求めているのか。

[土肥義則，ITmedia]

印刷／PDF　1031 ツイート　1187 いいね！　1187 シェア　295 Bookmark　189 Pocket　32 G+1　通知

PR “3-in-1”最新Win10タブレットのモニター募集中！

「ロボット」と聞いて、どんなモノを想像するだろうか。AI（人工知能）が搭載されていたり、複雑な動きをしたり、人間ができないことをしたり――そんなことを思い浮かべる人が多いかもしれないが、“手先が伸びて縮むだけ”のロボットが、各方面から注目を浴びて

http://www.itmedia.co.jp/business/articles/1608/24/news004.html

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

形式手法とは

数理論理学等に基づき品質の高いソフトウェアを効率よく開発するための
科学的・系統的アプローチ※

- 明確で厳密
- 高い信頼性が期待
- 分析・検証が可能
- 導入コストが高い（と言われる）

※ http://research.nii.ac.jp/~f-ishikawa/work/files/2012-05-11-SQiP-intro.pdf より

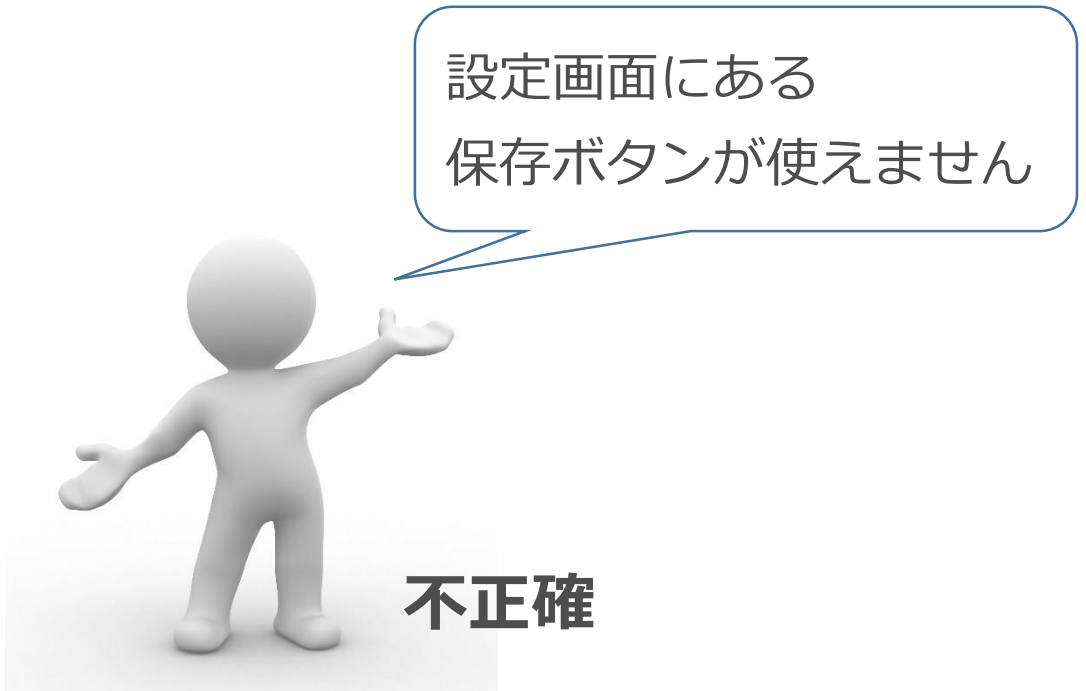
形式手法の適用レベル

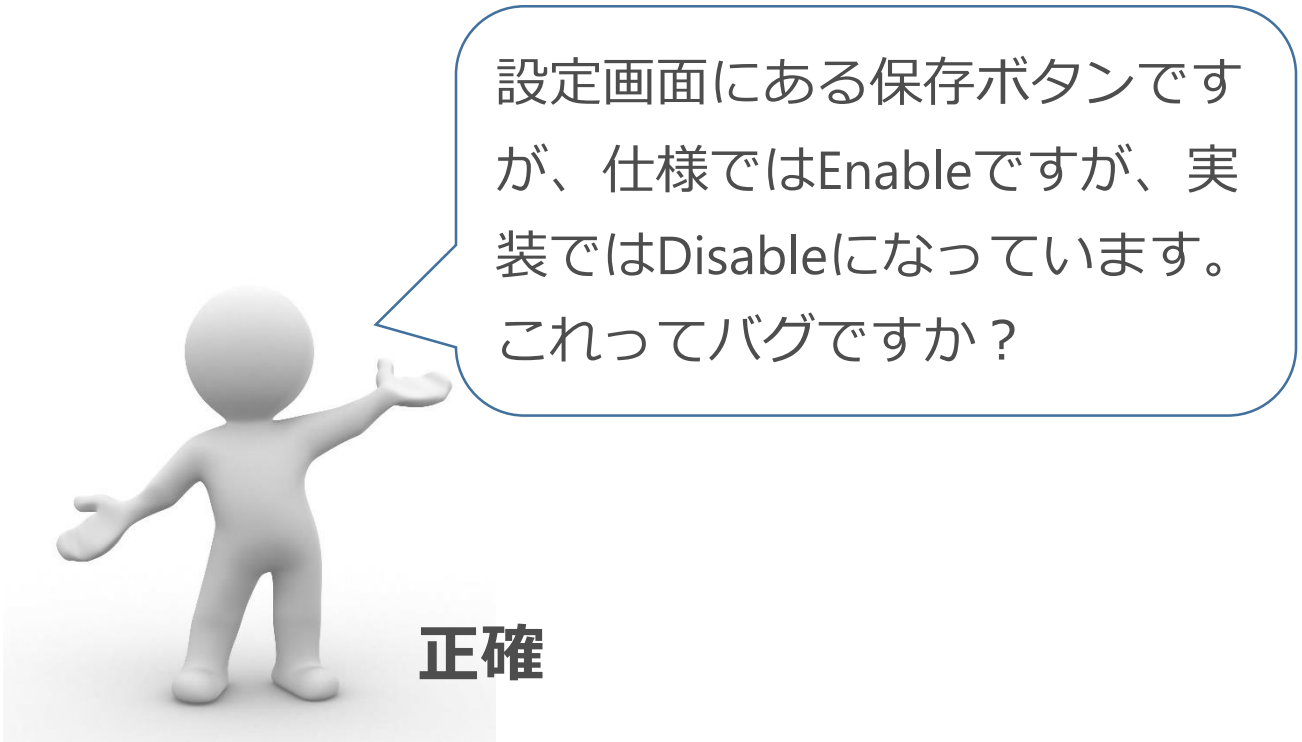
レベル 0	数学的な記述が可能である形式仕様記述言語を用いて仕様を記述する
レベル 1	形式仕様を詳細化することでプログラムを開発、または プログラムの性質を証明する
レベル 2	プログラムの性質を自動的に検証する

形式手法の適用レベルは、http://formal.mri.co.jp/outline/ より

なぜ形式手法か？

- 数理論理学を用いることで、明確で厳密なモデルを手にすることができる
- 明確で厳密であるため、自分の考えの推敲や他者との議論やコミュニーケーションが**正確**で**効率的**である





形式手法を導入するにあたっての懸念

1. 重くないか？開発効率落ちたりしないの？
2. 現実的に導入可能なコストなのか？
3. 仕様書を書くの大変そう。仕様書をメンテナンスするのも大変そう
4. 変更にかかるコストが形式的にしたことによって影響受けるか？
5. ステークホルダー全員がどうやってやりとりするのか？全員読めるのか？

導入事例の結果でふりかえります

形式的仕様記述を選択したねらい

厳密な記述で成果物の品質をあげる一方、
記述だけにとどめて導入コストを抑える狙い

明確で厳密

高い信頼性が期待

分析・検証が可能

導入コストが高い（と言われる）

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

形式的仕様とは

形式的仕様とは、数学的な記述で書かれた仕様のこと

仕様とは

- システムの仕様というとさまざまな視点がある
  - 機械的、電気的特性や見た目や外観、寸法など
- ソフトウェアにおいては振る舞いが最大の関心事のひとつ
- 以降は、どう振る舞うか？を定めたものを仕様として表現することに限定する

振る舞いを表す記法

- **シーケンス図**
  - 振る舞いのひとつの例に過ぎず網羅的ではない
- **コラボレーション図**
  - （主に内部の）構成要素の相互作用を書いたものである。外から見てどう振る舞うか？と抽象度が異なる
- **状態遷移図**
  - 外からみた振る舞いを網羅的に記述可能

状態遷移図から状態遷移モデル

- 状態遷移"図"である必要はなく、状態遷移を表すものであればよい。
  こういったものを総称して**状態遷移モデル**という

プリミティブな状態遷移モデル ー 遷移

定義

- プリミティブな状態遷移モデルは、**遷移の集合**
- 遷移とは次の3つ組

  (遷移元の状態，イベント，遷移先の状態)

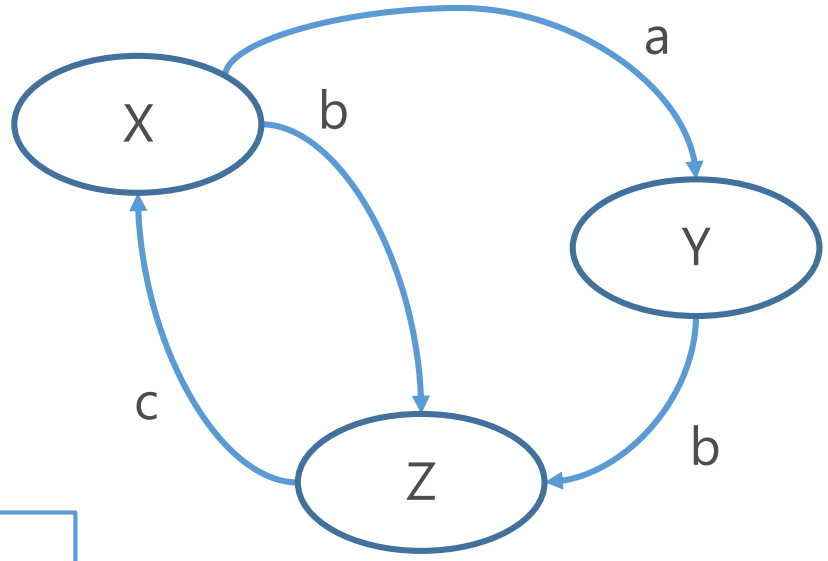
例

- 状態 X でイベント a をうけると状態 Y に遷移する遷移

  (X，a，Y)

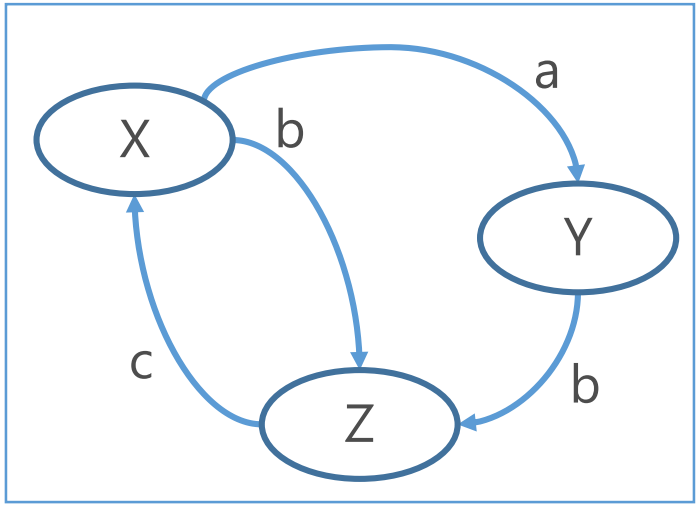
- 状態遷移

  {(X，a，Y)，(X，b，Z)，(Y，b，Z)，(Z，c，X)}



例の状態遷移図

プリミティブな状態遷移モデル ー 図と数学



状態遷移図

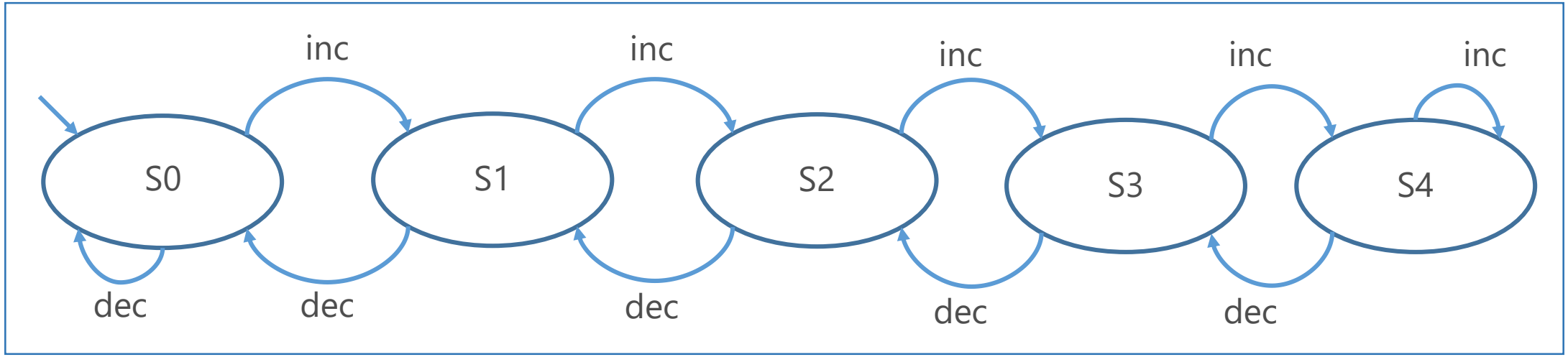
遷移を一覧したり、手で追ってシミュレーション
したりするのに便利

```
{(X, a, Y), (X, b, Z),
 (Y, b, Z), (Z, c, X)}
```

数学的記法

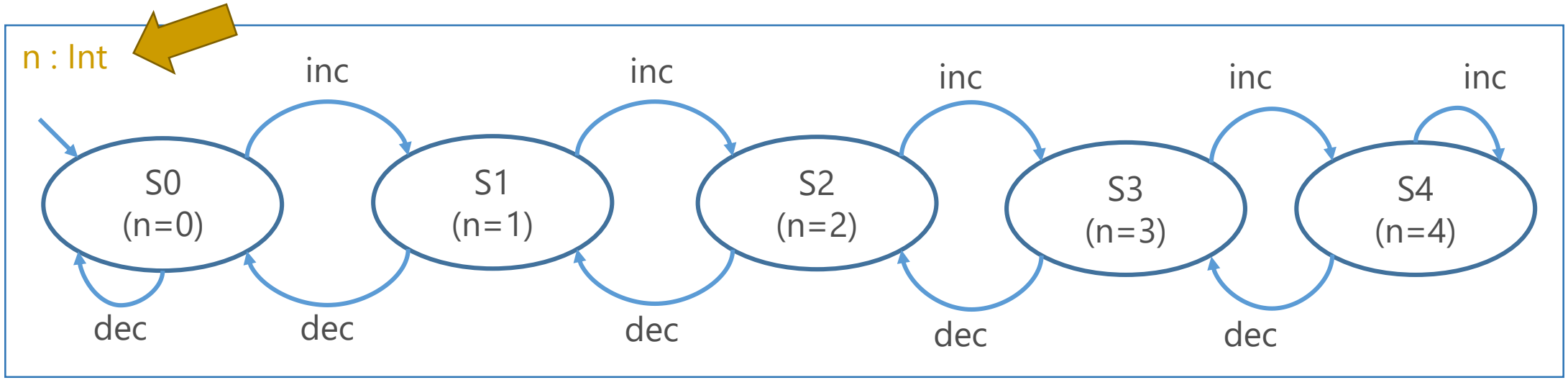
厳密な検証や分析するのに便利

状態遷移モデルに変数を導入する ー 変数なし



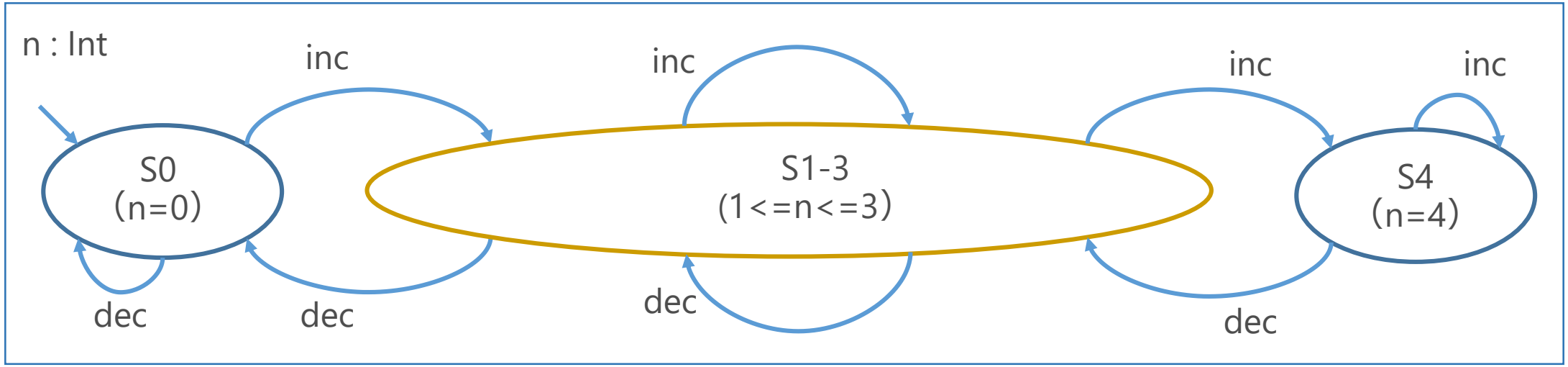
- 初期値が 0 でイベント inc で 1 つインクリメント、dec でデクリメントする状態遷移図（ただし、数値は 0 以上、5 未満）
- 状態（丸）が 5 個で、遷移が 10 個

状態遷移モデルに変数を導入する ー 変数追加



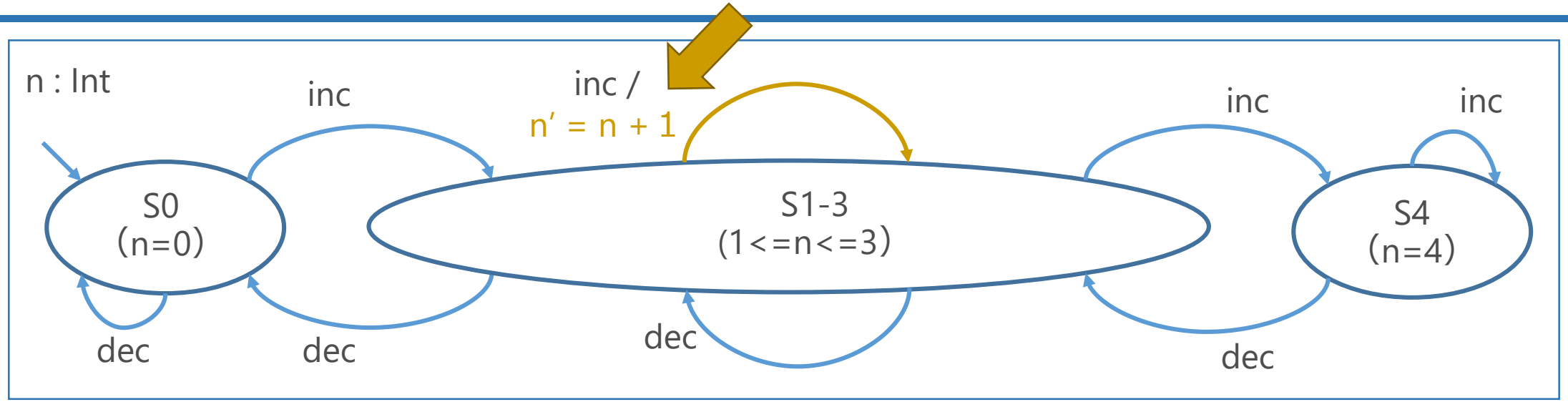
- 0 から 4 ではなく、0 から 99 だったら？数値が取りうる範囲が大きくなると図では表しづらい
- 整数型の変数 n を導入する

状態遷移モデルに変数を導入する ー グルーピング



- S1 から S3 の状態をグルーピングしてひとつにまとめる
- ただし、こうすると n = 1 のときにイベント inc をうけたときに n が 2 になることを表せない（1 <= n <= 3 であることはわかるが）

状態遷移モデルに変数を導入する ー 事後条件#1

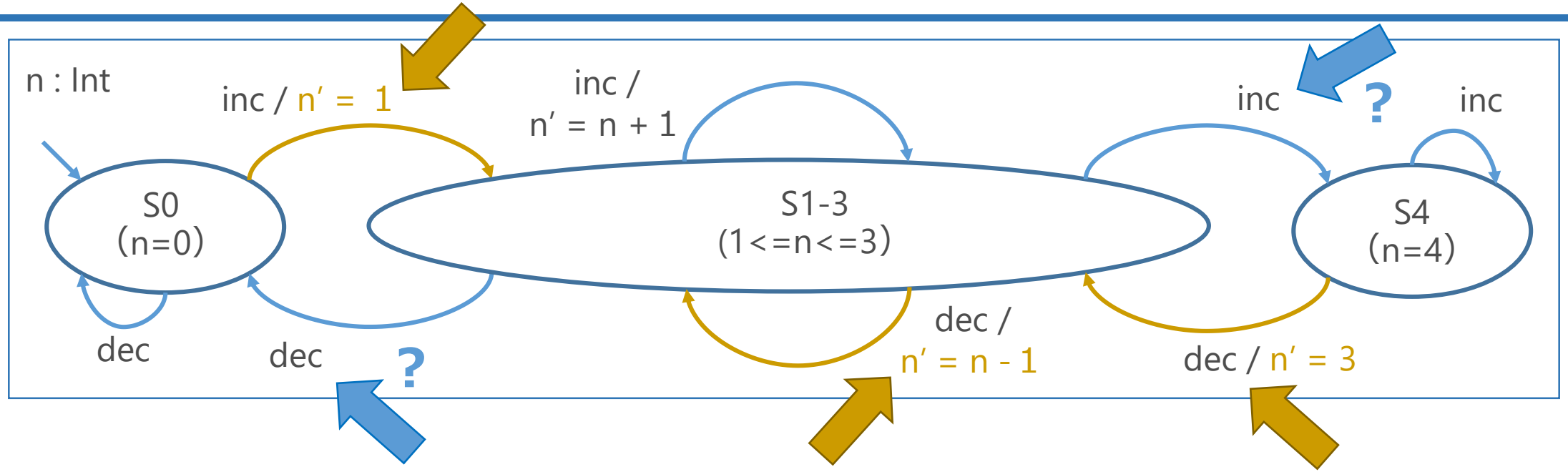


- n = 1 のときにイベント inc をうけたときの n の変化を条件で表す
- 遷移後の n は遷移前の n よりひとつ大きいものに等しい

n′ = n + 1　　（ n′ は遷移後の n, = は代入ではなく等しいことを表す等号）

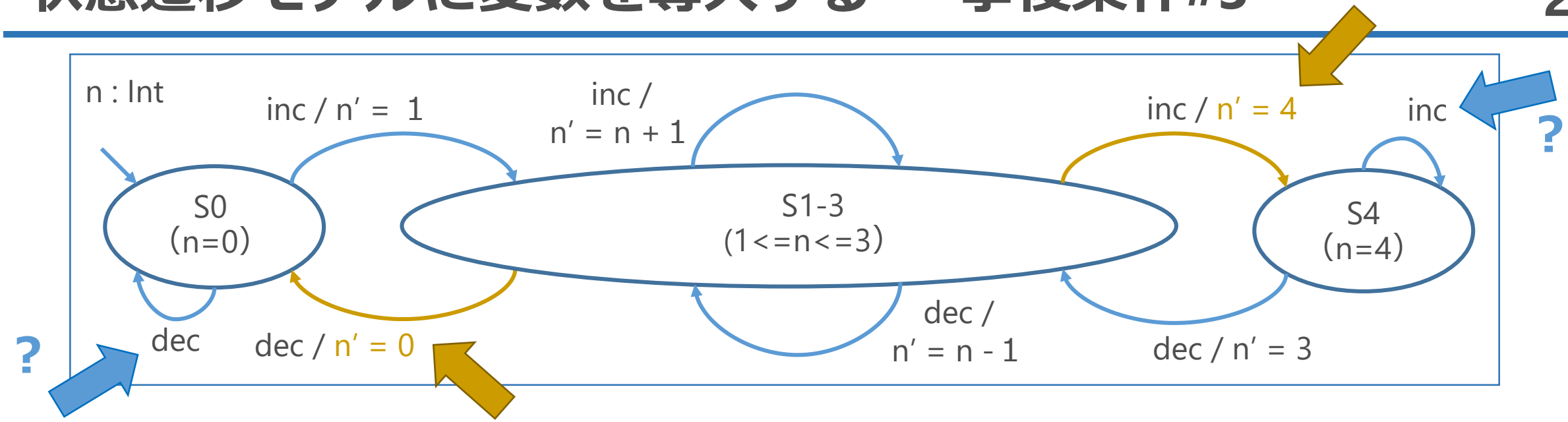
- これを**事後条件**と呼ぶ。事後条件は事前状態と事後状態の関係を表す

状態遷移モデルに変数を導入する ー 事後条件#2



- S1-3 では n は複数の値を取りうるので事後条件で表現
- S1-3 での dec, S1-3 での inc はどうか？

状態遷移モデルに変数を導入する ー 事後条件#3

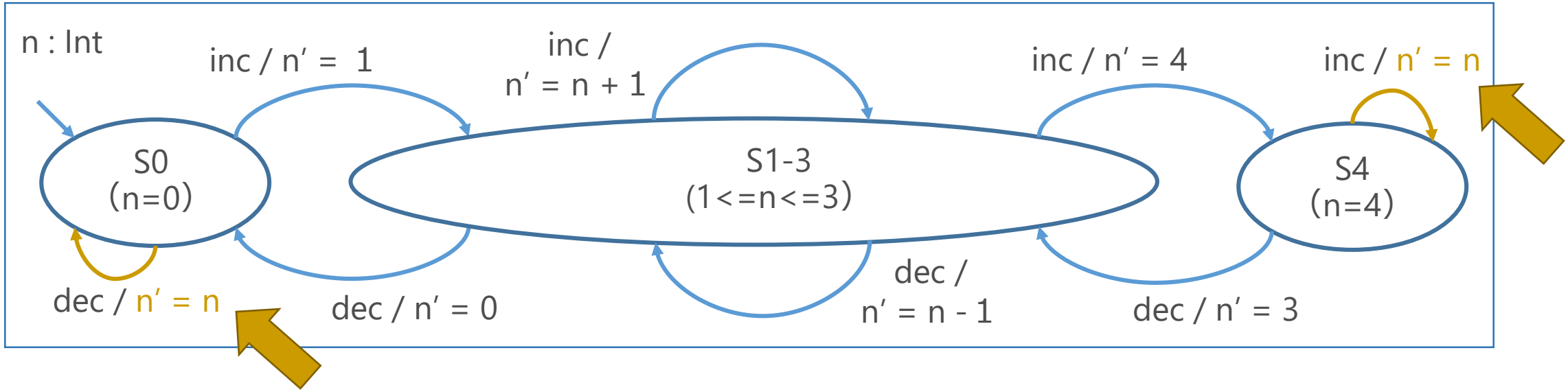


- S0, S4 ではそれぞれ n = 0, n = 4 であることが期待される

  それぞれの遷移で n がどのような変化（どのような値）をとっても良いわけではないので、事後条件を追記する

- S0 での dec, S4 での inc はどうか？

状態遷移モデルに変数を導入する ー 事後条件#4

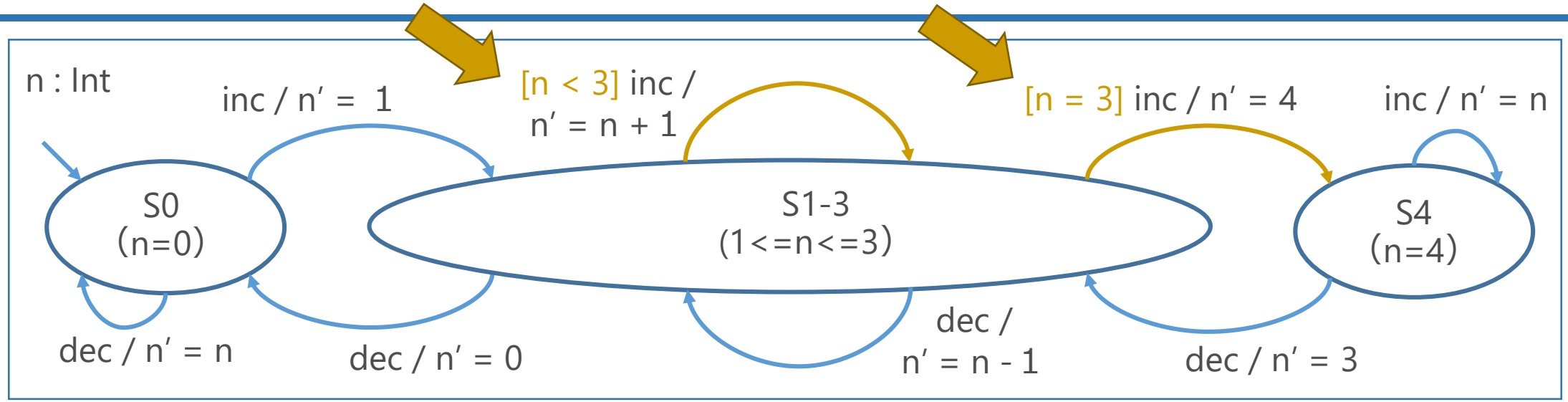


- S0, S4 ではそれぞれ n = 0, n = 4 であることが期待される

  それぞれの遷移で n がどのような変化（どのような値）をとっても良いわけではないので、事後条件を追記する

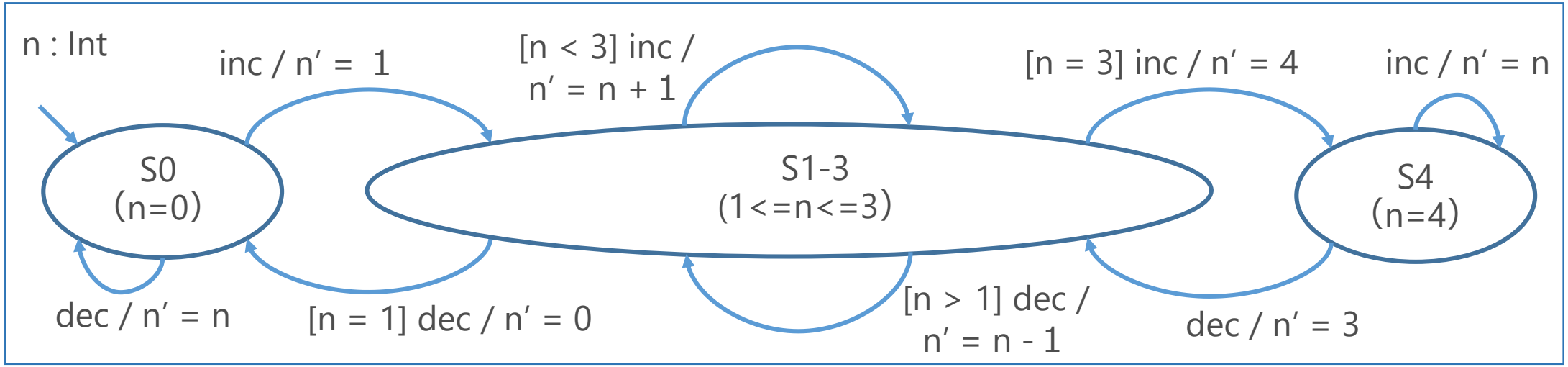
- **変わらないことも事後条件で明記する**

状態遷移モデルに変数を導入する ー ガード条件



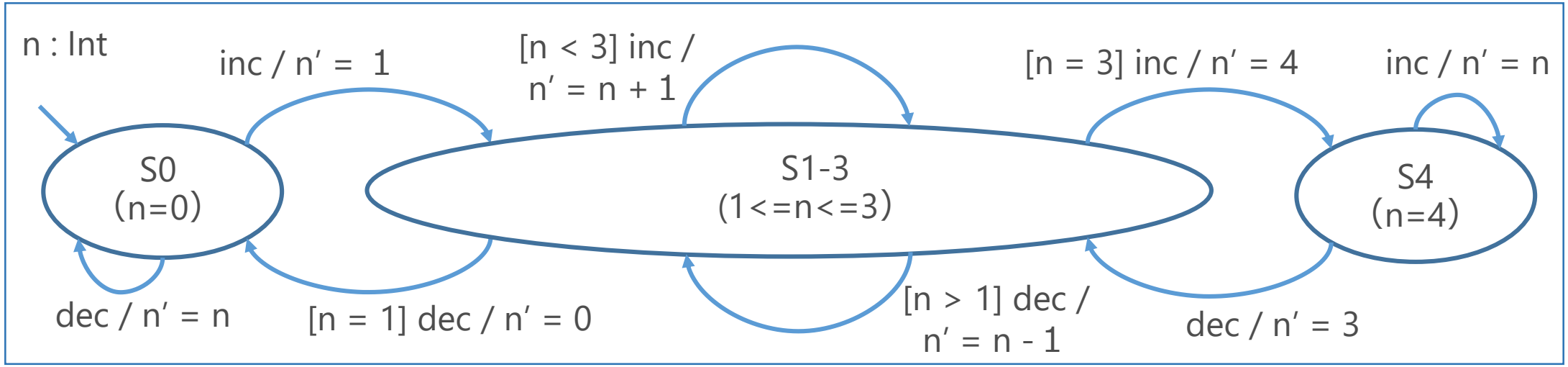
- また、n < 3 のときと n = 3 のときで、同じイベント inc の遷移後の状態が異なる
- これを表現するのに遷移に条件を付与する。イベントが発生したとき、条件が成り立っている遷移を遷移可能とする
- この条件を**ガード条件**と呼ぶ

状態遷移モデルに変数を導入する ー 完成



- 変数を導入すると、丸を少なくして状態遷移図を書くことができる
- 同じようにしてイベントもパラメータ抽象してまとめて扱うことができる
- イベントのパラメータはガード条件や事後条件にも表れうる

状態遷移モデルの形式的記述



- 状態遷移モデルの事後条件、ガード条件を数式（論理式）で表した記述
- システムの振る舞いを曖昧なく表現できる

まとめ

- 形式手法とは、数理論理学を用いたアプローチ。**厳密な記述**と分析が可能
- 形式的仕様記述のみの導入で、導入コストをおさえて記述の厳密さを手に入れる
- 仕様は状態遷移モデルで表す

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

導入プロジェクトの概要

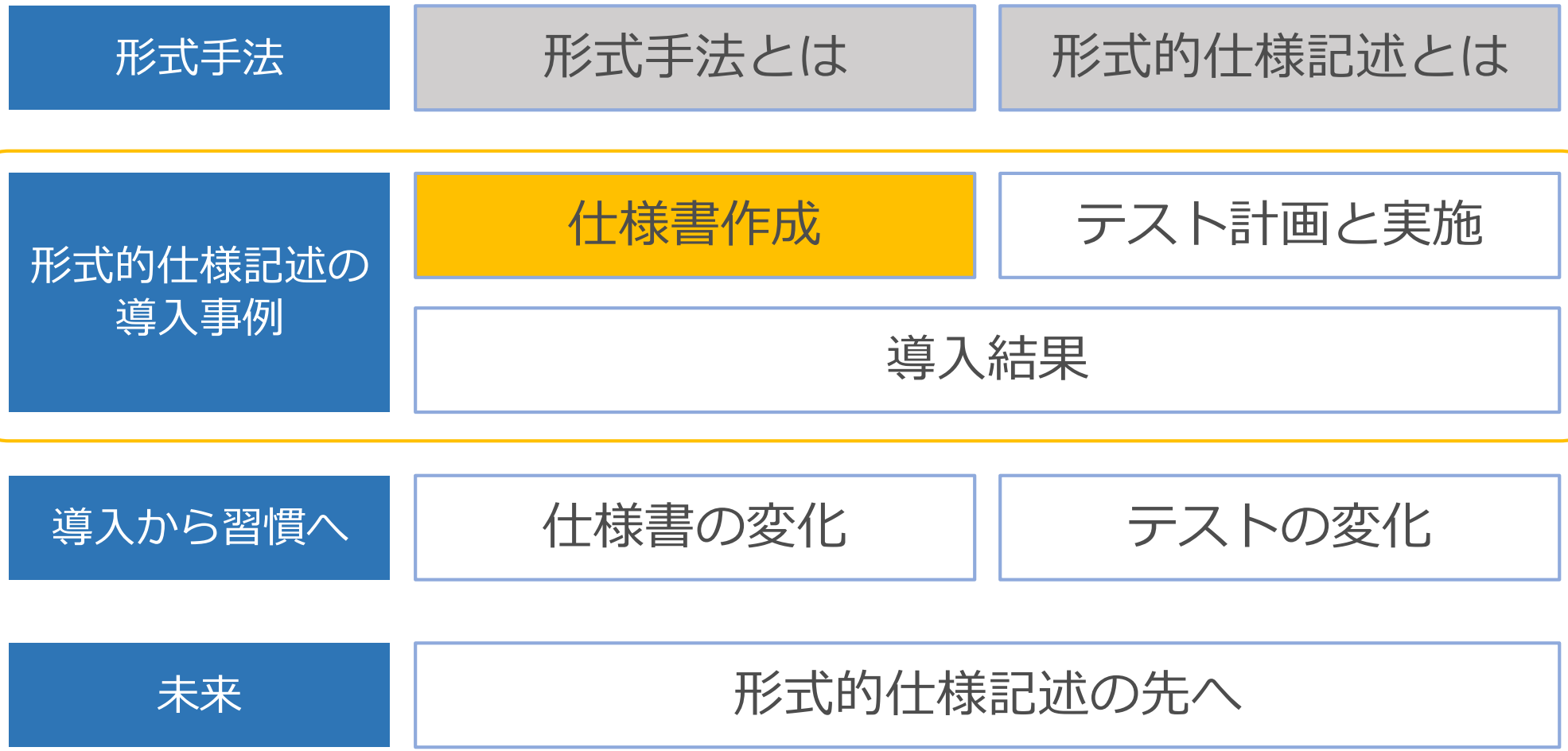
人の近くで動く協働ロボットCORO®の開発

スタートアップゆえにスピードが求められるが、
同時に品質も求められる

当時のSW開発者は５人のチーム

形式手法の経験者は自分ひとり。ほかは経験なし。
ただし、２人は関数型言語の知識があった

発表の構成


形式手法
形式手法とは
形式的仕様記述とは
形式的仕様記述の導入事例
仕様書作成
テスト計画と実施
導入結果
導入から習慣へ
仕様書の変化
テストの変化
未来
形式的仕様記述の先へ

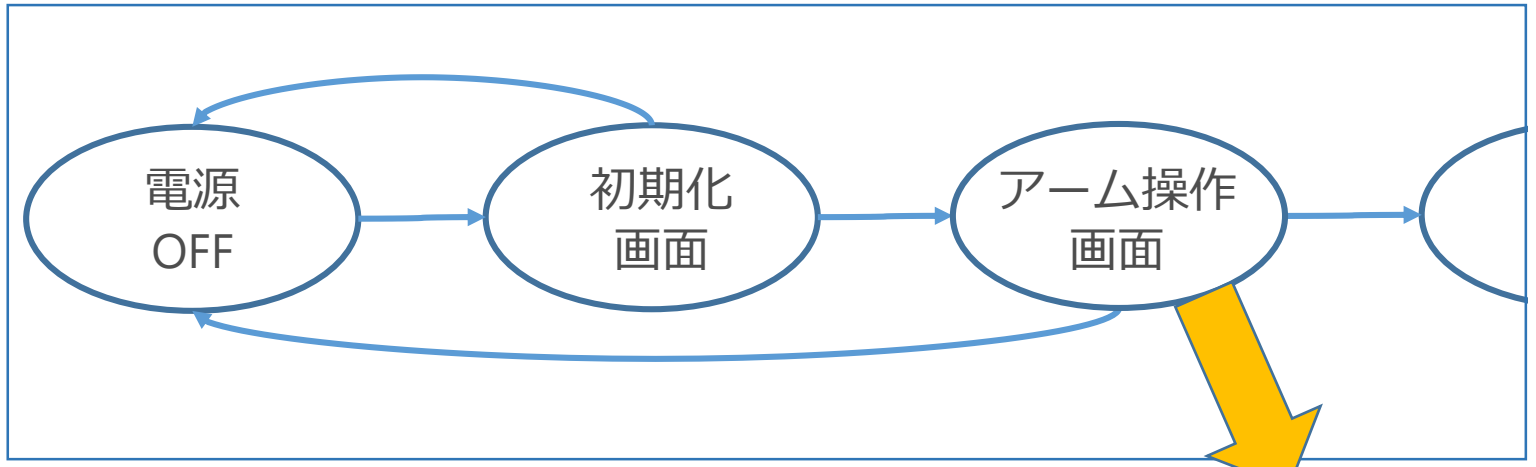
仕様書に何を書くか？書かないか？

- ユーザから観測できるシステムの特性を記述する。ユーザは単なる使用者のほか、メンテナンス担当者も含む
- 特性とは具体的には、
  - 振る舞い
  - 非機能要求
- 書かないとしたもの
  - 画面のUIパーツの大きさや色、配置

独自の仕様記述言語で状態遷移モデルを記述する

- 振る舞いは状態遷移モデルで記述する。遷移の構成要素は、
  - 遷移元の状態、イベント（パラメータを含む）、ガード条件、事後条件、遷移先の状態
- 状態は変数をもつことができる。ガード条件、事後条件の記述を**論理式**と**自然言語**で書く。
  論理式は、独自の仕様記述言語で書く
- 独自の言語を作った理由
  - 気に入ったものがなかった
  - 記述することに主眼をおいていたためツールのサポートはなくてもよいと割り切った
  - あとでパーサーが作れるように考慮して文法は決めた

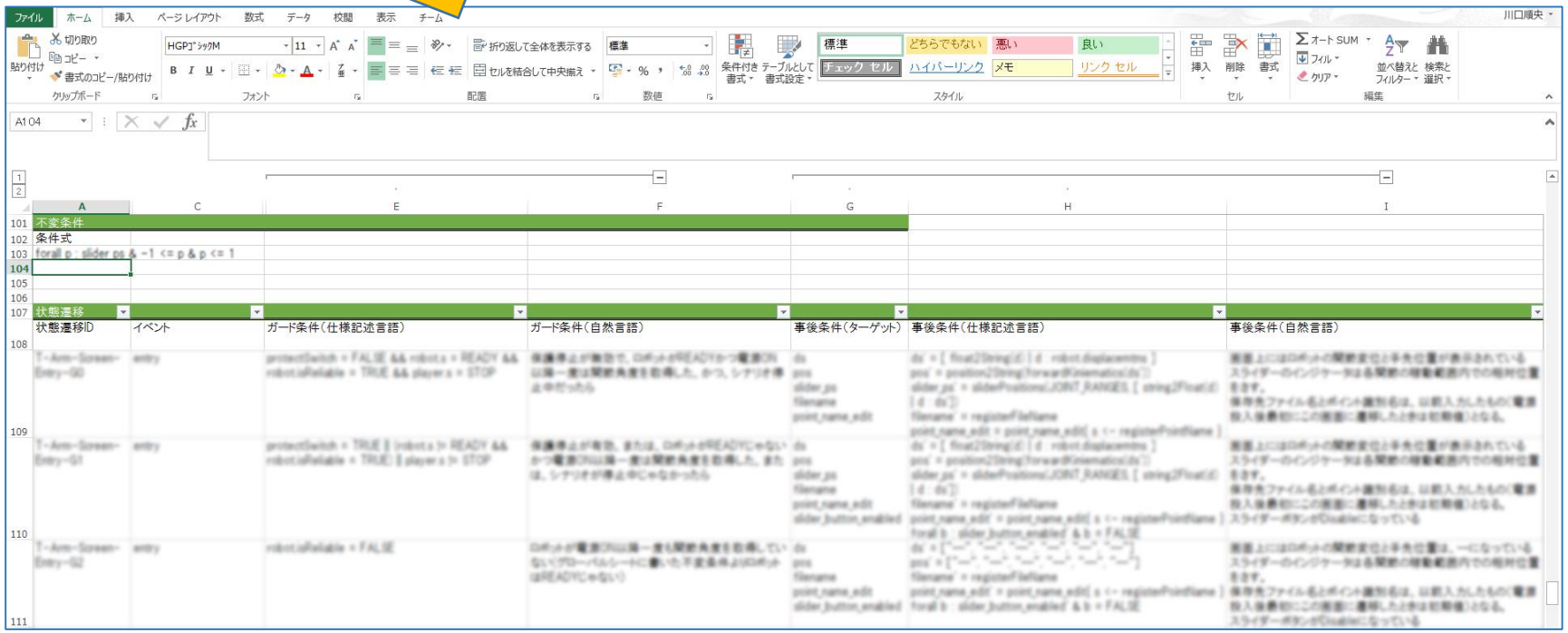
仕様書はEXCELに



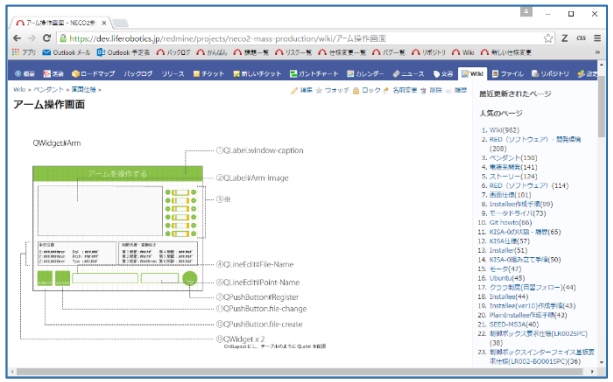
EXCELの理由

- エンジニア以外でも見慣れている
- エンジニア以外でもグルーピングや
  フィルターなど使える

- ひとつの画面を状態遷移機械のひとつの状態（丸）にする
- ひとつの状態の遷移をひとつのシートに書く

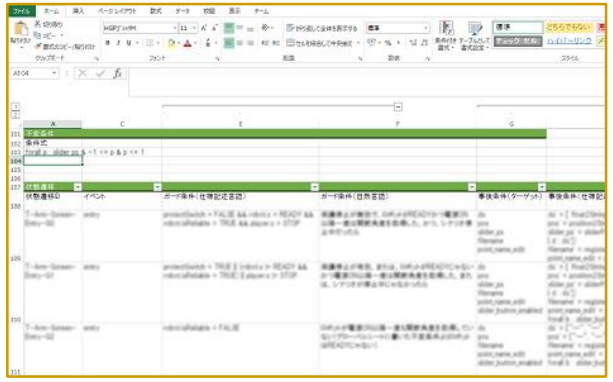


各画面の仕様はUI仕様をもとに



UI仕様

- 画面のイメージとボタンなどのUIパーツの配置
- 各UIパーツを操作したときにどんなことが起きるかが自然言語で説明してある



形式的仕様記述

- その画面で起きるすべてのイベントについて遷移について、論理式および自然言語による説明がある
- UIパーツの配置、色については言及しない

形式的仕様記述を書くだけで得られる効果

- **自然言語で書かれた仕様で気づくことができない考慮漏れをたくさん見つけた**
  - UI仕様の作成、レビュー時には気づかなかった
  - 事後条件を書くときに、変化しないものも含めてすべての状態変数について事後条件を考えるため。**プロトタイピングに近い効果**がある
  - 中にはソフトウェアのアーキテクチャに影響を及ぼしそうなものもあった

書くのは大変、でも読むことはできる

- 計画段階では、いくつかの画面の仕様を書いたら、あとはみんなで手分けして書くつもりだった
- しかし書くのは難しそうだったので、結局ひとりで全部書いた
- レビューは全員してくれた。**自然言語による説明**があったので意外と読めた（学習時間は状態遷移モデルと記述言語について**2時間ほど**講義しただけ）

仕様作成プロセスのまとめ

- 形式的に記述することで、多くの考慮漏れ、コンセプトの矛盾を発見することができた
- 書くのは難しいが、自然言語による説明があれば読むのはできる

発表の構成

テスト計画

- 当時は、開発者がテスト項目を作る必要があった
- 各遷移にテスト項目を記述する。テストの対象を遷移として、各遷移にその遷移をテストするため3つの項目を書いた
  - テストする遷移のガード条件
  - ガード条件を満たす手順
  - 遷移後に事後条件が成り立っているか確認する手順
- 一般的には、ひとつの遷移に対して複数のテスト項目
- すべての遷移に対してテストを実施すれば、振る舞いを網羅的にテストできると考えていた

テスト担当者への配慮

- テストケースの3項目は基本的に自然言語で書く。
  テスト担当者は論理式を見ないですむように

遷移以外のテストはランダムに

- 遷移視点だけのテストでは不安があったので、ランダムに触ってもらっておかしなところがあれば報告してもらうことにした

テスト実施の結果

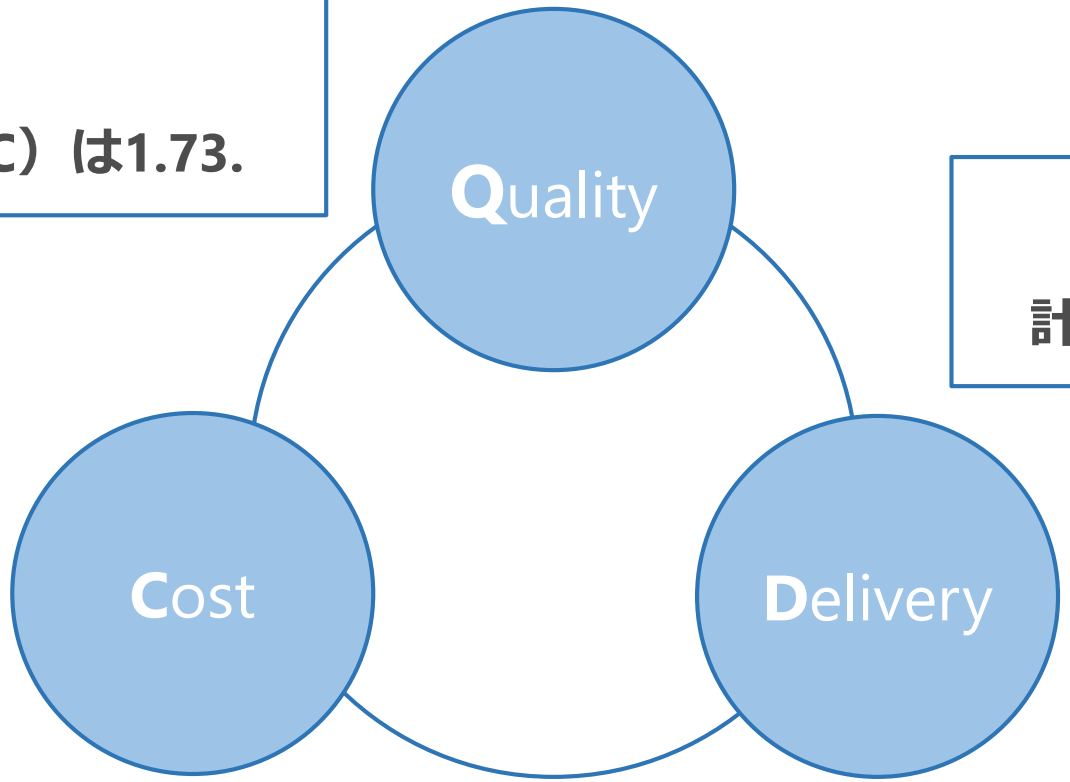
- ほとんどのバグはランダムテストで報告される
- 自然言語で書かれた事後条件では読み切れず、結局テスト担当者も論理式を読む

発表の構成

導入プロジェクトの結果 - QCD

So so

バグ密度（バグ数/1KLOC）は1.73.



Excellent !!

計画の日程で出荷判定合格

Good !

仕様が形式的だから増えたという感覚はない

品質への寄与

- 品質への貢献は期待したほどではなかった
- しかしそれ以外に見逃せない良い副次的な効果があった
  - 仕様に対する安心感
  - 合意形成の容易性
  - 設計・実装の少ない手戻り
  - 合理的なゴール設定が容易に

心配していた点はどうだったか？

1. 重くないか？開発効率落ちたりしないの？
2. 現実的に導入可能なコストなのか？
3. 仕様書を書くの大変そう。仕様書をメンテナンスするのも大変そう
4. 変更にかかるコストが形式的にしたことによって影響受けるか？
5. ステークホルダー全員がどうやってやりとりするのか？全員読めるのか？

心配していた点はそうでもなかった #1

1. **重くないか？開発効率落ちたりしないの？**

   → あくまで体感だが、**重いという印象はない**。スケジュール遅延なし

2. **現実的に導入可能なコストなのか？**

   → 最初に仕様書を書き上げるのは大変だけど、できなくもない

3. **仕様書を書くの大変そう。仕様書をメンテナンスするのも大変そう**

   → 最初に書くのは大変。でもこれは**形式的だからというよりシステムの複雑さに依存する**

   → メンテナンスはEXCELでなければよかった

心配していた点はそうでもなかった #2

4. **変更にかかるコストが形式的にしたことによって影響受けるか？**

   → 変更の必要の有無を判断しやすくなったのでむしろ良くなった

5. **ステークホルダー全員がどうやってやりとりするのか？全員読めるのか？**

   → **自然言語を併記**すれば割と読める。ただし、自然言語だけだと正確に読めないところもある

   → 自然言語だけでは記述が十分がどうかはほとんど判定できない

メンバーの感想 #1

形式的仕様記述について

- 自然言語で書かれると読む人によって解釈が多様になるけど、論理式だと曖昧さがない
- 日本語と併記している仕様を見ると、日本語の不明瞭さがよくわかる
- 齟齬が生じにくいのでコミュニケーションが取りやすかった
- 仕様の漏れに気づきやすい
- 自然言語と併記されていると読むのは何とかなるけど、書くのは難易度高い
- 最初はとまどった。今までは要求仕様書といったレベルのものばっかりだったので
- 仕様に基づいて作業するときに，何を実現すればいいのかが明確になった
- 検証作業を行った時も、どういった条件で何を検証すればいいかが分かりやすかった

メンバーの感想 #2

仕様書を中心にしたプロセスについて

- 最後まで仕様書メンテしたのは初めて
- 仕様にテスト項目まで書くとそのまま検証に出せるのは楽だった
- 遷移のすぐ隣にテスト項目が書いてあるので自分で検証する時すごい楽だった

ある開発者の胸あつな感想

もともとは自然言語やフローチャートで業務系のアプリケーションの仕様を書いていたため、最初は「形式仕様記述とか，まじかよ」と敷居を高く感じていたが、実際は慣れるまであまり時間もかからず、開発メンバー間でのコミュニケーションがとても取りやすいことに気づき、（今現在痛切に）**仕事が非常にやりやすかった**と感じた

導入に成功した秘訣

- UI仕様をもとに形式的に記述する時に次々に問題を見つけ、メンバーが形式化の力を目の当たりにした
- とにかく何かあれば仕様に戻るように促した。仕様に記述が足りなければ追記して、仕様書を見れば判断できることを浸透させた
- 記述を形式化したことに満足せずに基本を忘れない。仕様書の定義（何を書く、書かないの定義）、仕様書のバージョン管理、変更管理とメンテを怠らない

テストに与えるメリット

- 曖昧な表現ではないので、日本語での表記より仕様上での矛盾に気づきやすい
- 仕様の曖昧さが減り、どこで何が起こったか確認しやすい
- 各状態遷移に対するテストが必ず作成されるので、仕様に対してテスト漏れが発生し難い
- 抽象度がそろった表現になっていて仕様を理解しやすい。たとえば、エラーの仕様については“良しなに”という記述はできない。“良しなに”の定義がないので

テストに関する課題

- 抽象化した状態遷移モデルの視点だけでは不足しているものある
  - モデル上はアトミックな遷移で表現されていても実装上有限時間かかり、その間に入力できてしまうもの

    例）システムの電源が入ってからアプリケーションが起動するまでの操作
  - リソースリークのような欠陥を見つけられない
- 事後条件で使われている関数にテストケースが隠れている
- 遷移単位のテストはテスト実施時の効率がよくない。各遷移のテストがほかのテストが終わった後の状態から続けて行うことを考慮していない

発表の構成

EXCELからテキストへ

- 仕様書をGitでバージョン管理したが、diffとmergeができないことがストレスに
  - 特にmergeができないと複数人での編集が不可能
- プログラマは自分が好きなエディタで作業したいという要求も
- 導入プロジェクトの後半からエンジニアを中心にテキストベースに移行したい声があり、EXCELをすてる決意
  - 状態遷移モデル全体を記述できる言語を開発して移行することに

仕様記述言語KML

- KMLは、変数で拡張した状態遷移モデルを記述するための言語
  - 導入プロジェクトで自然言語の力を痛感したので、論理式の説明を単なるコメントではなく、構文要素として扱う
  - KMLで書いたものをマークダウン経由でHTMLに変換して今までのEXCELと似たフォーマットで出力が可能
- ツールは構文検査のみ。KMLは型つき言語なので型検査も導入可能だけど未実装
  - なおツールはOCamlで書かれている

KMLの記述例 #1


```
// 例）ロボットの移動点の編集状態の状態遷移．// でコメントが記述できます
state Edit@(移動点の編集状態) {

    var points : Point List = [];
    var index : Int = 0;

    invariant { // この状態で成り立つ不変条件
        (length points != 0) => (0 <= index && index < length points);
    }@{
        点のリストが空でなければ、インデックスは0以上、かつ、点のリスト長より小さい
    }
```

KMLの記述例 #2


```
transition addPoint(p : Point) --> {
    post {
        target points;

        points' = points ^ [p];
    }@{-
        事後の点のリストは、事前の点のリストの末尾に点pを追加したもの
    -}
}
```

KMLの記述例 #3


```
transition deletePoint
      when length points != 0 @[- 点のリストが空でないとき -] --> {
    post {
        target points, index;

        points' = [ points # i |
                    i : Int & 0 <= i && i < length points && i != index ];
        index' = if index + 1 = length points then index - 1 else index;
    }@{-
        事後の点のリストは、事前の点のリストからindex番目を除いたもの
        インデックスは末尾を指していた場合は1つ小さい値、そうでなければ変わらない
    -}
}
```

KMLの記述例 #4


```
    transition move
          when length points != 0 @[- 点のリストが空でないとき -] --> {
        post {
            state' = Moving(points);
        }@{-
            移動状態へ遷移する
        -}
    }
}
```

KML変換後のHTML

移動点の編集状態(Edit)

変数

#	識別子	型	初期値	補足説明	備考
1	points	List Point	[ ]		
2	index	Int	0		

不変条件

#	論理式	補足説明
1	(length points != 0) => (0 <= index && index < length points);	点のリストが空でなければ、インデックスは0以上、かつ、点のリスト長より小さい

状態遷移

#	イベント	イベント(説明)	ガード条件(論理式)	ガード条件(自然言語)	事後条件(ターゲット)	事後条件(論理式)	遷移(プロセス式)	事後条件(自然言語)
1	addPoint(p : Point)				points	points' = points ^ [p];		事後の点のリストは、事前の点のリストの末尾に点pを追加したもの
2	deletePoint	点を削除した	length points != 0	点のリストが空でないとき	points index	points' = { points # i | i : Int & 0 <= i && i < length points && i != index }; index' = if index + 1 = length points then index - 1 else index;		事後の点のリストは、事前の点のリストからindex番目を除いたものインデックスは末尾を指していた場合は1つ小さい値、そうでなければ変わらない
3	move	移動	length points != 0	点のリストが空でないとき			Moving((points))	移動状態へ遷移する

書き手も増えた

- 自分だけでなく、みんなも少しずつ書いてくれるように
- Gitでのバージョン管理もしやすくなって仕様レビューもソースコードレビューと同じやり方でできるようになった
- ツールのサポートをより求められるようになった

発表の構成

テストはテストの視点で

- 導入プロジェクトのやり方をやめて、テストエンジニアが従来のテストの視点でテストケースを設計することに

それでも形式的であるメリットは何か？

- 仕様の曖昧さがなくなるので、実施時の動作確認、テスト項目作成時のテスト項目実装ミスが減る
- 各状態遷移の挙動が把握しやすくなるので、テストの自動化ではかなり手間が減らせる（操作とそれに対する状態変化が機械的に判別しやすい）
- 開発者とテストエンジニアの関係を円滑にする。仕様かバグかで争わない

テストチームに表れた変化

- 仕様を理解しようとする姿勢が出てきた
  - 自然言語の仕様と違って一目で理解したつもりになれない
- KMLを読みたいという欲が出てきた
  - 自然言語では読み切れない部分は直接論理式を読みたい

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

なぜ仕様を形式的に書くことが習慣化されたか？

- 仕様書が安定していて（変更が少ないという意味ではない）、信頼感があるから
- 仕様書がない開発が不安になりつつある
- 仕様書をもとに議論、コミュケーションすることで開発を効率的に進められるという成功体験がモチベーションにつながっている
- これらは何も形式的に書かなくても良いのだが、形式的に書くからこそ実現できる仕様書の品質によってもたらされるもの
- 書いてて**楽しい**仕様記述言語だった

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

製品の品質向上へのむけての全体的な取り組み

- そもそも形式手法を導入した理由は品質の良いものをつくりたいから
- 製品の品質は、開発の全作業の成果物の品質の積み重ねである
- テストだけに頼るやり方には限界がある。製品はどんどん複雑になる傾向があり、いくら自動化しても網羅的にできない
- 開発のあらゆる工程で品質を高める活動が必要である

テスト以外の品質を高める活動例

仕様書レビュー

設計レビュー

ソースコードレビュー

UIの評価

- それぞれが寄与する品質の側面が異なる
- 特にレビューは品質面以外への貢献もあり有意義な活動
- しかし、いずれも人手で行なっていて部分的にしか行われない

形式手法がもつ他の手法との圧倒的な差

- 数理論理学をベースにしている。**推敲、議論、検証が厳密**に
- 自分が考えているものが正しいかどうか検算、証明できる。**理解したつもり**にはなれない
- 数学という世界共通言語で表現できる（表現させられる）。未定義な言葉で**説明したつもり**の文章は書けない
- 数学をベースとした表現はコンピュータが処理可能な表現にしやすい
- コンピュータを使って**網羅的な検証を正確に**行うことができる

仕様書をコンピュータで扱えることで出てくる可能性

- 形式的に仕様を記述することで、仕様書の検査をツールができるようになる
- 簡単なところでは
  - 構文検査
  - 型検査
- これらが検出するエラーは書き手にとっては単純で些細なものかもしれないが、
  読み手にとっては意図したものなのかミスなのか判別つきにくい

シミュレーション

- 構文検査と型検査が通っても書いた仕様が自分が書きたかったものと異なる可能性もある
- 仕様をシミュレーションすることで自分の意図したものか確認することが可能になる

モデル検査

- シミュレーションは部分的な検査に止まるが、モデル検査を使えば仕様がある性質を満たすかどうかを**網羅的に検査**することもできる
- たとえば、不変条件を違反する遷移がないかどうか

定理証明系

- モデル検査には状態爆発という問題があり、そのままでは検査できないものもあるが、
  定理証明系を使えば無限のケースについて扱える

本質的な問題に注力するために

- 最近スケジュールが優先され品質が軽視される傾向があるが、実際に求められているのは品質の高い製品を早く届けること
- 品質を犠牲にしてスピードを求める人たちは、いつまでたっても品質の高いものを速く作れるようになれない
- 品質の高いものをつくる技術を持った人たちが訓練を重ねれば、速く作れるようになれる

自分たちを訓練して当たり前のように品質の高い製品が作れるようになれば、ユーザにとっての価値は何か？とか、使い勝手とか、より**本質的な問題に注力**することができるようになる

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	

まとめ

- 形式的仕様記述を開発プロジェクトに適用した。スケジュール通り製品をリリース
- 仕様書を中心に効率的に開発を進められたのは、形式的に記述されていたからこそ得られた仕様書の品質のおかげ
- 仕様書を形式的に記述することでさらなる検証の力が得られる。テストだけに頼らず全体的な取り組みで品質向上をねらう

発表の構成

形式手法	形式手法とは	形式的仕様記述とは
形式的仕様記述の導入事例	仕様書作成	テスト計画と実施
	導入結果	
導入から習慣へ	仕様書の変化	テストの変化
未来	形式的仕様記述の先へ	