

テストケース自動生成における テスト仕様検証への形式手法適用

一般財団法人日本科学技術連盟

第35年度（2019年度）ソフトウェア品質管理研究会成果発表会

研究コース5 チーム形式仕様

研究員 : 井森 一樹（富士通株式会社）

柳田 憲彦（エヌ・ティ・ティ・コムウェア株式会社）

指導講師 : 栗田 太郎（ソニー株式会社）

石川 冬樹（国立情報学研究所）

2020年2月21日（金）

“要求工学上”の研究の位置づけ

①チームDHC

②チームKNT

③チーム形式仕様

↑ 本発表

④チームGOBIT



出典元：IPA 要求工学知識体系（REBOK）概説

研究スコープ テスト設計の自動化

テスト自動化

テスト識別の自動化

テスト設計の自動化

テスト仕様

提案

テストケース

自動化
ツール

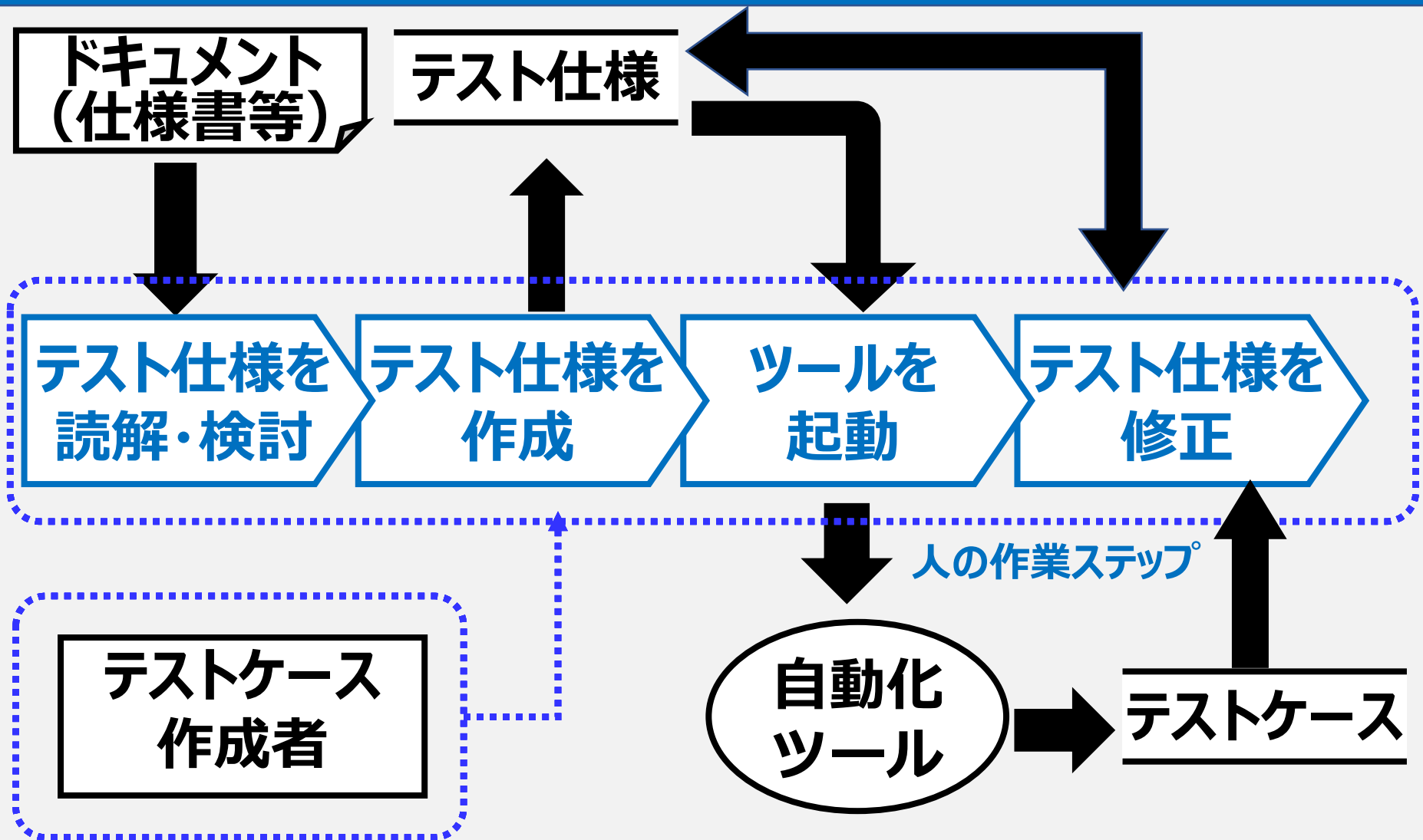
形式手法を
使った検証

テスト実装の自動化

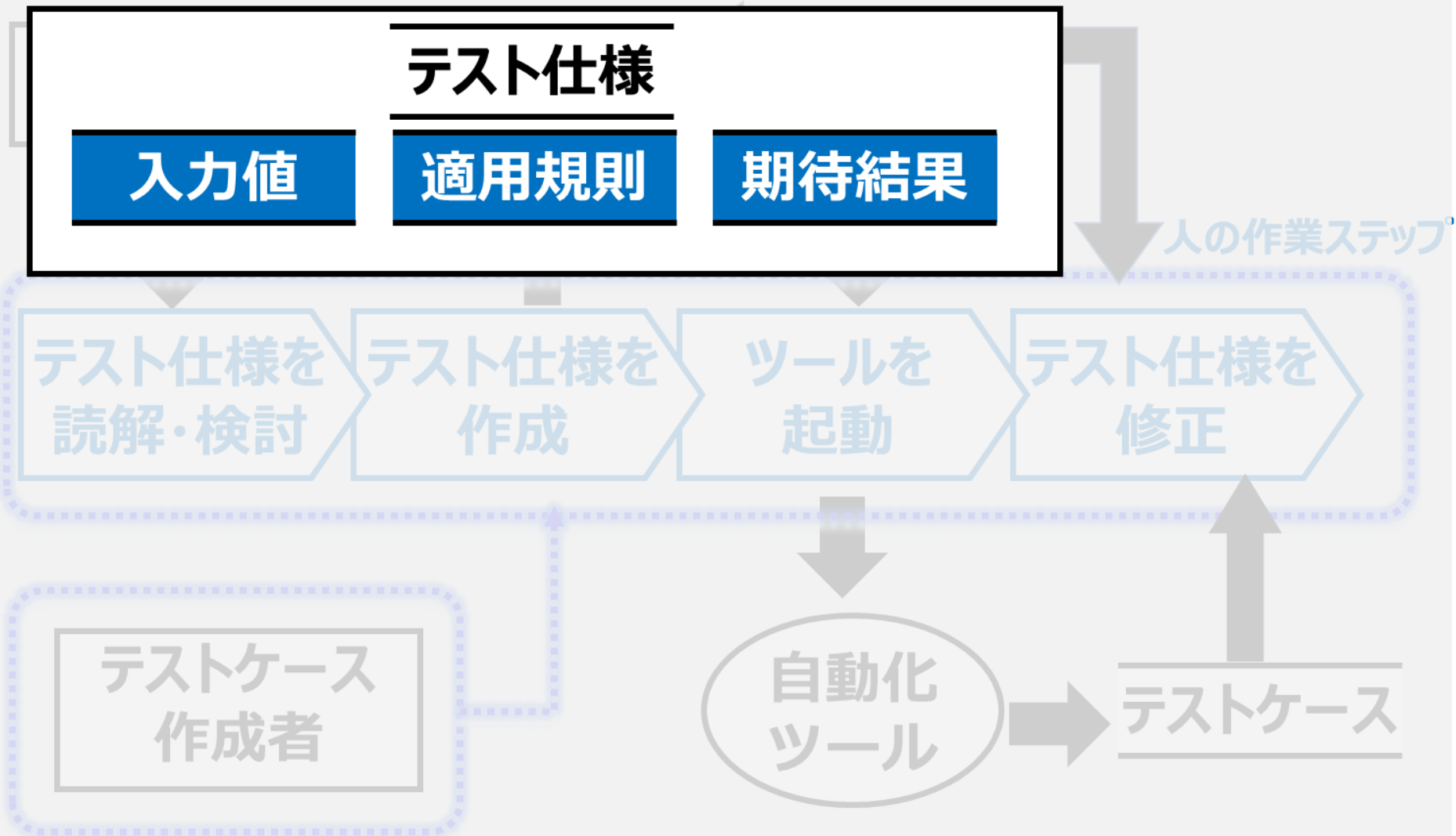
テスト実行の自動化

テスト結果比較の自動化

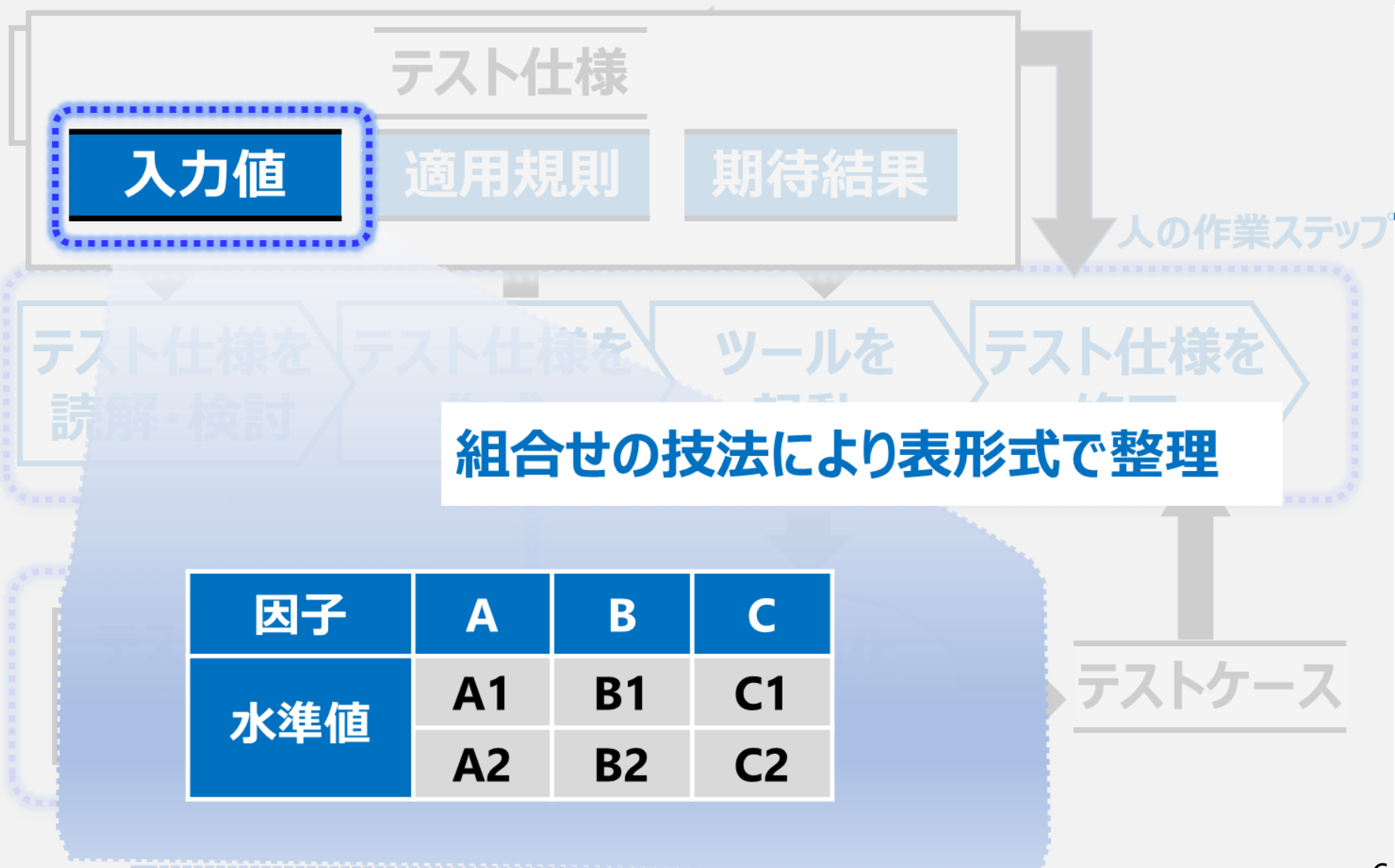
テスト設計の自動化 作業ステップ



テスト設計の自動化 テスト仕様



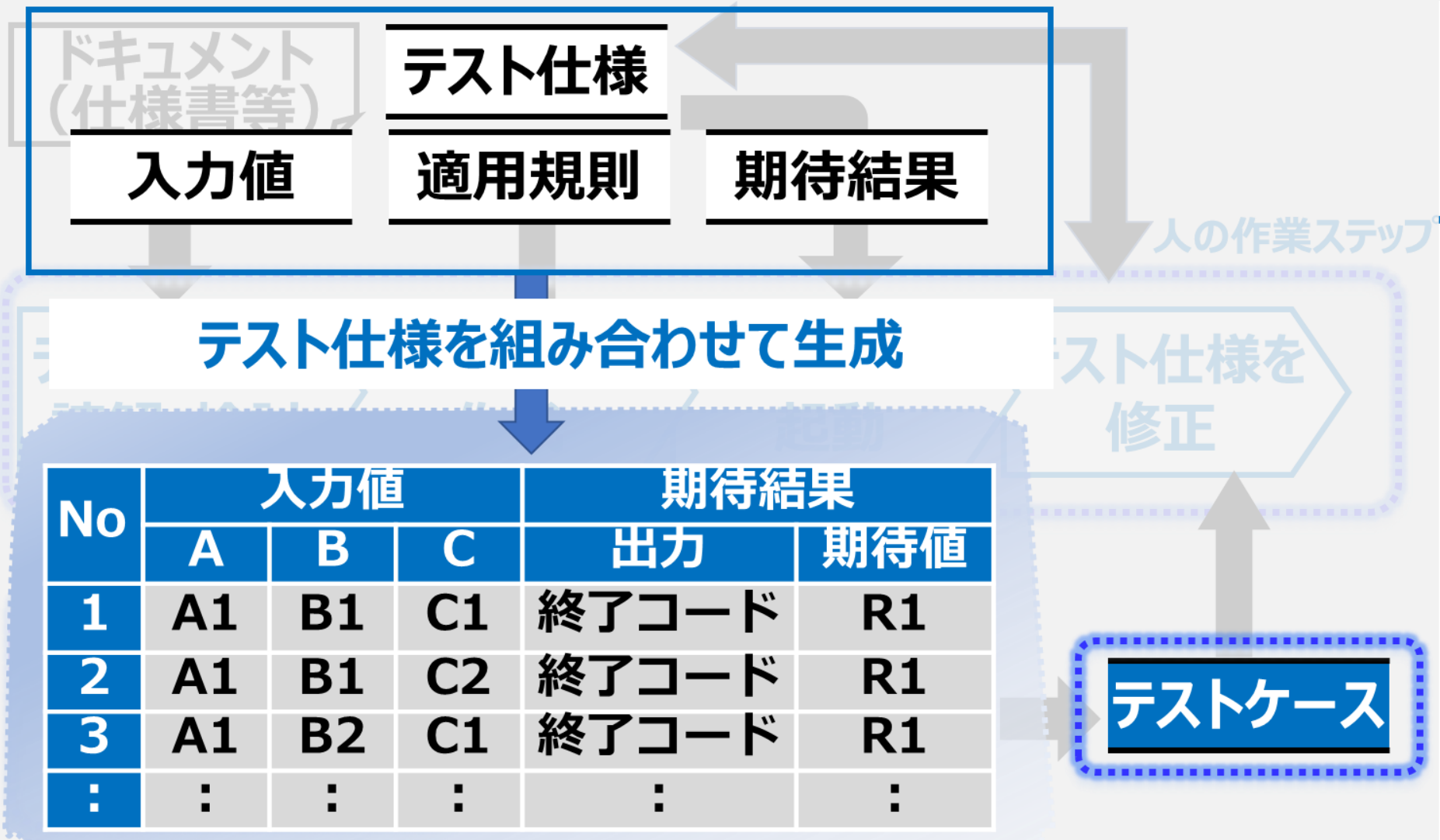
テスト設計の自動化 入力値



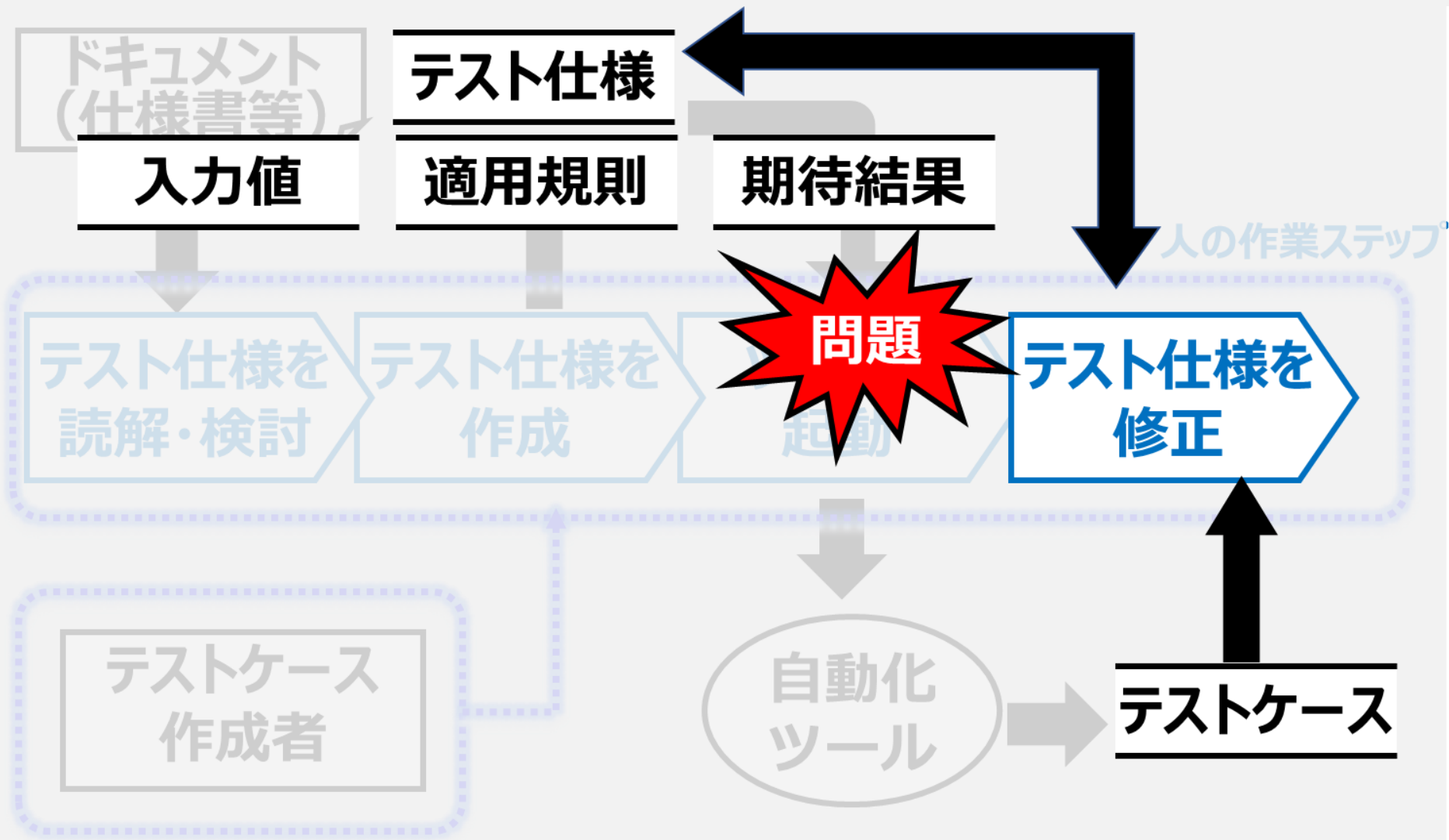
テスト設計の自動化 適用規則・期待結果



テスト設計の自動化 テストケース



問題ステップ° テスト仕様を修正



例 1 テスト仕様の誤り（修正し易い）

テストケース

No	入力値			期待結果	
	A	B	C	出力	期待値
1	A1	B1	C1	終了コード	R1,R2
:	:	:	:	:	:
5	A2	B1	C1	終了コード	
:	:	:	:	:	:

重複

漏れ

テスト仕様

因子	適用規則			期待結果	
	A	B	C	出力	期待値
水準値	A1	任意値	任意値	終了コード	R1
	A1	任意値	任意値	終了コード	R2
	A2	任意値	任意値	終了コード	

誤り箇所

誤り箇所

誤り箇所

テスト仕様の誤り箇所が明確

例 2 テスト仕様の誤り（修正し難い）

テストケース

No	入力値			期待結果	
	A	B	C	出力	期待値
:	:	:	:	:	:
3	A1	B2	C1	終了コード	R1,R3
:	:	:	:	:	:
8	A2	B2	C2	終了コード	

重複

漏れ

テスト仕様

適用規則				期待結果	
因子	A	B	C	出力	期待値
水準値	A1	任意値	任意値	終了コード	R1
	A2	B1	任意値	終了コード	R2
	任意値	B2	C1	終了コード	R3

テスト仕様の誤り有無が不明、誤り箇所も不明

テスト仕様の誤り箇所特定（1/2）

重複

				適用規則		
No	テスト入力値			因子	A	B
	A	B	C	水準値	A1	任意値
1	A1	B1	C1		A2	任意値
2	A1	B1	C2		任意値	B2
3	A1	B2	C1			C1
4	A1	B2	C2			
5	A2	B1	C1			
6	A2	B1	C2			
7	A2	B2	C1			
8	A2	B2	C2			

誤り箇所

誤り箇所

誤り特定には、テストケースと適用規則の突き合わせが必要

テスト仕様の誤り箇所特定 (2/2)

漏れ

全てのテストケース

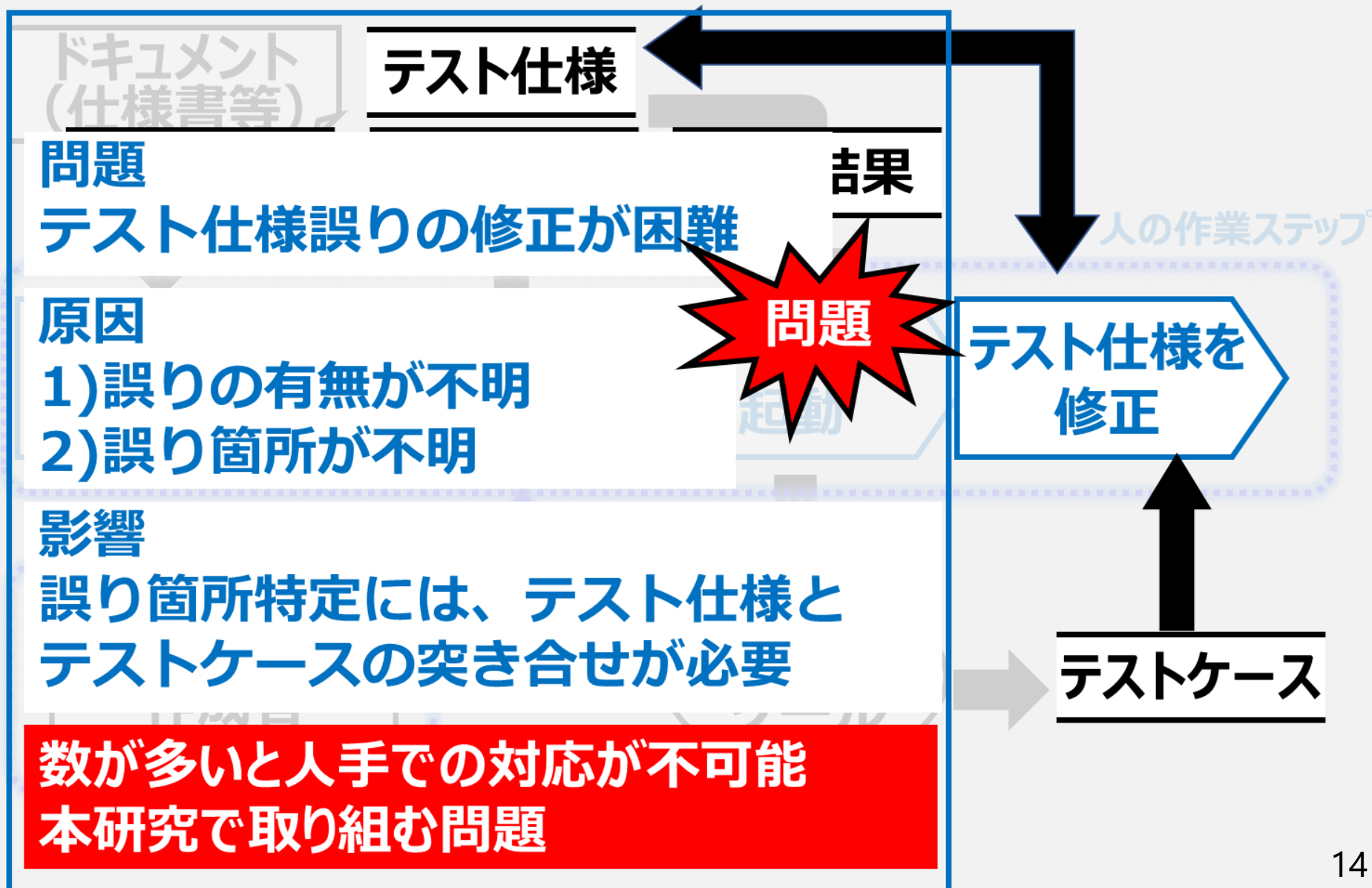
適用規則に該当

				適用規則		
No	テスト入力		因子	A	B	C
	A	B	水準値	A1	任意値	任意値
1	A1			A2	B1	任意値
2	A1			任意値	B2	C1
3	A1					
4	A1					
5	A2	B1				
6	A2	B1				
7		B2	C1			
8	A2	B2	C2			

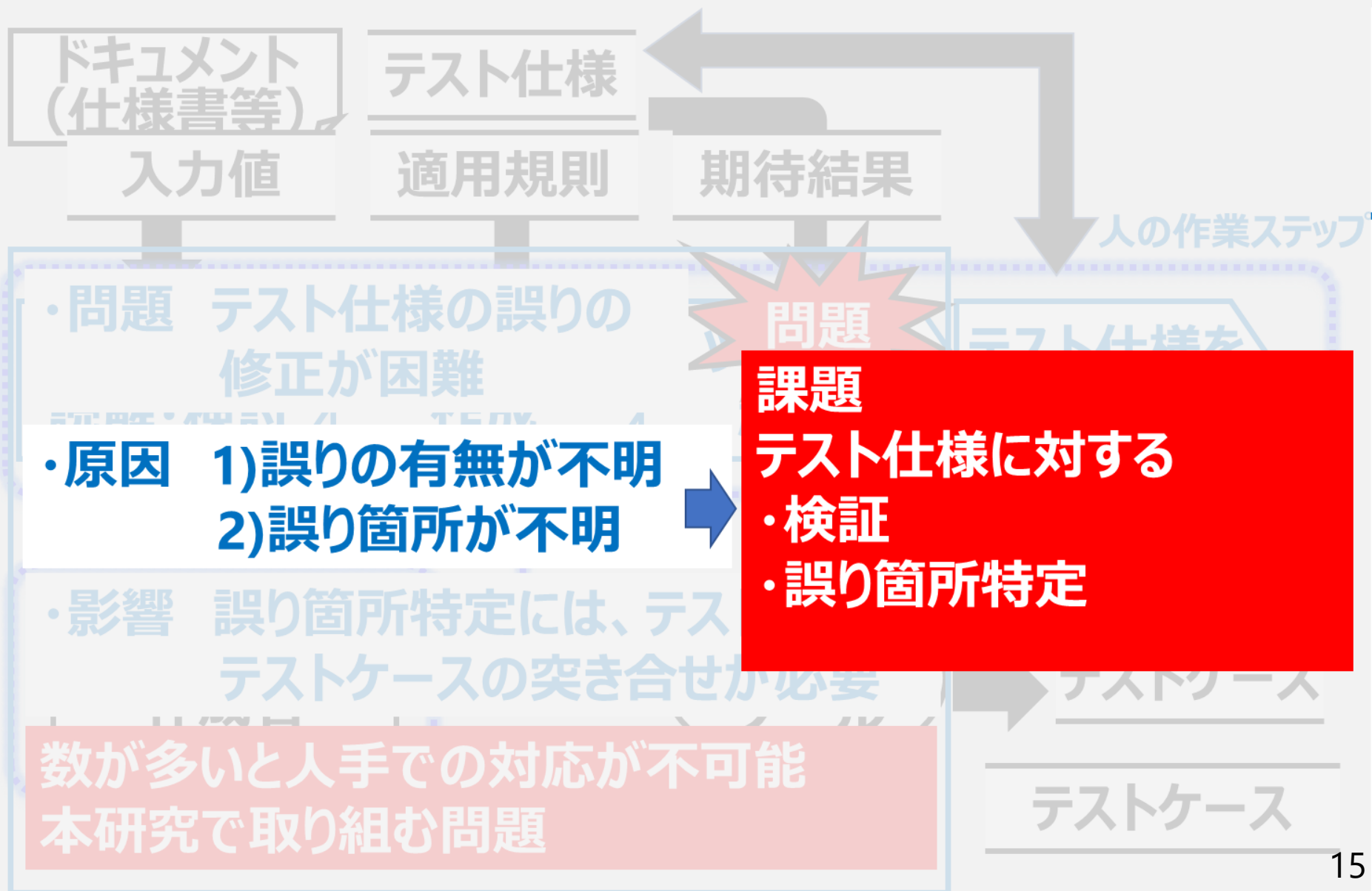
誤り箇所(この3つでは
No 8が漏れる)

誤り特定には、テストケースと適用規則の突き合わせが必要

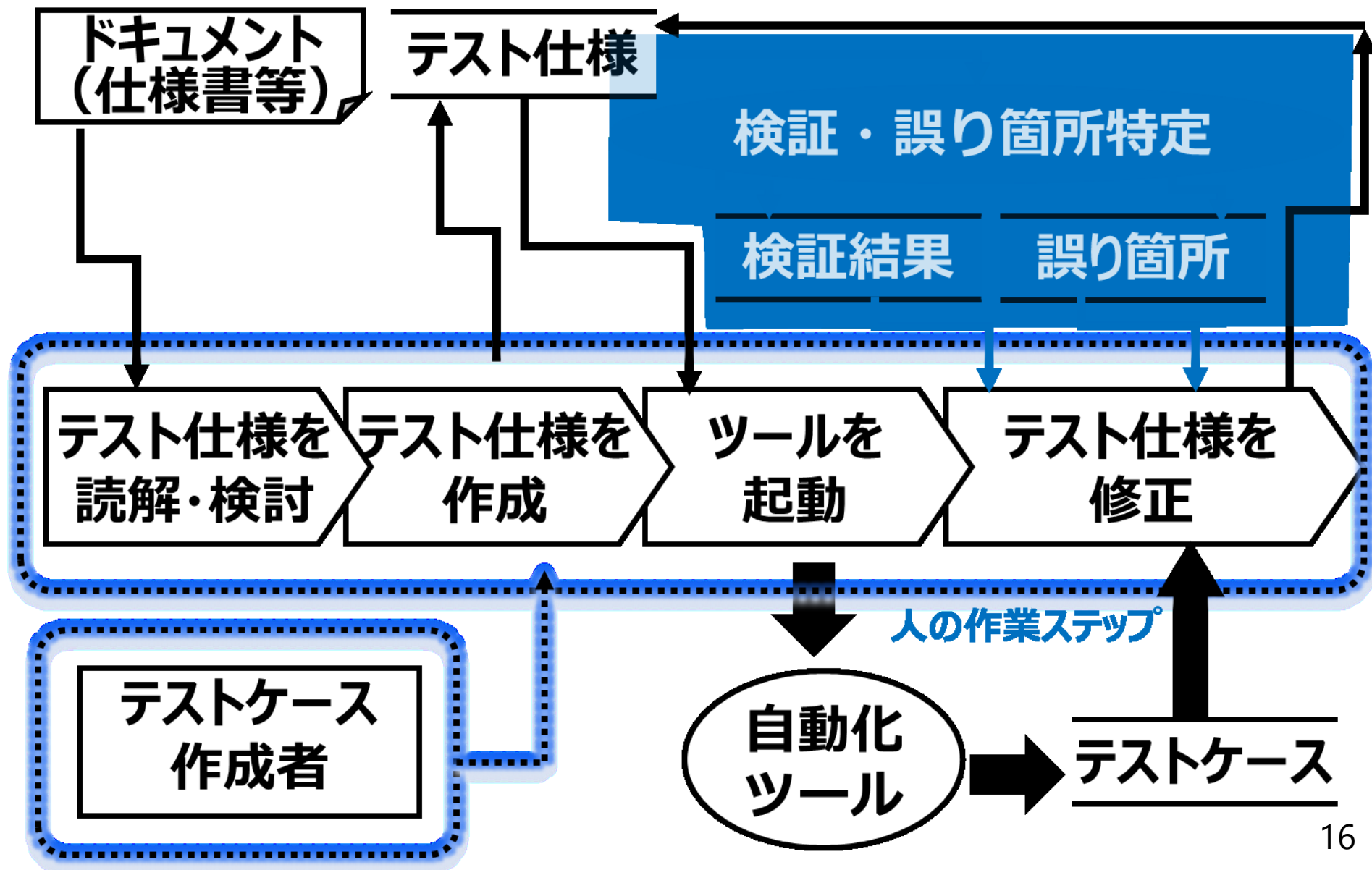
問題のまとめ

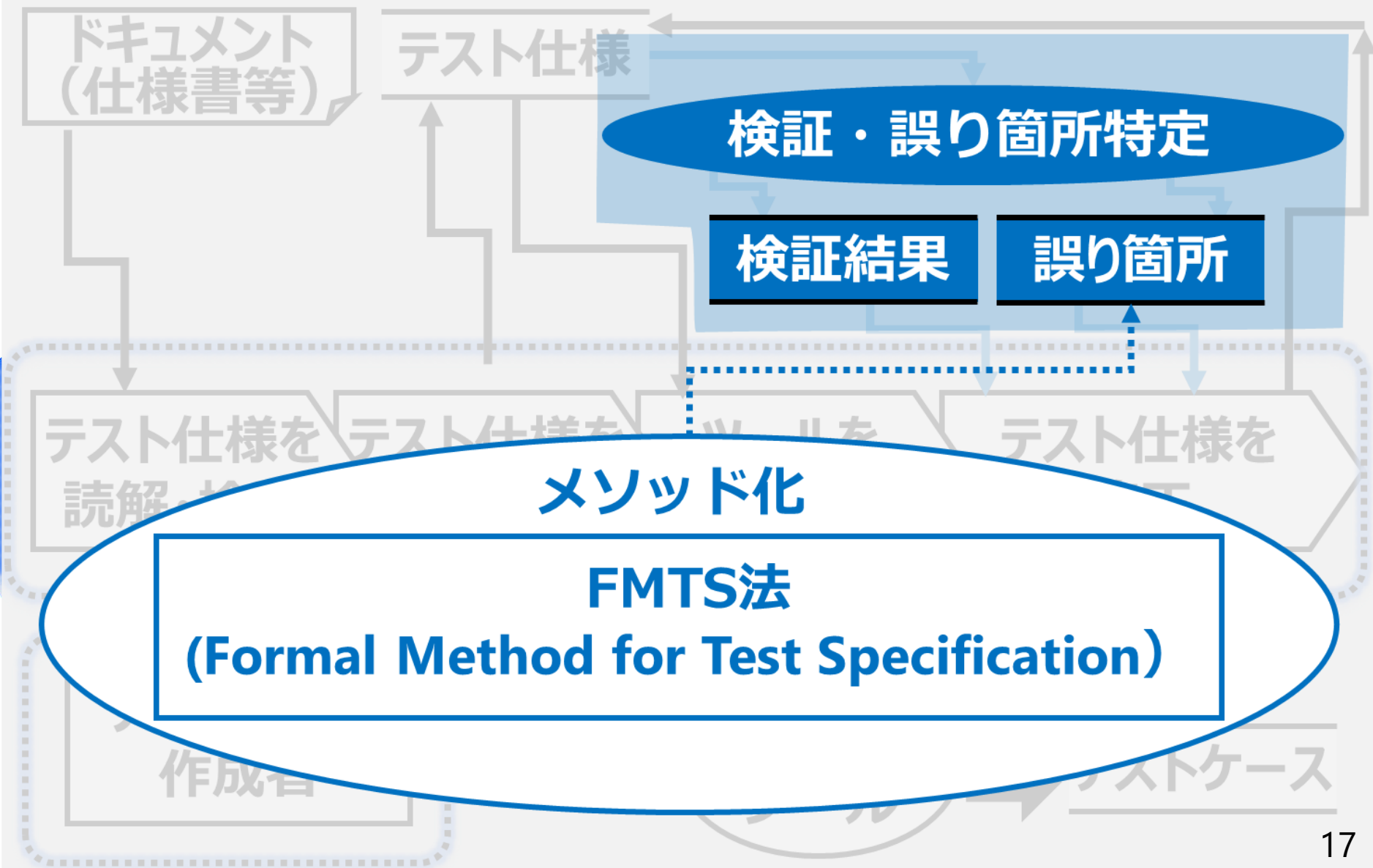


課題



解決策





FMTS法

検証

重複・漏れの有無を「検証結果」に出力

誤り箇所特定

「誤り箇所」として以下を出力

- ・ 重複した適用規則
- ・ 漏れたテストケースと関連適用規則

形式手法の利用

形式手法

- 形式仕様言語
厳密な定義
- 形式検証の方法
自動検証
- 支援ツール
デバッグ用表示

形式手法

- 形式仕様言語
厳密な定義

テスト仕様定義

- 形式検証の方法
自動検証

検証

誤り箇所特定

- 支援ツール
デバッグ用表示

検証結果表示

誤り箇所表示

FMTS法実装例 Alloyの採用

形式手法

- 形式仕様言語
厳密な定義

- 形式検証の方法
自動検証

- 支援ツール
デバッグ用表示

Alloy

- Alloy言語

- Alloy解析器

- メッセージ、ビューア

Alloy

1) 定義：表現しやすいAlloy言語

プロパティにより複数データを関連付け

※P38付録1参照

2) 解析：使いやすいAlloy解析器

即座に検証結果を取得

※P27参照

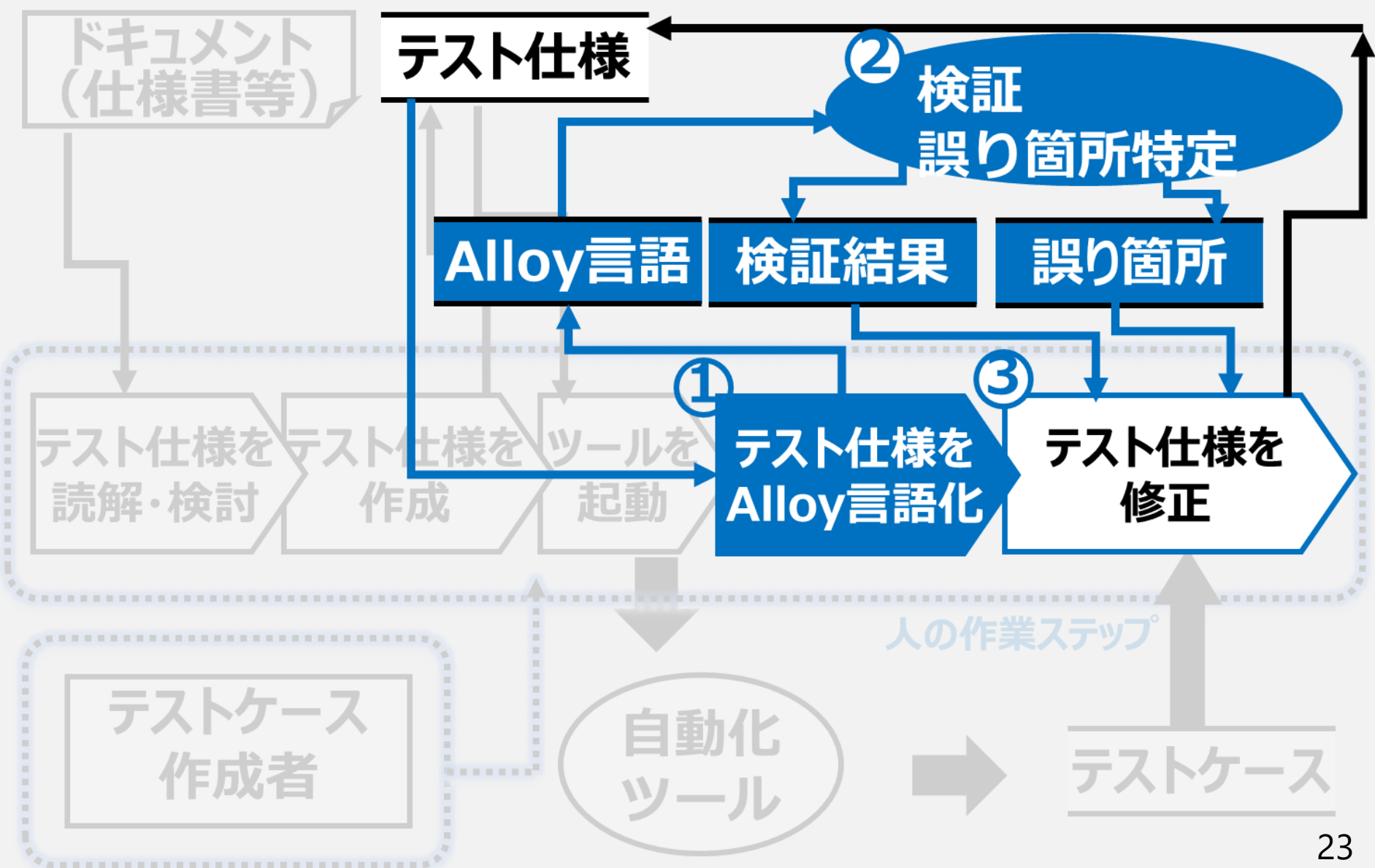
3) 出力：見やすいメッセージ、ビューア

判例情報から誤り箇所特定

情報をメッセージ・XMLで参照

※P28～P32参照

FMTS法実装例 実行手順



FMTS法実装例 Alloyを使った構成 (1/2)

Alloy

1) 定義 : Alloy言語

※P39付録2～P47付録6参照

クラス

型定義

インスタンス

テスト仕様定義

制約

検証用データ定義

重複検証ルール

漏れ検証ルール

2) 解析 : Alloy解析器

検証

誤り箇所特定

3) 出力 : メッセージ、ビューア

検証結果

誤り箇所

1) 定義 : Alloy言語

クラス

実装済
(FMTS法
から提供)

インスタンス

利用者が
テスト仕様を
Alloy言語化

制約

実装済
(FMTS法
から提供)

2) 解析 : Alloy解析器

Alloyをそのまま利用

3) 出力 : メッセージ、ビューア

Alloyをそのまま利用

評価方法

1) 事前準備

- ・ 誤りがあるケース、無いケースの2パターンの評価用データ（テスト仕様）、正解データ（テストケース）を作成 ※P48～P51付録7参照
- ・ テスト仕様をAlloy言語で定義
※P41～P42付録3参照

2) 実装例の実行

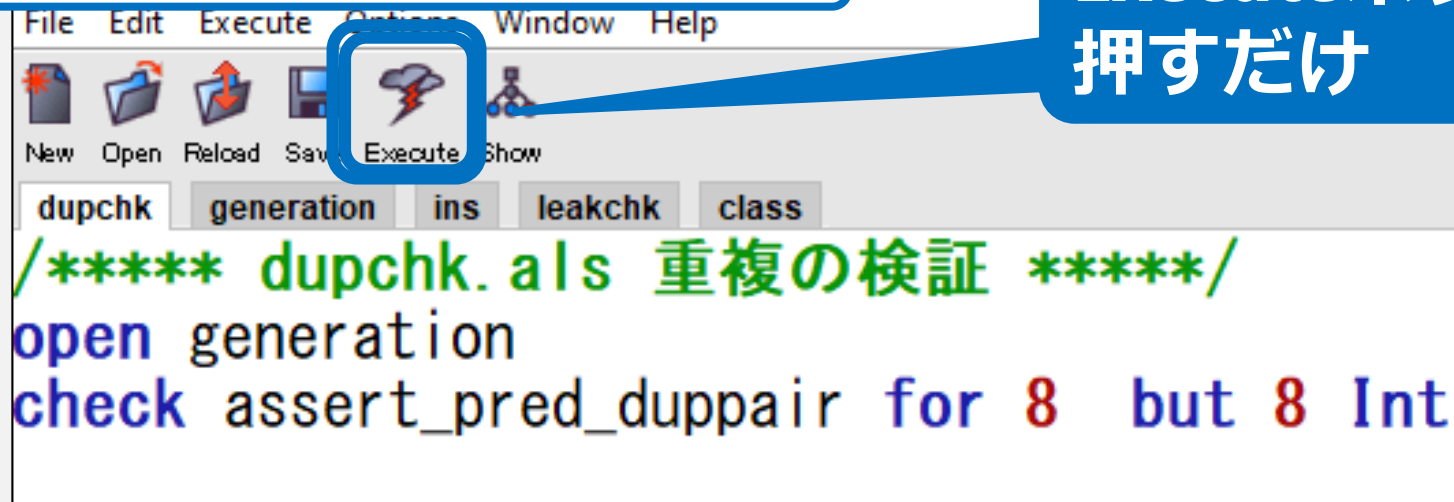
- ・ 検証・誤り箇所特定を実施

3) 結果確認

- ・ 出力（検証結果、誤り箇所）の確認
- ・ 出力と正解データの比較による評価

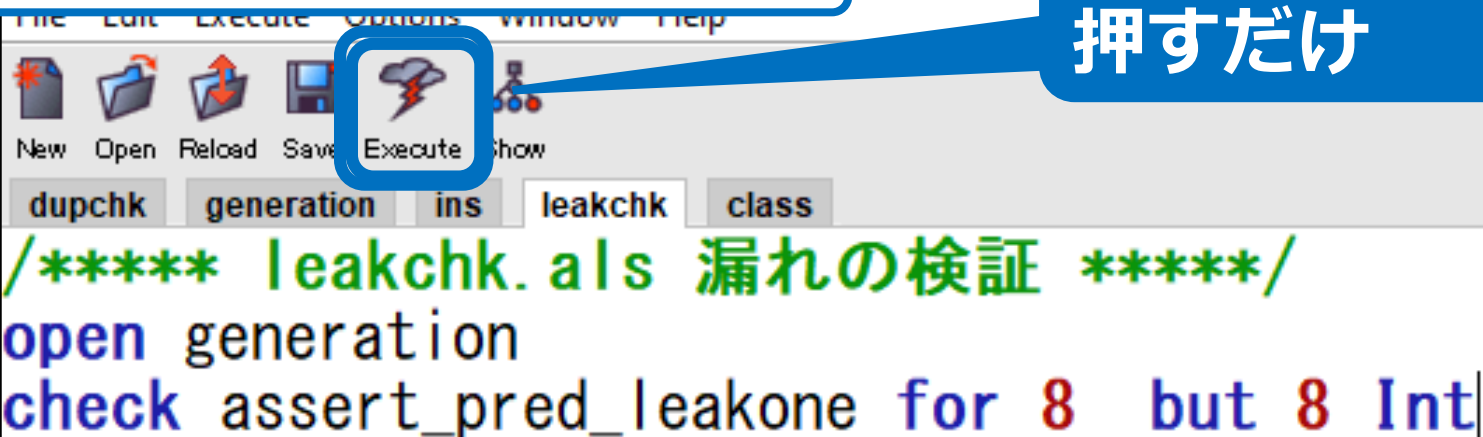
実装例の実行

重複の検証・誤り原因特定



Executeボタンを
押すだけ

漏れの検証・誤り原因特定



Executeボタンを
押すだけ

結果確認 検証結果 誤りが有るケース

・重複の検証結果⇒重複有り

```
Executing "Check assert_pred_duppair for 8 but 8 int"  
Solver=sat4j Bitwidth=8 MaxSeq=8 SkolemDepth=1 Symmetry=20  
185052 vars. 10966 primary vars. 430875 clauses. 3906ms.  
Counterexample found. Assertion is invalid. 1282ms.
```

・漏れの検証結果⇒漏れ有り

```
Executing "Check assert_pred_leakone for 8 but 8 int"  
Solver=sat4j Bitwidth=8 MaxSeq=8 SkolemDepth=1 Symmetry=20  
184990 vars. 10846 primary vars. 432193 clauses. 3000ms.  
Counterexample found. Assertion is invalid. 640ms.
```

評価結果：重複が有る事、漏れが有る事を検証可能

結果確認 検証結果 誤りが無いケース

・重複の検証結果⇒重複無し

```
Executing "Check assert_pred_duppair for 8 but 8 int"  
Solver=sat4j Bitwidth=8 MaxSeq=8 SkolemDepth=1 Symmetry=20  
185052 vars. 10966 primary vars. 430875 clauses. 2735ms.  
No counterexample found. Assertion may be valid. 891ms.
```

・漏れの検証結果⇒漏れ無し

```
Executing "Check assert_pred_leakone for 8 but 8 int"  
Solver=sat4j Bitwidth=8 MaxSeq=8 SkolemDepth=1 Symmetry=20  
184990 vars. 10846 primary vars. 432193 clauses. 2875ms.  
No counterexample found. Assertion may be valid. 297ms.
```

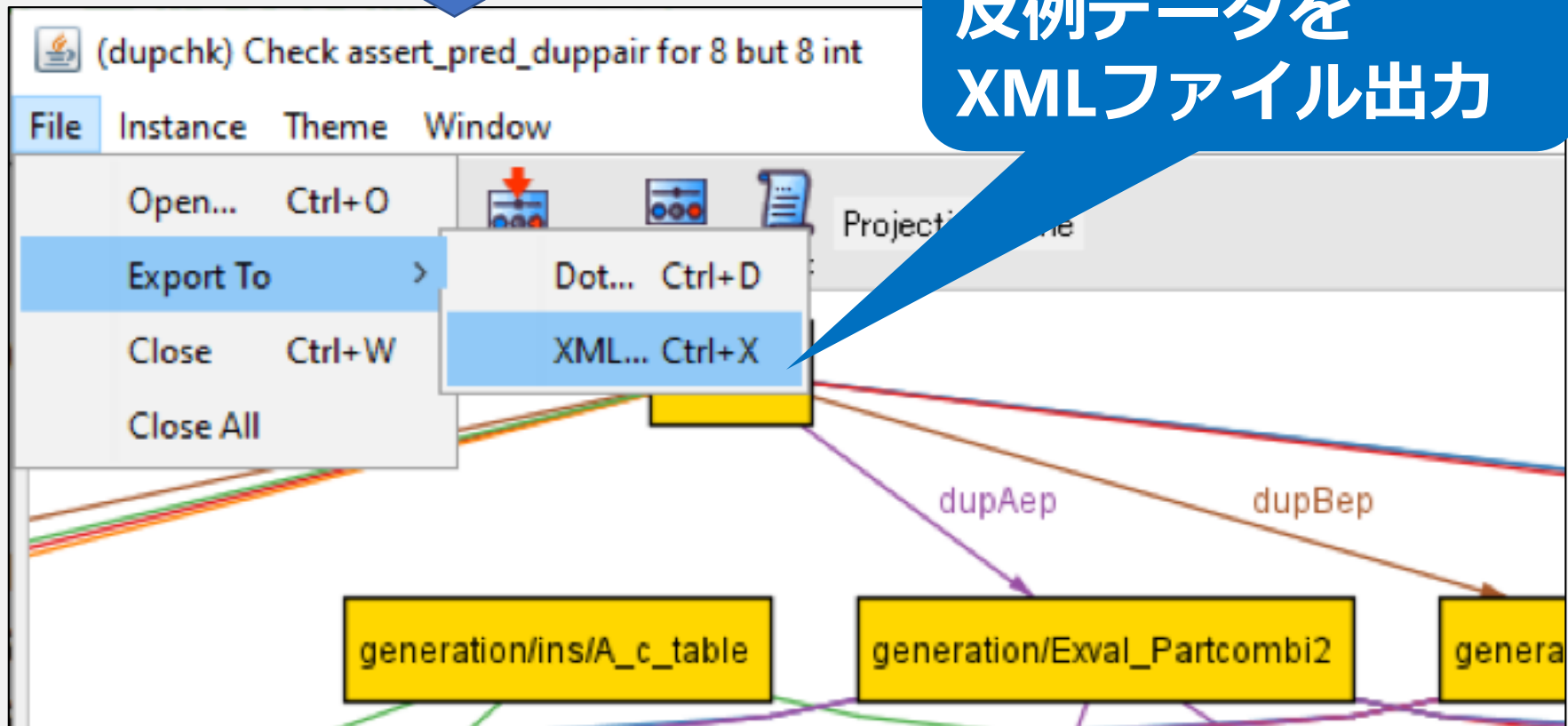
評価結果：重複が無い事、漏れが無い事を検証可能

結果確認 誤り箇所特定 反例確認の手順

Executing "Check assert
Solver=sat4j Bitwidth=
185052 vars. 10966 pr
Counterexample found.

クリックし反例を
ビューアで表示

ビューアから
反例データを
XMLファイル出力



結果確認 誤り箇所特定 重複

<sig label="this/DupPair . .
: : :
. . label="dupA_seqalv . .
. . . . /ins/B2\$0
. . . . /ins/C1\$0
: : :
. . label="dupB_seqalv
. . . . /ins/A1\$0

XMLファイルで
DupPairを確認

重複した
適用規則

適用規則				期待結果	
因子	A	B	C	出力	期待値
水準値	A1	任意値	任意値	終了コード	R1
	A2	B1	任意値	終了コード	R2
	任意値	B2	C1	終了コード	R3

評価結果：誤り箇所を特定可能

結果確認 誤り箇所特定 漏れ

<sig label="this/Leaks · ·

```
• • label="diff_alv_seq" • •
```

• • • • /ins/A2\$0

• • • • /ins/B2\$0

漏れた

漏れた テスト ケース

• • **label="leak_seqalv"** • •

• • • **/ins/A2\$0**

• • • /ins/B1\$0

• • • • /ins/B2\$0

• • • /ins/C1\$0

関連適用規則

適用規則

ケース力値				期待値	因子	A	B	C
No	A	B	C	出力	水準値	A1	任意値	任意値
1	A1	B1	C1	終了コード		A2	B1	任意値
	:	:	:	:		任意値	B2	C1
8	A2	B2	C2	終了コード				

漏れ

評価結果：誤り箇所を特定可能

FMTS法により以下の課題を解決可能

検証

重複・漏れの有無を「検証結果」に出力

➡ 評価結果：○

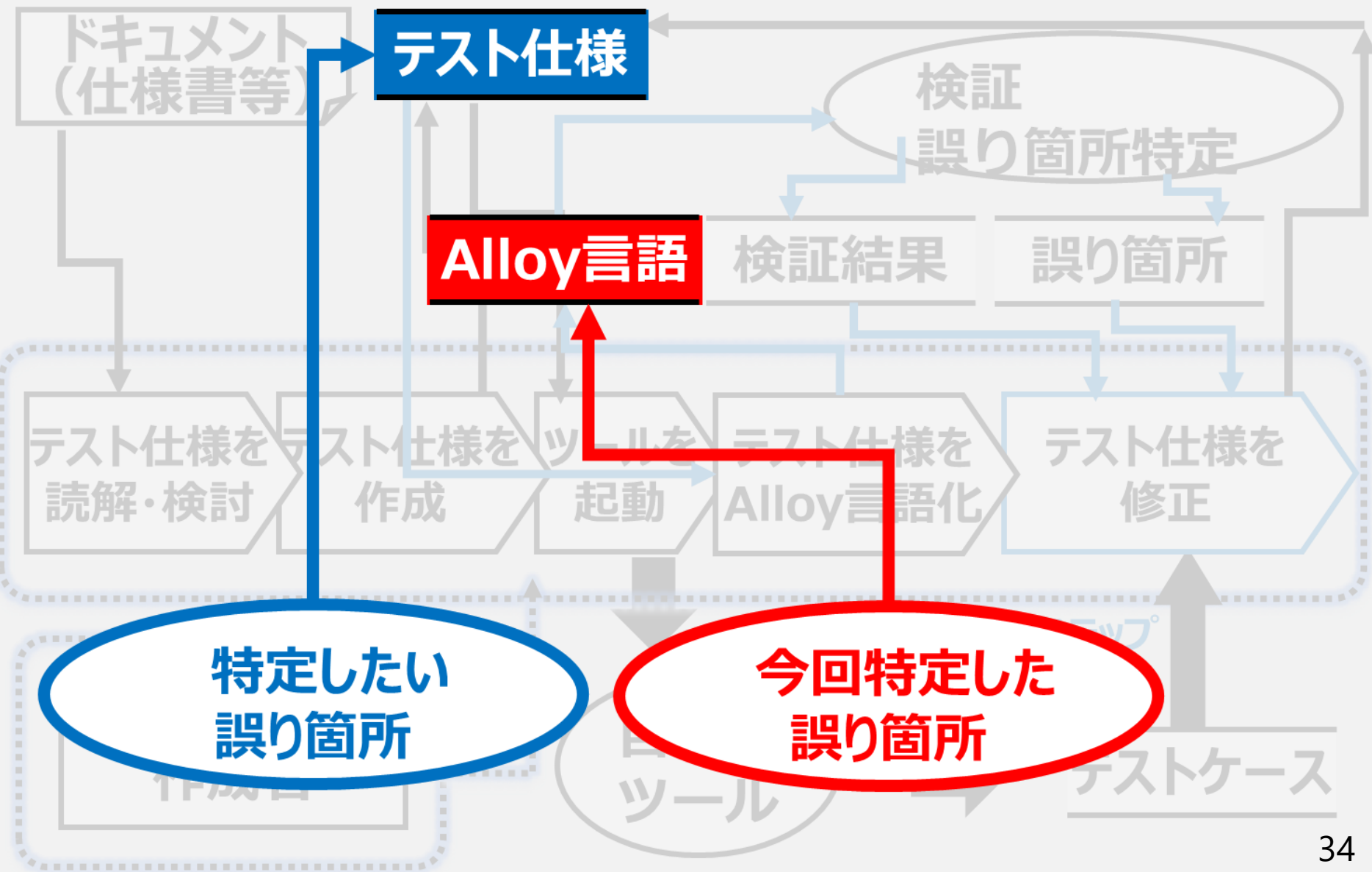
誤り箇所特定

「誤り箇所」として以下を出力

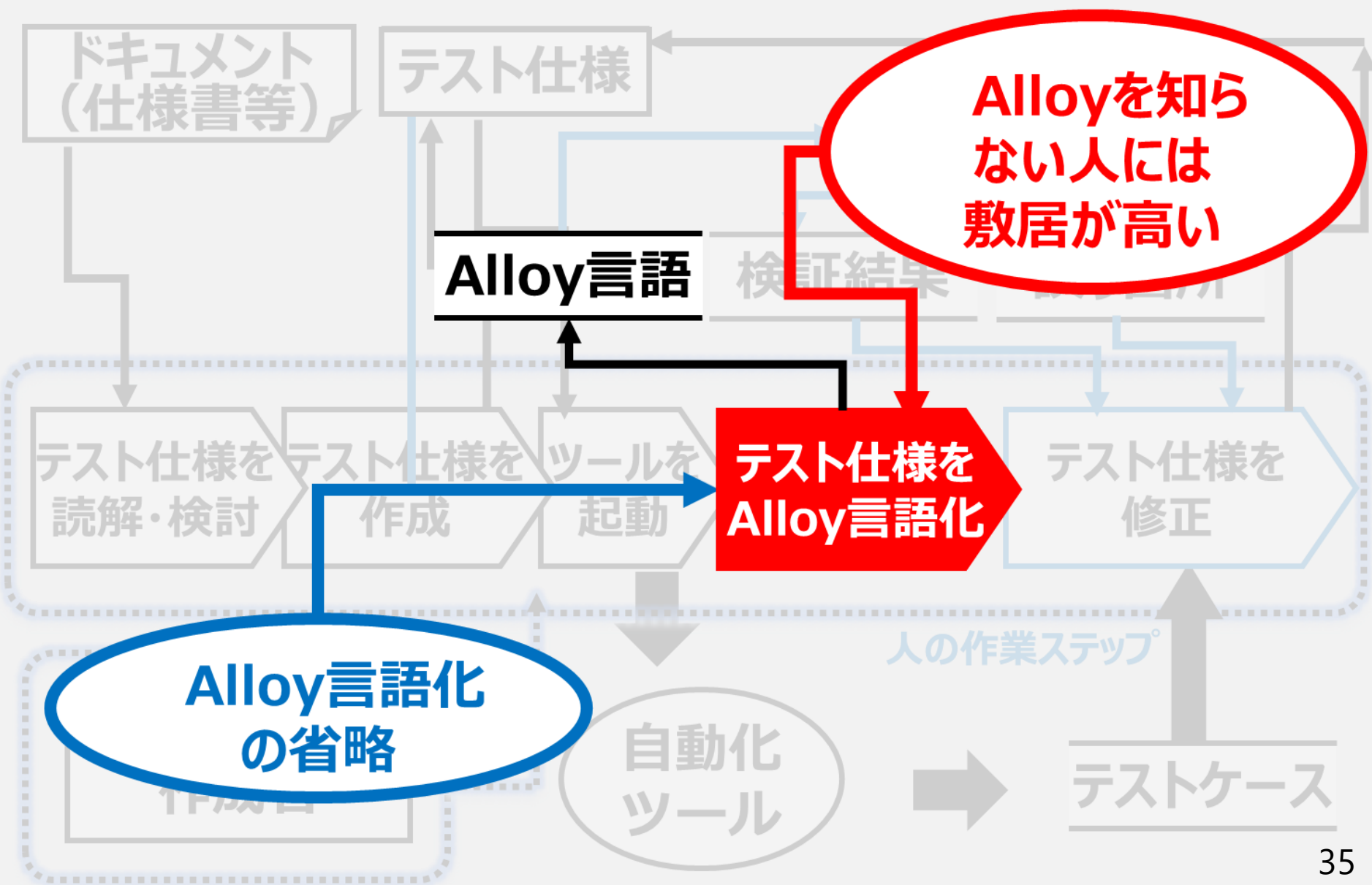
- ・ 重複した適用規則
- ・ 漏れたテストケースと関連適用規則

➡ 評価結果：○

残課題 (1/2)



残課題 (2/2)



ご清聴ありがとうございました

参考文献

- [1] 丹野治門, テスト自動化の基本的な考え方とNTTにおける研究開発の紹介, 第25回SPIトワイライトフォーラム, 2019
- [2] SQuBOK 策定部会編, ソフトウェア品質知識体系ガイド (第2版) – SQuBOK Guide V2 –, オーム社, 2014
- [3] Yogesh Singh, Automated Expected Output Generation-Is this a Problem that has been solved?, ACM SIGSOFT Software Engineering Notes, v.40 n.6, 2015
- [4] E.T. Barr, M. Harman, P. McMinn, M. Shahbaz, S. Yoo, The Oracle Problem in Software Testing: A Survey, IEEE Transactions on Software Engineering, vol 41 no.5, 507-525, 2015
- [5] S. J. Prowell and J. H. Poore, Foundations of sequence-based software specification, IEEE Transactions on Software Engineering, vol. 29 no. 5, 417-429, 2003
- [6] M. Balcer, W. Hasling, and T. Ostrand, Automatic generation of test scripts from formal test specifications, ACM SIGSOFT Software Engineering Notes, v14 n.8, 210-218, 1989
- [7] 林祥一, <https://qiita.com/sho1884/items/283f55c4e7e1ae374d74>, (2020年1月29日参照)
- [8] 石川冬樹, 「形式手法と仕様記述」の探求に向けて～形式手法とは?, 日本科学技術連盟 SQiP研究会「形式仕様と仕様記述」, 2012
- [9] D.Jackson, 抽象によるソフトウェア設計: Alloyではじめる形式手法, 中島震 (監訳), オーム社, 2011
- [10] 中島震, ソフトウェア工学の道具としての形式手法 Formal Methods as Software Engineering Tools, NII-2007-007J, National Information of Infomatics, 2007
- [11] 中島震, 形式手法入門ーロジックによるソフトウェア設計, オーム社, 2012
- [12] 中島 震, モデル検査法のソフトウェアデザイン検証への応用, コンピュータ ソフトウェア, 23 巻, 2 号, p. 2_72-2_86, 2006

付録1 Alloyの利点

表現しやすいAlloy言語

```
sig Command_Arg {  
  order : one Arg_Order,  
  level : some Arg_Level }
```

```
sig Arg_Level {  
  com_arg_id : one Arg_Order,  
  value : one Seq_ALV }
```

```
sig Seq_ALV {seq_alv : seq Arg_Level_Value}
```

付録2 class.als(データの型) (1/2)

```
/***** class.als 型の定義 *****/
```

```
//入力値
```

```
one sig Inputdata_Table { commands : some Command }
fact {Inputdata_Table.commands = Command
  all disj i1, i2 : Inputdata_Table | i1.commands != i2.commands }
sig Command { name : one Command_Name,
  order: one Execution_Order,
  args: some Command_Arg }
fact {Command.name = Command_Name }
fact {Command.order = Execution_Order
  all disj cm1, cm2 : Command |cm1. order !=cm2.order }
fact {Command.args = Command_Arg
  all disj cm1, cm2 : Command |cm1. args !=cm2.args }
sig Command_Name {}
sig Execution_Order{}
sig Command_Arg {
  order : one Arg_Order,
  level : some Arg_Level }
fact {
  Command_Arg.order = Arg_Order
  Command_Arg.level = Arg_Level
  all disj cma1, cma2 : Command_Arg |cma1. order !=cma2.order
  all disj cma1, cma2 : Command_Arg |cma1. level !=cma2.level }
sig Arg_Order{}
sig Arg_Level {
  com_arg_id : one Arg_Order,
  value : one Seq_ALV }
fact{
  Arg_Level.value = Seq_ALV
  all cm : Command | all cma : cm.args | all cmal : cma.level |
    cmal.com_arg_id = cma.order
  all disj cm : Command |
    all cma : cm.args |
      all disj cmal1, cmal2 : cma.level |
        cmal1.value != cmal2.value }
sig Seq_ALV {seq_alv : seq Arg_Level_Value}
{#(seq_alv.inds) = #(seq_alv.elems)}
sig Arg_Level_Value {}
```

```
//期待結果
```

```
one sig ExpectedValue_Condition_Table {
  commands : some Out_Command }
fact {
  ExpectedValue_Condition_Table.commands = Out_Command
  all disj ect1, ect2 : ExpectedValue_Condition_Table |
    ect1.commands != ect2.commands }
sig Out_Command {
  name : one Command_Name,
  order : one Execution_Order,
  retcode : set Out_Type_Rcode,
  stdout : set Out_Type_Stdout,
  stderr : set Out_Type_Stderr }
fact {
  Out_Command.retcode = Out_Type_Rcode
  Out_Command.stdout = Out_Type_Stdout
  Out_Command.stderr = Out_Type_Stderr
  all disj oc1, oc2 : Out_Command | oc1.order != oc2.order
  all disj oc1, oc2 : Out_Command | oc1.retcode != oc2.retcode
  all disj oc1, oc2 : Out_Command | oc1.stdout != oc2.stdout
  all disj oc1, oc2 : Out_Command | oc1.stderr != oc2.stderr }
sig TypeName {}{ TypeName = TypeR + TypeO + TypeE }
one sig TypeR, TypeO, TypeE extends TypeName{}
abstract sig Out_Type{}
sig Out_Type_Rcode extends Out_Type{
  name : one TypeName,
  value : one Rcode_Expected_Value,
  condition : some Rcode_Condition}
{ name = TypeR }
fact {
  Out_Type_Rcode.value = Rcode_Expected_Value
  Out_Type_Rcode.condition = Rcode_Condition
  all ocom : Out_Command |
    all disj orc1, orc2 : ocom.retcode |
      orc1.value != orc2.value
  all disj otr1, otr2 : Out_Type_Rcode | otr1.condition != otr2.condition }
sig Out_Type_Stdout extends Out_Type {
  name : one TypeName,
  value : one Stdout_Expected_Value,
  condition : some Stdout_Condition}
{ name = TypeO }
fact {
  Out_Type_Stdout.value = Stdout_Expected_Value
  Out_Type_Stdout.condition = Stdout_Condition
  all ocom : Out_Command |
    all disj oot1, oot2 : ocom.stdout |
      oot1.value != oot2.value
  all disj ots1, ots2 : Out_Type_Stdout | ots1.condition != ots2.condition }
```

付録2 class.als(データの型) (2/2)

```
sig Out_Type_Stderr extends Out_Type{
  name : one TypeName,
  value : one Stderr_Expected_Value,
  condition : some Stderr_Condition}
{ name = TypeE }
fact {
  Out_Type_Stderr.value = Stderr_Expected_Value
  Out_Type_Stderr.condition = Stderr_Condition
  all ocom : Out_Command |
    all disj oer1, oer2 : ocom.stderr |
      oer1.value != oer2.value
  all disj otse1, otse2 : Out_Type_Stderr |
    otse1.condition != otse2.condition }

//適用規則
one sig Applicable_Condition_Table
{conditions : some Condition }
fact {
  Applicable_Condition_Table.conditions = Condition
  all disj act1, act2 : Applicable_Condition_Table |
    act1.conditions != act2.conditions }
sig Condition {
  name : one Condition_Name,
  args_and : some Cond_Args_AND,
  pc_num : Int }
fact {
  Condition.name = Condition_Name
  Condition.args_and = Cond_Args_AND
  all disj cd1, cd2 : Condition | cd1.name != cd2.name
  all disj cd1, cd2 : Condition | cd1.args_and != cd2.args_and }
sig Condition_Name{}
sig Cond_Args_AND {
  id : one Cond_Args_AND_ID,
  order_in_cond : Int,
  name : one Condition_Name,
  order: one Arg_Order,
  levels_or : some Cond_Levels_OR,
  p_combi_caa : some Arg_Level, /* Set automatically */
  values : some Seq_ALV /* Set automatically */ }
fact {
  Cond_Args_AND.levels_or = Cond_Levels_OR
  all disj caa1, caa2 : Cond_Args_AND | caa1.levels_or != caa2.levels_or
  all caa3 : Cond_Args_AND | caa3.p_combi_caa = caa3.levels_or.p_combi_clor
  all caa4 : Cond_Args_AND | caa4.values = caa4.p_combi_caa.value }
```

```
sig Cond_Levels_OR{
  value : one Seq_ALV,
  p_combi_clor : one Arg_Level /* Set automatically */ }
sig Cond_Args_AND_ID {}
fact {
  all clor1 : Cond_Levels_OR | one ar1 : Arg_Level |
    clor1.value = ar1.value and
    clor1.p_combi_clor = ar1 }
abstract sig TypeCond { name : one Condition_Name }
sig Rcode_Condition extends TypeCond { }
fact {
  all ocom : Out_Command |
    all orc : ocom.retcode |
      all disj orcc1, orcc2 : orc.condition |
        orcc1.name != orcc2.name }
sig Stdout_Condition extends TypeCond { }
fact {
  all ocom : Out_Command |
    all oout : ocom.stdout |
      all disj ooutc1, ooutc2 : oout.condition |
        ooutc1.name != ooutc2.name }
sig Stderr_Condition extends TypeCond { }
fact {
  all ocom : Out_Command |
    all oerr : ocom.stderr |
      all disj oerrc1, oerrc2 : oerr.condition |
        oerrc1.name != oerrc2.name }
abstract sig Expected_Value {}
sig Rcode_Expected_Value extends Expected_Value {}
sig Stdout_Expected_Value extends Expected_Value {}
sig Stderr_Expected_Value extends Expected_Value {}

//インスタンス固有定数
one sig combinum {num : Int} // テスト入力値の数
one sig argordernum {num : Int} // Arg_Orderの数
sig InputCombi { // InputCombi(テスト入力値)
  combi : some Arg_Level,
  v_combi: some Seq_ALV}
sig Exv_Ic { // Exv_Ic(期待値毎のテスト入力値)
  exval : one Expected_Value,
  ex_ic : some InputCombi}
sig Com_Type_Exvlc{ // Com_Type_Exvlc(漏れ検証用テスト入力値)
  com_name : one Command_Name,
  type_name : one TypeName,
  ex_ic : some InputCombi}
```


付録3 ins.als(テスト仕様・誤り有り) (1/2)

/***** ins.als テスト仕様・入力データ定義 *****/

```
open class
//インスタンス固有定数
fact{ combinum.num = 8 // テスト入力値数
      argordernum.num = 3} // Arg_Orderの数

//入力値
one sig I_table extends Inputdata_Table{}
one sig CM1 extends Command{}
one sig CM1_name extends Command_Name{}
one sig CM1_order extends Execution_Order {}
one sig CM1_A,CM1_B,CM1_C extends Command_Arg {}
one sig CM1_A_order,CM1_B_order,CM1_C_order extends Arg_Order{}
one sig CM1_A_A1,CM1_A_A2,CM1_B_B1,CM1_B_B2,CM1_C_C1,CM1_C_C2
      extends Arg_Level {}
one sig SeqA1,SeqA2,SeqB1,SeqB2,SeqC1,SeqC2 extends Seq_ALV {}
one sig A1,A2,B1,B2,C1,C2 extends Arg_Level_Value {}
fact {I_table.commands = CM1
      Command      = CM1
      CM1.name     = CM1_name
      Command_Name = CM1_name
      CM1.order    = CM1_order
      Execution_Order = CM1_order
      CM1.args     = CM1_A + CM1_B + CM1_C
      Command_Arg  = CM1_A + CM1_B + CM1_C
      CM1_A.order  = CM1_A_order
      CM1_B.order  = CM1_B_order
      CM1_C.order  = CM1_C_order
      Arg_Order    = CM1_A_order + CM1_B_order + CM1_C_order
      CM1_A.level  = CM1_A_A1 + CM1_A_A2
      CM1_B.level  = CM1_B_B1 + CM1_B_B2
      CM1_C.level  = CM1_C_C1 + CM1_C_C2
      Arg_Level    = CM1_A_A1 + CM1_A_A2 + CM1_B_B1 + CM1_B_B2 + CM1_C_C1 + CM1_C_C2
      CM1_A_A1.value = SeqA1
      CM1_A_A2.value = SeqA2
      CM1_B_B1.value = SeqB1
      CM1_B_B2.value = SeqB2
      CM1_C_C1.value = SeqC1
      CM1_C_C2.value = SeqC2
      Seq_ALV = SeqA1 + SeqA2 + SeqB1 + SeqB2 + SeqC1 + SeqC2
      Arg_Level_Value = A1 + A2 + B1 + B2 + C1 + C2
      SeqA1.seq_alv.elems = A1
      SeqA2.seq_alv.elems = A2
      SeqB1.seq_alv.elems = B1
      SeqB2.seq_alv.elems = B2
      SeqC1.seq_alv.elems = C1
      SeqC2.seq_alv.elems = C2
```

```
CM1_A_A1.com_arg_id = CM1_A_order
CM1_A_A2.com_arg_id = CM1_A_order
CM1_B_B1.com_arg_id = CM1_B_order
CM1_B_B2.com_arg_id = CM1_B_order
CM1_C_C1.com_arg_id = CM1_C_order
CM1_C_C2.com_arg_id = CM1_C_order }
```

//期待結果

```
one sig EV_c_table extends ExpectedValue_Condition_Table{}
one sig O_CM1 extends Out_Command{}
one sig O_CM1_RC1,O_CM1_RC2,O_CM1_RC3
      extends Out_Type_Rcode{}
one sig O_CM1_RC1_cond,O_CM1_RC2_cond,
      O_CM1_RC3_cond extends Rcode_Condition{}
one sig O_CM1_RC1_R1,O_CM1_RC2_R2,
      O_CM1_RC3_R3 extends Rcode_Expected_Value {}
one sig Exvic_rcode1,Exvic_rcode2,Exvic_rcode3 extends Exv_Ic{}
one sig CM1_RC_Exvic extends Com_Type_Exvlc{}
fact {
      EV_c_table.commands = O_CM1
      Out_Command      = O_CM1
      O_CM1.name       = CM1_name
      O_CM1.order      = CM1_order
      O_CM1.retcode     = O_CM1_RC1 + O_CM1_RC2 + O_CM1_RC3
      Out_Type_Rcode   = O_CM1_RC1 + O_CM1_RC2 + O_CM1_RC3
      O_CM1_RC1.value  = O_CM1_RC1_R1
      O_CM1_RC2.value  = O_CM1_RC2_R2
      O_CM1_RC3.value  = O_CM1_RC3_R3
      Rcode_Expected_Value = O_CM1_RC1_R1 + O_CM1_RC2_R2 + O_CM1_RC3_R3
      O_CM1_RC1.condition = O_CM1_RC1_cond
      O_CM1_RC2.condition = O_CM1_RC2_cond
      O_CM1_RC3.condition = O_CM1_RC3_cond
      Rcode_Condition = O_CM1_RC1_cond + O_CM1_RC2_cond + O_CM1_RC3_cond
      O_CM1_RC1_cond.name = Cond1_name
      O_CM1_RC2_cond.name = Cond2_name
      O_CM1_RC3_cond.name = Cond3_name
      O_CM1.stdout      = none
      Out_Type_Stdout   = none
      Stdout_Expected_Value = none
      Stdout_Condition  = none
      O_CM1.stderr     = none
      Out_Type_Stderr   = none
      Stderr_Expected_Value = none
      Stderr_Condition  = none
```

付録3 ins.als(テスト仕様・誤り有り) (2/2)

```
Exv_ic = Exvic_rcode1 + Exvic_rcode2 + Exvic_rcode3
Exvic_rcode1.exval = O_CM1_RC1_R1
Exvic_rcode2.exval = O_CM1_RC2_R2
Exvic_rcode3.exval = O_CM1_RC3_R3
CM1_RC_Exvic.com_name = CM1_name
CM1_RC_Exvic.type_name = TypeR
CM1_RC_Exvic.ex_ic
    = Exvic_rcode1.ex_ic + Exvic_rcode2.ex_ic + Exvic_rcode3.ex_ic
Com_Type_Exvlc = CM1_RC_Exvic

//適用規則
one sig A_c_table extends Applicable_Condition_Table{
one sig Cond1,Cond2,Cond3 extends Condition{
one sig Cond1_name,Cond2_name,Cond3_name extends Condition_Name{
one sig Cond1_args_and1, Cond2_args_and1,Cond2_args_and2,
    Cond3_args_and1,Cond3_args_and2 extends Cond_Args_AND{
one sig Cond1_args_and1_level1,
    Cond2_args_and1_level1,Cond2_args_and2_level1,
    Cond3_args_and1_level1,Cond3_args_and2_level1
    extends Cond_Levels_OR{
one sig Cond1_args_and1_ID,Cond2_args_and1_ID,Cond2_args_and2_ID,
    Cond3_args_and1_ID,Cond3_args_and2_ID
    extends Cond_Args_AND_ID {}
fact {
    A_c_table.conditions = Cond1 + Cond2 + Cond3
    Condition          = Cond1 + Cond2 + Cond3
    Cond1.name         = Cond1_name
    Cond2.name         = Cond2_name
    Cond3.name         = Cond3_name
    Condition_Name     = Cond1_name + Cond2_name + Cond3_name
    Cond1_args_and     = Cond1_args_and1
    Cond2_args_and     = Cond2_args_and1 + Cond2_args_and2
    Cond3_args_and     = Cond3_args_and1 + Cond3_args_and2
    Cond_Args_AND      = Cond1_args_and1 + Cond2_args_and1 + Cond2_args_and2 +
        Cond3_args_and1 + Cond3_args_and2
    Cond1.pc_num = 1
    Cond2.pc_num = 1
    Cond3.pc_num = 1
    Cond1_args_and1.name = Cond1_name
    Cond2_args_and1.name = Cond2_name
    Cond2_args_and2.name = Cond2_name
    Cond3_args_and1.name = Cond3_name
    Cond3_args_and2.name = Cond3_name
```

```
Cond1_args_and1.id = Cond1_args_and1_ID
Cond2_args_and1.id = Cond2_args_and1_ID
Cond2_args_and2.id = Cond2_args_and2_ID
Cond3_args_and1.id = Cond3_args_and1_ID
Cond3_args_and2.id = Cond3_args_and2_ID
Cond_Args_AND_ID
    = Cond1_args_and1_ID + Cond2_args_and1_ID + Cond2_args_and2_ID +
        Cond3_args_and1_ID + Cond3_args_and2_ID
Cond1_args_and1.order_in_cond = 0
Cond2_args_and1.order_in_cond = 0
Cond2_args_and2.order_in_cond = 1
Cond3_args_and1.order_in_cond = 0
Cond3_args_and2.order_in_cond = 1
Cond1_args_and1.order = CM1_A_order
Cond2_args_and1.order = CM1_A_order
Cond2_args_and2.order = CM1_B_order
Cond3_args_and1.order = CM1_A_order
Cond3_args_and2.order = CM1_B_order
Cond1_args_and1.levels_or = Cond1_args_and1_level1
Cond2_args_and1.levels_or = Cond2_args_and1_level1
Cond2_args_and2.levels_or = Cond2_args_and2_level1
Cond3_args_and1.levels_or = Cond3_args_and1_level1
Cond3_args_and2.levels_or = Cond3_args_and2_level1
Cond_Levels_OR
    = Cond1_args_and1_level1 +
        Cond2_args_and1_level1 + Cond2_args_and2_level1 +
        Cond3_args_and1_level1 + Cond3_args_and2_level1
Cond1_args_and1_level1.value = SeqA1
Cond2_args_and1_level1.value = SeqA2
Cond2_args_and2_level1.value = SeqB1
Cond3_args_and1_level1.value = SeqB2
Cond3_args_and2_level1.value = SeqC1 }
```

付録4 generation.als(検証用データ定義) (1/3)

```
/***** generation.als 検証用データ生成 *****/
open ins
//Execution_OrderからCommand_Nameを求める関数
fun cname_by_order [ex_o : Execution_Order] : one Command_Name {
  {c_name : Command_Name |
    all o_com : Out_Command |
      o_com.order = ex_o implies(c_name = o_com.name)}
}

fact {
  // Exv_ic(期待値毎のテスト入力値)の生成
  all ep1 : Exval_Partcombi |
    all exic1 : Exv_ic |
      (exic1.exval = ep1.exval
        implies(all ep_one_ic1a : ep1.pc_ic |
          one exv_one_ic1a : exic1.ex_ic |
            ep_one_ic1a = exv_one_ic1a and
          all exv_one_ic1b : exic1.ex_ic |
            one ep_one_ic1b : ep1.pc_ic |
              ep_one_ic1b = exv_one_ic1b))
  all disj exic2, exic3 : Exv_ic |
    exic2.exval != exic3.exval and
    exic2.ex_ic != exic3.ex_ic }
// Exval_Partcombi 期待結果毎のデータ
sig Exval_Partcombi {
  exorder : one Execution_Order,
  otype : one Out_Type,
  tyname : one TypeName,
  exval : one Expected_Value, //期待値
  conds : some TypeCond, //期待値に対する適用規則
  condsargsand : some PartCombCondArgsAnd, //適用規則毎の入力値
  p_combi_exval_seq : some Combi_Seq_ALV, //期待値&適用規則の入力値
  pc_ic : some InputCombi} //期待値の入力値
fact{
  all ep_a : Exval_Partcombi |
    ep_a.p_combi_exval_seq = ep_a.condsargsand.p_combi_cond and
  all oc4 : Out_Command |
    (oc4.retcode != none
      implies(
        all ret4 : oc4.retcode | one ep4 : Exval_Partcombi |
          ep4.tyname = TypeR and
          ep4.exorder = oc4.order and
          ep4.otype = ret4 and
          ep4.exval = ret4.value and
          ep4.conds = ret4.condition ))
}
```

```
all oc5 : Out_Command |
  (oc5.stdout != none
    implies(
      all ret5 : oc5.stdout | one ep5 : Exval_Partcombi |
        ep5.tyname = TypeO and
        ep5.exorder = oc5.order and
        ep5.otype = ret5 and
        ep5.exval = ret5.value and
        ep5.conds = ret5.condition))
all oc6 : Out_Command |
  (oc6.stderr != none
    implies(
      all ret6 : oc6.stderr | one ep6 : Exval_Partcombi |
        ep6.tyname = TypeE and
        ep6.exorder = oc6.order and
        ep6.otype = ret6 and
        ep6.exval = ret6.value and
        ep6.conds = ret6.condition))
all ep13 : Exval_Partcombi |
  one oc13 : Out_Command |
    (ep13.tyname = TypeR
      implies(oc13.retcode != none
        implies(one ret13 : oc13.retcode |
          ep13.exorder = oc13.order and
          ep13.otype = ret13 and
          ep13.exval = ret13.value and
          ep13.conds = ret13.condition)
        )else(ep13.tyname = TypeO
          implies(oc13.stdout != none
            implies(one out13 : oc13.stdout |
              ep13.exorder = oc13.order and
              ep13.otype = out13 and
              ep13.exval = out13.value and
              ep13.conds = out13.condition)
            )else(ep13.tyname = TypeE
              implies(oc13.stderr != none
                implies(one err13 : oc13.stderr |
                  ep13.exorder = oc13.order and
                  ep13.otype = err13 and
                  ep13.exval = err13.value and
                  ep13.conds = err13.condition))))))
all ep9 : Exval_Partcombi | all onecond9 : ep9.conds |
  one pccaa9 : ep9.condsargsand |
    onecond9.name = pccaa9.one_cond_name
```

付録4 generation.als(検証用データ定義) (2/3)

```
all ep10 : Exval_Partcombi | all pccaa10 : ep10.condsargsand |
  one onecond10 : ep10.conds |
    onecond10.name = pccaa10.one_cond_name
all ep11 : Exval_Partcombi | all onecond11 : ep11.conds |
  all pccaa11 : PartCombCondArgsAnd |
    (onecond11.name = pccaa11.one_cond_name
    implies (pccaa11 in ep11.condsargsand))
all pccaa12 : PartCombCondArgsAnd |
  all ep12 : Exval_Partcombi | all onecond12 : ep12.conds |
    (onecond12.name = pccaa12.one_cond_name
    implies (pccaa12 in ep12.condsargsand))
Exval_Partcombi.exorder = Out_Command.order
all disj ep5, ep6 : Exval_Partcombi |
  (ep5.exorder != ep6.exorder) or (ep5.otype != ep6.otype)
all disj ep7, ep8 : Exval_Partcombi |
  (ep7.condsargsand != ep8.condsargsand)
all disj ep_b, ep_c : Exval_Partcombi | ep_b != ep_c
all ep14 : Exval_Partcombi |
  all exval_seq14 : ep14.p_combi_exval_seq |
    all ic14 : InputCombi |
      (exval_seq14.seq_alv.elems in ic14.v_combi.seq_alv.elems
      implies (one pc_ic14 : ep14.pc_ic |
        pc_ic14 = ic14))
// Exval_Partcombi.pc_ic(期待値に対する入力値)を
// InputCombi(全テスト入力値)から抽出
all ep16 : Exval_Partcombi |
  all pc_ic16 : ep16.pc_ic |
    all exval_seq16 : ep16.p_combi_exval_seq |
      (exval_seq16.seq_alv.elems in pc_ic16.v_combi.seq_alv.elems
      implies (one ic16 : InputCombi |
        exval_seq16.seq_alv.elems in
        ic16.v_combi.seq_alv.elems and
        pc_ic16 = ic16))
all ep17 : Exval_Partcombi |
  all pc_ic17 : ep17.pc_ic |
    some exval_seq17 : ep17.p_combi_exval_seq |
      exval_seq17.seq_alv.elems in pc_ic17.v_combi.seq_alv.elems
all disj ep15_1, ep15_2 : Exval_Partcombi |
  ep15_1.pc_ic != ep15_2.pc_ic }
```

```
// PartCombCondArgsAnd 適用規則毎のデータ
sig PartCombCondArgsAnd {
  one_cond_name : one Condition_Name,
  cond_args_and : some Cond_Args_AND,
  pc_num : Int,
  p_combi_cond : some Combi_Seq_ALV }
fact {
  all cn1 : Condition_Name | one pccaa1 : PartCombCondArgsAnd |
    cn1 = pccaa1.one_cond_name
  all pccaa2 : PartCombCondArgsAnd | one cn2 : Condition_Name |
    cn2 = pccaa2.one_cond_name
  all disj pccaa3, pccaa4 : PartCombCondArgsAnd |
    pccaa3.one_cond_name != pccaa4.one_cond_name
  all pccaa5 : PartCombCondArgsAnd | all c5 : Condition |
    (pccaa5.one_cond_name = c5.name
    implies (pccaa5.cond_args_and = c5.args_and and
    pccaa5.pc_num = c5.pc_num))
  all c6 : Condition | all pccaa6 : PartCombCondArgsAnd |
    (pccaa6.one_cond_name = c6.name
    implies (pccaa6.cond_args_and = c6.args_and and
    pccaa6.pc_num = c6.pc_num))
  all disj aa10, aa11 : PartCombCondArgsAnd |
    aa10.cond_args_and != aa11.cond_args_and
  PartCombCondArgsAnd.one_cond_name = Condition_Name
  all disj pccaa_a, pccaa_b : PartCombCondArgsAnd |
    pccaa_a != pccaa_b
  all disj pccaa_c1, pccaa_c2 : PartCombCondArgsAnd |
    pccaa_c1.p_combi_cond != pccaa_c2.p_combi_cond
  all pccaa_d : PartCombCondArgsAnd |
    all pcc_d : pccaa_d.p_combi_cond |
      #(pccaa_d.cond_args_and) = #(pcc_d.seq_alv.elems) and
      #(pcc_d.seq_alv.indxs) = #(pcc_d.seq_alv.elems)
  all pccaa_e : PartCombCondArgsAnd |
    #(pccaa_e.p_combi_cond) = pccaa_e.pc_num
  all pccaa_f : PartCombCondArgsAnd |
    all caa_f : pccaa_f.cond_args_and |
      all lvor_f : caa_f.levels_or |
        some pcc_f : pccaa_f.p_combi_cond |
          pcc_f.seq_alv[caa_f.order_in_cond]
          = lvor_f.value.seq_alv[0]
// 適用規則毎の入力値を集計
all pccaa_g : PartCombCondArgsAnd |
  all pcc_g : pccaa_g.p_combi_cond |
    all caa_g : pccaa_g.cond_args_and |
      one lvor_g : caa_g.levels_or |
        pcc_g.seq_alv[caa_g.order_in_cond]
        = lvor_g.value.seq_alv[0] }
```

付録4 generation.als(検証用データ定義) (3/3)

```
// Combi_Seq_ALV 適用規則の入力値
sig Combi_Seq_ALV {
  seq_alv : seq Arg_Level_Value }
fact{
  Combi_Seq_ALV = Exval_Partcombi.condsargsand.p_combi_cond
  all disj csalv1,csalv2 : Combi_Seq_ALV |
    csalv1 != csalv2 and
    csalv1.seq_alv != csalv2.seq_alv and
    (#(csalv1.seq_alv.elems) = #(csalv2.seq_alv.elems)
      implies(csalv1.seq_alv.elems != csalv2.seq_alv.elems)) }
// InputCombi(入力値)の生成
fact {
  #InputCombi = combinum.num
  all ic_4 : InputCombi | #(ic_4.combi) = argordernum.num
  all ic : InputCombi | all disj c1, c2 : ic.combi |
    c1.com_arg_id != c2.com_arg_id
  all disj ic1,ic2 : InputCombi | ic1.combi != ic2.combi
  all ic3 : InputCombi | ic3.v_combi = ic3.combi.value
  Arg_Level = InputCombi.combi
  Arg_Level_Value in InputCombi.v_combi.seq_alv.elems }
```

付録5 dupchk.als(重複検証ルール)

```
/***** dupchk.als 重複の検証 *****/
open generation
check assert_pred_duppair for 8 but 8 Int

//重複の検証処理
assert assert_pred_duppair {
  pred_duppair [DupPair]}
pred pred_duppair [dp :DupPair] {
  dp.dupAep.exorder != dp.dupBep.exorder or //1.コマンドが異なる
  dp.dupAep.tyname != dp.dupBep.tyname or //2.出力が異なる
  dp.dupAep.exval = dp.dupBep.exval or //3.期待値が同じ
  all pcic_a : dp.dupAep.pc_ic |
    no pcic_b : dp.dupBep.pc_ic | //4.テストケースが異なる
    pcic_a = pcic_b }
//重複の検証・誤り箇所（期待結果）特定用signature
one sig DupPair {
  dupAep : one Exval_Partcombi, //判定データA
  dupBep : one Exval_Partcombi, //判定データB
  dup_exorder : one Execution_Order,
  dup_comname : one Command_Name,
  dup_tyname : one TypeName,
  dup_exval : one Expected_Value,
  dupA_seqalv : seq Arg_Level_Value, //適用規則A
  dupB_seqalv : seq Arg_Level_Value //適用規則B
}
{
  dup_exorder = dupAep.exorder
  dup_comname = comname_by_order[dupAep.exorder]
  dup_tyname = dupAep.tyname
  dup_exval = dupAep.exval
  dupA_seqalv = dupAep.p_combi_exval_seq.seq_alv
  dupB_seqalv = dupBep.p_combi_exval_seq.seq_alv
  dupAep != dupBep }
```

付録6 leakchk.als(漏れ検証ルール)

```

/***** leakchk.als 漏れの検証 *****/
open generation
check assert_pred_leakone for 8 but 8 Int

//漏れの検証処理
assert assert_pred_leakone {
  pred_leakone[Leaks] }
pred pred_leakone[lk : Leaks]{
  InputCombi = lk.leak_ic} //全テストケースInputCombiと比較
//漏れの検証・誤り箇所（テストケース）特定用signature
one sig Leaks {
  leak_ctv_ic : one Com_Type_ExvIc, //出力毎のデータ
  leak_com    : one Command_Name,
  leak_tyname : one TypeName,
  leak_ic     : some InputCombi, //出力毎のテストケース
  leak_differ : set InputCombi, //漏れたテストケース
  diff_alv_seq : some Arg_Level_Value, //漏れたテスト入力値
  leak_ep     : some Exval_Partcombi,
  leak_alv    : some Combi_Seq_ALV,
  leak_seqalv : some Arg_Level_Value //出力毎の適用規則
}{
  leak_com    = leak_ctv_ic.com_name
  leak_tyname = leak_ctv_ic.type_name
  leak_ic     = leak_ctv_ic.ex_ic
  leak_differ = InputCombi - leak_ic
  diff_alv_seq = leak_differ.v_combi.seq_alv.elems
  all disj e1, e2 : leak_ep | e1 != e2
  leak_alv = leak_ep.p_combi_exval_seq
  leak_seqalv = leak_alv.seq_alv.elems
}
fact {
  all ep1 : Exval_Partcombi |
    (comname_by_order[ep1.exorder] = Leaks.leak_com and
     ep1.tyname = Leaks.leak_tyname
     implies(one l_ep : Leaks.leak_ep | l_ep = ep1))
  all lk2 : Leaks | all ep2 : lk2.leak_ep |
    comname_by_order[ep2.exorder] = lk2.leak_com and
    ep2.tyname = lk2.leak_tyname
}
```

付録7 事前準備 誤りが有るケース (1/4)

入力値

因子	A	B	C
水準値	A1	B1	C1
	A2	B2	C2

期待結果

適用規則				期待結果	
因子	A	B	C	出力	期待値
水準値	A1	任意値	任意値	終了コード	R1
	A2	B1	任意値	終了コード	R2
	任意値	B2	C1	終了コード	R3

付録7 事前準備 誤りが有るケース (2/4)

テストケース

No	入力値			期待結果	
	A	B	C	出力	期待値
1	A1	B1	C1	終了コード	R1
2	A1	B1	C2	終了コード	R1
3	A1	B2	C1	終了コード	R1,R3
4	A1	B2	C2	終了コード	R1
5	A2	B1	C1	終了コード	R2
6	A2	B1	C2	終了コード	R2
7	A2	B2	C1	終了コード	R3
8	A2	B2	C2	終了コード	

同一テストケースに期待値が重複

漏れ

付録7 事前準備 誤りが無いケース (3/4)

入力値

因子	A	B	C
水準値	A1	B1	C1
	A2	B2	C2

期待結果

適用規則				期待結果	
因子	A	B	C	出力	期待値
水準値	A1	任意値	任意値	終了コード	R1
	A2	B1	任意値	終了コード	R2
	A2	B2	任意値	終了コード	R3

付録7 事前準備 誤りが無いケース (4/4)

テストケース

No	入力値			期待結果	
	A	B	C	出力	期待値
1	A1	B1	C1	終了コード	R1
2	A1	B1	C2	終了コード	R1
3	A1	B2	C1	終了コード	R1
4	A1	B2	C2	終了コード	R1
5	A2	B1	C1	終了コード	R2
6	A2	B1	C2	終了コード	R2
7	A2	B2	C1	終了コード	R3
8	A2	B2	C2	終了コード	R3