2006.5.11

ソフトウェアテストシンポジウム2006 in 大阪

形式手法の概要

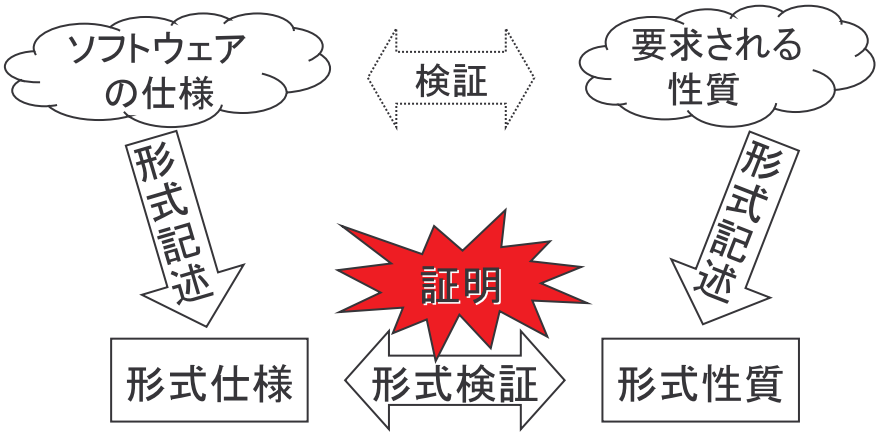
産業技術総合研究所
システム検証研究センター
水口大知

発表内容

- 形式手法のこれまで
  - 形式手法とは？
- 形式手法の今
  - モデル検査
- 形式手法のこれから
  - 産官学の動向

形式手法のこれまで

- 形式手法 = formal method とは?



著名なもの

- Z notation
- VDM
- B method
- OBJ
- CSP
- CCS
- LOTOS

sequential system 向け

concurrent system 向け

複合型

形式化の意義

- 論理的に証明するには形式化が不可欠
- 仕様の意味が一意に決まる
- ロジックと実装を切り離して議論できる（しやすい）
  - Howではなく、Whatについて

Z記法による記述

- The birthday book

$[NAME, DATE]$ ← データ型

$$
\begin{array}{|l}
BirthdayBook \\
known : \mathbb{P} NAME \\
birthday : NAME \pfun DATE \\
\hline
known = \mathrm{dom}\ birthday
\end{array}
$$

宣言部

条件部

状態空間の定義

状態

$known = \{$太郎, 花子$\}$

$birthday = \{$太郎 $\mapsto$ 5月5日,
花子 $\mapsto$ 3月3日$\}$

$AddBirthday$

$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

入力変数

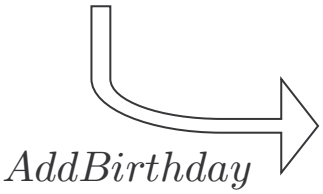
事前条件

事後条件

操作の定義

状態遷移

$known$ = {太郎, 花子}
$birthday$ = {太郎 ↦ 5月5日,
花子 ↦ 3月3日}



$known$ = {太郎, 花子, 次郎}
$birthday$ = {太郎 ↦ 5月5日,
花子 ↦ 3月3日,
次郎 ↦ 1月1日}

性質と証明

- $AddBirthday$ において、
  $known' = known \cup \{name?\}$

$\because$ $known'$ = dom $birthday'$
= dom($birthday \cup \{name? \mapsto date?\}$)
= dom $birthday \cup$ dom $\{name? \mapsto date?\}$)
= dom $birthday \cup \{name?\}$
= $known \cup \{name?\}$

$$
\begin{array}{|l}
\hline
FindBirthday \\
\Xi BirthdayBook \\
name? : NAME \\
date! : DATE \\
\hline
name? \in known \\
date! = birthday(name?) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
Remind \\
\Xi BirthdayBook \\
today? : DATE \\
cards! : \mathbb{P} NAME \\
\hline
cards! = \{n : known \mid birthday(n) = today?\} \\
\hline
\end{array}
$$

CSPによる記述

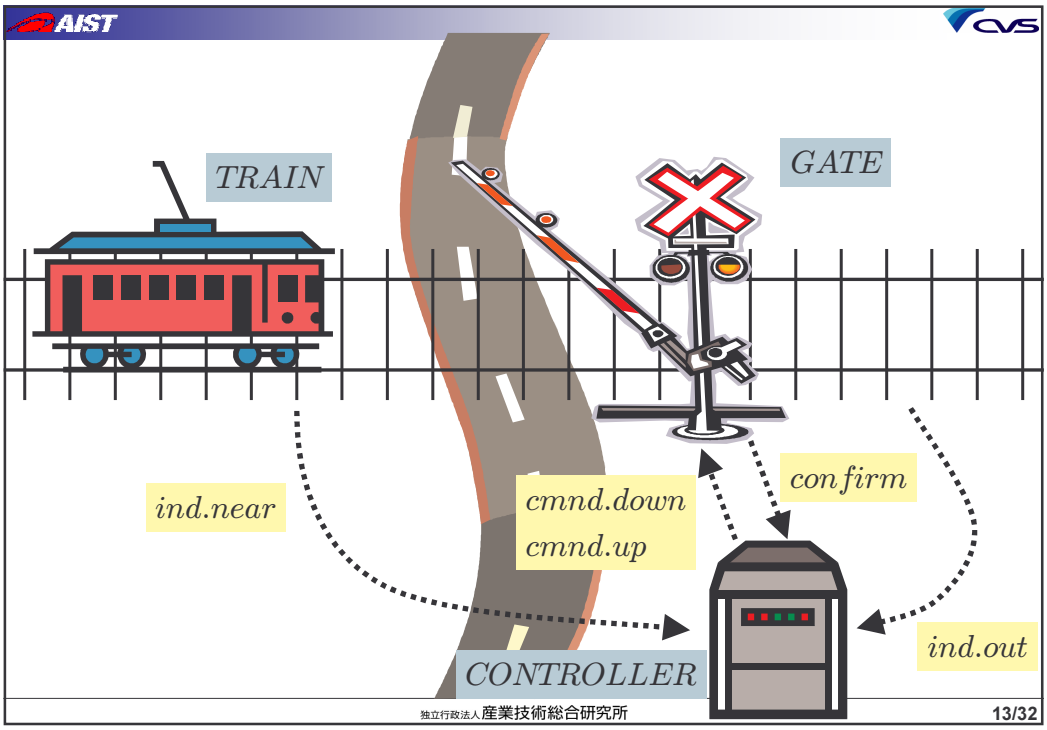
$$CROSSING = CONTROLLER \ {}_C\|_G \ GATE$$

$$
\begin{aligned}
CONTROLLER = \\
&ind.near \rightarrow cmnd.down \rightarrow confirm \rightarrow CONTROLLER \\
&\Box \ ind.out \rightarrow cmnd.up \rightarrow confirm \rightarrow CONTROLLER
\end{aligned}
$$

$$
\begin{aligned}
GATE = \\
&cmnd.down \rightarrow down \rightarrow confirm \rightarrow GATE \\
&\Box \ cmnd.up \rightarrow up \rightarrow confirm \rightarrow GATE
\end{aligned}
$$

$$
\begin{aligned}
TRAIN = near \rightarrow ind.near \rightarrow enter \rightarrow leave \rightarrow ind.out \rightarrow \\
TRAIN
\end{aligned}
$$

$$SYSTEM = CROSSING \ {}_{C \cup G}\|_T \ TRAIN$$



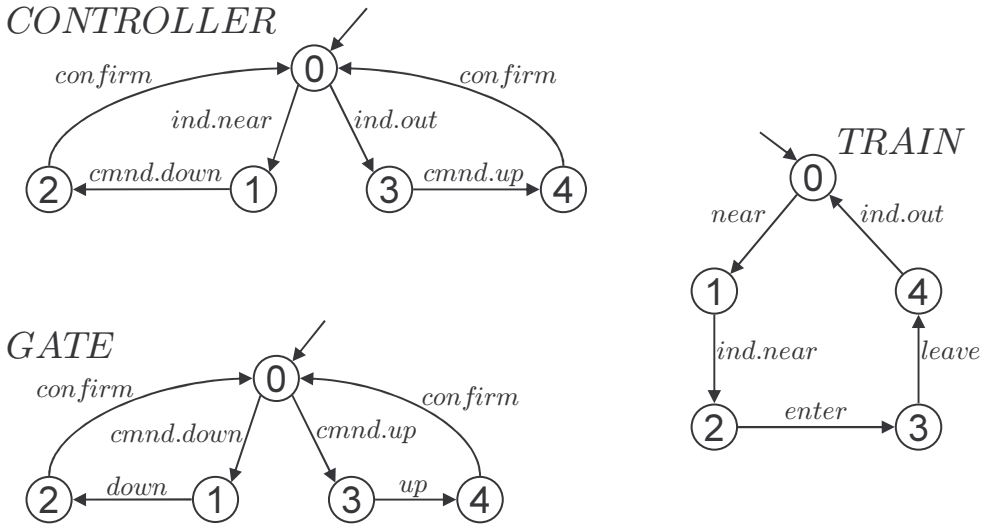
性質と証明

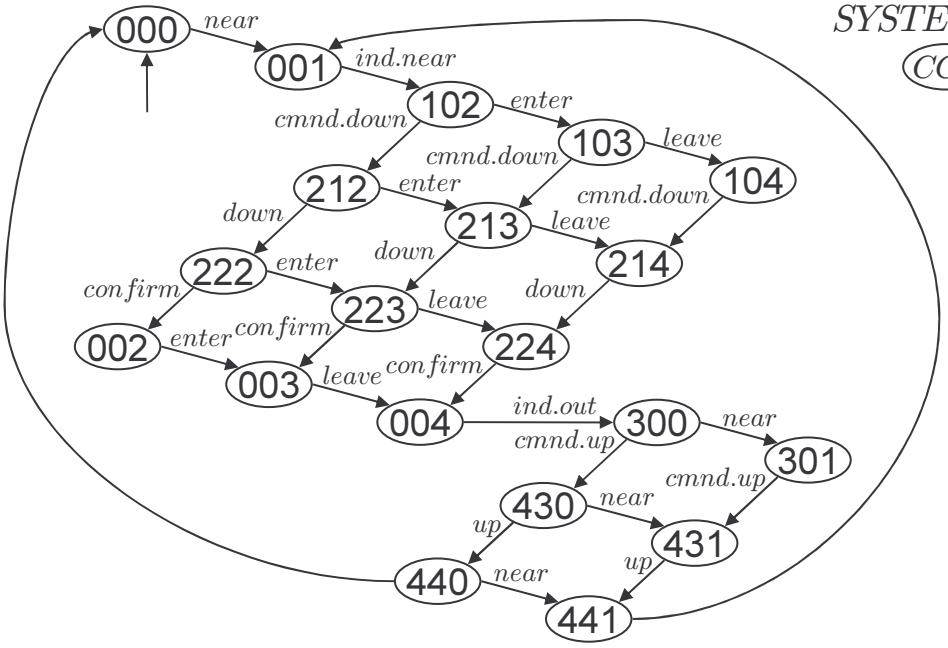
- $\mathrm{last}(tr) = up \Rightarrow \mathrm{last}(tr \upharpoonright \{enter, leave\}) = leave$
  - 遮断機を上げるのは、列車が踏切を通過した後
- $\mathrm{last}(tr) = enter \Rightarrow \mathrm{last}(tr \upharpoonright \{down, up\}) = down$
  - 列車が踏切に進入するのは、遮断機が下りた後

形式手法の今

- モデル検査による形式検証が実用化段階へ
- モデル検査とは？
  - 有限状態遷移系の上で、状態を網羅的に探索し性質が正しいことを確認して証明とする
    ⇒ 自動化可能
  - ツール：NuSMV, SPIN, UPPAAL,...
  - いわゆる定理証明 = theorem proving と対比される

例えば

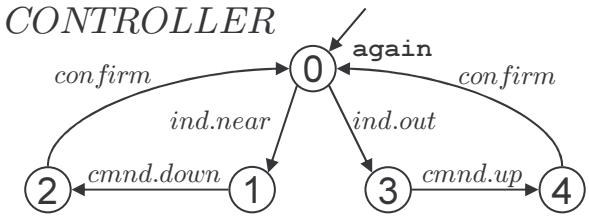




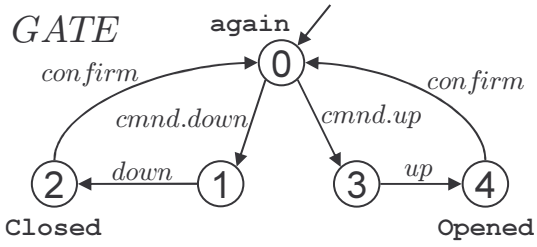
SPINでは

```
mtype = {near,out,down,up};
chan ind = [0] of {mtype};
chan cmnd = [0] of {mtype};
chan confirm = [0] of {mtype};

active proctype CONTROLLER(){
again: do
       ::ind?near;
         cmnd!down;
         confirm?down;
         goto again
       ::ind?out;
         cmnd!up;
         confirm?up;
         goto again
       od
}

active proctype GATE(){
again: do
       ::cmnd?down;
         printf("down¥n");
Closed:  confirm!down;
         goto again
       ::cmnd?up;
         printf("up¥n");
Opened:  confirm!up;
         goto again
       od
}

active proctype TRAIN(){
again:   printf("near¥n");
         ind!near;
         printf("enter¥n");
Entered: printf("leave¥n");
Left:    ind!out;
         goto again
}
```

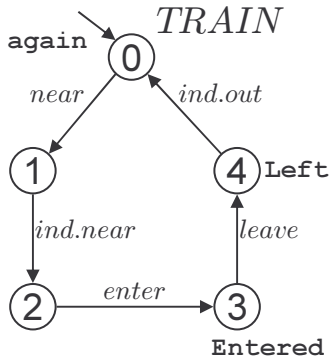
```
active proctype
  CONTROLLER(){
again: do
       ::ind?near;
         cmnd!down;
         confirm?down;
         goto again
       ::ind?out;
         cmnd!up;
         confirm?up;
         goto again
       od
}
```




```
active proctype GATE(){
again: do
       ::cmnd?down;
         printf("down¥n");
Closed:  confirm!down;
         goto again
       ::cmnd?up;
         printf("up¥n");
Opened:  confirm!up;
         goto again
       od
}
```



```
active proctype TRAIN(){
again:   printf("near¥n");
         ind!near;
         printf("enter¥n");
Entered: printf("leave¥n");
Left:    ind!out;
         goto again
}
```



NuSMVでは

```
MODULE main
VAR
CONTROLLER: 0..4;
GATE: 0..4;
TRAIN: 0..4;
aux: boolean;

DEFINE
ind_near := TRAIN=1 &
   CONTROLLER=0 & aux;
ind_out :=  TRAIN=4 &
   CONTROLLER=0 & aux;
cmnd_down := CONTROLLER=1 &
   GATE=0 & !aux;
cmnd_up := CONTROLLER=3 &
   GATE=0 & !aux;
confirm :=(CONTROLLER=2|CONTRO
   LLER=4)&(GATE=2|GATE=4)
   & !aux;
down := GATE=1 & !aux;
up := GATE=3 & !aux;
near := TRAIN=0 & aux;
enter := TRAIN=2 & aux;
leave := TRAIN=3 & aux;


ASSIGN
init(CONTROLLER) := 0;
next(CONTROLLER) := case
   ind_near: 1;
   ind_out : 3;
   cmnd_down: 2;
   cmnd_up: 4;
   confirm: 0;
   1 : CONTROLLER;
esac;
```

```
init(GATE) := 0;
next(GATE) := case
   cmnd_down: 1;
   cmnd_up: 3;
   down : 2;
   up : 4;
   confirm : 0;
   1 : GATE;
esac;

init(TRAIN) := 0;
next(TRAIN) := case
   near : 1;
   ind_near : 2;
   enter : 3;
   leave : 4;
   ind_out : 0;
   1 : TRAIN;
esac;
```

```
init(aux) := 1;
next(aux) := {0,1};


DEFINE
Left := TRAIN=4;
Entered := TRAIN=3;
Opened := GATE=4;
Closed := GATE=2;

LTLSPEC !G(!Left U Opened)
LTLSPEC !G(!Closed U Entered)
```

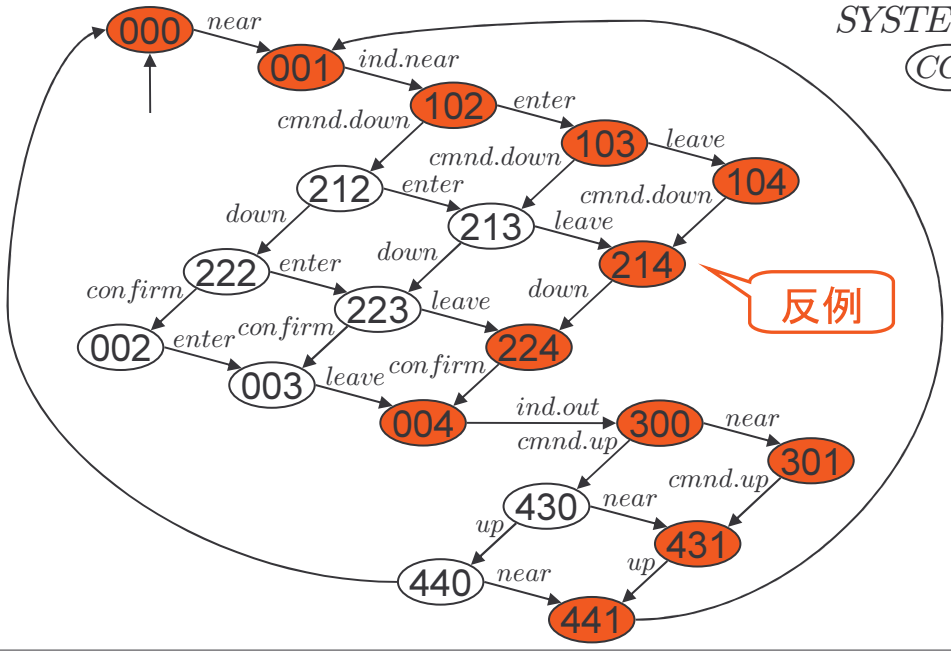
性質

時相論理

- 命題論理に時相演算子を加えて拡張した論理
  - CTL, LTL, modal μ calculus,...
- LTL = Linear Temporal Logic
  - $\wedge$ (かつ), $\vee$ (または), $\neg$ (でない), $\Rightarrow$ (ならば)
  - G (つねに), F (いつか), X (次に), U (…まで…)
- LTL式の例:
  - $\neg$ G (($\neg$ **Left**) U **Opened**)
    『遮断機を上げるのは、列車が踏切を通過した後』
  - $\neg$ G (($\neg$ **Closed**) U **Entered**)
    『列車が踏切に進入するのは、遮断機が下りた後』

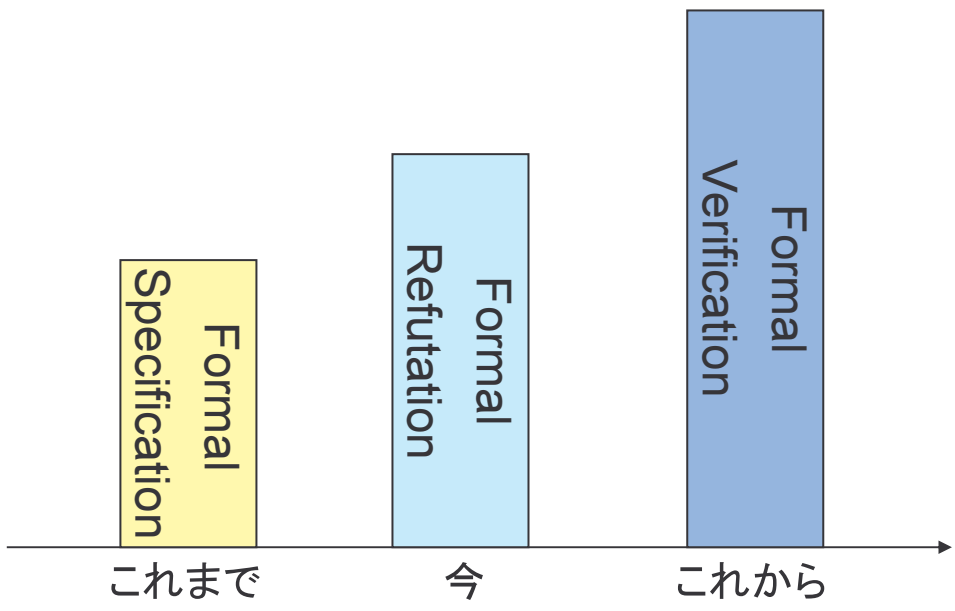
検査結果

```
*** This is NuSMV 2.3.0
-- specification  ! G ((!Left)
   U Opened)  is true
-- specification  ! G
   ((!Closed) U Entered)  is
   false
-- as demonstrated by the
   following execution sequence
Trace Description: LTL
   Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  CONTROLLER = 0
  GATE = 0
  TRAIN = 0
  aux = 1
  gateOpen = 0
  trainIn = 0
  leave = 0
  enter = 0
  near = 1
  up = 0
  down = 0
  confirm = 0
  cmnd_up = 0
  cmnd_down = 0
  ind_out = 0
  ind_near = 0
  Closed = 0
  Opened = 0
  Entered = 0
  Left = 0
-> State: 1.2 <-
  TRAIN = 1
  near = 0
  ind_near = 1
-> State: 1.3 <-
  CONTROLLER = 1
  TRAIN = 2
  enter = 1
  ind_near = 0     ...(続く)
```

注目の理由

- モデルの記述がしやすい
- 反例が見える
  - バグ検出のための有益な情報
- 動的性質の検証に向いている
  - 組み込み系への適用
- スケーラビリティが向上した
  - 計算機能力の向上
  - データ構造、アルゴリズムの進歩
- 優れたツールが手に入る
  - NuSMV, SPIN, UPPAAL, ...
- 従来工程との親和性がよい



形式手法のこれから

- 各種手法の統合
  - UMLとの連携
  - Z系とCSP系の融合
  - モデル検査と定理証明の連携
- より下流へ
  - コード検証
  - コード生成

形式手法のこれから

- 規格化
  - LOTOS, VDM, Z,...
- 規格からの要請
  - IEC 61508 SIL4
  - ISO/IEC 15408 (Common Criteria) EAL7
  - 経産省：情報システムの信頼性向上に関するガイドライン（案）
- 国内での適用の加速

規格化されたもの

- LOTOS
  - ISO 8807:1989
    Information processing systems -- Open Systems Interconnection -- LOTOS -- A formal description technique based on the temporal ordering of observational behaviour
- VDM
  - ISO/IEC 13817-1:1996
    Information technology -- Programming languages, their environments and system software interfaces -- Vienna Development Method -- Specification Language -- Part 1: Base language
- Z
  - ISO/IEC 13568:2002
    Information technology -- Z formal specification notation -- Syntax, type system and semantics

参考

- Formal Methods Virtual Library, http://vl.fmnet.info/
- Formal Methods Europe, http://www.fmeurope.org/
- Zの記述例：The Z Notation, J. M. Spivey, http://spivey.oriel.ox.ac.uk/mike/zrm/index.html
- CSPの記述例：Concurrent and Real-time Systems; The CSP Approach, S. Schneider, Wiley
- Wikipedia,
  http://en.wikipedia.org/wiki/Model_checking
  http://en.wikipedia.org/wiki/Theorem_proving
  http://en.wikipedia.org/wiki/Formal_method