Garakabu2: A Formal Verification Tool for ZIPC

Weiqiang Kong, Kyushu University, Japan

[Jointly developed by]
Fukuoka IST & CATS Co. Ltd. & Kyushu University of Japan

[With technical supports from]
AIST of Japan

Industry Day Program of FM2012
August 30, 2012

Outline

- Garakabu2 and ZIPC – A general introduction
- The SMT-based BMC approach in Garakabu2
- Towards practical usability for on-site software engineers

Outline

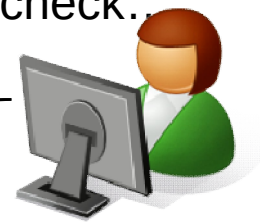
- Garakabu2 and ZIPC – A general introduction

  The SMT-based BMC approach in Garakabu2

  Towards practical usability for on-site software engineers

Garakabu2 in General

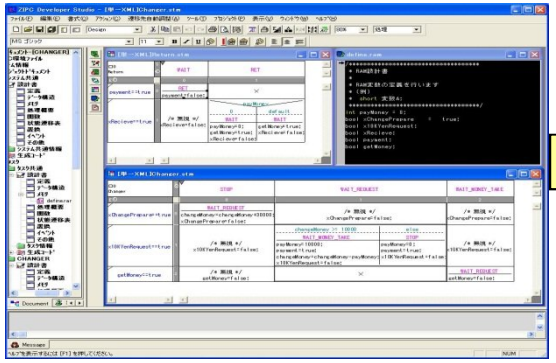
⑤ Revising the design and recheck..

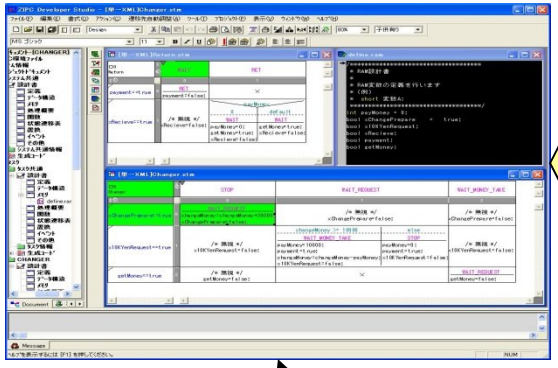


① Software designs in ZIPC-HSTM

② Properties in LTL

ZIPC

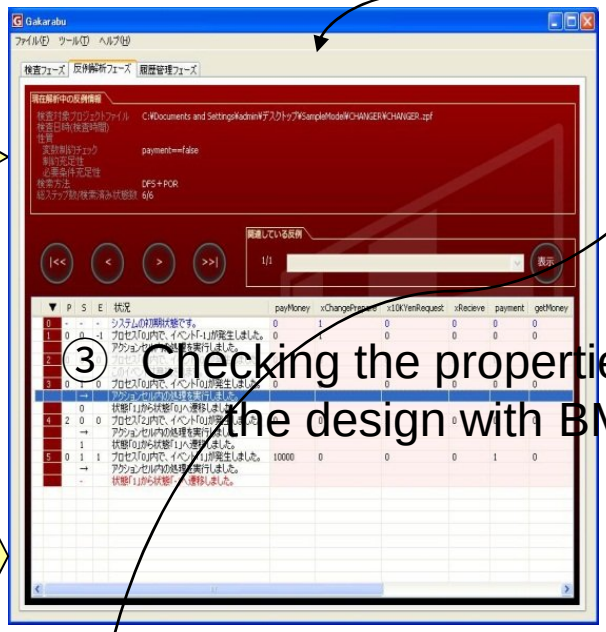




input

output counterexample

Garakabu2



③ Checking the properties against the design with BMC

④ Tracking and understanding error reasons

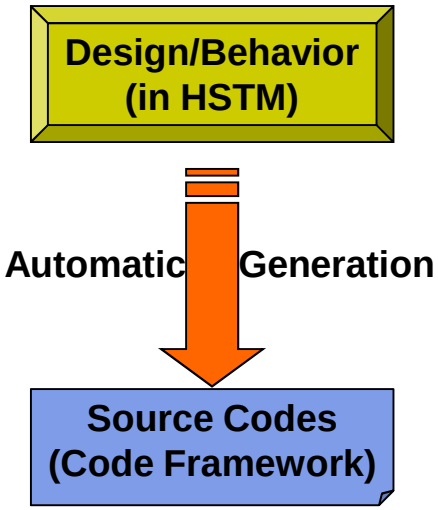
Features of Garakabu2

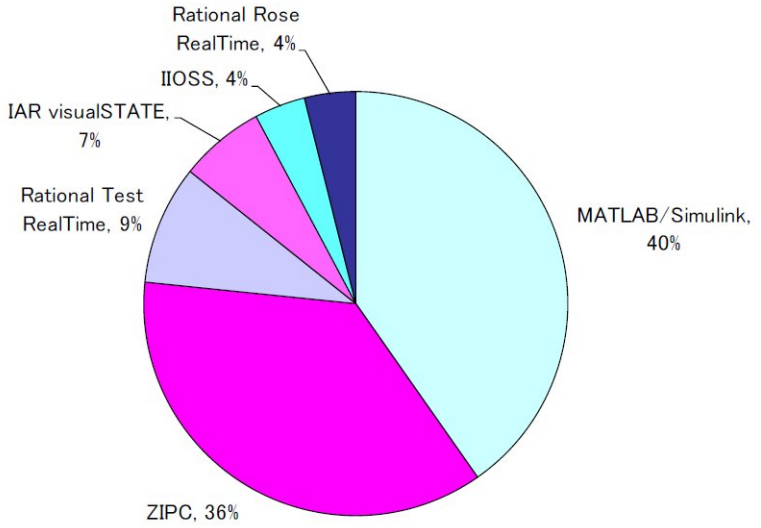
- Designs developed with ZIPC can be checked as they are
  - Easy to use: no need to learn a 2nd MC specific description language
- Traces of CX can be illustrated in ZIPC
  - Easy to understand: relatively easier to understand the reasons for errors
- Previous MC results can be saved and replayed
  - Easy to make a later confirmation for previous designs/checks (traceability)
- LTL properties can be specified more intuitively
  - Write/Draw LTL properties by patterns or figures (with SpecEditor of AIST)

Our intention:

Make Garakabu2 an easy-to-use formal verification tool for on-site software engineers who have not much knowledge in formal methods.

ZIPC in General



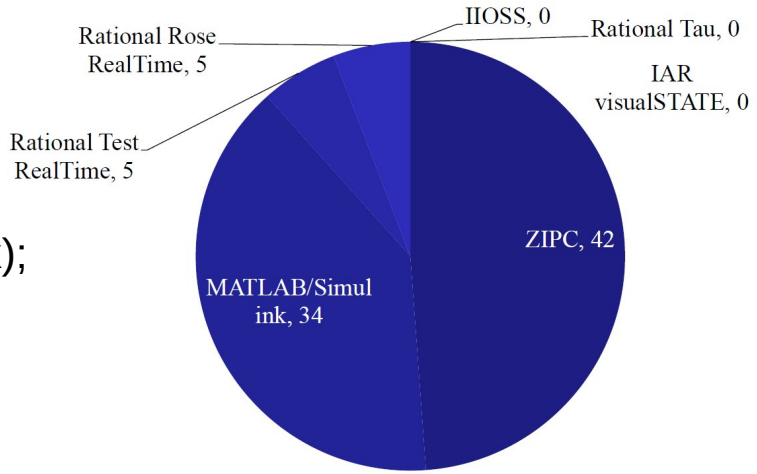


JASA Questionnaire 2009

A CASE Tool (or … model-based development tool)

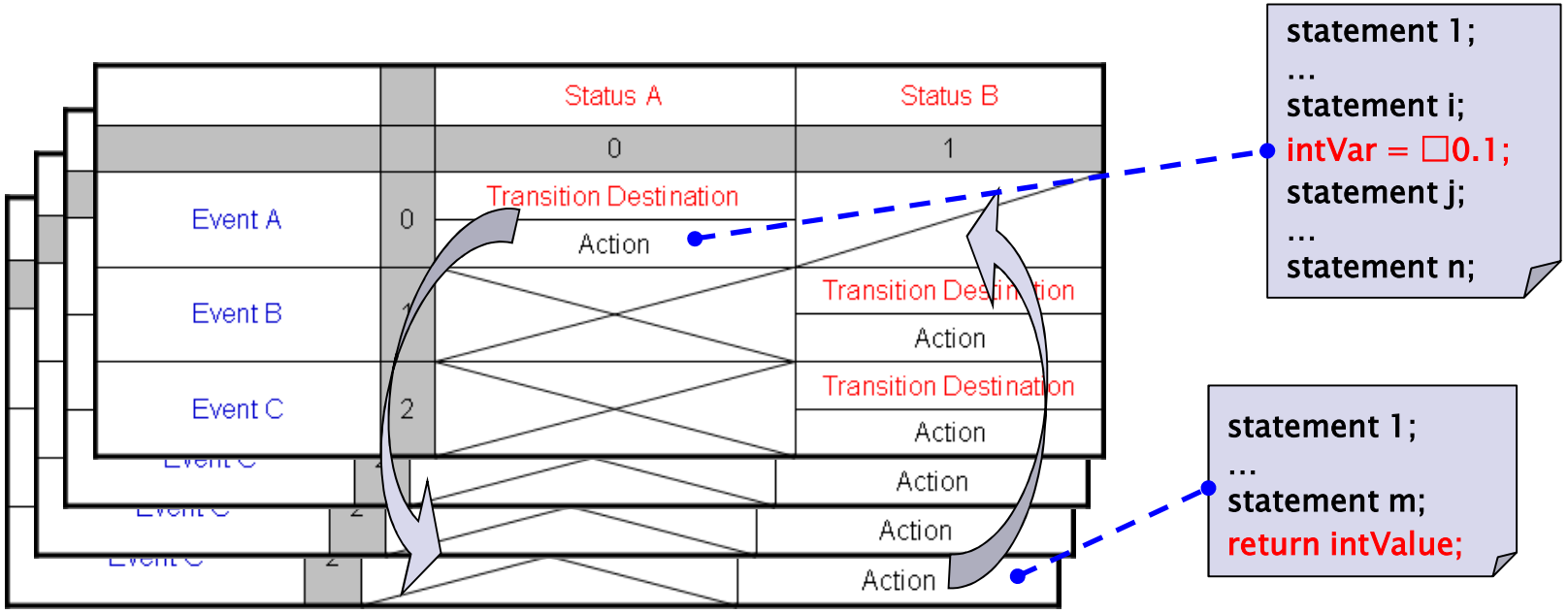
- ✓ No overlooking of possibly abnormal cases;
- ✓ Automatically generating source codes (framework);
- ✓ Syntax check, graphical simulation;
- ✓ …

➢ No formal verification supports



JASA Questionnaire 2010

Hierarchical State Transition Matrix (HSTM)



- STM is a table that captures an object's behaviors under event-state match;
- An HSTM is a set of STMs organized in a hierarchical structure with calling-to and returning-from the execution of Child STMs;
- An HSTM design is a set of HSTMs executing in an interleaving manner.

A Simplified Money-Exchange Machine (MEM) in HSTM

□0 MainController	S	M_IDLE	M_ACTIVE	M_WAIT_BILL_TAKEN
E		0	1	2
xMachinePrepare	0	M_ACTIVE Balance = Balance + 10000; xMachinePrepare = false;	／	／
xUserOperation	1	／	□0.1; xUserOperation = false;	／

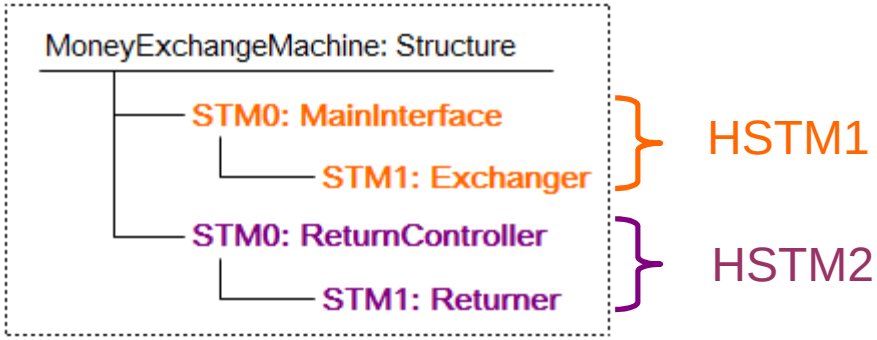
RequestProcessed	2	×	BillOutputAmount>0
			M_WAIT_BILL_TAKEN
			BillReady=true;
			RequestProcessed = false;
			else
			M_IDLE
			//BillReady=true;
			RequestProcessed = false;
				／

UserBillObtained	3	×	／	M_ACTIVE UserBillObtained = false;

□0 ReturnControl	S	R_IDLE	R_ACTIVE
E		0	1
BillReady	0	R_ACTIVE BillOutput = true; BillReady = false;	×
BillOutput	1	×	R_IDLE □0.1; BillOutput = false;

□0.1 Returner	S	R_Bill_RETURN
E		0
xUserBillTake	0	R_Bill_RETURN BillOutputAmount = 0; UserBillObtained = true; xUserBillTake = false; return;

□0.1 Exchanger	S	M_WAIT_REQUEST	
E		0	
xUser1OKRequest	0	Balance>=10000 M_WAIT_REQUEST Balance = Balance - 10000; BillOutputAmount = 10000; RequestProcessed = true; xUser1OKRequest = false; return;	Balance<10000 M_WAIT_REQUEST RequestProcessed = true; xUser1OKRequest=false; return;



HSTM1: Specifies interaction with users and actual bill exchange functionality;

HSTM2: Specifies exchanged bill payback functionality.

ZIPC – Image Demo (Japanese)

Outline

Garakabu2 and ZIPC – A general introduction

- The SMT-based BMC approach in Garakabu2

Towards practical usability for on-site software engineers

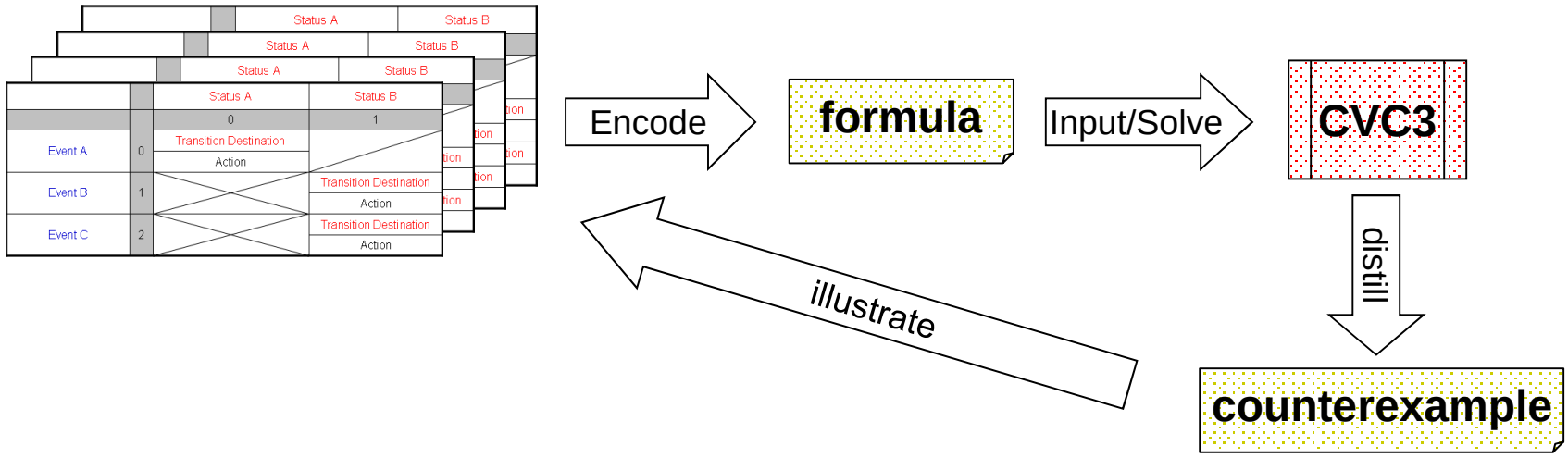
SMT solving and SMT solvers

- SMT (Satisfiability Modulo Theories) Solving Technique

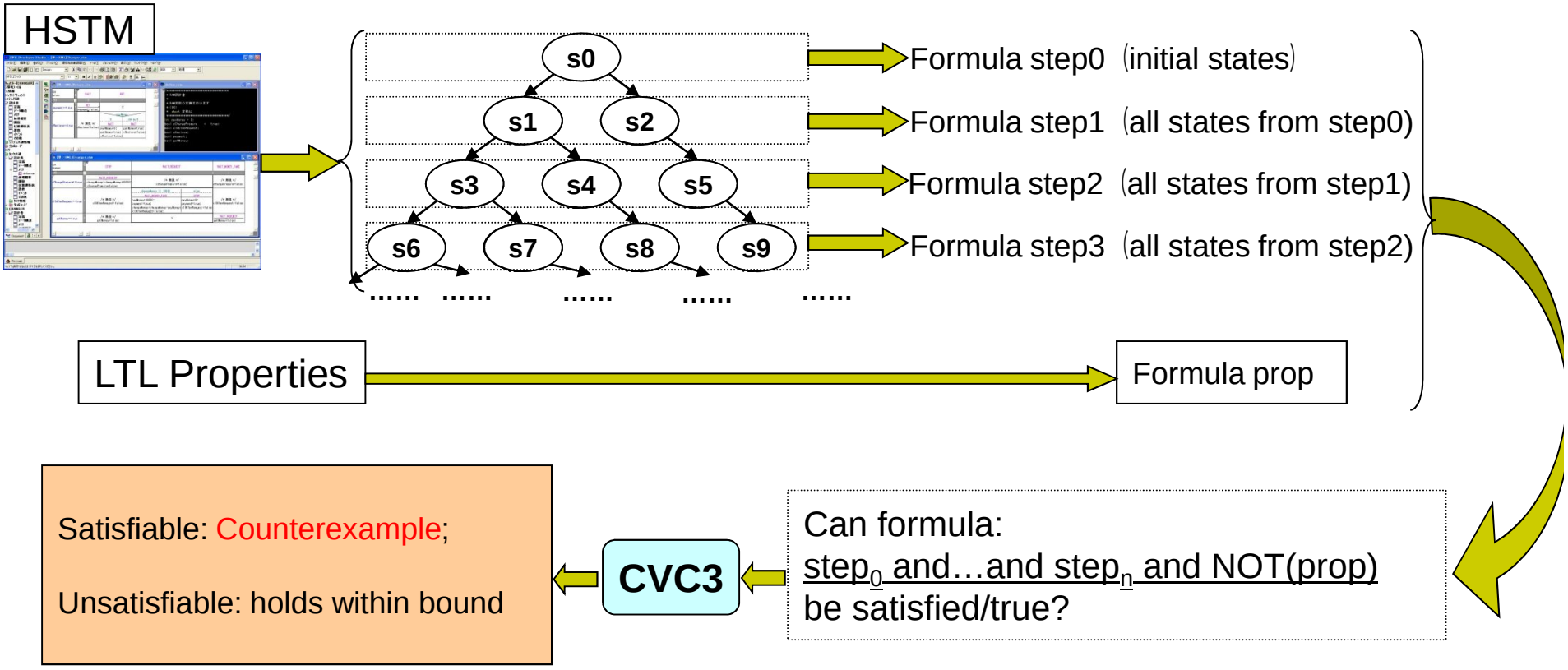
  A technique of deciding satisfiability of a given quantifier-free formula, i.e., finding a variable-assignment that makes the formula TRUE.

- SMT Solvers

  CVC3 (New York Univ.), Yices (SRI International), Z3 (Microsoft), etc.



Basic ideas for the encoding in Garakabu2



Basic Ideas (Informal) for Encoding an HSTM Design

Basic Encoding Rules

$$
\begin{aligned}
\text{CELL} &:= (\text{cell.event} \wedge \text{cell.status}) \Rightarrow (\text{cell.action} \wedge \text{cell.statusTransit} \wedge \text{untouchedVars}) \\
\text{STM} &:= (\text{CELL1} \wedge \text{flagC1}) \vee (\text{CELL2} \wedge \text{flagC2}) \vee \ldots \vee (\text{CELLN} \wedge \text{flagCN}) \\
\text{HSTM} &:= (\text{STM1} \wedge \text{flagS1}) \vee (\text{STM2} \wedge \text{flagS2}) \vee \ldots \vee (\text{STMP} \wedge \text{flagSN}) \\
\text{DESIGN} &:= (\text{HSTM1} \vee \text{HSTM2} \vee \ldots \vee \text{HSTMQ}) \wedge \text{asynchConstraints}
\end{aligned}
$$

※ Formula **asynchConstraints** is defined on flag variables to restrict the interleaving execution manner.

※ There are many subtle, but not technically difficult, details for encoding.

Basic Ideas (Informal) for Encoding an HSTM Design

Step 0 Formula (representing Initial States)

$$\text{InitState} := (\text{var1}_0 = \text{initValueVar1}) \wedge (\text{var2}_0 = \text{initValueVar2}) \wedge \ldots \wedge (\text{varM}_k = \text{initValueVarM})$$

Step k Formula (representing the set of states reachable at step k)

$$\text{CELL}_K := (\text{cell.event}_{K-1} \wedge \text{cell.status}_{K-1}) \Rightarrow (\text{cell.action}_K \wedge \text{cell.statusTransit}_K \wedge \text{untouchedVars}_K)$$

$$\text{STM}_K := (\text{CELL1}_K \wedge \text{flagC1}_K) \vee (\text{CELL2}_K \wedge \text{flagC2}_K) \vee \ldots \vee (\text{CELLN}_K \wedge \text{flagCN}_K)$$

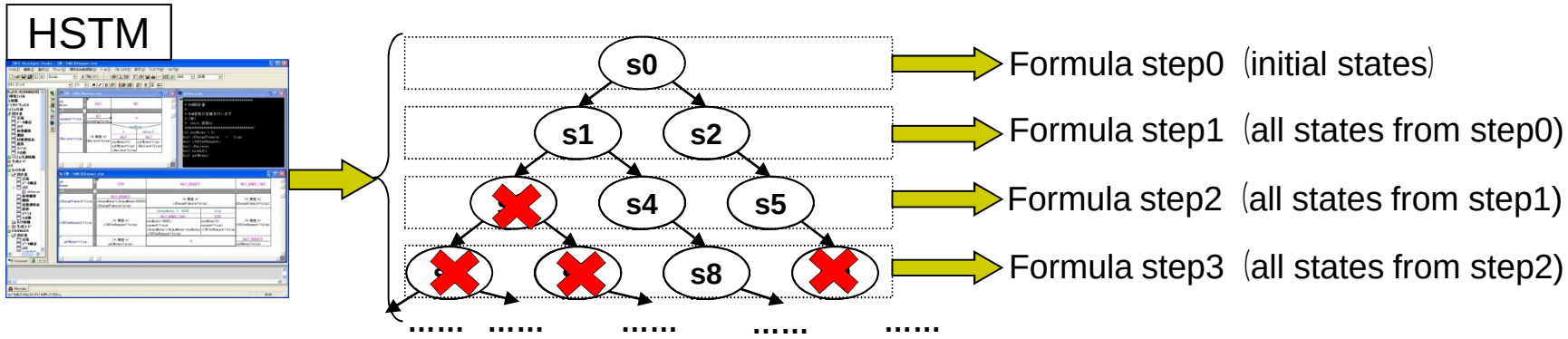
$$\text{HSTM}_K := (\text{STM1}_K \wedge \text{flagS1}_k) \vee (\text{STM2}_K \wedge \text{flagS2}_k) \vee \ldots \vee (\text{STMP}_K \wedge \text{flagSN}_k)$$

$$\text{DESIGN}_K := (\text{HSTM1}_K \vee \text{HSTM2}_K \vee \ldots \vee \text{HSTMQ}_K) \wedge \text{asynchConstraints}_K$$

※ Formulas **cell.action$_k$** and **untouchedVars$_k$** are defined using variables that belong to either step k or step k-1.

※ Hierarchical structure is represented by flag variables and changes of their values.

Basic ideas for accelerating BMC in Garakabu2



- Avoid encoding all transitions in $step_k$ by explicitly pre-traversing the state space.
- Stateless traversing; only interested in transitions executable in $step_k$.

- Saturation may happen for deep BMC bounds.
- Generally becomes faster, especially for safety properties.

Outline

Garakabu2 and ZIPC – A general introduction

The SMT-based BMC approach in Garakabu2

- Towards practical usability for on-site software engineers
  (with demo of Garakabu2)

Step 1: Input ZIPC-HSTM designs into Garakabu2

- Syntax errors or specifications that could not be handled by Garakabu2 will be reported and pinpointed.

Garakabu2 – Image Demo

Step 2: Select STMs to be checked

- It is possible to select and check partial designs that are of interesting
- Partial selection that violates predefined rules are not allowed
  - E.g., it is not allowed to select a child STM, whose parent STM is not selected but STMs of other tasks are selected.

Garakabu2 – Image Demo

Step 3: Set initial/threshold values for variables

- Garakabu2 automatically reads initial values defined in ZIPC RAM file;
- It is possible to set the min/max values of variables (optional)

Variable Types	Range Values	
	Min	Max
Bool	False	True
Byte	-128	127
Char	-128	127
Short	-32768	32767
Int	-2147483648	2147483647

Garakabu2 – Image Demo

Step 4: Input properties to be checked

- Supported properties
  - Reachability of Invalid Cells;
  - General LTL properties;
  - Deadlock;
  - Range values violation;
  - HSTM-specific properties:
    - Correlation between status of different STMs

Garakabu2 – Image Demo

Specify LTL properties by patterns

- It is difficult to specify LTL properties correctly
  - Property in text:

    Before the bill exchanged for a previous session is taken, no new to-be-exchanged bills could be inserted into the machine.

  - Property in LTL:

```
[G](((sigBillExchanged == true) && !(exchangeTaken == true) &&
                [F](exchangeTaken == true)) =>
                        ((sigExchangeOK == false) [U] (exchangeTaken == true)))
```

- SpecEditor – under-development in AIST of Japan
  - Specify LTL properties with Dwyer's LTL patterns
  - Specify LTL properties with graph drawing (AIST LTL Notations)

SpecEditor – Image Demo

Step 5: Read/Track the check results

- After checking, the results in
  - Black indicates no counterexamples (within the bound)
  - Red indicates a counterexample

- By clicking a check result in red
  - Trace of the counterexample could be illustrated in ZIPC environment,
  - By which to confirm the undesired behaviors that violate the property

Garakabu2 – Image Demo

Step 6: Read/Replay previous check results

- Previous check results are recorded for further confirmation.
  - Select the history management label,
  - Double-click a checking item to confirm the model, property, and results to help understand design revision history.

Garakabu2 – Image Demo

This is all Garakabu2 about.

Simple and Easy-to-use
are what we expect!

Future work

- Accelerating BMC implemented in Garakabu2;
  - Further integrating explicit model checking techniques into BMC
    - Preprocessing with explicit traversing and applying explicit abstraction techniques, e.g., partial order reductions etc.
- Extending HSTM components checkable by Garakabu2;
  - E.g., Concurrent states are not checkable by current Garakabu2
    - According to comments from practical users of ZIPC and Garakabu2
- (Further) Implementing SpecEditor;
- ……

Questions and Comments?