Counterexample Discovery with a Combination of Induction and Bounded Model Checking*

Kazuhiro Ogata[1]

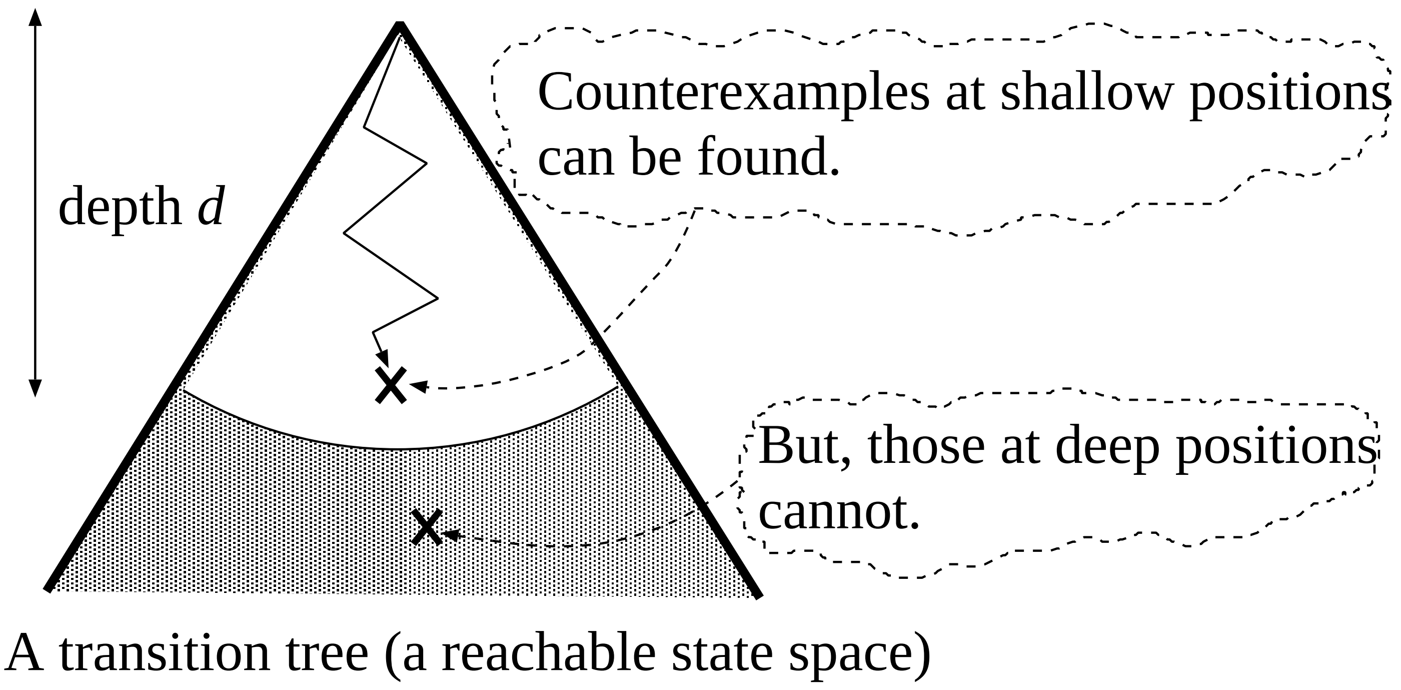
In collaboration with *Masahiro Nakano*[1], *Weiqiang Kong*[1], and *Kokichi Futatsugi*[1]

[1] School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)

*Presented at the JAIST 21st Century COE Symposium 2007 on Verifiable and Evolvable e-Society, March 7, 2007, JAIST

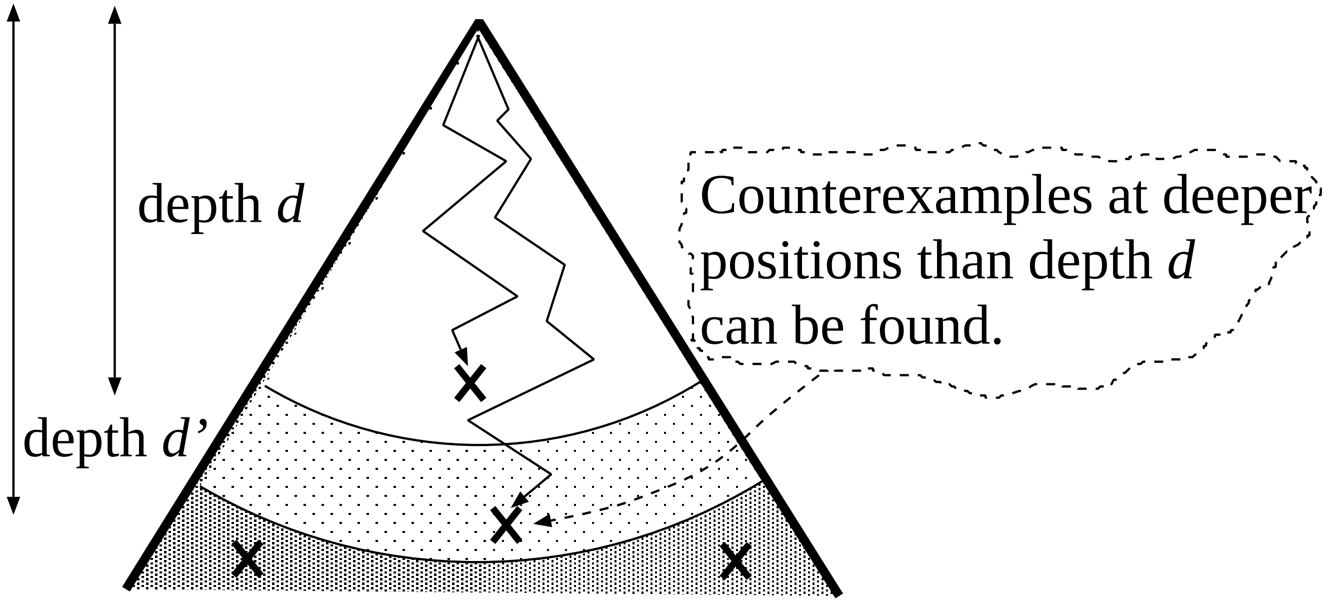
Background

- *Bounded model checking* (BMC) is often used to find counterexamples.
  - But, it cannot find counterexamples that exist at deep positions: the *state explosion problem.*


depth d
Counterexamples at shallow positions can be found.
But, those at deep positions cannot.
A transition tree (a reachable state space)

Contribution

- To make it possible to find counterexamples at deeper positions than depth d.


depth *d*
depth *d'*
Counterexamples at deeper positions than depth *d* can be found.

A transition tree (a reachable state space)

To make it possible to find counterexamples at positions between depth d and depth d', where d' is greater than d.

Proposed Solution

- A way to find counterexamples showing that state machines do not satisfy invariant properties by combining BMC and induction on the structure of reachable state spaces.

  When a state predicate p is proved invariant wrt a state machine by induction, other state predicates $q_1, \ldots q_N$ are made as lemmas. Instead of finding a counterexample of p, we try to find a counterexample of either of $q_1, \ldots q_N$.

  – Observational transition systems (OTSs) are used as state machines,
  – Alg spec lang & sys Maude is used as a model checker, and
  – Alg spec lang & sys CafeOBJ is used as a proof assistant.

Publication

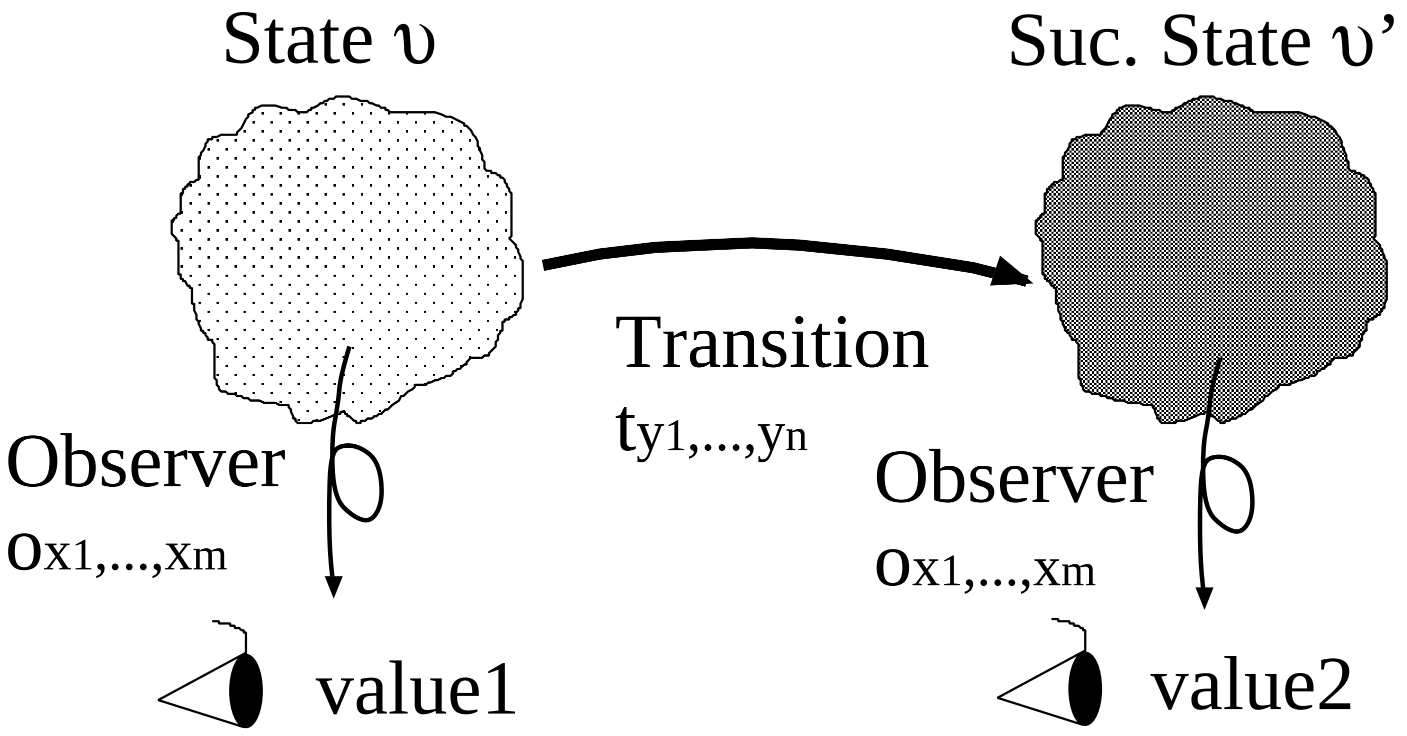
1. Kazuhiro Ogata, Masahiro Nakano, Weiqiang Kong and Kokichi Futatsugi: Induction-Guided Falsification, Prceedings of the 8th International Conference on Formal Engineering Methods (8th ICFEM), LNCS 4260, Springer, pp.114-131 (2006).
2. Weiqiang Kong, Kazuhiro Ogata, Masaki Nakamura and Kokichi Futatsugi: A Review of Induction-Guided Falsification and Towards its Automation, Preliminary Proceedings of the 1st Asian Working Conference on Verified Software (1st AWCVC), pp48-59 (2006).
3. Kazuhiro Ogata, Weiqiang Kong and Kokichi Futatsugi: Falsification of OTSs by Searches of Bounded Reachable State Spaces, Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (18th SEKE), Knowledge Systems Institute, pp.440-445 (2006).

Outline of Talk

A mutual exclusion protocol is used as an example to describe the proposed solution.

Informal Description of OTSs

OTSs are transition systems, which can be straightforwardly written as algebraic specifications.



Definition of OTSs

Suppose a universal state space Υ and data types D_* used in OTSs.

An OTS $\mathcal{S}$ is $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ such that

- $\mathcal{O}$: A finite set of observers.

  Each observer is an indexed function $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \rightarrow D_o$.

  $\upsilon_1 =_{\mathcal{S}} \upsilon_2$ iff $\forall o_{x_1,\ldots,x_m} : \mathcal{O}.\forall x_1 : D_{o1} \ldots (o_{x_1,\ldots,x_m}(\upsilon_1) = o_{x_1,\ldots,x_m}(\upsilon_2))$.
- $\mathcal{I}$: The set of initial states such that $\mathcal{I} \subseteq \Upsilon$.
- $\mathcal{T}$: A finite set of transitions.

  Each transition is an indexed function $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \Upsilon$ provided that $t_{y_1,\ldots,y_n}(\upsilon_1) =_{\mathcal{S}} t_{y_1,\ldots,y_n}(\upsilon_2)$ for each $[\upsilon] \in \Upsilon/{=_{\mathcal{S}}}$, each $\upsilon_1, \upsilon_2 \in [\upsilon]$ and $\forall y_k : D_{tk}$ for $k = 1, \ldots, n$.

  $t_{y_1,\ldots,y_n}(\upsilon)$ is called *the successor state* of υ wrt $t_{y_1,\ldots,y_n}$.

  The condition $c\text{-}t_{y_1,\ldots,y_n}$ of $t_{y_1,\ldots,y_n}$ is called *the effective condition.*

Reachable States and Invariants

Reachable States: Given an OTS $\mathcal{S}$, reachable states wrt $\mathcal{S}$ are inductively defined:

- Each $\upsilon_{\text{init}} \in \mathcal{I}$ is reachable.
- For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1, \ldots, n$, $t_{y_1,\ldots,y_m}(\upsilon)$ is reachable if υ is reachable.

Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states wrt $\mathcal{S}$.

Invariants: A state predicate $p : \Upsilon \to \text{Bool}$ is called invariant wrt $\mathcal{S}$ if p holds in all reachable states wrt $\mathcal{S}$, i.e. $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.

Suppoe that every quantifier in state predicates is $\forall$.

Transition Trees and Bounded Reachable States

Bounded Reachable States: Given $\mathcal{S}$, bounded reachable states wrt $\mathcal{S}$ are inductively defined:

- Each $\upsilon_{\text{init}} \in \mathcal{I}$ is 0-bounded reachable.
- For each $t_{y_1,\ldots,y_n}$ and each y_k for $k = 1, \ldots, n$, $t_{y_1,\ldots,y_m}(\upsilon)$ is $(n+1)$-bounded reachable if υ is n-bounded reachable and $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$.

m-bounded reachable states are also n-bounded reachable if $m \le n$.
Let $\mathcal{R}_{\mathcal{S}}^{\le n}$ be the set of all n-bounded reachable states wrt $\mathcal{S}$.

Transition Trees: Given $\mathcal{S}$, transition trees wrt $\mathcal{S}$, where nodes are states and edges are transitions, are constructed:

- The roots are initial states of $\mathcal{S}$.
- For each $t_{y_1,\ldots,y_n}$ and each y_k for $k = 1, \ldots, n$, $t_{y_1,\ldots,y_m}(\upsilon)$ is a node and $t_{y_1,\ldots,y_m}$ is the edge from υ to the node if υ is a node and $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$.

A Mutual Exclusion Protocol (Ticket)

The pseudo-code executed by each process i :

```
Loop
    l1: $ticket[i] := tvm$;
    l2: $tvm := tvm + 1$;
    l3: repeat until $ticket[i] = turn$;
        Critical section;
    cs: $turn := turn + 1$;
```

- Initially, each process is at label l1, tvm is 0, $turn$ is 0 and $ticket[i]$ is 0 for each i.

OTS $\mathcal{S}_{\text{Ticket}}$ Modeling Ticket

- $\mathcal{O}_{\text{Ticket}} \triangleq \{\text{tvm} : \Upsilon \rightarrow \text{Nat}, \text{turn} : \Upsilon \rightarrow \text{Nat}, \text{ticket}_{i:\text{Pid}} : \Upsilon \rightarrow \text{Nat},$
  $\text{pc}_{i:\text{Pid}} : \Upsilon \rightarrow \text{Label}\}$
- $\mathcal{I}_{\text{Ticket}} \triangleq \{\upsilon_{\text{init}} \,|\, \text{tvm}(\upsilon_{\text{init}}) = 0 \wedge \text{turn}(\upsilon_{\text{init}}) = 0 \wedge$
  $\forall i : \text{Pid}.\,(\text{ticket}_i(\upsilon_{\text{init}}) = 0) \wedge \forall i : \text{Pid}.\,(\text{pc}_i(\upsilon_{\text{init}}) = \text{l1})\}$
- $\mathcal{T}_{\text{Tlock}} \triangleq \{\text{get}_{j:\text{Pid}} : \Upsilon \rightarrow \Upsilon, \text{inc}_{j:\text{Pid}} : \Upsilon \rightarrow \Upsilon, \text{enter}_{j:\text{Pid}} : \Upsilon \rightarrow \Upsilon,$
  $\text{leave}_{j:\text{Pid}} : \Upsilon \rightarrow \Upsilon\,\}$

$\text{inc}_j(\upsilon) \triangleq \upsilon'$ **if** $\text{c-inc}_j(\upsilon)$ **s.t.**

$\quad \text{tvm}(\upsilon') = \text{tvm}(\upsilon) + 1$

$\quad \text{turn}(\upsilon') = \text{turn}(\upsilon)$

$\quad \text{ticket}_i(\upsilon') = \text{ticket}_i(\upsilon)$

$\quad \text{pc}_i(\upsilon') =$ **if** $i = j$ **then** l3 **else** $\text{pc}_i(\upsilon)$

where $\text{c-inc}_j(\upsilon) \triangleq (\text{pc}_j(\upsilon) = \text{l2})$

An Invariant (Candidate) wrt $\mathcal{S}_{\text{Ticket}}$

One desired property Ticket should satisfy is *the mutual exclusion property*, which informally means that there is always at most one process in the critical section.

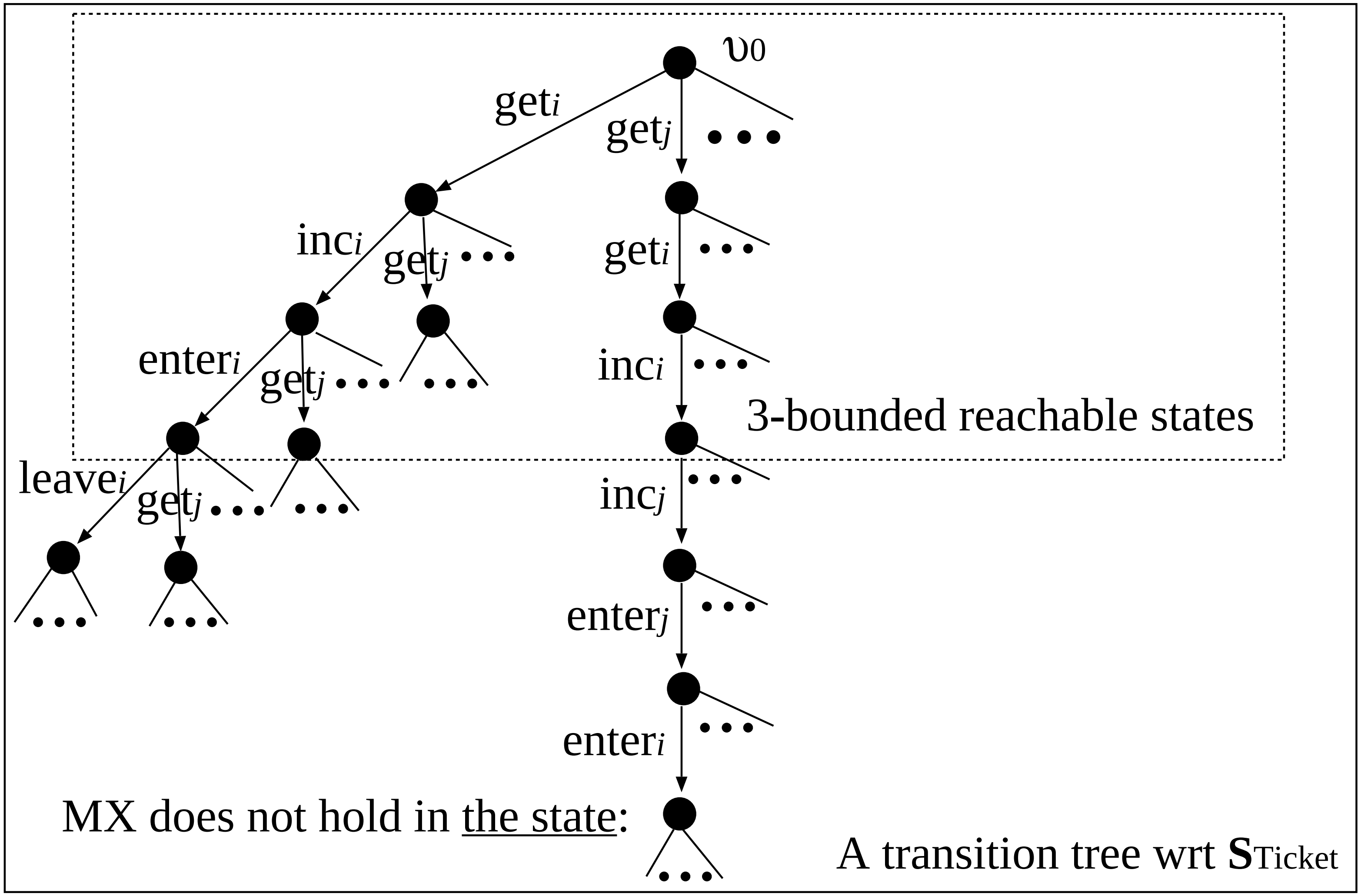
The property is expressed as

$$\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\text{Ticket}}}.\, \text{MX}(\upsilon)$$

where

- $\text{MX}(\upsilon) \triangleq \forall i, j : \text{Pid}\,(pc_i(\upsilon) = \text{cs} \wedge pc_j(\upsilon) = \text{cs} \Rightarrow i = j)\,.$

Bounded Reachable & Transition Tree wrt $\mathcal{S}_{\text{Ticket}}$



Maude

- An alg spec lang & sys, mainly developed at SRI Int'l & UIUC.
  - A member of the OBJ language family.
- State machines as well as data types.
  - Data types are specified in initial algebras.
  - State machines are specified in rewriting logic.
- A spec : a sig (sorts & ops decls) & a set of eqs and/or rewriting rules
  - Functions on data types are defined in equations.
  - Transitions of state machines are defined in rewriting rules.
- Equipped with model checking facilities.

Two Ways of Model Checking

1. The command `search` finds states that are reachable from a given state and satisfy some coditions. It can be used to find counterexamles showing that state machines do not satisfy invariant properties.
2. The LTL model checker checks if a state machine satisfies a property written in LTL.

For both ways,

- Inductive data types can be used freely.
  - No need to encode complex data types in basic data types.
- The state space of a state machine does not have to be finite.
  - The LTL model checker requires the reachable state space to be finite.
  - The command `search` even does not so.
- The performance is comparable to SPIN.

BMC with the Command `search`

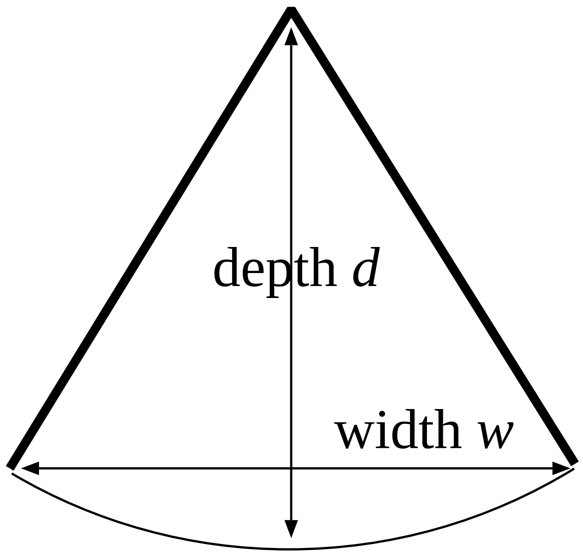
- The older versions of Maude did not provide any BMC functionalities.
- Even in the latest version Maude 2.3, the LTL model checker does not provide any BMC functionalities.
- In the latest version Maude 2.3, however, the command **`search`** has an optional argument n stating the maximum depth of the search.

  For an OTS $\mathcal{S}$ and $v_0 \in \mathcal{I}$, **`search`** searches the n-bounded reachable states in the transition tree made from $\mathcal{S}$ and v_0 for a counterexample showing that $\mathcal{S}$ does not satisfy an invariant property.

Since properties we take into account are invariant properties only, the command **`search`** can be used for our purpose.

Instantiation of the Numbers of Entities

- Even if d is finite, w may not be finite and then $\mathcal{R}_{\mathcal{S}}^{\leq d}$ may not be finite.
- To make it possible for the command `search` to make an exhautive search for $\mathcal{R}_{\mathcal{S}}^{\leq d}$, w should be finite.



A transition tree wrt **S**

To this end, the numbers of some entities such as processes have to be finite.
Such instantiation should be made to write state machines in Maude, precisely to write initial states, although transitions can be mostly written in Maude independent of the numbers of entities.

Maude Specification of $\mathcal{S}_{\mathrm{Ticket}}$ (1)

The signature is as follows:

```
sorts TRule OValue Sys .
subsorts TRule OValue < Sys .
*** Configuration
op none : -> Sys .
op __ : Sys Sys -> Sys [assoc comm id: none] .
*** Observable values
op pc[_] :_     : Pid Label -> OValue .
op ticket[_] :_ : Pid Nat -> OValue .
op turn :_      : Nat -> OValue .
op tvm :_       : Nat -> OValue .
*** Transition rules
ops get inc enter leave : -> TRule .
```

Maude Specification of $\mathcal{S}_{\mathrm{Ticket}}$ (2)

The def of transition $\mathrm{inc}_{\mathtt{I}}$ is written as follows:

```
rl [inc] : inc (pc[I] : l2) (ticket[I] : T) (tvm : V)
   => inc (pc[I] : l3) (ticket[I] : T) (tvm : (T + 1)) .
```

When the number of processes is 2, the initial state is written as follows:

```
eq init = get inc enter leave
          (pc[p1] : l1) (ticket[p1] : 0)
          (pc[p2] : l1) (ticket[p2] : 0)
          (turn : 0) (tvm : 0) .
```

Bounded Model Checking $\mathcal{S}_{\mathrm{Ticket}}$

- When the maximum depth n of the search is 6 (or more), the command `search` finds a counterexample showing that $\mathcal{S}_{\mathrm{Ticket}}$ does not satisfy the mutual exclusion property.
- When n is 5 or less, no counterexamples are found.
- But, what if the 6-bounded reachable state space were too large to be exhaustively traversed within a reasonable time?

  — *the state explosion problem.*

  – Some possible solutions to the problem have been proposed such as use of abstraction.

  – We propose another possible solution to the problem, which uses mathematical induction.

CafeOBJ

- An alg spec lang & sys, mainly developed at JAIST.
  - A member of the OBJ language family.
- State machines as well as data types.
  - Data types are specified in initial algebras.
  - State machines are specified in hidden algebras.
- A spec : a signature (sort & operator decls) & a set of eqs.
  - Both functions and transitions are defined in equations.
- Used as a proof assistant.

Necessary Lemmas

Given an OTS $\mathcal{S}$ and a state predicate $p : \Upsilon \rightarrow \text{Bool}$, state predicates are called *necessary lemmas* for $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$ if the state predicates are needed to prove $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$ and are also invariant wrt $\mathcal{S}$ when p is invariant wrt $\mathcal{S}$.

- Given an OTS $\mathcal{S}$ and a state predicate $p : \Upsilon \rightarrow \text{Bool}$, necessary lemmas for $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$ can be systematically constructed.
- It may be hard to prove necessary lemmas invariant wrt $\mathcal{S}$.

  Necessary lemmas must be often strengthened to make good lemmas.
- Necessary lemmas are appropriate for falsification.

Let $\mathcal{NL}_{\mathcal{S},p}$ be the set of all necessary lemmas for $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$.

A Way to Construct Necessary Lemmas

Suppose that $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$ is proved by induction on the structure of $\mathcal{R}_{\mathcal{S}}$.

- In invariant verification with CafeOBJ, the main task is case splitting.

  Each case is represented by a set of equations.

  For each induction case, the case is split into sub-cases such that a formula to prove in the induction case reduces to either `true` or `false`.

- For a sub-case such that the formula reduces to `false` where E is the set of equations that represents the sub-case,

  $\neg(\bigwedge_{e \in E} e)$ is a necessary lemma for $\forall v : \mathcal{R}_{\mathcal{S}}.\, p(v)$.

CafeOBJ Specification of $\mathcal{S}_{\text{Ticket}}$

The signature is as follows:

```
*[Sys]*
bops tvm turn : Sys -> Nat
bop ticket    : Sys Pid -> Nat
bop pc        : Sys Pid -> Label
bops get inc enter leave : Sys Pid -> Sys
```

The def of transition $\text{inc}_{\texttt{J}}$ is written as follows:

```
ceq tvm(inc(S,J))        = s(tvm(S)) if pc(S,J) = l2 .
eq  turn(inc(S,J))       = turn(S) .
eq  ticket(inc(S,J),I) = ticket(S,I) .
eq  pc(inc(S,J),I)
    = (if I = J then l3 else pc(S,I) fi) if pc(S,J) = l2 .
ceq inc(S,J)             = S if not(pc(S,J) = l2) .
```

An Example of Constructing Necessary Lemmas

A Fragment of the proof of $\forall v : \mathcal{R}_{\mathcal{S}_{\text{Ticket}}}.\,\text{MX}(v)$ in CafeOBJ:

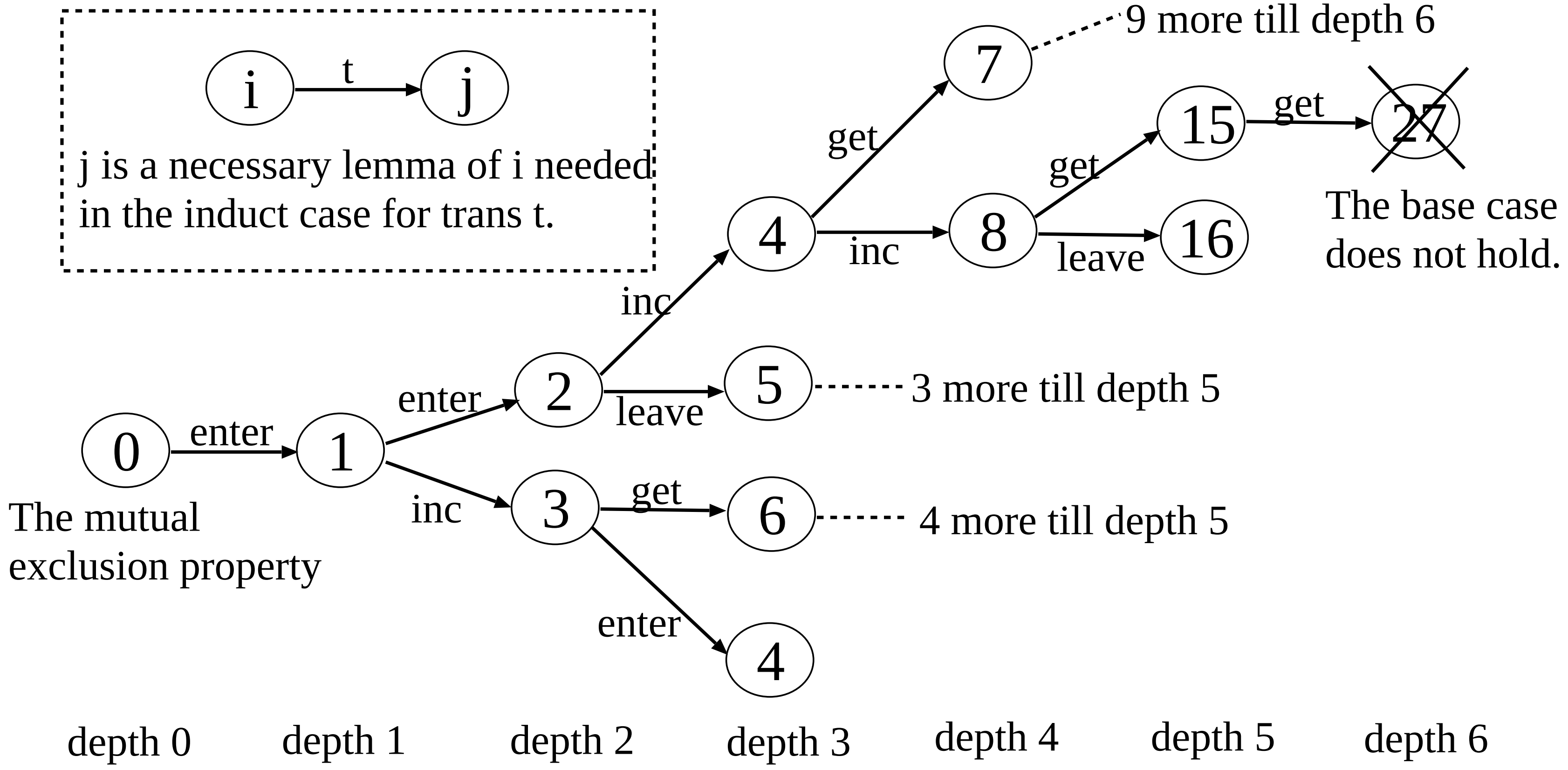
```
open ISTEP
-- arbitrary values
  op k : -> Pid .
-- assumptions
  eq pc(s,k) = l3 .
  eq ticket(s,k) = turn(s) .
  eq (i = k) = false .
  eq j = k .
  eq pc(s,i) = cs .
-- successor state
  eq s' = enter(s,k) .
-- enter
  red istep1(i,j) .
close
```

By substituting `j` for `k` in the 5 eqs that characterize the sub-case and replacing constants `s`, `i` and `j` with variables `S`, `I` and `J`, the following necessary lemma is constructed:

```
eq inv2(S,I,J)
   = not(pc(S,J) = l3             and
         ticket(S,J) = turn(S) and
         not(I = J)               and
         pc(S,I) = cs) .
```

Falsification of $\forall v : \mathcal{R}_{\mathcal{S}_{\text{Ticket}}}.\,\text{MX}(v)$ by Induction

Only induction on the structure of $\mathcal{R}_{\mathcal{S}_{\text{Ticket}}}$ can be used to find a counterexample of $\forall v : \mathcal{R}_{\mathcal{S}_{\text{Ticket}}}.\,\text{MX}(v)$.



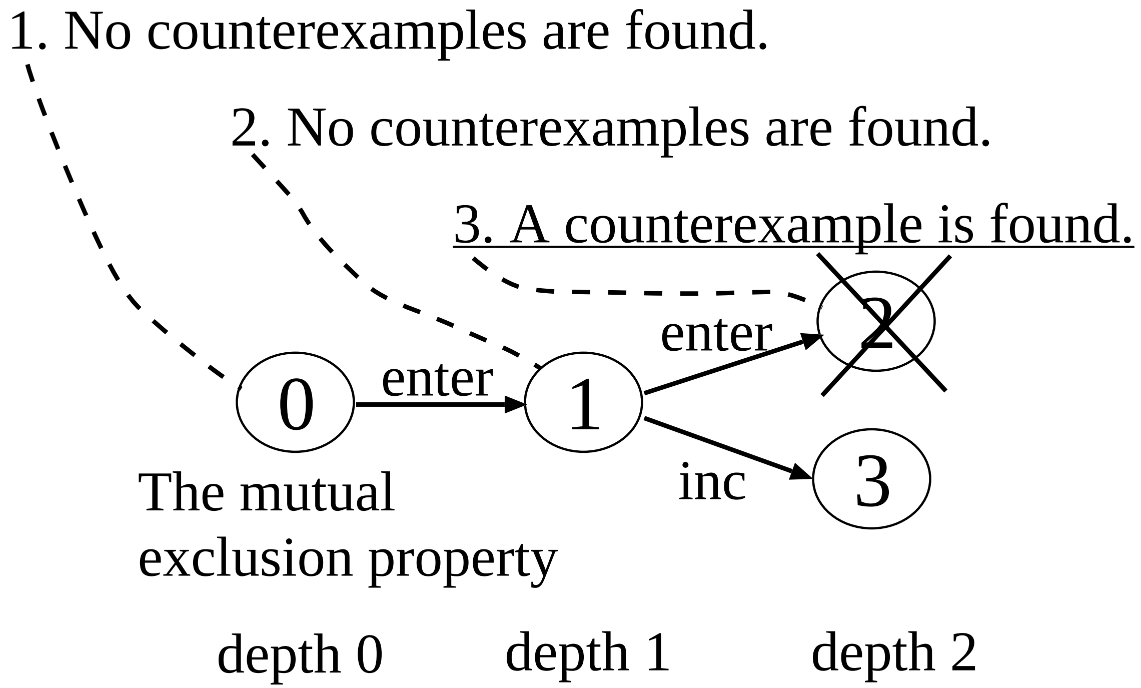
Procedure IGF

Given an OTS $\mathcal{S}$, a state predicate p and a natural number n, the procedure is defined as follows:

1. $\mathcal{Q} := \mathrm{put}(\mathrm{emptyQueue}, p)$ and $\mathcal{P} := \emptyset$.
2. Repeat the following until $\mathcal{Q} = \mathrm{emptyQueue}$.
   - 2.1. Let q be $\mathrm{top}(\mathcal{Q})$.
   - 2.2. Search $\mathcal{R}_{\mathcal{S}}^{\leq n}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$.
     If a counterexample is found, terminate and return Falsified.
   - 2.3. Compute all necessary lemmas $\mathcal{NL}_{\mathcal{S},q}$ for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, q(\upsilon)$.
   - 2.4. $\mathcal{P} := \{q\} \cup \mathcal{P}$ and $\mathcal{Q} := \mathrm{get}(\mathcal{Q})$.
   - 2.5. For each $r \in \mathcal{NL}_{\mathcal{S},q}$,
     if $\neg[\exists r' \in \mathrm{q2s}(\mathcal{Q}) \cup \mathcal{P}.\, (r' \Rightarrow r)]$, then $\mathcal{Q} := \mathrm{put}(\mathcal{Q}, r)$.
3. Terminate and return Verified.

Falsification of $\forall v : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\,\mathrm{MX}(v)$ by IGF

When n is 4, IGF finds a counterexample of $\forall v : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\,\mathrm{MX}(v)$ as follows:


1. No counterexamples are found.
2. No counterexamples are found.
3. A counterexample is found.
0
enter
1
enter
2
inc
3
The mutual exclusion property
depth 0
depth 1
depth 2

Summary

- We have proposed a way to find counterexamples showing that OTSs do not satisfy invariant properties by combining BMC and induction, which are done by Maude and CafeOBJ.

  The proposed solution can be regarded as a possible solution to the state explosion problem.

- An example (Ticket) has been used to describe the proposed solution.

Another case study:

⋆ The proposed solution has been applied to the NSPK authentication protocol.

Ongoing and Future Work (1)

- Neither Ticket nor NSPK can demonstrate the usefulness of the proposed solution.

  This is because only BMC can find counterexamples showing that Ticket does not satisfy the mutual exclusion property and that NSPK does not satisfy the secrecy property.

- We are planning to apply the proposed solution to the iKP electronic payment protocol.

  We found in an ad-hoc way a counterexample showing that iKP does not satisfy a property called the payment agreement property.

  We believe that only BMC cannot find such a counterexample, but the proposed solution can systematically find such a counterexample.

Ongoing and Future Work (2)

- Develop a tool that automates procedure IGF based on Cafe2Maude and Creme.
  - Cafe2Maude: A translator from CafeOBJ specs of OTSs into Maude specs of instances of the OTSs.
  - Creme: An automatic invariant verifier of OTSs, which uses an automatic case splitter.
- It depends on the numbers of some entities in an instance of an OTS whether the instance has a counterexample for an invariant property when the OTS has.

  We want to come up with something that can decide how many entities are enough to make sure that the instance has a counterexample when the OTS has.