Model checking GARP protocol using Spin and VRS

Igor Konnov, Lomonosov Moscow State University
O.A. Letichevsky, V.M. Glushkov Institute of Cybernetics NA of Sciences of Ukraine
Rationale

• We were looking for a case study:
  – parameterized by the number of communicating processes,
  – non-trivial one,
  – based on real-life problem.

• It will be nice to have a study with all the sources available
  – (like in Lafuente’s Promela database)
Introduction to GARP

- Generic Attribute Registration Protocol
- It is defined in IEEE 802.1D-2004
- End stations and MAC Bridges dynamically register (de-register) attributes
- An attribute is identified by a type and its value
Operation: registration of attributes

2-port bridge

End stations
Operation: de-registration

2-port bridge

3: LeaveEmpty

LAN

1: LeaveIn

2: LeaveEmpty

End stations
Operation: LeaveAll

1: LeaveAll

2: De-register (leave)
GARP Architecture

• GARP Participant: Application + GID
• Each port (on a bridge or a workstation) has a connected Participant
• Bridge participants are connected by GARP Information Propagation (GIP)
• GARP Information Directory (GID) has FSMs for each attribute:
  – Applicant State Machine
  – Registrar State Machine (optional)
• GID has also the only FSM:
  – LeaveAll State Machine (optional)
GID Components

• **Applicant** is looking after the interests of all would-be Participants and also recovers from the loss of one message
• **Registrar** records attribute value registrations declared by other Participants
• **LeaveAll** periodically initiates garbage collection by the Participant
The model by Tadashi Nakatani

• “Verification of Group Address Registration Protocol using Promela and Spin” (1997) based on IEEE 802.1p (outdated)

• The model of single segment LAN, i.e. that of one bridge (Registrar + LeaveAll) and two end stations (2 Applicants) in one LAN was considered
  – Tadashi modeled multicasting by sending messages to all the processes
  – Fast state explosion was observed
2. Our models in Promela
General abstractions (1)

- The only attribute and the only attribute value
- Several LANs are connected by bridges
- No more than 2 participants reside in the same LAN
- Two port and three port bridges are considered
General abstractions (2)

• Participants form a tree (assumed to be built by Rapid Spanning Tree Protocol) with:
  – end stations as leaves
  – and bridge ports as nodes
• Participants do not fail
• Spanning tree is not reconfigured
• Each channel may loose at most one message in a row
Promela processes

• Each participant is modeled as a Promela process:
  – GID_ARLA which has Application, Applicant, Registrar, and LeaveAll (bridge port)
  – GID_A which has Application and Applicant (end station)

• Participants are connected by asynchronous channels of length 1

• The participants in each bridge are interconnected via GIP process which forwards requests
B1: 1 bridge, 2 end stations

Diagram:

- GID_ARLA
- GID_A
- GIP2
- GID_ARLA
- GID_A
B2: 2 bridges, 3 end stations

```
+-----------------+-----------------+
| GID_ARLA        | GID_ARLA        |
|                 |                 |
| GIP3            | GIP2            |
|                 |                 |
| GID_ARLA        | GID_ARLA        |
|                 |                 |
| GID_A           | GID_A           |
+-----------------+-----------------+
```
B3: 3 bridges, 4 end stations

- GID_ARLA
  - GIP3
    - GID_ARLA
      - GID_A
    - GID_ARLA
      - GID_A
  - GID_ARLA
    - GID_A
  - GID_ARLA
    - GID_A
  - GID_ARLA
    - GID_A
Specifications: the basic ones

- Safety, i.e. absence of deadlock and livelock (D1)
- The coverage of all process local states, i.e. the reachability of a state containing a given local state, $T_2$: $\langle\rangle p$
- The reachability of a global state, where all the processes have registered the attribute, $T_3$: $\langle\rangle p$, $\neg p: (rs[1] == \text{IN} \&\& ... \&\& rs[N] == \text{IN})$
Specifications: an ambiguity

• An ambiguity in the standard:
  – according to separate tables of Applicant and Registrar (our GID) machine must move from LA.IN to VO.IN
  – in the combined table the machine moves from LA.IN to VO.LV

– S4: \([\mathcal{L} \ (p \rightarrow \leftrightarrow \ q) \ [\text{fairness required!}]]\)
  • p: GID_ARLA[1]:as == LA && rs[1] == IN
  • q: GID_ARLA[1]:as == VO && rs[1] == LV
Propagation of a request to join

• If a given participant declares the attribute and no participant leaves the attribute, then eventually the attribute is registered in all the participants

• S5: \([p \rightarrow (!r \lor (q \lor r)))\]
  - \(p\): GID_A[k]@join
  - \(q\): rs[1] == IN && ... && rs[N] == IN
  - \(r\): GID_A[k]@leave
Propagation of a request to leave

• If a given participant requests to leave the attribute, no participant re-joins the attribute and leavetimer does not expire infinitely often, then all the participants eventually leave the attribute

• S6: \[
\begin{align*}
\[](p \rightarrow (<> q || \[](!q && <>s) || (!q) U r))
\end{align*}
\]

- p: GID_A[k]@leave
- q: GID_A[1]@join && ... && GID_A[N]@join
- r: rs[1] == MT && ... && rs[N] == MT
- s: GID_ARLA[1]@leaveall && ... && GID_ARLA[N]@leaveall
No declaration without prior request

- If a given participant does not request join then it will never go to a state when it has declared the attribute

- \[ S7: []([[!p) || (!q) U p) \]
  - \( p: \text{GID}_A[k]@\text{join} \)
  - \( q: \text{GID}_A[k]:\text{as} == \text{AA} || \text{GID}_A[k]:\text{as} == \text{VA} || \text{GID}_A[k]:\text{as} == \text{QA} || \text{GID}_A[k]:\text{as} == \text{LA} \)
Propagation of ReqLeave by GIP

• A request to leave the attribute is forwarded by GIP iff all neighbour participants have already left it

• S8: $\Box (p \rightarrow q)$
  - $p$: $\text{gip\_out}[1]?[\text{ReqLeave}]$
  - $q$: $\text{rs}[1] == \text{MT} \land \ldots \land \text{rs}[N] == \text{MT}$

• This one should be probably invalid
Promela: sizes of processes

- **GID_A**: 19 control states
  - as (11 values), leave_timer (2 values), m (8 values)
  - 2 channels of mtype (8 values)

- **GID_ARLA**: 34 control states
  - as (11 v.), leave_timer (2 v.), m (8 v.), rs (3 v.)
  - 4 channels of mtype (8 v.)

- **GIP2**: 7 control states
  - 2 channels (8 v.)
3. Checking by Spin
Exhaustive search results

• Exhaustive search in various modes ran out of memory on B1, B2, and B3 (with limit of 1.9 G)

• For example, safety checking on B1:
  – COLLAPSE: depth > 22 M, #states > 43 M, #transitions > 87 M
  – HC: depth > 23 M, #states > 68 M, #transitions > 138 M
  – MA: depth > 18 M, #states > 631 M, #transitions > 1450 M
Bitstate search results

- Spin has a mode to perform an approximate search using bit hash table
- Parameters: 20 hash functions, $2^{31}$ entries in hash table
- Results for safety checking
  - B1: depth = 10.6 M, states = 363 M, transitions = 862 M, time = 37 min, memory = 968 M, hash factor = 5.9
  - B2: depth = 16.2 M, states = 19 M, transitions = 45 M, time = 127 sec, memory = 1891 M, hash factor = 112 (very good)
  - B3: depth > 10.9 M, states > 13 M, transitions > 33.4 M, time = 101 sec, out of memory, hash factor = 169
Checking other specifications

• As in the case of safety we checked the models using BITSTATE

• Spin had not reported any error on specifications assumed to be valid
  – in one run on B2 it missed an error on S8
  – in a run on B1 it found a counter-example on S8

• Coverage is predicted by hash factor value
  – The authors suggest to trust the results with hash factor greater than 100
  – We have got unexpectedly high hash factor (>100) on B2 and B3 opposite to that of B1 (5.9)
Troubleshooting

• Stack depth in B1, B2, and B3 is very large. Stack consumes the most of available memory.
  – Fortunately, Spin supports on-disk stack
• In order to decrease state space we used explicit state merging using d_step wherever possible (it decreases the number of control states)
How to deal with the explosion?

• **Abstraction?** There is no obvious abstract domain: all types are pretty simple (no counters, only enums) and control flow depends on different local variable values

• **Symmetry?** Maybe. There is no built-in support for it in Spin (though there are several papers even for Spin).

• **Symbolic MC?** We will try NuSMV.

• **Other techniques?** We are going to try SWARM tool which merges Spin’s results ran in different directions.
Checking by VRS

presented by O. Letichevsky…
We use deductive tools of **VRS (Verification Requirements System)** for proving of deadlock absence in GARP protocol.

Specifications and input language in VRS are presented as the set of basic protocols:

Basic Protocol is Hoara triple (Precondition, Action, Postcondition)

Especially for GARP example we formalize the transition form GARP protocol as basic protocol. For example:

Usage of symbolic modeling gives the possibility to consider the behavior of single GID. If behavior of single GID is free from deadlock then the system of the same GID is free from deadlocks.
We use method of static checking. Absence of deadlocks is defined by completeness of basic protocol system. System is incomplete if:

\[ \text{Precd}(1) \lor \text{Precd}(2) \lor \ldots \lor \text{Precd}(N) = 1 \]

If we cannot prove this disjunction we will consider its negation and get the formula of symbolic state of environment which present the set of concrete candidates for deadlocks.

We should transform the obtained formula to DNF and prove the reachability (unreachability) for every conjunct.

For proving of reachability we use two methods:

- backward symbolic modeling;
- proving the safety property as negation of deadlock formula
Results of proving:

380 – deadlocks candidates

230 – unreachability is proved by backward symbolic modeling;

80 – unreachability is proved as safety

70 candidates still not proved as deadlocks

Next actions:

1. Provide enhancements for backward symbolic modeling;

2. Use invariant searching technique (together with A.Godlevsky)