Verifying LTL properties of concurrent Ada programs with Quasar

ACM SIGAda 2003

S. Evangelista(*), C. Kaiser, J-F Pradat-Peyre and P. Rousseau(*)

CEDRIC CNAM - Paris

December 10, 2003
Quasar Analyzes concurrent Ada programs

Method: from source code to model

Based on the Petri nets formalism

Simple to use

- Automatic tool
- No Petri nets knowledge required
- Graphical interface
Quasar Presentation (2/2)

- **Quasar** proceeds in four steps:
  - **Slicing**: suppressing all the elements of the source code not related to the property to verify
  - **Translation**: translating the sliced source code into a Petri net
  - **Verification**: using structural and model-checking techniques to validate the property
  - Construction of a **report**: using counter-example and making the link between the formal model and the source code
task Type_T;
task body Type_T is
    My_Id : Id := 1;
begin
    loop
        Put_Line ("Before actions, task " & Id’Image (My_Id));
        Peterson.Enter (My_Id);
        Set_Controller_Instruction (My_Id);
        Peterson.Quit (My_Id);
        Put_Line ("After actions section, task " & Id’Image (My_Id));
    end loop;
end Type_T;

T_One : Type_T;
T_Two : Type_T;
Peterson Example (2/2)

Priority  :  Id := 1;
Candidate :  Tab.Candidate := (others => False);

procedure Enter (X : in Id) is
  Other : Id := (X mod 2) + 1;
begin
  Candidate (X) := True;
  Priority := Other;
  while Condition_Not_Satisfied loop null; end loop;
end Enter;

Let us check three solutions

• not ((Candidate (X)) and (Priority = X))
• (Candidate (Other)) and (Priority = Other)
• (Candidate (Other)) or (Priority = Other)
First Step: Slicing

- Sliced program: without the colored lines

```ada
task Type_T;
task body Type_T is
  My_Id : Id := 1;
begin
  begin
    loop
      Put_Line ("Before actions, task " & Id'Image (My_Id));
      Peterson.Enter (My_Id);
      Set_Controller_Instruction (My_Id);
      Peterson.Quit (My_Id);
      Put_Line ("After actions section, task " & Id'Image (My_Id));
    end loop;
  end loop;
end Type_T;

T_One : Type_T;
T_Two : Type_T;
```
Second Step : Translation - Patterns

- Building the Petri net with components: **patterns**
  - **sub-net**: a partial Petri net corresponding to an element of the Ada language
  - **meta-net**: an abstraction of sub-net used to represent general part of an element (example: the statements of a loop)

- and with **operators**:
  - **Substitution**: replacing a meta-net by its corresponding sub-nets
  - **Merging**: merging two sub-nets
Second Step: Translation - Example

- Substitution example
Third Step: Verification - Process

- **Expressing the properties with a formal temporal logic**
  - LTL (Linear Time Temporal Logic)
  - Atomic propositions
  - Propositional operators: ¬, ∧, ∨
  - Temporal operators:
    - $U$ [until] ($G$ [always], $F$ [eventually]), $X$ (next)

- **Verifying the properties** by model-checking
We want to verify the property:
“*If the task T_One is candidate to enter in the critical section, T_One will access the critical section*”

\[
\text{LTL} \rightarrow (\text{T\_One is Candidate}) \Rightarrow F (\text{T\_One is in CS})
\]

LTL manipulation:

\- **Not intuitive** and **error prone**

\- Difficulty to make **reference to specific parts of the program**
Third Step : Verification - Our solution

- Using **templates** for simplifying LTL manipulation
  - Concerns **usual properties**
  - Keeps the **advantages of LTL** (precision and expressiveness)
  - Keeps the **advantages of automatization**
State accessibility

- **Inevitable state**: $\neg s_0 \cup (s_0 \Rightarrow F s_1)$

\[\begin{align*}
\cdots & \quad s_0 & \quad \cdots & \quad s_1 \\
\neg s_0 & \quad & \neg s_1
\end{align*}\]

- **Inevitable state with condition**: $\neg s_0 \cup (s_0 \Rightarrow (\text{Cond} \cup F s_1))$

- **Home state**: $\neg s_0 \cup (s_0 \Rightarrow G(F s_1))$
Third Step: Verification - Templates (2/2)

- **Bounded Wait**: $G(s_0 \Rightarrow F s_1)$

- **Safety property**: $G(\neg s)$

- **Stability property**: $\neg f U G(f)$

- **Expert mode**: all LTL properties
Third Step: Verification - Program reference

- Semi-graphical definition of atomic properties based either on:
  - Value of variable
  - State of tasks (selected by line number)
  - ... still under development
    * Length of entry queues
Third Step: Verification - Example (cont.)

- Choice of the template → Bounded Wait: G (s₀ ⇒ F s₁)
  → s₀ : T_One is Candidate
  → s₁ : T_One is in CS

- Atomic proposition definition:
  - T_One is Candidate
    → value of a variable
    → (Candidate(1) = True)
  - T_One is in CS
    → Selection of a task variable and task body line
    → T_One@8
Fourth Step : Report

- Automatic detection of the sequence leading to the error
- Step by step graphical representation of this sequence
- Programmers can understand easily the design error using the generated trace
- Correction and new check of the program

Quasar allows us to verify that the second solution of Peterson example is the only valid one
Conclusion

- An easy way to **add verification of LTL properties** in **Quasar using templates**

- Future works
  - Extending **coverage of the language** (pointers, dynamic tasking, objects, ...)
  - Extending temporal properties to Computational Tree Logic (**CTL**)
  - **Improving** specific verification techniques
    - **Structural techniques** with colored Petri nets reductions
    - **Model-checking** using the knowledge of the generated Petri nets structure