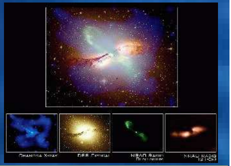
Quasar

A New Tool for Concurrent Ada Programs Analysis

S. Evangelista, C. Kaiser, J.F. Pradat-Peyre, P. Rousseau

June 2003

{evangelista, kaiser, peyre, rousseau}@cnam.fr
http://quasar.cnam.fr



The context

Quasar basics

Quasar in works

Conclusion

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Page 1 of 18

Go Back

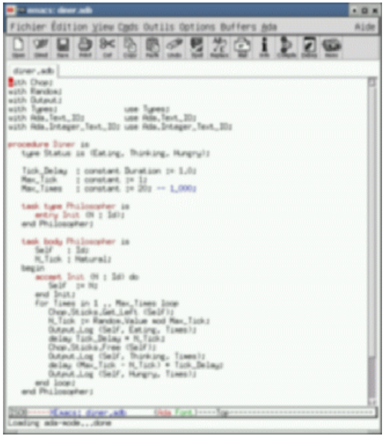
Full Screen

Close

Quit

1. The context

Given ANY program P



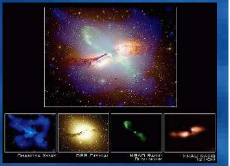
```
with Chop;
with Random;
with Output;
with Types;                 use Types;
with Ada.Text_IO;           use Ada.Text_IO;
with Ada.Integer_Text_IO;   use Ada.Integer_Text_IO;

procedure Diner is
   type Status is (Eating, Thinking, Hungry);

   Tick_Delay  : constant Duration := 1.0;
   Max_Tick    : constant := 1;
   Max_Times   : constant := 20; -- 1_000;

   task type Philosopher is
      entry Init (N : Id);
   end Philosopher;

   task body Philosopher is
      Self   : Id;
      N_Tick : Natural;
   begin
      accept Init (N : Id) do
         Self  := N;
      end Init;
      for Times in 1 .. Max_Times loop
         Chop_Sticks.Get_Left (Self);
         N_Tick := Random.Value mod Max_Tick;
         Output.Log (Self, Eating, Times);
         delay Tick_Delay * N_Tick;
         Chop_Sticks.Free (Self);
         Output.Log (Self, Thinking, Times);
         delay (Max_Tick - N_Tick) * Tick_Delay;
         Output.Log (Self, Hungry, Times);
      end loop;
   end Philosopher;
```

```
ISO8----XEmacs: diner.adb      (Ada Font)----Top----
Loading ada-mode...done
```

Given ANY property F

$$\forall x \in E, \exists y \mid P(x,y) = 1 \Longrightarrow F(x) \geq F(y)$$

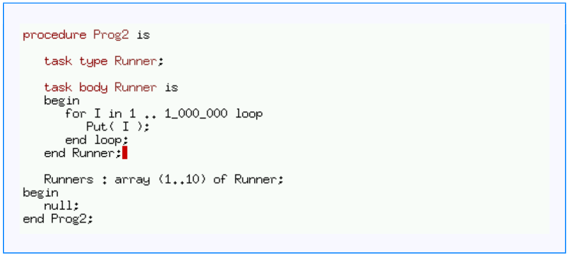
The problem to verify that P satisfies F is **undecidable**



The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 2 of 18
Go Back
Full Screen
Close
Quit

However, thinks are not so bad

Given **a specific** program P



```
procedure Prog2 is

   task type Runner;

   task body Runner is
   begin
      for I in 1 .. 1_000_000 loop
         Put( I );
      end loop;
   end Runner;

   Runners : array (1..10) of Runner;
begin
   null;
end Prog2;
```

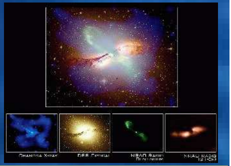
Given **a specific** context W

no exception occur during outputs

And given **a specific** property F

does the program ends ?

We can verify that P satisfies F under conditions W.



The context

Quasar basics

Quasar in works

Conclusion

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Page 3 of 18

Go Back

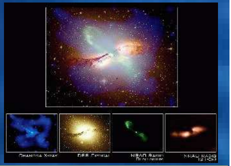
Full Screen

Close

Quit

General framework

- Write programs with well defined languages and limit language expression in order to enforce predictable behaviour.
  - for instance, with Ada, use the Ravenscar profil,
  - with Java, try to use the RT-Java definition,
  - with C/C++, use POSIX standards and environment profiles (PSE__)
- Make some assumptions on the execution context (such as exceptions occurrence, atomicity for specific statements, ...) in order to simplify the analysis process.
- Focus on specific properties in order to obtain a high level of expertise.
- Construct a formal representation of the program with respect to the analyzed property
- Use automatic tools that implement the most adequate techniques or strategies for proving or disproving the property.

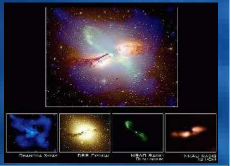


The context
Quasar basics
Quasar in works
Conclusion

Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 4 of 18
Go Back
Full Screen
Close
Quit

Automatic verification strategies

1. Use methods that are based on states enumeration :
   - + fully automatic,
   - + many properties can be checked with this strategy,
   - + very efficient for small or medium models or when the number of states can be reduced (with partial order methods, with bdd, ...),
   - − suffer from the combinatory explosion problem.
2. Use methods that are based on the analysis of the structure of the model :
   - + depend on the size of the model and not on the number of states that can be reached by the model,
   - + can perform parameterized analysis, and give “high level” comprehension of abnormal behaviours when necessary,
   - − partially automatic,
   - − some properties are difficult to be analyzed with this strategy (almost each property needs a specific method).



The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 5 of 18
Go Back
Full Screen
Close
Quit

2. Quasar basics

QUASAR is an automatic verification tool that focuses on software concurrent behaviour

- concurrent behaviour is very difficult to predict / understand by "manual" reasoning and a very little modification in the code can produce a major behaviour transformation

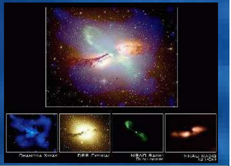
  $\Longrightarrow$ *an automatic tool is needed.*
- it is very difficult to reproduce an error once detected

  $\Longrightarrow$ *traditional debugging process cannot be employed*

  $\Longrightarrow$ *systematic "a-priori" analysis is required*

QUASAR is based on colored Petri nets

- colored Petri nets are a good compromise between modeling facilities and analysis possibilities;
- a colored Petri net can be analyzed both with
  - ∗ structural and parameterized techniques (e.g. invariants, reductions, ...)
  - ∗ and efficient states enumerations techniques;
- many experienced tools supporting colored Petri nets analysis are available (see Petri Nets home page)



The context

Quasar basics

Quasar in works

Conclusion

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Page 6 of 18

Go Back

Full Screen

Close

Quit

Quasar inside

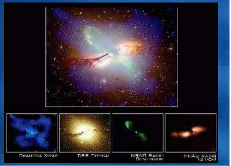
Quasar is composed of 3 modules

- A first module **extracts the relevant concurrent part** of the analyzed program and **produces a colored Petri net** by combining (merging or substitution) elementary patterns.
- A second module **analyses the colored Petri nets** produced at first step; this module first reduces the net by applying structural reductions and then uses states enumeration techniques.
- A third module **reports result of analysis**, in particular, a faulty trace can be reported when the property is not satisfied.

Quasar supports a large subset of Ada (see quasar.cnam.fr) and language restrictions are progressively removed.

Quasar is written in Ada and uses an implementation of the ASIS standard for the first module.

Quasar uses model checkers Prod and Maria for properties verification



The context

Quasar basics

Quasar in works

Conclusion

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Page 7 of 18

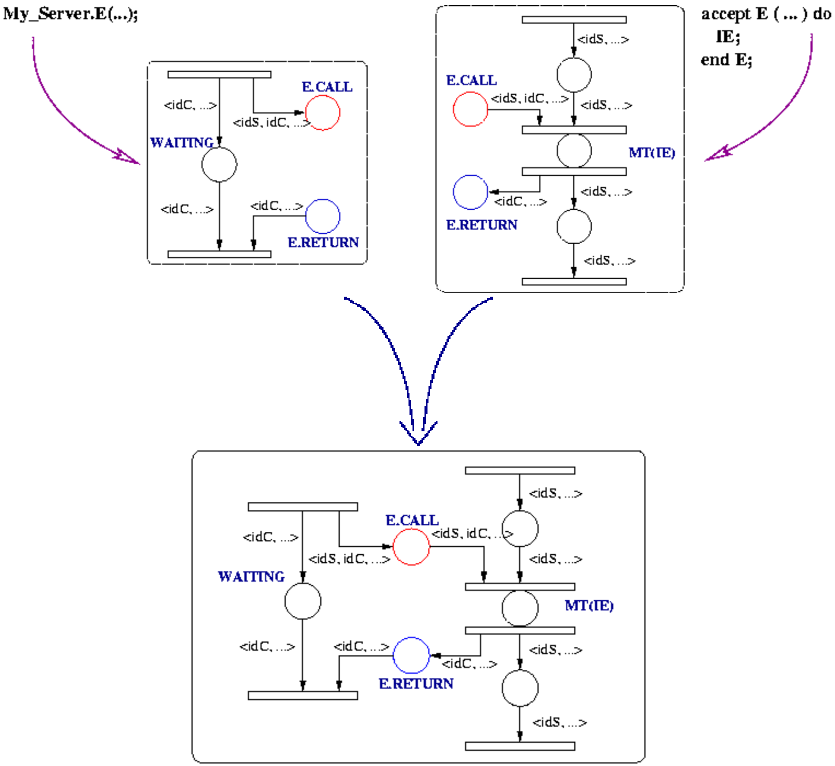
Go Back

Full Screen

Close

Quit

Patterns merging



```
My_Server.E(...);
```

```
accept E ( ... ) do
  IE;
end E;
```

The context

Quasar basics

Quasar in works

Conclusion

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Page 8 of 18

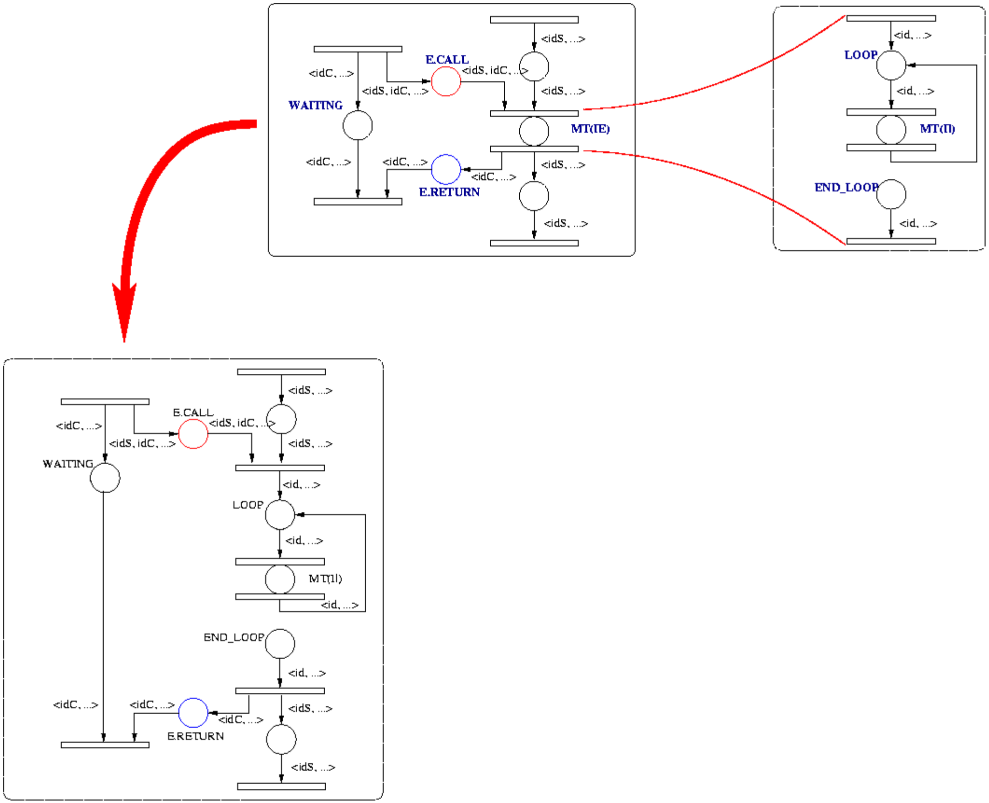
Go Back

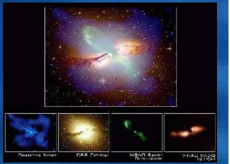
Full Screen

Close

Quit

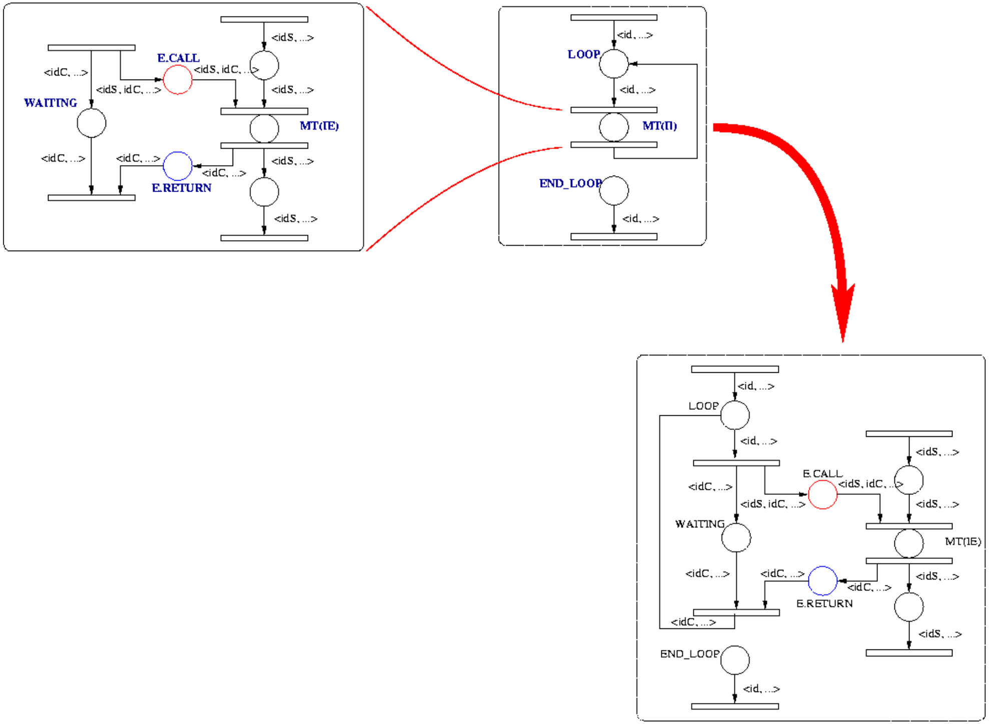
Patterns substitution





The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 9 of 18
Go Back
Full Screen
Close
Quit

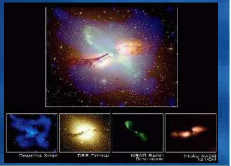
Patterns substitution (continued)



3. Quasar in works

Consider the following declarations :

```
type Money is new Natural;

 task Pump is
    entry Activate;
    entry Start;
    entry Finish;
end Pump;

 task Operator is
    entry Prepay (P : Money);
    entry Charge (V : Money);
end Operator;

 task Customer is
    entry Change (P : in Money);
end Customer;
```



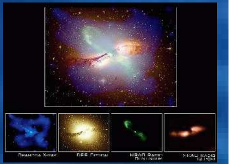
The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
Page 11 of 18
Go Back
Full Screen
Close
Quit

Quasar in works (continued)

```
task body Pump is
begin
   loop
      accept Activate;
      accept Start;
      accept Finish do
         Operator.Charge (5);
      end Finish;
   end loop;
end Pump;

task body Operator is
   Cash_Box : Money := 500;
   Paid      : Money := 0;
   Back      : Money := 0;
begin
   loop
      accept Prepay (P : Money) do
         Put_Line (”Pump_activation”);
         Pump.Activate;
         Paid := P;
      end Prepay;
      accept Charge (V : Money) do
         Back := Paid - V;
         Customer.Change (Back);
         Cash_Box := Cash_Box + V;
      end Charge;
   end loop;
end Operator;
```

```
task body Customer is
   Purse : Money := 20;
begin
   loop
      Purse := 20;
      Put_Line (”The_Custmer_calls_the_Operator”);
      Operator.Prepay (10);
      Purse := Purse - 10;
      Put_Line (”The_Custmer_starts_to_use_the_pump”);
      Pump.Start;
      Pump.Finish;
      Put_Line (”The_Custmer_stops_to_use_the_pump”);
      accept Change (P : in Money) do
         Purse := Purse + P;
      end Change;
   end loop;
end Customer;
```

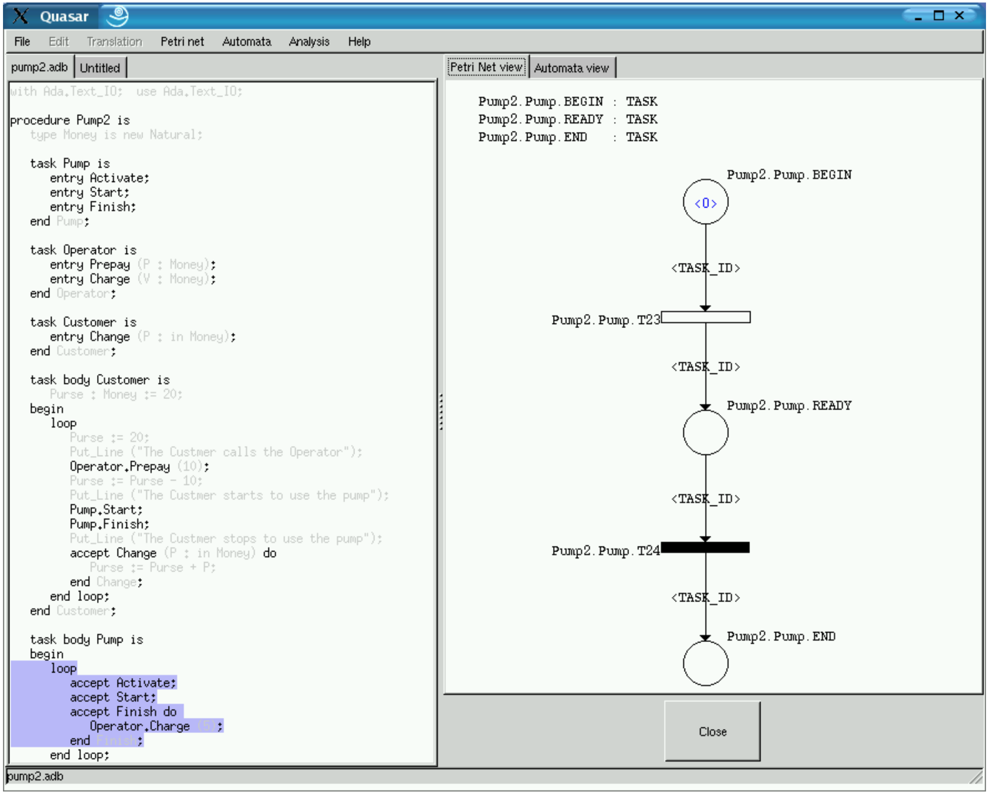
With QUASAR we can analyse and correct this program



The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶

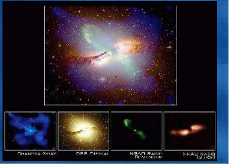
Go Back
Full Screen
Close
Quit

Translating the program

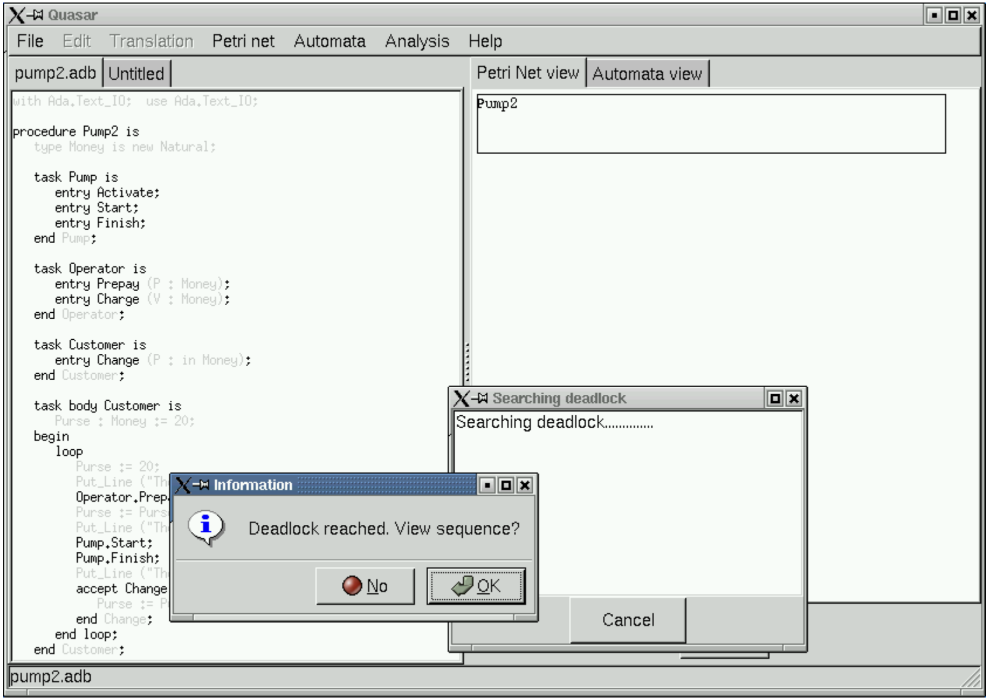


```
with Ada.Text_IO;  use Ada.Text_IO;

procedure Pump2 is
   type Money is new Natural;

   task Pump is
      entry Activate;
      entry Start;
      entry Finish;
   end Pump;

   task Operator is
      entry Prepay (P : Money);
      entry Charge (V : Money);
   end Operator;

   task Customer is
      entry Change (P : in Money);
   end Customer;

   task body Customer is
      Purse : Money := 20;
   begin
      loop
         Purse := 20;
         Put_Line ("The Custmer calls the Operator");
         Operator.Prepay (10);
         Purse := Purse - 10;
         Put_Line ("The Custmer starts to use the pump");
         Pump.Start;
         Pump.Finish;
         Put_Line ("The Custmer stops to use the pump");
         accept Change (P : in Money) do
            Purse := Purse + P;
         end Change;
      end loop;
   end Customer;

   task body Pump is
   begin
      loop
         accept Activate;
         accept Start;
         accept Finish do
            Operator.Charge (5);
         end Finish;
      end loop;
```

```
Pump2.Pump.BEGIN  :  TASK
Pump2.Pump.READY  :  TASK
Pump2.Pump.END    :  TASK
```



Tracking deadlock



```
with Ada.Text_IO;  use Ada.Text_IO;

procedure Pump2 is
   type Money is new Natural;

   task Pump is
      entry Activate;
      entry Start;
      entry Finish;
   end Pump;

   task Operator is
      entry Prepay (P : Money);
      entry Charge (V : Money);
   end Operator;

   task Customer is
      entry Change (P : in Money);
   end Customer;

   task body Customer is
      Purse : Money := 20;
   begin
      loop
         Purse := 20;
         Put_Line ("Th
         Operator.Prep
         Purse := Purs
         Put_Line ("Th
         Pump.Start;
         Pump.Finish;
         Put_Line ("Th
         accept Change
            Purse := P
         end Change;
      end loop;
   end Customer;
```

Searching deadlock.............

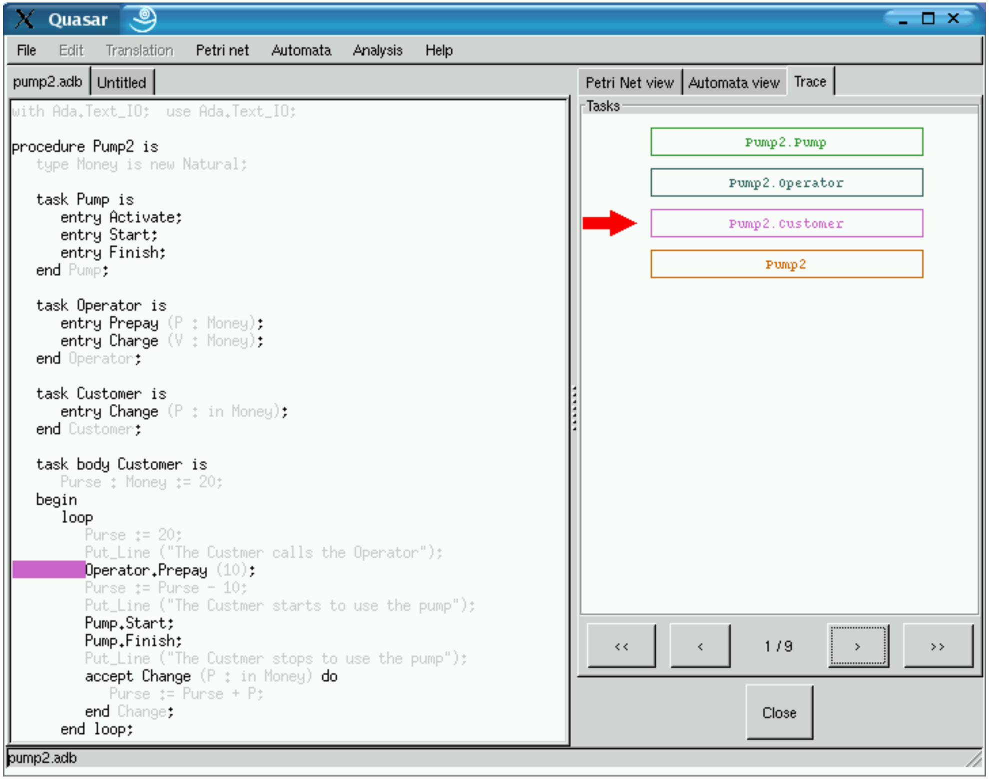
Deadlock reached. View sequence?

pump2.adb

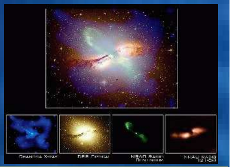
The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶

Go Back
Full Screen
Close
Quit

Analyzing the sequence leading to deadlock

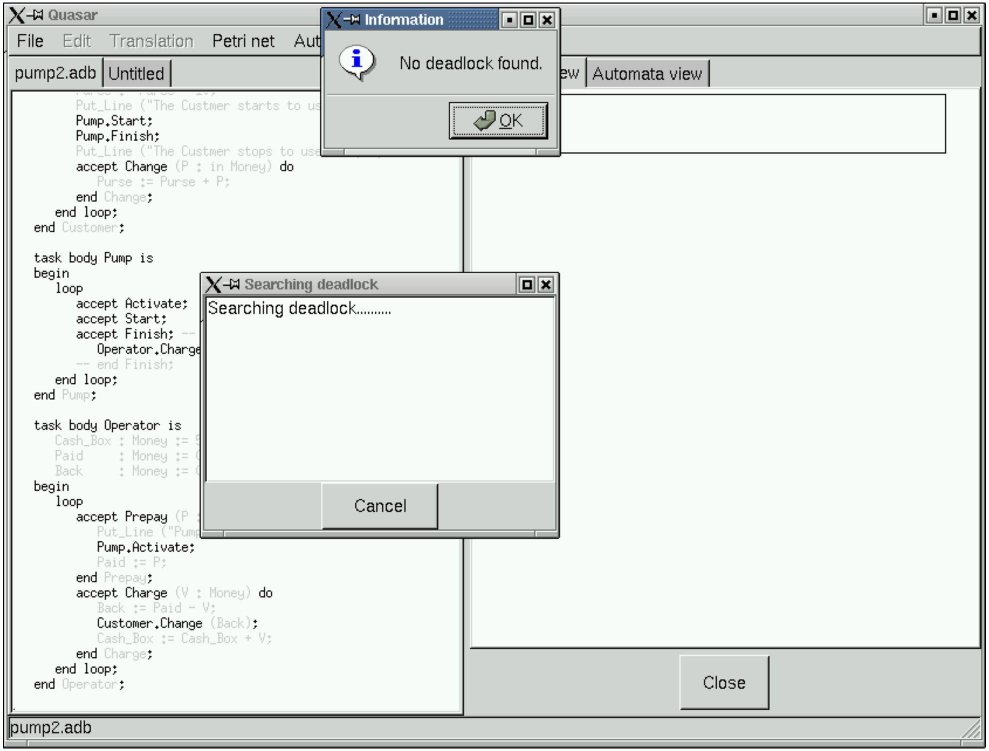


```
with Ada.Text_IO;  use Ada.Text_IO;

procedure Pump2 is
   type Money is new Natural;

   task Pump is
      entry Activate;
      entry Start;
      entry Finish;
   end Pump;

   task Operator is
      entry Prepay (P : Money);
      entry Charge (V : Money);
   end Operator;

   task Customer is
      entry Change (P : in Money);
   end Customer;

   task body Customer is
      Purse : Money := 20;
   begin
      loop
         Purse := 20;
         Put_Line ("The Custmer calls the Operator");
         Operator.Prepay (10);
         Purse := Purse - 10;
         Put_Line ("The Custmer starts to use the pump");
         Pump.Start;
         Pump.Finish;
         Put_Line ("The Custmer stops to use the pump");
         accept Change (P : in Money) do
            Purse := Purse + P;
         end Change;
      end loop;
```



The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
Page 15 of 18
Go Back
Full Screen
Close
Quit

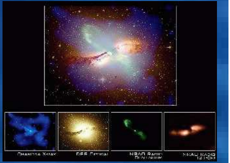
Verifying that correction suppresses the deadlock



```
        Put_Line ("The Custmer starts to us
        Pump.Start;
        Pump.Finish;
        Put_Line ("The Custmer stops to use
        accept Change (P : in Money) do
           Purse := Purse + P;
        end Change;
     end loop;
  end Customer;

  task body Pump is
  begin
     loop
        accept Activate;
        accept Start;
        accept Finish; --
           Operator.Charge
        -- end Finish;
     end loop;
  end Pump;

  task body Operator is
     Cash_Box : Money := 5
     Paid     : Money := (
     Back     : Money := (
  begin
     loop
        accept Prepay (P :
           Put_Line ("Pump
           Pump.Activate;
           Paid := P;
        end Prepay;
        accept Charge (V : Money) do
           Back := Paid - V;
           Customer.Change (Back);
           Cash_Box := Cash_Box + V;
        end Charge;
     end loop;
  end Operator;
```

No deadlock found.

Searching deadlock.........



The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 16 of 18
Go Back
Full Screen
Close
Quit

4. Conclusion

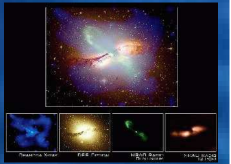
- Quasar is an automatic tool that can verify many properties related to concurrency

- Quasar is based on colored Petri nets that allow to combine structural verification techniques and states enumeration verification techniques

- Quasar translates the analyzed program into a colored Petri nets using pre-defined patterns and two compositional operators (merging and substitution); this methodology allows us to treat easily more and more program constructions.

- Quasar is modular and can be adapted to other languages or specific profile.

- Quasar uses well defined tools such as ASIS, GtkAda, Prod or Maria and is portable

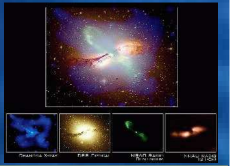


The context
Quasar basics
Quasar in works
Conclusion
Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 17 of 18
Go Back
Full Screen
Close
Quit

Future works

We make efforts in different directions :

- Augment the part of accepted language constructions
- Adapt or develop specific analyzis tools that take advantage of the structure of colored Petri nets produced by the translation module
- Develop more user-friendly interfaces for investigating concurrent properties corresponding to LTL formulae
- Enrich reports given when a required property is detected as non satisfied



The context
Quasar basics
Quasar in works
Conclusion

Home Page
Title Page
◀◀ ▶▶
◀ ▶
Page 18 of 18
Go Back
Full Screen
Close
Quit