

# Synchronous Languages—Lecture 08

Prof. Dr. Reinhard von Hanxleden

Christian-Albrechts Universität Kiel  
Department of Computer Science  
Real-Time Systems and Embedded Systems Group

14 May 2020

*Last compiled: May 19, 2020, 10:59 hrs*



*Schizophrenia Problems*

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program?

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program? (**Hint:** 2-step procedure)

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program? (**Hint:** 2-step procedure)
2. When does this fail?

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program? (**Hint:** 2-step procedure)
2. When does this fail?
3. What is the difference to the *logical behavioral semantics*?

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program? (**Hint:** 2-step procedure)
2. When does this fail?
3. What is the difference to the *logical behavioral semantics*?
4. What is the physical explanation/equivalent for constructiveness?

## The 5-Minute Review Session

1. How can we determine the *constructive behavioral semantics* of a program? (**Hint:** 2-step procedure)
2. When does this fail?
3. What is the difference to the *logical behavioral semantics*?
4. What is the physical explanation/equivalent for constructiveness?
5. What circuit property is equivalent to logical correctness?

# Overview

## Schizophrenia and Reincarnation

The Problem

Solving the Reincarnation Problem

Tardieu and de Simone (2004)



# Schizophrenia Problems

## Recall

- ▶ Synchronous programs consist of macro steps
- ▶ Macro steps consist of micro steps
- ▶ Transition rules define micro steps

## Questions:

- ▶ Can a statement be executed more than once in a macro step?
- ▶ If so, does this cause any problems?

# Schizophrenia Problems

## Recall

- ▶ Synchronous programs consist of macro steps
- ▶ Macro steps consist of micro steps
- ▶ Transition rules define micro steps

## Questions:

- ▶ Can a statement be executed more than once in a macro step?
- ▶ If so, does this cause any problems?

## Schizophrenic statements

- ▶ are statements that are started more than once in a macro step (eg., an emit), or left and entered in the same macrostep (eg., an abort)
- ▶ Although signal values do not change in the further starts, the repeated execution might differ!

## A Related Problem with Abortion

```

loop
  abort
    emit A;
    pause;
    emit B
  when I
end loop

```

Assume the control is at the pause and I is present

- ~> emit B is aborted
- ~> emit A is executed

Hence, we cannot simply say that

- ▶ Weak abortion executes all actions of the macro step
- ▶ And strong abortion kills these actions

Instead, it depends on whether the actions belong to the surface of the abort statement or to its depth

- ▶ **Surface** of a statement: parts that are reachable in one macrostep.
- ▶ **Depth** of a statement: all parts reachable in later macrosteps.

## Example for Schizophrenic Emission

```
loop
  present I then
    pause
  end present;
  emit A;
||
  pause
end loop
```

- ▶ The previous example was not yet schizophrenic
- ▶ However, consider Schizo1 on the left
- ▶ Assume I was present in the first instance and is absent in the second

## Example for Schizophrenic Emission

```
loop
  present I then
    pause
  end present;
  emit A;
||
  pause
end loop
```

- ▶ The previous example was not yet schizophrenic
- ▶ However, consider Schizo1 on the left
- ▶ Assume I was present in the first instance and is absent in the second
  - ↪ emit A is executed
  - ↪ loop restarts its body
  - ↪ present I ... is skipped
  - ↪ emit A is executed twice
- ▶ Hence, schizophrenic statements exist

## Schizophrenic Actions

- ▶ Is it a problem that statements may be executed more than once in a macro step?

## Schizophrenic Actions

- ▶ Is it a problem that statements may be executed more than once in a macro step?
- ▶ Since the value of a valued signal is always computed for a whole macrostep, it appears (at a first glance) not to be a problem
  - ▶ Executing `emit S` more than once makes `S` present
  - ▶ Executing `emit(S(i))` more than once has the same effect as the execution of multiple `emit(S(i))`
- ▶ So, the synchrony of the valued signal updates and the causal ordering of variable updates seems to make everything consistent

## Schizophrenic Actions

- ▶ Is it a problem that statements may be executed more than once in a macro step?
  - ▶ Since the value of a valued signal is always computed for a whole macrostep, it appears (at a first glance) not to be a problem
    - ▶ Executing `emit S` more than once makes `S` present
    - ▶ Executing `emit(S(i))` more than once has the same effect as the execution of multiple `emit(S(i))`
  - ▶ So, the synchrony of the valued signal updates and the causal ordering of variable updates seems to make everything consistent
  - ▶ However, scopes of local variables may be re-entered
  - ▶ This can change the environment in micro steps
- ~> *Reincarnation problem*



# The Reincarnation Problem

- ▶ The reincarnation problem is related to schizophrenia
- ▶ **Reincarnation** takes place, iff a local declaration is left and re-entered within the same macro step
- ▶ This is not necessarily a problem
- ▶ However, it may lead to unexpected behaviours
- ▶ In particular, in combination with schizophrenic statements, since **these may behave different in the second execution**

# The Simplest Example for Reincarnation

```
loop
  signal S in
    present S then
      emit S_on
    else
      emit S_off
    end;
  pause
  emit S;
  present S then
    emit S_on
  else
    emit S_off
  end;
end signal
end loop
```

## The Simplest Example for Reincarnation

```
loop
  signal S in
    present S then
      emit S_on
    else
      emit S_off
    end;
  pause
  emit S;
  present S then
    emit S_on
  else
    emit S_off
  end;
end signal
end loop
```

- ▶ If control starts at the pause, then S is emitted

## The Simplest Example for Reincarnation

```
loop
  signal S in
    present S then
      emit S_on
    else
      emit S_off
    end;
  pause
  emit S;
  present S then
    emit S_on
  else
    emit S_off
  end;
end signal
end loop
```

- ▶ If control starts at the pause, then S is emitted
- ▶ Second conditional emits S\_on
- ▶ Scope of local signal is left
- ▶ Loop restarts its body
- ▶ Scope of local signal is entered
- ▶ First conditional emits S\_off
- ▶ Control stops at pause

## The Simplest Example for Reincarnation

```

loop
  signal S in
    present S then
      emit S_on
    else
      emit S_off
    end;
  pause
  emit S;
  present S then
    emit S_on
  else
    emit S_off
  end;
end signal
end loop

```

- ▶ If control starts at the pause, then S is emitted
  - ▶ Second conditional emits S\_on
  - ▶ Scope of local signal is left
  - ▶ Loop restarts its body
  - ▶ Scope of local signal is entered
  - ▶ First conditional emits S\_off
  - ▶ Control stops at pause
- ~> *Both S\_on and S\_off are present for  $t > 0$*

## Compilation to Software

- ▶ Reincarnating local declarations is well-known from sequential imperative languages

## Compilation to Software

- ▶ Reincarnating local declarations is well-known from sequential imperative languages
  - ▶ It is handled by maintaining a stack that holds the current visible variables together with their values
  - ▶ If a local declaration is entered, an entry for the variable is put on the stack
  - ▶ During execution, the values of the variables on the stack may be changed; to this end, the stack is searched from top to bottom to find a variable
  - ▶ If a local declaration is left, the entry is deleted from the stack
- ~> No problem in [software](#)

# Schizophrenia

```
module P17:  
  output 0;  
  loop  
    signal S in  
      present S  
        then emit 0  
      end present;  
      pause;  
      emit S;  
    end signal  
  end loop  
end module
```



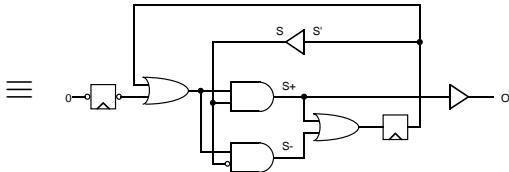


## Schizophrenia

```

module P17:
output 0;
loop
  signal S in
  present S
    then emit 0
    end present;
  pause;
  emit S;
  end signal
end loop
end module

```



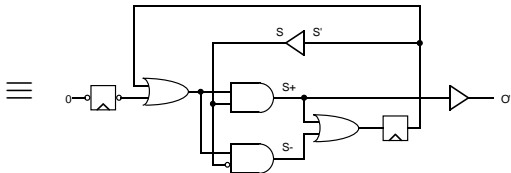


## Schizophrenia

```

module P17:
output 0;
loop
  signal S in
  present S
    then emit 0
  end present;
  pause;
  emit S;
  end signal
end loop
end module

```



- ▶ The circuit resulting from the translation rules (as given so far) does **not** behave as P17!
- ▶ **The Problem:** The circuit translation rules do not consider signal scoping rules
- ▶ Different signal incarnations are treated as identical

## Compilation Problem

The proposed hardware synthesis can still be used with the following adaptations:

- ▶ generate copies of locally declared signals (one for the surface and one for the depth)
- ▶ decide for every occurrence of these signals which copy is meant

**Note:** more than one copy may be required this way  
~> multiple reincarnation

# Schizophrenia

```
module P17:
  output 0;
  loop
    signal S in
      present S
        then emit 0
      end present;
      pause;
      emit S;
    end signal
  end loop
end module
```

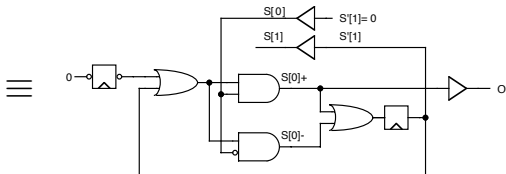
≡

## Schizophrenia

```

module P17:
  output 0;
  loop
    signal S in
      present S
        then emit 0
        end present;
      pause;
      emit S;
    end signal
  end loop
end module

```

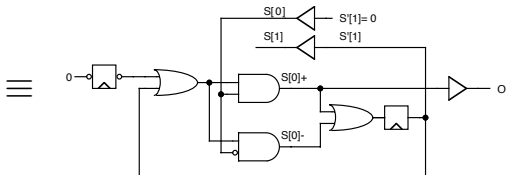


## Schizophrenia

```

module P17:
  output 0;
  loop
    signal S in
      present S
        then emit 0
        end present;
      pause;
      emit S;
    end signal
  end loop
end module

```



- ▶ In this circuit, signal is handled correctly by separating surface and depth

## Multiple Reincarnation

```

loop (a)
  trap T1 in (1)
    pause;
    exit T1
  ||
  loop (b)
    trap T2 in (2)
      pause;
      exit T2
    ||
    loop (c)
      emit 0(1);
      pause (3)
    end loop
  end trap
end loop
end trap
end loop

```

- ▶ 0 is an integer signal, combined by +



## Multiple Reincarnation

```

loop                                     (a)
  trap T1 in
    pause;                               (1)
  exit T1
||
  loop                                     (b)
    trap T2 in
      pause;                             (2)
    exit T2
  ||
    loop                                     (c)
      emit 0(1);
      pause                               (3)
    end loop
  end trap
end loop
end trap
end loop

```

- ▶ 0 is an integer signal, combined by +
- ▶ After first macrostep, control rests on all three pause statements in parallel

# Multiple Reincarnation

```

loop (a)
  trap T1 in
    pause; (1)
  exit T1
||
loop (b)
  trap T2 in
    pause; (2)
  exit T2
||
loop (c)
  emit 0(1);
  pause (3)
end loop
end trap
end loop
end trap
end loop

```

- ▶ 0 is an integer signal, combined by +
  - ▶ After first macrostep, control rests on all three pause statements in parallel
  - ▶ In the second macrostep:
    - ▶ pause (3) is left → restart loop (c) → 0(1) emitted
    - ▶ pause (2) is left → execute exit T2 → restart loop (b) → emit 0(1)
    - ▶ pause (1) is left → execute exit T1 → restart loop (a) → emit 0(1)
- ~> 0(1) is emitted three times

## Multiple Reincarnation

- ▶ Nested loops may even lead to multiple reincarnations
- ▶ **Note:** leaving and restarting a local declaration can only be done by a surrounding loop
- ▶ Number of nested loops around the local declaration corresponds with the number of possible reincarnations
- ▶ **Remark:** generated copies can, in principle, be substituted, however, the compilation is then even more complicated

## Schizophrenia

- ▶ Schizophrenia can be a problem even without local signal reincarnations

## Schizophrenia

- ▶ Schizophrenia can be a problem even without local signal reincarnations
- ▶ To illustrate, first consider the following circuit translation (which is equivalent to `sustain S`):

## Schizophrenia

- ▶ Schizophrenia can be a problem even without local signal reincarnations
- ▶ To illustrate, first consider the following circuit translation (which is equivalent to sustain S):

```
module SUSTAIN:  
  output S;  
  loop  
    emit S;  
    pause  
  end loop;  
end module
```

≡

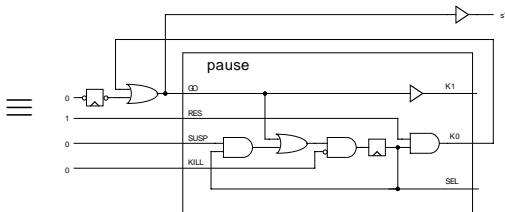
## Schizophrenia

- ▶ Schizophrenia can be a problem even without local signal reincarnations
- ▶ To illustrate, first consider the following circuit translation (which is equivalent to sustain S):

```

module SUSTAIN:
  output S;
  loop
    emit S;
    pause
  end loop;
end module

```



- ▶ K0 output of pause subcircuit feeds back to the GO input

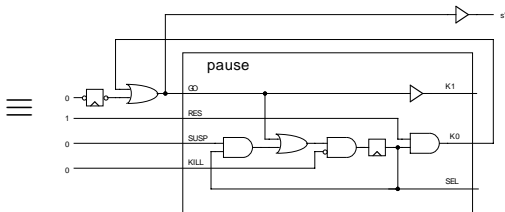
## Schizophrenia

- ▶ Schizophrenia can be a problem even without local signal reincarnations
- ▶ To illustrate, first consider the following circuit translation (which is equivalent to sustain S):

```

module SUSTAIN:
  output S;
  loop
    emit S;
    pause
  end loop;
end module

```



- ▶ K0 output of pause subcircuit feeds back to the GO input
- ▶ However, signal levels are always fully determined



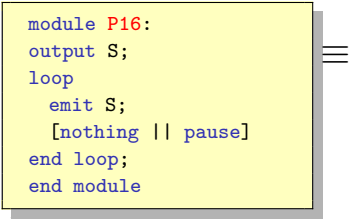
## Schizophrenia

- ▶ Now consider the circuit translation for P16, which should be equivalent to SUSTAIN:

## Schizophrenia

- ▶ Now consider the circuit translation for P16, which should be equivalent to SUSTAIN:

```
module P16:  
  output S;  
  loop  
    emit S;  
    [nothing || pause]  
  end loop;  
end module
```







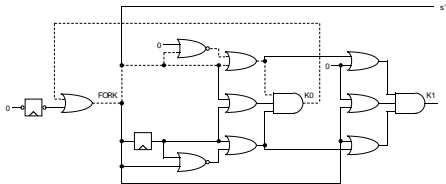
# Schizophrenia

- ▶ Now consider the circuit translation for P16, which should be equivalent to SUSTAIN:

```

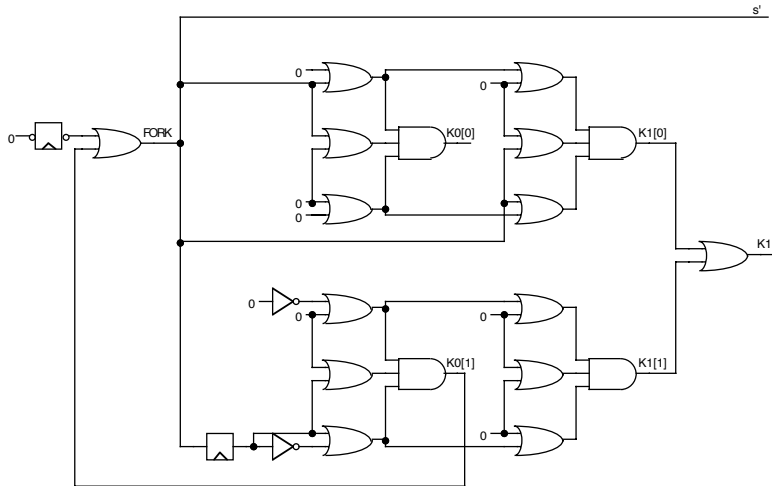
module P16:
  output S;
  loop
    emit S;
    [nothing || pause]
  end loop;
end module

```



- ▶ This circuit contains an unstable combinational loop (see dotted lines)
- ▶ Hence, the circuit is **not constructive!**
- ▶ **The problem:** reincarnation of parallel

## Schizophrenic Synchronizer



Correct circuit of  $(!s; (0 \mid 1))^*$

## Solutions to the Reincarnation Problem

Problematic for hardware circuit synthesis

- ▶ Variables are translated to wires and registers
- ▶ Wires must have unique values for every cycle!

### Questions

- ▶ Do schizophrenic local declarations require more than one wire?
- ▶ How to separate the scopes in the circuit?

### Solutions:

- ▶ Simple loop duplication
- ▶ Poigné and Holenderski (1995)  $\rightsquigarrow$  circuit level
- ▶ Berry (1996/1999)  $\rightsquigarrow$  circuit level
- ▶ Schneider and Wenz (2001)  $\rightsquigarrow$  program level
- ▶ Tardieu and de Simone (2004)  $\rightsquigarrow$  program level

## Reincarnation: Simple Solution

- ▶ A simple approach to eliminate schizophrenia (and hence reincarnation), is to duplicate loop bodies:

```
loop p end
```

⇒

```
loop p;p end
```

- ▶ Since  $p$  is not instantaneous, no part of  $p$  can be restarted immediately
  - ▶ We have to do this recursively
- ~> Worst-case increase of program size:



## Reincarnation: Simple Solution

- ▶ A simple approach to eliminate schizophrenia (and hence reincarnation), is to duplicate loop bodies:

```
loop p end
```

 $\Rightarrow$ 

```
loop p;p end
```

- ▶ Since  $p$  is not instantaneous, no part of  $p$  can be restarted immediately
  - ▶ We have to do this recursively
- ~> Worst-case increase of program size: Exponential

## Tardieu and de Simone (2004)

- ▶ Add unique labels to each pause statement
- ▶ New Esterel statement `gotopause` jumps to a labeled pause

## Tardieu and de Simone (2004)

- ▶ Add unique labels to each pause statement
- ▶ New Esterel statement `gotopause` jumps to a labeled pause
- ▶ Define function  $surf(p)$  to compute surface of  $p$  as:
  - ▶  $surf(\text{loop } p \text{ end}) = surf(p)$
  - ▶  $surf(p; q) = surf(p); surf(q)$  if  $p$  can be instantaneous
  - ▶  $surf(p; q) = surf(p)$  otherwise
  - ▶  $surf(\ell : \text{pause}) = \text{gotopause } \ell$

## Tardieu and de Simone (2004)

- ▶ Add unique labels to each pause statement
- ▶ New Esterel statement `gotopause` jumps to a labeled pause
- ▶ Define function  $surf(p)$  to compute surface of  $p$  as:
  - ▶  $surf(\text{loop } p \text{ end}) = surf(p)$
  - ▶  $surf(p; q) = surf(p); surf(q)$  if  $p$  can be instantaneous
  - ▶  $surf(p; q) = surf(p)$  otherwise
  - ▶  $surf(\ell : \text{pause}) = \text{gotopause } \ell$
- ▶ Define function  $dup(p)$  that expands loop bodies:
  - ▶  $dup(\text{loop } p \text{ end}) = \text{loop } surf(p); dup(p) \text{ end}$
- ▶ Omitted rules correspond to simple recursive calls

## Example with gotopause

Expand loop body by applying dup():

```
loop
  signal S in
    present S then emit 0 end;
    pause;
    emit S;
  end;
  present I then emit 0 end;
end loop
```

## Example with gotopause

Expand loop body by applying dup():

```

loop
  signal S in
    present S then emit 0 end;
    pause;
    emit S;
  end;
  present I then emit 0 end;
end loop

```

```

loop
  signal S in
    present S then emit 0 end;
    gotopause 1;
  end;
  signal S in
    present S then emit 0 end;
    1: pause;
    emit S;
  end;
  present I then emit 0 end;
end loop

```

► Optimization: remove dead code

## Tardieu and de Simone (2004)

- ▶ Program size grows quadratic in worst case, but linear in practice
- ▶ As by Schneider and Wenz, no new registers are introduced
- ▶ But there is still room for improvement . . .

## Tardieu and de Simone (2004)

- ▶ Program size grows quadratic in worst case, but linear in practice
- ▶ As by Schneider and Wenz, no new registers are introduced
- ▶ But there is still room for improvement ...
- ▶ **Observation 1:** Whether a program  $p$  is instantly re-started depends on both  $p$  and the context of  $p$

```
trap T in
  loop
     $p_1$ 
  end loop
end trap
```

```
loop
  trap T in
     $p_2$ ;
    pause
  end trap
end loop
```



## Tardieu and de Simone (2004)

- ▶ Program size grows quadratic in worst case, but linear in practice
- ▶ As by Schneider and Wenz, no new registers are introduced
- ▶ But there is still room for improvement ...
- ▶ **Observation 1:** Whether a program  $p$  is instantly re-started depends on both  $p$  and the context of  $p$

```

trap T in
  loop
     $p_1$ 
  end loop
end trap

```

```

loop
  trap T in
     $p_2$ ;
    pause
  end trap
end loop

```

- ▶  $p_1$  is instantly restarted when it returns completion code 0

## Tardieu and de Simone (2004)

- ▶ Program size grows quadratic in worst case, but linear in practice
- ▶ As by Schneider and Wenz, no new registers are introduced
- ▶ But there is still room for improvement ...
- ▶ **Observation 1:** Whether a program  $p$  is instantly re-started depends on both  $p$  and the context of  $p$

```

trap T in
  loop
    p1
  end loop
end trap

```

```

loop
  trap T in
    p2;
    pause
  end trap
end loop

```

- ▶  $p_1$  is instantly restarted when it returns completion code 0
- ▶  $p_2$  is instantly restarted when it returns completion code 2

## Tardieu and de Simone (2004)

Based on Observation 1, the program transformation can be enhanced with static program analysis

- ▶ Compute **potential completion codes** for each program fragment  $p$
- ▶ Compute **unsafe completion codes** for the context of  $p$
- ▶ If intersection is not empty, then  $p$  is potentially schizophrenic

## Tardieu and de Simone (2004)

Based on Observation 1, the program transformation can be enhanced with static program analysis

- ▶ Compute **potential completion codes** for each program fragment  $p$
- ▶ Compute **unsafe completion codes** for the context of  $p$
- ▶ If intersection is not empty, then  $p$  is potentially schizophrenic

**Observation 2:** Only signal declarations and parallel statements can lead to schizophrenic behavior

- ▶ The improved transformation does not blindly duplicate whole loop bodies, but instead duplicates only potentially schizophrenic signal declarations and parallel statements

## To Go Further

- ▶ Gérard Berry, The Constructive Semantics of Pure Esterel, Draft book, current version 3.0, Dec. 2002, Chapter 12, <http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.zip>
- ▶ Klaus Schneider and M. Wenz, A New Method for Compiling Schizophrenic Synchronous Programs, CASES 2001, <http://es.cs.uni-kl.de/publications/datarsg/ScWe01.pdf>
- ▶ Oliver Tardieu and Robert de Simone, Curing Schizophrenia by Program Rewriting in Esterel, MEMOCODE 2004 <http://www1.cs.columbia.edu/~tardieu/papers/memocode04.pdf>