Informal Description of RSML-e

- Specification consists of Variables and Interfaces
  - Variables maintain internal state of model
  - Interfaces describe interaction with the external environment
- Assignment Relations assign values to interfaces and variables
- Dataflow model of computation
  - Value of object can be computed as soon as objects on which it is dependent have been computed.
Informal Description of RSML$^e$

**Input Variables:**
- Light_Level_InVar: Integer
- Light_Level_Detectable_InVar: Boolean
- Occupied_InVar: Boolean
- Occupied_Detectable_InVar: Boolean
- OpIn_Window_LB_Intensity_InVar: Integer
- OpIn_Wall_LB_Intensity_InVar: Integer

**Output State Variables:**
- Con_Window_LB_Intensity_OutVar: Integer
- Con_Wall_LB_Intensity_OutVar: Integer
- Failed_OutVar: Boolean

---

**Variables** represent internal behavior and state

- **Input Variables** represent quantities from the environment
  - Sensor data
  - Commands from user-interface
  - Data from other software systems

- **State Variables** represent the internal state of the RSML$^e$ specification.
  - System modes
  - Outputs for external environment
  - Hierarchical and parallel composition
Informal Description of RSML\textsuperscript{e}

- **Interfaces** define interaction with the external environment
  - Communicate via messages
  - **Input Interfaces** define how the specification receives information from the environment
    - Read interfaces
    - Receive interfaces
  - **Output Interfaces** define how the specification sends information to the environment
    - Publish interfaces
    - Send interfaces

- Step behavior is described by the *next-state-relation*
  - Created from *assignment relations*: each state variable, interface is assigned by an assignment relation.
  - Assignment relations are ordered based on their data dependencies
  - Next-state relation is the relational composition of all assignment relations in the ordering.
Informal Description of RSML\textsuperscript{-e}

Tool Support
Environment—How??

RSML-e as a Specification Language

An Example and Discussion
Objectives

- To learn the basics of RSML-e
  - We will use a simple example to illustrate the language features and the basic modeling approach

Dataflow Languages

\[
\begin{align*}
X &= 2a / (b - c) \\
Y &= X + d
\end{align*}
\]
RSML-\(e\) is a Dataflow Language

The Clean Room

- Consider a room that is supposed to be sealed at all times

We do NOT want this
Clean Room Requirements

- To enter the room you have to go through an airlock
- To get in, you have to open the front door, step into the airlock, close the door, open the inside door, step into the room, and finally close the inside door
- To open a door, a person must request the door to be opened using some means (e.g. a button)
- Only one person should be allowed in the airlock at a time, and if the airlock is in use, other requests should be denied until the airlock is unoccupied
- At no point should both doors to the airlock be open, unless a power failure or catastrophic event occurs

More Clean Room

- If both doors are open, then the clean room must be considered contaminated, with serious financial consequences
- When entering the clean room, an individual must be “cleaned” using air scrubbers to remove particles from their clothes
- The duration for this cleaning is some application-defined constant
- Until the individual is clean, they should not be allowed into the clean room.
Even More Clean Room

- The system shall provide two alarm features
  - If an airlock is occupied for longer than a specified duration, a timeout alarm shall be generated
  - In case of some system malfunction or other catastrophe, pressing buttons within the clean room and the airlock will generate a panic alarm
    - In this event, both doors should be unlocked and people should be able to leave the clean room unhindered
- If an alarm is generated, it continues until the system is reset by an administrator
- The clean room may have 1-\(n\) airlocks, all of which behave identically

Simplifications

- The clean room only contains one airlock
- It is the administrator’s responsibility to ensure that the system is in a consistent state when the system is reset (i.e. no people in the airlocks)
- The sensors/actuators do not malfunction
- The cleaning interval is 60 seconds
- The timeout interval is 5 minutes
### Output Variables

<table>
<thead>
<tr>
<th>Output:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner_door_lock: bool</td>
<td>inner_door_lock is true when the inner door of the airlock is locked</td>
</tr>
<tr>
<td>outer_door_lock: bool</td>
<td>outer_door_lock is true when the outer door of the airlock is locked</td>
</tr>
<tr>
<td>decontaminate: bool</td>
<td>decontaminate is true during the decontamination interval when the system should “scrub” a user who is entering the clean room</td>
</tr>
<tr>
<td>panic_alarm: bool</td>
<td>panic_alarm is true at the instant when a user presses the panic_button, and true thereafter until the reset_button has been pressed</td>
</tr>
<tr>
<td>timeout_alarm: bool</td>
<td>timeout_alarm is true at the instant when the user has been in an airlock for longer than 300 seconds</td>
</tr>
</tbody>
</table>

### Input Variables

<table>
<thead>
<tr>
<th>Input:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>panic_button: bool</td>
<td>panic_button is true when any of the panic buttons are pressed</td>
</tr>
<tr>
<td>reset_button: bool</td>
<td>reset_button is true when a system reset request is generated</td>
</tr>
<tr>
<td>inner_door_request: bool</td>
<td>inner_door_request is true in the duration when a user is requesting to exit the clean room</td>
</tr>
<tr>
<td>outer_door_request: bool</td>
<td>outer_door_request is true in the duration when a user is requesting to enter the clean room</td>
</tr>
<tr>
<td>inner_door_open: bool</td>
<td>inner_door_open is true when the inner door is open</td>
</tr>
<tr>
<td>outer_door_open: bool</td>
<td>outer_door_open is true when the outer door is open</td>
</tr>
<tr>
<td>airlock_occupied: bool</td>
<td>airlock_occupied is true when the airlock is occupied</td>
</tr>
<tr>
<td>clock: bool</td>
<td>A clock pulse that is issued once per second</td>
</tr>
</tbody>
</table>
A Typical Scenario

- Initially, both doors to the airlock are locked.
- The user requests to enter (exit) the airlock
  - If the airlock is ‘in-use’ the request is denied
  - Otherwise, unlock the outer (inner) door
- The user opens the outer (inner) door
- The user closes the outer (inner) door
  - If the airlock is occupied, then lock the outer (inner) door and proceed to the next stage
  - If the airlock is unoccupied, the user must have decided not to enter the airlock, so the airlock is no longer ‘in-use’

Scenario Continued

- If the user is entering, then clean the user for 60 seconds
- Unlock the inner (outer) door and wait for the user to exit
- The user opens the door
- The user closes the door
  - If the airlock is still occupied, then the user must not yet have exited; repeat
  - If the airlock is unoccupied, then the process is complete and the airlock is no longer ‘in-use’
Clean Room in RSML-e

TYPE_DEF on_off         { off, on }  
TYPE_DEF door_status    { closed, open } 
TYPE_DEF door_lock_status { unlocked, locked } 
TYPE_DEF button_status  { not_pressed, pressed } 

CONSTANT decontamination_interval : TIME 
   VALUE: 60 S 
END CONSTANT 

CONSTANT timeout_interval : TIME 
   VALUE : 5 M 
END CONSTANT 

Start with Outputs

STATE_VARIABLE panic_alarm : on_off 
   PARENT : NONE 
   INITIAL_VALUE : off 
   CLASSIFICATION : State 

   TRANSITION off TO on IF panic_button 
   TRANSITION on TO off IF reset_button 

END STATE_VARIABLE
Start with Outputs

```plaintext
STATE_VARIABLE outer_door_lock : door_lock_status
    PARENT: NONE
    INITIAL_VALUE : locked
    CLASSIFICATION : State

    EQUALS unlocked IF outer_door_unlocked()
    EQUALS locked IF !outer_door_unlocked()

END STATE_VARIABLE

MACRO outer_door_unlocked() :
    TABLE
    someone wants to enter : T * *
    someone wants to exit : * T *
    panic_alarm = on        : * * T
    END TABLE

END MACRO
```

Define Variables and Organize

```
Example:
airlock_status

entry
- airlock_entry
  - entering
  - decontamination
  - awaiting_exit
  - exiting
  - completed

exit
- airlock_exit
  - entering
  - awaiting_exit
  - exiting
  - completed

outer_door_lock
- unlocked
- locked

panic_alarm
- off
- on

inner_door_lock
- unlocked
- locked

timeout_alarm
- off
- on
```

http://www.umsec.umn.edu
Start with Outputs

STATE_VARIABLE outer_door_lock : door_lock_status
  PARENT: NONE
  INITIAL_VALUE : locked
  CLASSIFICATION : State

  EQUALS unlocked IF outer_door_unlocked()
  EQUALS locked IF !outer_door_unlocked()
END STATE_VARIABLE

MACRO outer_door_unlocked() :
  TABLE
    ..airlock_entry IN_STATE entering : T * *
    ..airlock_exit IN_ONE_OF {awaiting_exit, exiting} : T *
    panic_alarm = on : * * T
  END TABLE
END MACRO

Start with Outputs

STATE_VARIABLE inner_door_lock : door_lock_status
  PARENT: NONE
  INITIAL_VALUE : locked
  CLASSIFICATION : State

  EQUALS unlocked IF inner_door_unlocked()
  EQUALS locked IF !inner_door_unlocked()
END STATE_VARIABLE

MACRO inner_door_unlocked() :
  TABLE
    ..airlock_entry IN_ONE_OF {awaiting_exit, exiting} : T * *
    ..airlock_exit IN_STATE entering : T *
    panic_alarm = on : * * T
  END TABLE
END MACRO
Define Variables and Organize

Example 1

`timeout_alarm` off on
`airlock_status` unoccupied entry exit
`airlock_entry` entering decontamination awaiting exit exiting completed
`airlock_exit` entering awaiting exit exiting completed
`inner_door_lock` unlocked locked
`panic_alarm` off on
`outer_door_lock` unlocked locked
`timeout_alarm` off on

Airlock Status

```plaintext
STATE_VARIABLE airlock_status :
  VALUES : {unoccupied, entry, exit}
  PARENT : None
  INITIAL_VALUE : unoccupied
  CLASSIFICATION: State

Transition unoccupied TO exit IF inner_door_request
Transition unoccupied TO entry IF
TABLE
  outer_door_request : T;
  inner_door_request : F;
END TABLE
```
Airlock Status (Cont.)

Transition exit TO unoccupied IF
TABLE
   PREV_STEP(.airlock_exit) IN_STATE completed : T *;
   reset_button                                    : * T;
END TABLE

Transition entry TO unoccupied IF
TABLE
   PREV_STEP(.airlock_entry) IN_STATE completed : T *;
   reset_button                                     : * T;
END TABLE
END STATE_VARIABLE

Define Variables and Organize

Example 1

airlock_status
  - unoccupied
  - entry
  - airlock_entry
    - entering
    - decontamination
    - awaiting_exit
    - exiting
    - completed
  - exit
    - airlock_exit
      - entering
      - awaiting_exit
      - exiting
      - completed

outer_door_lock
  - unlocked
  - locked

panic_alarm
  - off
  - on

inner_door_lock
  - unlocked
  - locked

timeout_alarm
  - off
  - on
Entering the Airlock

STATE_VARIABLE airlock_entry :
VALUES : {entering, decontamination, awaiting_exit, exiting, completed}
PARENT : airlock_status.entry
INITIAL_VALUE: UNDEFINED
CLASSIFICATION: State

Transition UNDEFINED TO entering IF TRUE
Transition entering TO decontamination IF
  TABLE
      outer_door_request : F ;
      outer_door_open   : F ;
      airlock_occupied  : T ;
  END TABLE

Transition decontamination TO awaiting_exit IF
  TIME - TIME_CHANGED(PREV_STEP(airlock_entry)) > decontamination_interval

Entering the Airlock (Cont.)

Transition awaiting_exit TO exiting IF inner_door_open
Transition exiting TO completed IF
  TABLE
      inner_door_open : F ;
      airlock_occupied : F ;
  END TABLE

Transition entering TO completed IF
  TABLE
      outer_door_request : F ;
      outer_door_open   : F ;
      airlock_occupied  : F ;
  END TABLE
END STATE_VARIABLE
Define Variables and Organize

Example 1

**airlock_status**

- unoccupied
- entry
- airlock_entry
  - entering
  - decontamination
  - awaiting_exit
  - exiting
  - completed
- exit
  - entering
  - awaiting_exit
  - exiting
  - completed

**outer_door_lock**

- unlocked
- locked

**panic_alarm**

- off
- on

**inner_door_lock**

- unlocked
- locked

**timeout_alarm**

- off
- on

---

**Final Outputs**

**STATE_VARIABLE** timeout_alarm : on_off

- **PARENT**: NONE
- **INITIAL_VALUE**: off
- **CLASSIFICATION**: State

**TRANSITION** off TO on IF

**TABLE**

- **airlock_status** IN_STATE unoccupied : F;
- **TIME** - **TIME_CHANGED**(airlock_status) > timeout_interval : T;

**END TABLE**

**TRANSITION** on TO off IF reset_button

**END STATE_VARIABLE**

**STATE_VARIABLE** decontaminate : boolean

- **PARENT**: NONE
- **INITIAL_VALUE**: FALSE
- **CLASSIFICATION**: Output

**EQUALS** TRUE IF ..airlock_entry IN_STATE decontamination

**EQUALS** FALSE IF not (.airlock_entry IN_STATE decontamination)

**END STATE_VARIABLE**
Inputs—Where are They??

IN_VARIABLE panic_button : boolean
    INITIAL_VALUE : FALSE
    CLASSIFICATION: MONITORED
END IN_VARIABLE

IN_VARIABLE reset_button : boolean
    INITIAL_VALUE : FALSE
    CLASSIFICATION: MONITORED
END IN_VARIABLE

IN_VARIABLE inner_door_request : boolean
    INITIAL_VALUE : FALSE
    CLASSIFICATION: MONITORED
END IN VARIABLE

How do the Inputs Get In??

MESSAGE Update_Message {
    f_panic_button IS boolean,
    f_reset_button IS boolean,
    f_inner_door_request IS boolean,
    f_outer_door_request IS boolean,
    f_inner_door_open IS boolean,
    f_outer_door_open IS boolean,
    f_airlock_occupied IS boolean
}

IN_INTERFACE Update_Interface :
    MIN SEP : UNDEFINED
    MAX SEP : UNDEFINED
    INPUT_ACTION : READ(Update_Message)
    HANDLER :
        CONDITION : TIME - Update_Interface::LAST_IO > 50 MS
        ASSIGNMENT
            panic_button := f_panic_button,
            reset_button := f_reset_button,
            inner_door_request := f_inner_door_request,
            outer_door_request := f_outer_door_request,
            inner_door_open := f_inner_door_open,
            outer_door_open := f_outer_door_open,
            airlock_occupied := f_airlock_occupied
        END ASSIGNMENT
    END HANDLER
END IN_INTERFACE
How do the Outputs Get Out??

MESSAGE Actuator_Message {
    f_inner_door_lock IS door_lock_status,
    f_outer_door_lock IS door_lock_status,
    f_decontaminate IS boolean,
    f_panic_alarm IS on_off,
    f_timeout_alarm IS on_off
}

OUT_INTERFACE Actuator_Interface :
MIN_SEP : UNDEFINED
MAX_SEP : UNDEFINED
OUTPUT_ACTION : SEND(Actuator_Message)
HANDLER :
    CONDITION : TRUE
    ASSIGNMENT
    f_inner_door_lock := inner_door_lock,
    f_outer_door_lock := outer_door_lock,
    f_decontaminate := decontaminate,
    f_panic_alarm := panic_alarm,
    f_timeout_alarm := timeout_alarm
END ASSIGNMENT
ACTION : SEND
END HANDLER
END OUT_INTERFACE

We Have Learned

- RSML-e is a synchronous dataflow language
- The basics of constructing an RSML-e model
  - Applied to the clean room example
    - Adopted from Mike Whalen
  - Short paper and full clean room available on web site
- Next time
  - Writing Proposals
Objectives

- To learn the syntax and semantics of RSML-e
  - Interfaces
  - State variables
  - Expressions
  - Time

http://www.umsec.umn.edu
The RSML-c Specification

```
component_def : def_list ;
def_list : /* empty */ | def_list def ;
def : type_def |
constant_def |
state_variable_def |
in_variable_def |
in_interface_def |
out_interface_def |
macro_def |
function_def |
message_def ;
```

Data From the Altimeters

```
TYPE_DEF AltitudeQualityType { Good, Bad }

IN_VARIABLE Altitude : INTEGER
  INITIAL_VALUE : Undefined
  UNITS : ft
  EXPECTED_MIN : 0
  EXPECTED_MAX : 40000
END IN_VARIABLE

IN_VARIABLE AltitudeQuality : AltitudeQualityType
  INITIAL_VALUE : Undefined
END IN_VARIABLE

MESSAGE AltitudeMessage {
  Alt IS INTEGER,
  aq IS AltitudeQualityType
}
```
Type Definition Syntax

```plaintext
type_def : TYPE_DEF IDENTIFIER '{' enum_element_list '}'

enum_element_list : IDENTIFIER
| enum_element_list ',' IDENTIFIER

type_ref : IDENTIFIER
| INTEGER_TYPE
| REAL_TYPE
| BOOLEAN_TYPE
| TIME
```

Input Interfaces in RSML-e

```plaintext
IN_INTERFACE AltitudeMessageInterface :
  MIN_SEP : 50 MS
  MAX_SEP : 100 MS
  INPUT_ACTION : RECEIVE(AltitudeMessage)
  RECEIVE_HANDLER :
    CONDITION : TRUE
    ASSIGNMENT
      Altitude := Alt,
      AltitudeQuality := aq
    END ASSIGNMENT
  END HANDLER
END IN_INTERFACE
```
In Interface Syntax

```
in_interface_def : IN_INTERFACE IDENTIFIER ':'
    MIN_SEP ':' expression  /* checked to be const */
    MAX_SEP ':' expression  /* checked to be const */
    INPUT_ACTION ':' in_interface_type_spec '{' IDENTIFIER '}'
    in_handler_list
    END IN_INTERFACE

in_interface_type_spec : RECEIVE
    | READ

http://www.umsec.umn.edu
```

Handlers?

```
in_handler : in_handler_type ':'
    CONDITION ':' condition
    in_assignment
    END HANDLER

in_handler_type : RECEIVE_HANDLER
    | HANDLER

http://www.umsec.umn.edu
```
### Input Interfaces in RSML-ε

**IN_INTERFACE AltitudeMessageInterface:**

- **MIN_SEP**: UNDEFINED
- **MAX_SEP**: UNDEFINED
- **INPUT_ACTION**: READ(AltitudeMessage)

**HANDLER:**

- **CONDITION**: TIME - AltitudeMessageInterface::LAST_IO > 1 s
  
  **ASSIGNMENT**
  
  - Altitude := Alt,
  - AltitudeQuality := aq

**END ASSIGNMENT**

**END HANDLER**

**END IN_INTERFACE**

### Error Checking in the Interface

**IN_INTERFACE AltitudeMessageInterface:**

- **MIN_SEP**: 50 MS
- **MAX_SEP**: 100 MS
- **INPUT_ACTION**: RECEIVE(AltitudeMessage)

**RECEIVE_HANDLER:**

- **CONDITION**: TABLE
  
  - Alt <= Altitude::EXPECTED_MAX : T;
  - Alt >= Altitude::EXPECTED_MIN : T;

- **ASSIGNMENT**
  
  - Altitude := Alt,
  - AltitudeQuality := aq

**END ASSIGNMENT**

**END HANDLER**

**END IN_INTERFACE**
No Input or Too Many Inputs??

HANDLER :
  CONDITION :
  TABLE
    TIME - AltitudeMessageInterface::LAST_IO < AltitudeMessageInterface::MIN_SEP : T *;
    TIME - AltitudeMessageInterface::LAST_IO > AltitudeMessageInterface::MIN_SEP : * T;
  END TABLE
  ASSIGNMENT
    Altitude := UNDEFINED,
    AltitudeQuality := Bad
  END ASSIGNMENT
END HANDLER

Pure Signals

MESSAGE EmptyMessage {}

IN_INTERFACE ResetMessageInterface :
  MIN_SEP : 50 MS
  MAX_SEP : 100 MS

  INPUT_ACTION : RECEIVE(EmptyMessage)

  RECEIVE_HANDLER :
    CONDITION : TRUE
    ASSIGNMENT
      ivReset := TRUE
    END ASSIGNMENT
END HANDLER

  HANDLER :
    CONDITION : TRUE
    ASSIGNMENT
      ivReset := FALSE
    END ASSIGNMENT
END HANDLER
END IN_INTERFACE
Constants Syntax

constant_def : CONSTANT IDENTIFIER ':' type_ref
    UNITS ':' IDENTIFIER
    VALUE ':' expression /* checked to be const */
    END CONSTANT
| CONSTANT IDENTIFIER ':' type_ref
    VALUE ':' expression /* checked to be const */
    END CONSTANT
|

Constants

CONSTANT AltitudeThreshold : INTEGER
    UNITS : ft
    VALUE : 2000
    END CONSTANT

CONSTANT Hysteresis : INTEGER
    UNITS : ft
    VALUE : 100
    END CONSTANT
State Variables Altitude Status

STATE_VARIABLE AltitudeStatus :
VALUES : { Unknown, Above, Below, AltitudeBad }
PARENT : NONE
INITIAL_VALUE : Unknown
CLASSIFICATION : State

EQUALS Unknown IF ivReset = TRUE
EQUALS Below IF
TABLE
BelowThreshold() : T;
AltitudeQualityOK() : T;
ivReset : F;
END TABLE

EQUALS Above IF
TABLE
BelowThreshold() : F;
AltitudeQualityOK() : T;
ivReset : F;
END TABLE

EQUALS AltitudeBad IF
TABLE
AltitudeQualityOK() : F;
ivReset : F;
END TABLE

END STATE_VARIABLE

Sending or Publishing??

OUT_INTERFACE FaultDetectionInterface :
MIN_SEP : UNDEFINED
MAX_SEP : UNDEFINED
OUTPUT_ACTION : PUBLISH(FaultMessage)

HANDLER :
CONDITION :
TABLE
ASWOpModes IN_STATE OK : T * ;
ASWOpModes IN_STATE FailureDetected : * T;
END TABLE
ASSIGNMENT
fault := FaultDetectedVariable
END ASSIGNMENT
ACTION : PUBLISH
END HANDLER
END OUT_INTERFACE
Sending or Publishing??

OUT_INTERFACE FaultDetectionInterface :

  MIN_SEP : UNDEFINED
  MAX_SEP : UNDEFINED
  OUTPUT_ACTION : SEND(FaultMessage)

  HANDLER :
    CONDITION :
      TABLE
      ASWOpModes IN_STATE OK : T * ;
      ASWOpModes IN_STATE FailureDetected : * T;
    END TABLE
    ASSIGNMENT
      fault := FaultDetectedVariable
    END ASSIGNMENT
    ACTION : SEND
  END HANDLER
END OUT_INTERFACE

Out Interface Syntax

out_interface_def : OUT_INTERFACE IDENTIFIER ':'

  MIN_SEP ':' expression /* checked to be const */
  MAX_SEP ':' expression /* checked to be const */
  OUTPUT_ACTION ':' out_interface_type_spec '(' IDENTIFIER ')' output_handler_list
END OUT_INTERFACE

out_interface_type_spec : SEND

  | PUBLISH


Out Handler Syntax

```
output_handler          : HANDLER ':'
                        |
                        CONDITION ': ' condition
                        |
                        out_assignment
                        |
                        ACTION ': ' out_handler_type
                        |
                        END HANDLER
                        ;
```

```
out_handler_type        : SEND
                        |
                        PUBLISH
                        |
                        NONE

```

Macros

```
MACRO BelowThreshold() :
    Altitude < AltitudeThreshold
END MACRO

MACRO AltitudeQualityOK() :
    TABLE
    AltitudeQuality = Good : T;
    END TABLE
END MACRO
```
Macro Syntax

optional_formalParms : /* EMPTY */
    | '(' formalParameterList ')
macroDef : MACRO IDENTIFIER optional_formalParms ':'
          condition
          END MACRO
          ;

Expressions

• All standard expressions on integer, real, and enumerated variables
• Some other interesting expressions

eventExpression : AT_TRUE '(' expression ')
    | AT_FALSE '(' expression ')
    | AT_CHANGED '(' expression ')
    ;
prevStepExpression : identifierExpression
    | PREV_STEP '(' identifierExpression ')'
More Expressions

static_variable_info_decl : EXPECTED_MIN
| EXPECTED_MAX
| UPPER_BOUND
| LOWER_BOUND
;

identifier_expression : identifier_name_path
| identifier_name_path DOUBLE_COLON static_variable_info_decl
| identifier_name_path DOUBLE_COLON MAX_SEP
| identifier_name_path DOUBLE_COLON MIN_SEP
| identifier_name_path DOUBLE_COLON THIS
| identifier_name_path DOUBLE_COLON TIME
| identifier_name_path DOUBLE_COLON LAST_IO

‘Previous’ Expressions

prev_expression : prev_step_expression
| PREV_ASSIGN '(' prev_step_expression optional_pv ')' '
| PREV_VALUE '(' prev_step_expression optional_pv ')' '
| TIME_ASSIGNED
  '(' prev_step_expression optional_ta ')' '
| TIME_CHANGED
  '(' prev_step_expression optional_ta ')' '
;