

Interactive Trace Replay for Event-B Models

Jan Gruteser, Michael Leuschel

12th Rodin Workshop
Düsseldorf, 10.06.2025



prob.hhu.de
stups.hhu.de

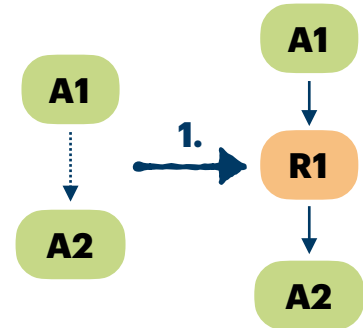
- Model checker, animator, and constraint solver
- High-level formal specification languages (like Event-B and TLA+)
- Tooling:
 - ▶ CLI
 - ▶ ProB Java API
 - ▶ ProB Rodin Plugin
 - ▶ ProB2-UI: VisB, SimB, *trace replay*
- Developed by STUPS group at HHU Düsseldorf (open source)
- Focus of this talk: **Trace Export and Replay**

- ProB allows to store and replay *traces* (automatically)
 - ▶ sequence of transitions
 - ▶ validate the model's behaviour at different stages of its development
- But: often breaking changes between model versions
 - ▶ e.g. renamed variables/events
 - ▶ Old traces can only be partially replayed (or not at all)
 - ▶ Refactoring/repairing traces can be hard for complex models
- Idea: instead of trying to replay everything automatically, let the user decide interactively in case of conflicts

Position ▲	Transition
0	---root---
1	SETUP_CONSTANTS
2	INITIALISATION
3	close_elevator_door
4	move_to_floor(f=1)
5	open_elevator_door
6	close_elevator_door
7	move_to_floor(f=4)
8	open_elevator_door
9	close_elevator_door
10	move_to_floor(f=-1)
11	open_elevator_door

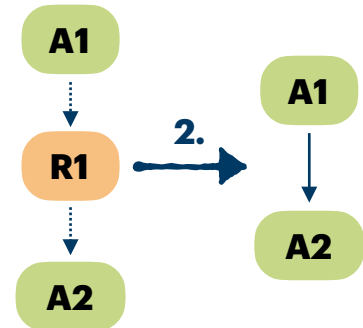
1. Refinement of Traces

- ▶ Insert new trace steps between two abstract steps by manual animation of new or refined events



2. Abstraction of Traces

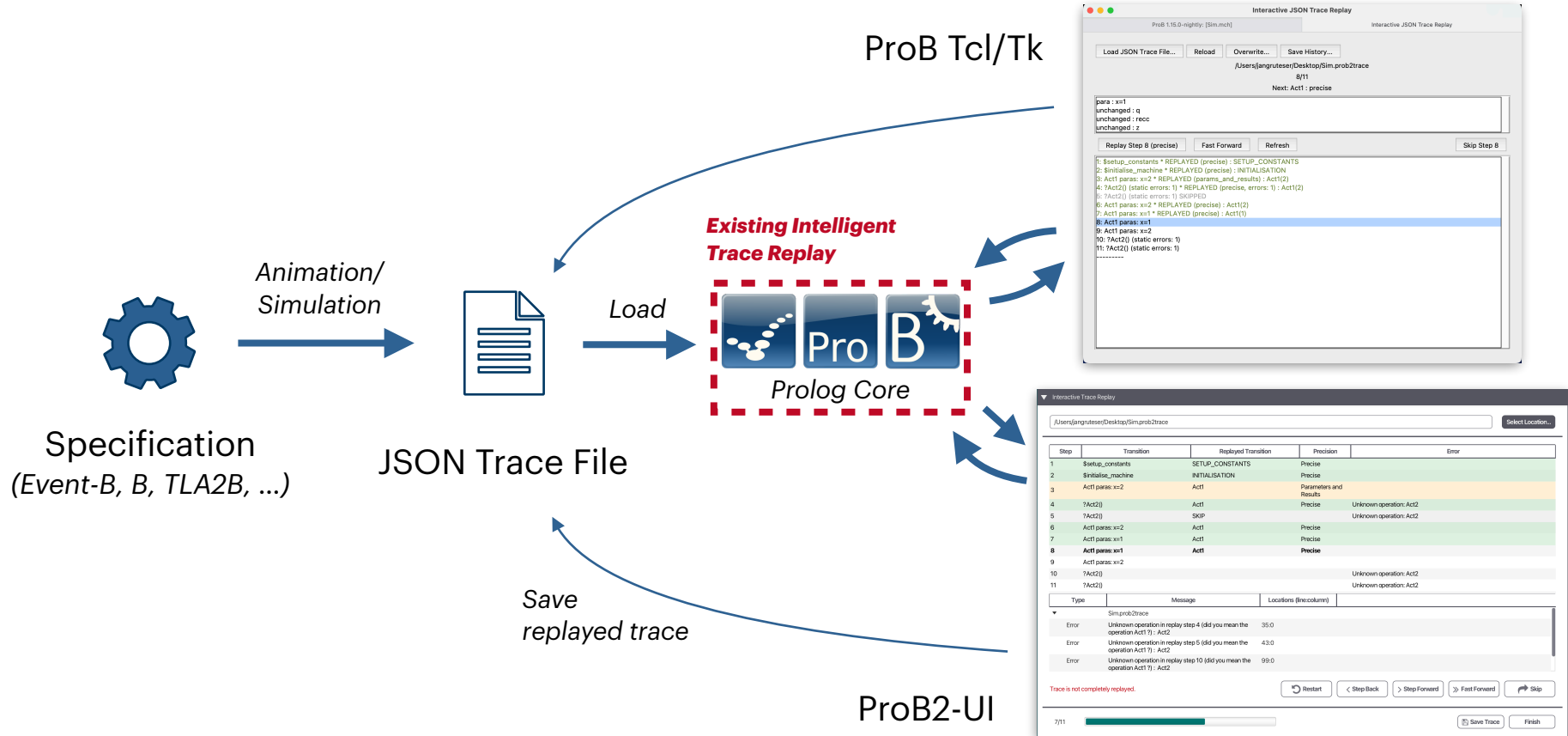
- ▶ Skip steps with unavailable concrete events



3. Refactoring/Repairing of Traces

- ▶ Refactor/repair traces of complex models after major changes

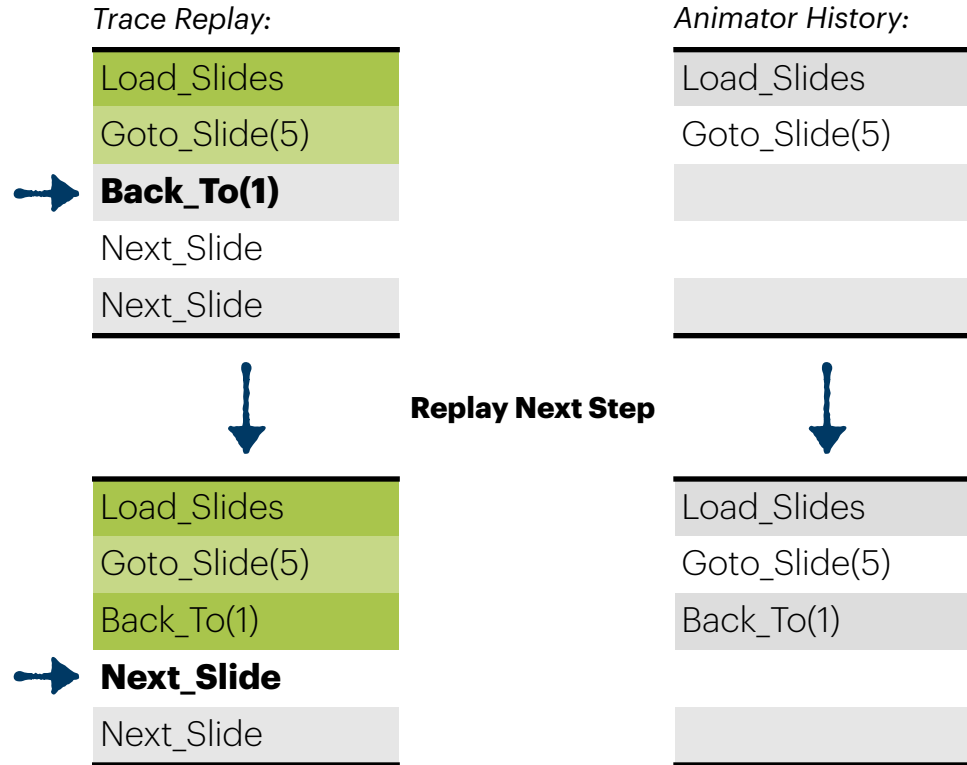
Overview



- “Intelligent” trace replay, automatic replay without user input
- Tries to resolve possible failure scenarios with different precision
 1. Perfect replay: *matching operation name, parameters and variable values after the trace step*
 2. Parameters: *matching operation name and parameters*
 3. Operation name: *matching operation name*
 4. Use another operation with the same effect (e.g. when an operation has been renamed)
 5. Skip steps where the original operation is unavailable
- This is not always possible.
- We combine the existing logic with the interactive replay
 - ▶ conflicts that cannot be handled can be resolved by the modeller
 - ▶ add / change / remove trace steps flexibly during replay

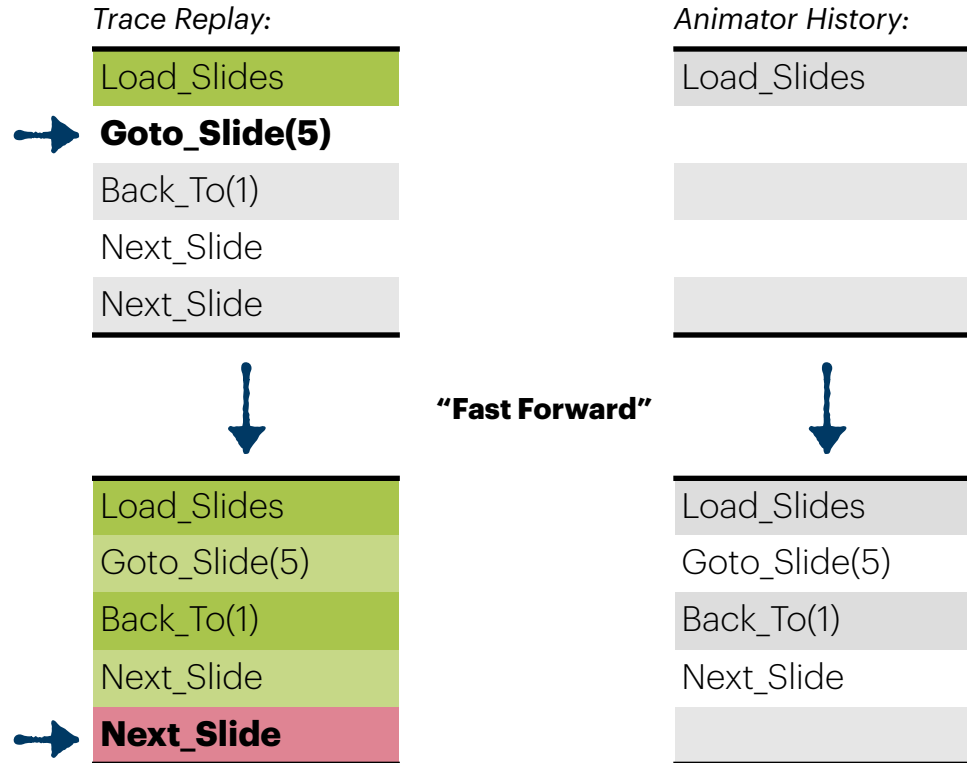
- **Two** states:
 - ▶ Current position in the replayed trace
 - ▶ State/history of the animator
- Based on this:
 - ▶ **Replay next step using a matching transition, if possible**
(selected by intelligent replay as before)
 - ▶ “Fast Forward”: automatic replay until the next step cannot be replayed
(this is basically the previous implementation)
 - ▶ Skip the current trace step, always possible
 - ▶ Add manual animation steps anywhere in the trace using the animator
 - ▶ Undo the last replayed transition (from manual animation or replayed)

Current Control Options



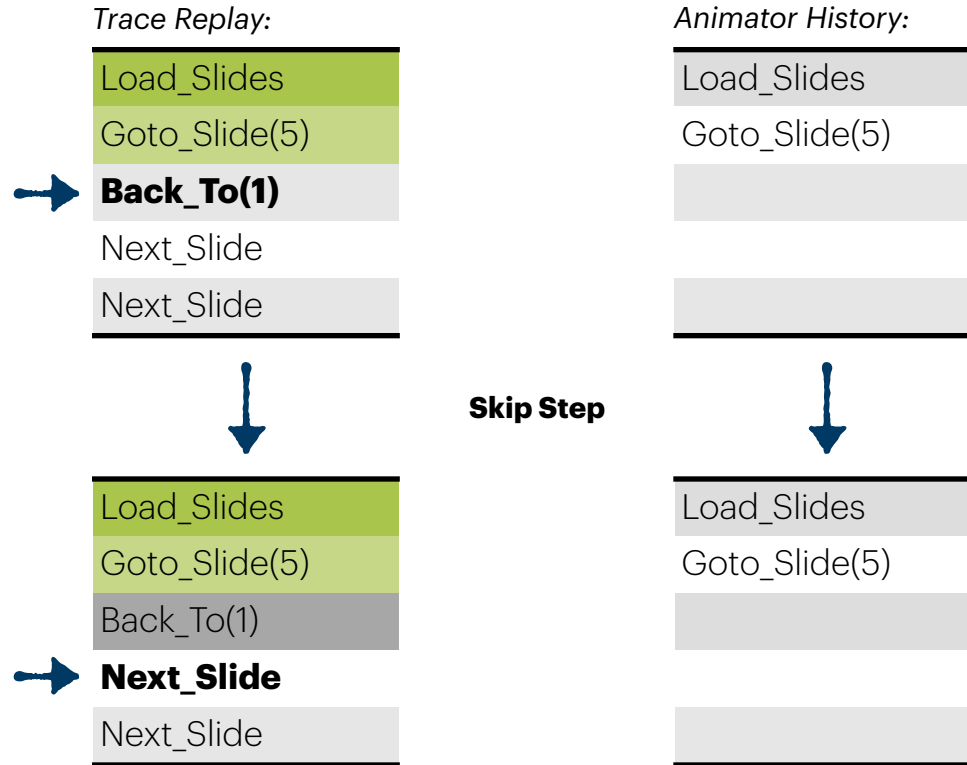
- **Two** states:
 - ▶ Current position in the replayed trace
 - ▶ State/history of the animator
- Based on this:
 - ▶ Replay next step using a matching transition, if possible
(selected by intelligent replay as before)
 - ▶ **“Fast Forward”**: automatic replay until the next step cannot be replayed
(this is basically the previous implementation)
 - ▶ Skip the current trace step, always possible
 - ▶ Add manual animation steps anywhere in the trace using the animator
 - ▶ Undo the last replayed transition (from manual animation or replayed)

Current Control Options



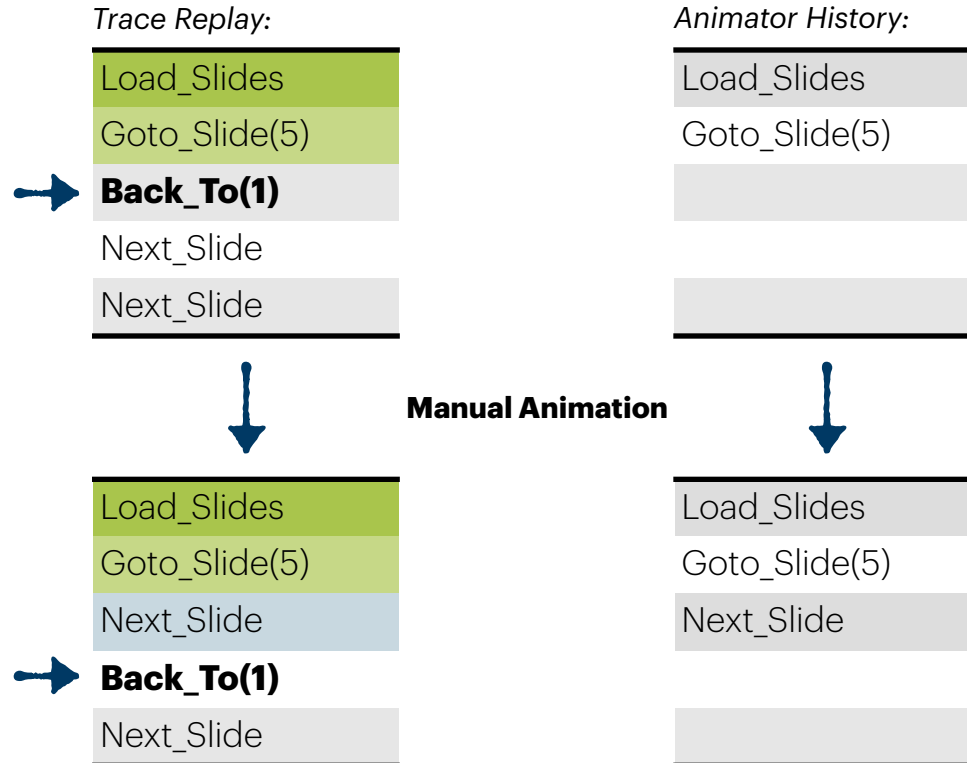
- **Two** states:
 - ▶ Current position in the replayed trace
 - ▶ State/history of the animator
- Based on this:
 - ▶ Replay next step using a matching transition, if possible
(selected by intelligent replay as before)
 - ▶ “Fast Forward”: automatic replay until the next step cannot be replayed
(this is basically the previous implementation)
 - ▶ **Skip the current trace step, always possible**
 - ▶ Add manual animation steps anywhere in the trace using the animator
 - ▶ Undo the last replayed transition (from manual animation or replayed)

Current Control Options

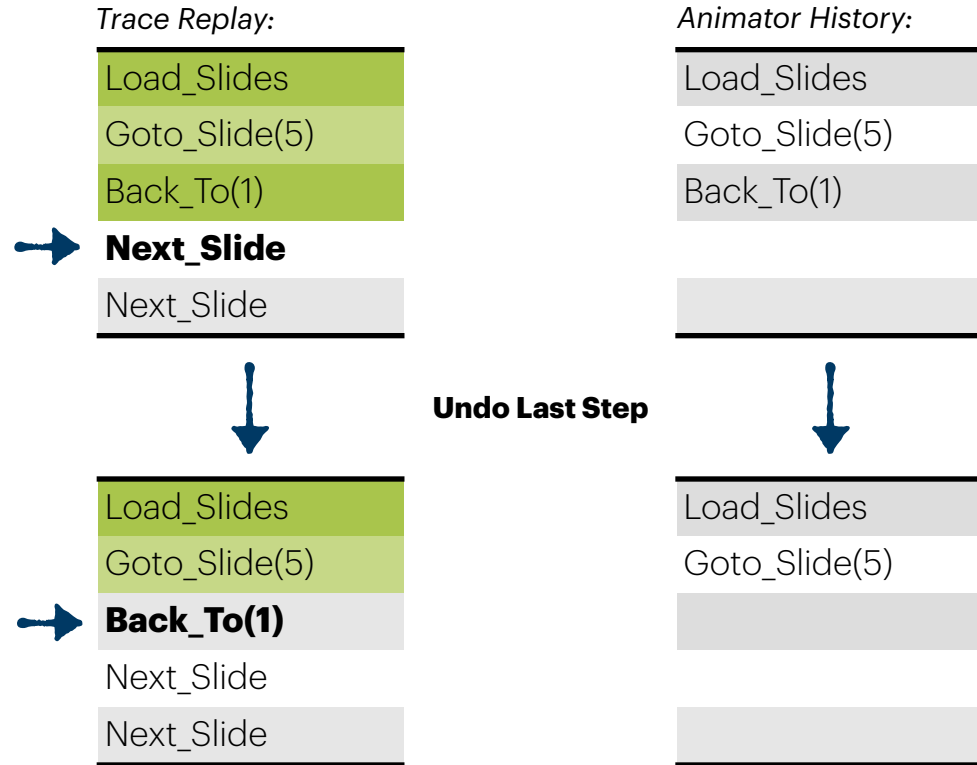


- **Two** states:
 - ▶ Current position in the replayed trace
 - ▶ State/history of the animator
- Based on this:
 - ▶ Replay next step using a matching transition, if possible
(selected by intelligent replay as before)
 - ▶ “Fast Forward”: automatic replay until the next step cannot be replayed
(this is basically the previous implementation)
 - ▶ Skip the current trace step, always possible
 - ▶ Add manual animation steps anywhere in the trace using the animator
 - ▶ Undo the last replayed transition (from manual animation or replayed)

Current Control Options

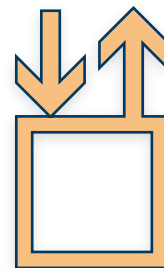


- **Two** states:
 - ▶ Current position in the replayed trace
 - ▶ State/history of the animator
- Based on this:
 - ▶ Replay next step using a matching transition, if possible
(selected by intelligent replay as before)
 - ▶ “Fast Forward”: automatic replay until the next step cannot be replayed
(this is basically the previous implementation)
 - ▶ Skip the current trace step, always possible
 - ▶ Add manual animation steps anywhere in the trace using the animator
 - ▶ Undo the last replayed transition (from manual animation or replayed)



Example: Lift

- Model of a simple lift
- Consider abstraction and first refinement
 - ▶ Abstraction has only one event: `move_to_floor`
 - ▶ Refinement adds the cabin door: `close_door`,
`open_door`
- We want to replay a trace:
 - ▶ created for Lift0 on Lift1 (refinement of a trace)
 - ▶ created for Lift1 on Lift0 (abstraction of a trace)



Lift0



Lift1

The screenshot displays the ProB2-UI interface for the Interactive Trace Replay of Event-B models. The main window is titled "Elevator1_mch.eventb - AbstractElevator_mch - ProB 2.0".

State View: The code editor shows the following Event-B code:

```

61 open_elevator_door =
62   EVENT open_elevator_door = /* of machine Elevator1 */
63   WHEN
64     /*@label Elevator1:grd1 */ elevator_door_open = closed
65   THEN
66     elevator_door_open := open
67   END;
68
69 close_elevator_door =
70   EVENT close_elevator_door = /* of machine Elevator1 */
71   WHEN
72     /*@label Elevator1:grd1 */ elevator_door_open = open
73   THEN
74     elevator_door_open := closed
75   END
76 END

```

Interactive Trace Replay: The table below shows the replayed transitions and their precision.

S...	Transition	Replayed Transition	Precision	Error
1	\$setup_constants	SETUP_CONSTANTS	Precise	
2	\$initialise_machine	INITIALISATION	Precise	
2		close_elevator_door()	Manual	
3	move_to_floor paras: f=0	move_to_floor	Precise	
3		open_elevator_door()	Manual	
3		close_elevator_door()	Manual	
4	move_to_floor paras: f=2	move_to_floor	Precise	
4		open_elevator_door()	Manual	
4		close_elevator_door()	Manual	
5	move_to_floor paras: f=4	move_to_floor	Precise	

Below the table are controls for the replay: Restart, < Step Back, > Step Forward, >> Fast Forward, and Skip. A progress bar shows 5/5 steps completed. Buttons for Save Trace and Finish are also present.

History (state 10 of 10):

Position	Transition
0	---root---
1	SETUP_CONSTANTS
2	INITIALISATION
3	close_elevator_door
4	move_to_floor(f=0)
5	open_elevator_door
6	close_elevator_door
7	move_to_floor(f=2)
8	open_elevator_door
9	close_elevator_door
10	move_to_floor(f=4)

The bottom status bar indicates "Everything is OK".

The screenshot displays the ProB2-UI interface for an interactive trace replay. The main window is titled "AbstractElevator_mch.eventb - AbstractElevator_mch - ProB 2.0".

Code Editor: Shows the Event-B model code for the elevator machine.

```

EVENT move_to_floor = /* of machine AbstractElevator */
ANY f
WHERE
/*@label AbstractElevator:grd1 */ f : (min_floor .. max_floor) \ (current_floor)
THEN
current_floor := f
END
    
```

Interactive Trace Replay: A table showing the sequence of events and transitions during the replay.

Step	Transition	Replayed Transition	Precision	Error
2	\$initialise_machine	INITIALISATION	Precise	
3	?close_elevator_door()	SKIP		Unknown operation: close_elevator_door
4	move_to_floor paras: f=5	move_to_floor	Precise	Ignoring unknown variable at step 4 for move_to_floor: elevator_door_open
5	?open_elevator_door()	SKIP		Unknown operation: open_elevator_door
6	?close_elevator_door()	SKIP		Unknown operation: close_elevator_door
7	move_to_floor paras: f=2	move_to_floor	Precise	Ignoring unknown variable at step 7 for move_to_floor: elevator_door_open
8	?open_elevator_door()	SKIP		Unknown operation: open_elevator_door
9	?close_elevator_door()	Failed		Unknown operation: close_elevator_door
10	move_to_floor paras: f=3			Ignoring unknown variable at step 10 for move_to_floor: elevator_door_open

History (state 4 of 5): A list of states and transitions.

Position	Transition
0	--root--
1	SETUP_CONSTANTS
2	INITIALISATION
3	move_to_floor(f=5)
4	move_to_floor(f=2)
5	move_to_floor(f=3)

Trace Status: The trace is not completely replayed. The progress bar shows 8/11 steps completed.

Buttons: Restart, < Step Back, > Step Forward, >> Fast Forward, Skip, Save Trace, Finish.

Bottom Status: Everything is OK

The screenshot displays the ProB2-UI interface for the 'AbstractElevator_mch' model. The main window is titled 'AbstractElevator_mch.eventb - AbstractElevator_mch - ProB 2.0*'. It features several panels:

- Operations:** A list of operations on the left, including 'move_to_floor(f=-1)', 'move_to_floor(f=-2)', 'move_to_floor(f=0)', 'move_to_floor(f=1)', 'move_to_floor(f=2)', 'move_to_floor(f=3)', and 'move_to_floor(f=5)'.
- State View / Edit:** The central area showing the Event-B code for the 'move_to_floor' event. The code includes a 'WHERE' clause for the floor range and a 'THEN' clause for updating the current floor.
- Interactive Trace Replay:** A section below the code showing a table of replayed transitions. The table has columns for Step, Transition, Replayed Transition, Precision, and Error.
- Animation:** A panel on the bottom left with 'Replay', 'Symbolic', and 'Test Case Generation' tabs. It includes a 'Status' table with columns for Name and Steps.
- Statistics:** A panel on the top right showing 'Statistics (states 8 of 10)', 'Verifications', and 'Project' information.
- History:** A panel on the bottom right showing a list of states and transitions.

The 'Interactive Trace Replay' table shows the following data:

Step	Transition	Replayed Transition	Precision	Error
1	\$setup_constants	SETUP_CONSTANTS	Precise	
2	\$initialise_machine	INITIALISATION	Precise	
3	move_to_floor paras: f=0	move_to_floor	Precise	
3		move_to_floor(5)	Manual	
3		move_to_floor(-2)	Manual	
4	move_to_floor paras: f=2	SKIP		
5	move_to_floor paras: f=4	move_to_floor	Precise	

The 'Status' table in the Animation panel shows:

✓	Status	Name	Steps
✓	●	Elevator1_m...	11
✓	✓	AbstractElev...	5

The 'History' panel shows a list of states and transitions:

Position	Transition
0	--root--
1	SETUP_CONSTANTS
2	INITIALISATION
3	move_to_floor(f=0)
4	move_to_floor(f=5)
5	move_to_floor(f=2)
6	move_to_floor(f=4)

At the bottom of the interface, there are buttons for 'Restart', '< Step Back', '> Step Forward', '>> Fast Forward', and 'Skip'. A progress bar shows '5/5' and a 'Save Trace' button is also present.

- Interactive trace replay initially motivated by refactoring of trace for complex model
 - ▶ Also useful for further applications: abstraction and refinement of traces
- Future Plans:
 - ▶ Provide more options how the next transition should be selected
 - ▶ Allow the user to select the next replayed transitions explicitly
 - ▶ Replace not available operations in the entire trace with another one, e.g. if a operation has been renamed
 - ▶ Restrict the replay precision, e.g. allowing only precise replay
- Comments, questions? What other functionality/application would be interesting?