Reasoning about Triggered Scenarios in Logic Programming

Dalal Alrajeh, Alessandra Russo, Sebastian Uchitel
Imperial College London

Rob Miller
University College London
Scenario-based Specifications

- Describe how the software-to-be should, should not or may interact with its environment.
- Informal or semi-formal language for specifying system behaviour at a functional level.
Scenario-based Specifications

- Describe how the software-to-be should, should not or may interact with its environment.
- Informal or semi-formal language for specifying system behaviour at a functional level.

**StoryBoards**

**Use cases**

**Sequence charts**
Triggered Scenarios (TS)

• Expressive variant of sequence charts
  - triggers of systems behaviours vs actual system responses
  - behaviours that *should always be* performed after a trigger vs behaviours that *should be possible to perform* within the system.
Triggered Scenarios (TS)

- Expressive variant of sequence charts
  - triggers of systems behaviours vs actual system responses
  - behaviours that *should always be* performed after a trigger vs behaviours that *should be possible to perform* within the system.
Triggered Scenarios (TS)

- Expressive variant of sequence charts
  - triggers of systems behaviours vs actual system responses
  - behaviours that *should always be* performed after a trigger vs behaviours that *should be possible to perform* within the system.
Triggered Scenarios (TS)

- Expressive variant of sequence charts
  - triggers of systems behaviours vs actual system responses
  - behaviours that should always be performed after a trigger vs behaviours that should be possible to perform within the system.

- Tools for automated analysis of TS specifications exist but they are “incomplete”:
  - models generated from TSs do not cover all possible implementations.
  - counter-examples to properties are missed.

Triggered Scenarios (TS)

- Expressive variant of sequence charts
  - triggers of systems behaviours vs actual system responses
  - behaviours that should always be performed after a trigger vs behaviours that should be possible to perform within the system.

- Tools for automated analysis of TS specifications exist but they are “incomplete”:
  - models generated from TSs do not cover all possible implementations.
  - counter-examples to properties are missed.

New analysis approach for TSs
  - Event Calculus
  - Answer Set Programming

Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.
Example of Triggered Scenario

Air Traffic Control system interacts with fight planner, radar and the aircraft to provide safe passage through airspace.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. *update-details*, *send-details*,.... are not included in the *scope* of this scenario.

**If** ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, **then** the ATC **must** send instructions to the aircraft and the radar **must** scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.

Other actions, e.g. update-details, send-details,.... are not included in the scope of this scenario.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details,.... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with fight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. *update-details*, *send-details*,..... are not included in the *scope* of this scenario.

*If* ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, *then* the ATC **must** send instructions to the aircraft and the radar **must** scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details,.... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details,..... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details,.... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details, .... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with fight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details, etc., are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Example of Triggered Scenario

Air Traffic Control system interacts with flight planner, radar and the aircraft to provide safe passage through airspace.

Other actions, e.g. update-details, send-details,.... are not included in the scope of this scenario.

If ATC receives a request to change a flight path, approves it, and then receives a signal from the radar, then the ATC must send instructions to the aircraft and the radar must scan for other flights before any other event in the scenario’s “scope” occurs.
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

But we need to model branching execution traces.

EC traditionally uses linear time.
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: \( time = run \times position \)
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: $time = run \times position$

Clipped:

$$clipped(R, P_1, F, P_2) \leftarrow \text{happens}(A, R, P), P_1 \leq P < P_2, \text{terminates}(A, F, R, P).$$

Holds at:

$$\text{holdsAt}(F, R, P) \leftarrow \text{happens}(A, R, P_1), P_1 < P, \text{initiates}(A, F, R, P_1),$$
$$\text{not clipped}(R, P_1, F, P).$$

$$\text{holdsAt}(F, R, P) \leftarrow \text{holdsAt}(F, R, P_1), P_1 < P, \text{not clipped}(R, P_1, F, P).$$
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: \( time = run \times position \)

\[
\text{clipped}(R, P_1, F, P_2) \leftarrow \text{happens}(A, R, P), P_1 \leq P < P_2, \text{terminates}(A, F, R, P).
\]

\[
\text{holdsAt}(F, R, P) \leftarrow \text{happens}(A, R, P_1), P_1 < P, \text{initiates}(A, F, R, P_1),
\]

\[
\text{not clipped}(R, P_1, F, P).
\]

\[
\text{holdsAt}(F, R, P) \leftarrow \text{holdsAt}(F, R, P_1), P_1 < P, \text{not clipped}(R, P_1, F, P).
\]

\[
\text{happens}(A_1, R, P), \text{happens}(A_2, R, P), A_1 \neq A_2.
\]

\[
\text{not occurEvent}(R, P).
\]

\[
\text{occurEvent}(R, P) \leftarrow \text{happens}(A, R, P).
\]
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: \( time = run \times position \)

\[
\begin{align*}
\text{clipped}(R, P_1, F, P_2) & \leftarrow \text{happens}(A, R, P), P_1 \leq P < P_2, \text{terminates}(A, F, R, P). \\
\text{holdsAt}(F, R, P) & \leftarrow \text{happens}(A, R, P_1), P_1 < P, \text{initiates}(A, F, R, P_1), \\
& \quad \text{not clipped}(R, P_1, F, P). \\
\text{holdsAt}(F, R, P) & \leftarrow \text{holdsAt}(F, R, P_1), P_1 < P, \text{not clipped}(R, P_1, F, P).
\end{align*}
\]

\[
\begin{align*}
\leftarrow \text{happens}(A_1, R, P), \text{happens}(A_2, R, P), A_1 \neq A_2. \\
\leftarrow \text{not occurEvent}(R, P). \\
\text{occurEvent}(R, P) & \leftarrow \text{happens}(A, R, P).
\end{align*}
\]

Exactly one action must occur at each position of each run.
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: $time = run \times position$

\[
\text{clipped}(R, P_1, F, P_2) \leftarrow \text{happens}(A, R, P), P_1 \leq P < P_2, \text{terminates}(A, F, R, P).
\]

\[
\text{holdsAt}(F, R, P) \leftarrow \text{happens}(A, R, P_1), P_1 < P, \text{initiates}(A, F, R, P_1), \neg \text{clipped}(R, P_1, F, P).
\]

\[
\text{holdsAt}(F, R, P) \leftarrow \text{holdsAt}(F, R, P_1), P_1 < P, \neg \text{clipped}(R, P_1, F, P).
\]

\[
\neg \text{happens}(A_1, R, P), \text{happens}(A_2, R, P), A_1 \neq A_2.
\]

\[
\neg \text{not occurEvent}(R, P).
\]

\[
\text{occurEvent}(R, P) \leftarrow \text{happens}(A, R, P).
\]

\[
\text{sameHistory}(R_1, R_2, P) \leftarrow \neg \text{differentHistory}(R_1, R_2, P).
\]

\[
\text{differentHistory}(R_1, R_2, P) \leftarrow P_1 < P, \text{happens}(A, R_1, P_1) \neg \text{happens}(A, R_2, P_1).
\]

Exactly one action must occur at each position of each run.
Extended Event Calculus

EC is a well known formalism for reasoning about events and causal effects. Already used for representing and analysing event-driven specifications.

EC traditionally uses linear time.

But we need to model branching execution traces.

Extend EC with parallel time lines: \( time = run \times position \)

- \( \text{clipped}(R, P_1, F, P_2) \leftarrow \text{happens}(A, R, P), P_1 \leq P < P_2, \text{terminates}(A, F, R, P). \)
- \( \text{holdsAt}(F, R, P) \leftarrow \text{happens}(A, R, P_1), P_1 < P, \text{initiates}(A, F, R, P_1), \text{not clipped}(R, P_1, F, P). \)
- \( \text{holdsAt}(F, R, P) \leftarrow \text{holdsAt}(F, R, P_1), P_1 < P, \text{not clipped}(R, P_1, F, P). \)

- \( \text{exactly one action must occur at each position of each run} \)
- \( \text{two runs contain same sequence of actions up to a given position} \)

- \( \text{occurEvent}(R, P) \leftarrow \text{happens}(A, R, P). \)

- \( \text{sameHistory}(R_1, R_2, P) \leftarrow \text{not differentHistory}(R_1, R_2, P). \)
- \( \text{differentHistory}(R_1, R_2, P) \leftarrow P_1 < P, \text{happens}(A, R_1, P_1) \text{ not happens}(A, R_2, P_1). \)
EC Representation of TSs

Domain Independent Axioms
EC Representation of TSs

Domain Independent Axioms

\[
\text{mainChart}((SM, P_{\text{start}}, R) \leftarrow P_{\text{start}} < P_{\text{end}}, \text{lpoc}(SM, P_{\text{start}}, P_{\text{end}}, R)).
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{linearisationOf}(L, SM),
\]

\[
\text{not existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R).
\]

\[
\text{existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R) \leftarrow \text{sameHistory}(R, R_{1}, P), \text{linearisation}(SM, L, P, R_{1})
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{not mainChart}(SM, P_{\text{end}}, R).
\]
EC Representation of TSs

Domain Independent Axioms

mainChart(SM, Pstart, R) ← Pstart < Pend, lpoc(SM, Pstart, Pend, R).

→ trigger(ST), isIn(ST, S), isIn(SM, S), lpoc(ST, Pstart, Pend, R), linearisationOf(L, SM),
   not existsLinearisationSameHistory(SM, L, Pend,R).

existsLinearisationSameHistory(SM, L, Pend, R) ← sameHistory(R, R1, P), linearisation(SM, L, P, R1)

→ trigger(ST), isIn(ST, S), isIn(SM, S), lpoc(ST, Pstart, Pend, R), not mainChart(SM, Pend, R).

Domain Dependent Axioms
EC Representation of TSs

Domain Independent Axioms

\[
\text{mainChart}(SM, P_{\text{start}}, R) \leftarrow P_{\text{start}} < P_{\text{end}}, \text{lpoc}(SM, P_{\text{start}}, P_{\text{end}}, R).
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{linearisationOf}(L, SM), \\
\text{not existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R).
\]

\[
\text{existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R) \leftarrow \text{sameHistory}(R, R_1, P), \text{linearisation}(SM, L, P, R_1)
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{not mainChart}(SM, P_{\text{end}}, R).
\]

Domain Dependent Axioms

Diagram:

- **Aircraft**
  - Request-change
  - Approve
  - Send-signal
- **ATC**
  - Send-instructions
- **Radar**
  - Scan
EC Representation of TSs

Domain Independent Axioms

mainChart(SM, Pstart, R) ← Pstart < Pend, Ipoc(SM, Pstart, Pend, R).

← trigger(ST), isIn(ST, S), isIn(SM, S), Ipoc(ST, Pstart, Pend, R), linearisationOf(L, SM),
not existsLinearisationSameHistory(SM, L, Pend,R).

existsLinearisationSameHistory(SM, L, Pend, R) ← sameHistory(R, R1, P), linearisation(SM, L, P, R1)

← trigger(ST), isIn(ST, S), isIn(SM, S), Ipoc(ST, Pstart, Pend, R), not mainChart(SM, Pend, R).

Domain Dependent Axioms

Ipoc(t1, Pstart, Pend, R) ← Pstart ≤ P1, P1 < P2, P2 < P3, P3 < Pend,
happens(request-change, R, P1),
happens(approve, R, P2),
happens(send-signal, R, P3),
not scopeActionHappens(t1,Pstart,Pend,R,P1,P2,P3).
EC Representation of TSs

Domain Independent Axioms

\[
\text{mainChart}(SM, P_{\text{start}}, R) \leftarrow P_{\text{start}} < P_{\text{end}}, \text{lpoc}(SM, P_{\text{start}}, P_{\text{end}}, R).
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{linearisationOf}(L, SM), \text{not existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R).
\]

\[
\text{existsLinearisationSameHistory}(SM, L, P_{\text{end}}, R) \leftarrow \text{sameHistory}(R, R1, P), \text{linearisation}(SM, L, P, R1).
\]

\[
\leftarrow \text{trigger}(ST), \text{isIn}(ST, S), \text{isIn}(SM, S), \text{lpoc}(ST, P_{\text{start}}, P_{\text{end}}, R), \text{not mainChart}(SM, P_{\text{end}}, R).
\]

Domain Dependent Axioms

\[
\text{lpoc}(t1, P_{\text{start}}, P_{\text{end}}, R) \leftarrow P_{\text{start}} \leq P1, P1 < P2, P2 < P3, P3 < P_{\text{end}}, \text{happens(request-change, R, P1)}, \text{happens(approve, R, P2)}, \text{happens(send-signal, R, P3)}, \text{not scopeActionHappens}(t1, P_{\text{start}}, P_{\text{end}}, R, P1, P2, P3).
\]

\[
\text{lpoc}(m1, P_{\text{start}}, P_{\text{end}}, R) \leftarrow P_{\text{start}} \leq P1, P1 < P_{\text{end}}, P_{\text{start}} \leq P2, P2 < P_{\text{end}}, \text{happens(send-instructions, R, P1)}, \text{happens(scan, R, P2)}, \text{not scopeActionHappens}(m1, P_{\text{start}}, P_{\text{end}}, R, P1, P2).
\]
Abstracting Computation Trees into Finite ASP Models

Computation trees are infinite, but ASP models are finite.

How can we overcome this apparent mismatch?
Abstracting Computation Trees into Finite ASP Models

Computation trees are infinite, but ASP models are finite.

How can we overcome this apparent mismatch?

A system state is a current value for each fluent together with a current position along each potential pathway through each TS.

There are only a finite number of system states.
Abstracting Computation Trees into Finite ASP Models

Computation trees are infinite, but ASP models are finite.

How can we overcome this apparent mismatch?

A system state is a current value for each fluent together with a current position along each potential pathway through each TS.

There are only a finite number of system states.

In our ASP models, the end position of each finite run must match with a system state already reached further back in the same or another run.
Abstracting Computation Trees into Finite ASP Models

Computation trees are infinite, but ASP models are finite. How can we overcome this apparent mismatch?

A system state is a current value for each fluent together with a current position along each potential pathway through each TS. There are only a finite number of system states.

In our ASP models, the end position of each finite run must match with a system state already reached further back in the same or another run.

From this other matching run position, progress must be made through each mainchart that has already started.
Abstracting Computation Trees into Finite ASP Models

Computation trees are infinite, but ASP models are finite.

How can we overcome this apparent mismatch?

A system state is a current value for each fluent together with a current position along each potential pathway through each TS.

There are only a finite number of system states.

In our ASP models, the end position of each finite run must match with a system state already reached further back in the same or another run.

From this other matching run position, progress must be made through each mainchart that has already started.

These properties are captured through domain independent integrity constraints together with domain dependent assertions of execution states per position in each potential pathway.
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.

Property: \( \text{The trigger } t1 \text{ is never exhibited} \)
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.

Property: \( \text{The trigger } t_1 \text{ is never exhibited} \)

ASP: \( \leftarrow 0 \ [ \text{lpoc}(t_1, P_1, P_2, R) : \text{position}(P_1) : \text{position}(P_2) : \text{run}(R)] \ 0 \)
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.

Property: The trigger \( t_1 \) is never exhibited

ASP: \( \leftarrow 0 \ [ \text{l poc}(t_1, P_1, P_2, R) : \text{position}(P_1) : \text{position}(P_2) : \text{run}(R)] 0 \)

Single-state Fluent Properties
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.

Property: The trigger t1 is never exhibited

ASP: \[ \leftarrow 0 \ [ \text{lpc}(t1, P1, P2, R) \ : \ \text{position}(P1) \ : \ \text{position}(P2) \ : \ \text{run}(R) ] 0 \]

Single-state Fluent Properties

- Assertions required to hold in every state of a system ("safety properties").
Bounded Verification

Vacuity Detection

- TS may be vacuously satisfied if the trigger is never executed.
- Useful to detect TSs that may have incompatible maincharts.

Property: *The trigger t1 is never exhibited*

ASP: $\leftarrow 0 \ [ \text{lpoc}(t1, P1, P2, R) : \text{position}(P1) : \text{position}(P2) : \text{run}(R)] \ 0$

Single-state Fluent Properties

- Assertions required to hold in every state of a system ("safety properties").
- ASP solver detects violations by searching for solutions of the TS specification and the negation of the property.
Conclusion & Future Work

- Paper lays the foundation for ASP-based analysis of TSs and conditional scenarios.

- Achieved by introducing multiple time-lines and same action histories to the Event Calculus.

- Abstraction using a notion of “system state” allows infinite computation trees to be represented as finite ASP models.

- The ASP representation allows detection of vacuity and violation of a class of safety properties.

- Next step is to investigate more general forms of safety properties and “liveness” properties.

- Future plans include using inductive learning methods for revising incomplete or inconsistent TSs.
Thank You for Your Attention
Incompleteness of Behaviour Model Synthesis from TS

TS specifications are synthesised into system behaviour models expressed as Modal Transition Systems (MTS) and analysed:
Incompleteness of Behaviour Model Synthesis from TS

TS specifications are synthesised into system behaviour models expressed as Modal Transition Systems (MTS) and analysed:

![Diagram 1](image1)

![Diagram 2](image2)
Incompleteness of Behaviour Model Synthesis from TS

TS specifications are synthesised into system behaviour models expressed as Modal Transition Systems (MTS) and analysed:
Incompleteness of Behaviour Model Synthesis from TS

TS specifications are synthesised into system behaviour models expressed as Modal Transition Systems (MTS) and analysed:

But not
Incompleteness of Behaviour Model Synthesis from TS

TS specifications are synthesised into system behaviour models expressed as Modal Transition Systems (MTS) and analysed:

MTS are not closed under the "merge" operator, as they cannot capture disjunctive behaviours.