Verifying Business Processes using SPIN

Wil Janssen

Telematics Institute

(Enschede, The Netherlands)

Radu Mateescu

INRIA Rhône-Alpes / VASY

(Montbonnot, France)

Sjouke Mauw

Eindhoven University of Technology

(Eindhoven, The Netherlands)

Jan Springintveld

CWI / SEN2

(Amsterdam, The Netherlands)

Introduction

Framework :

Testbed Project (Telematics Institute)

Objectives :

automated functional analysis of *business processes*

Approach :

verification by model-checking using Promela and SPIN

Plan

- **Overview of the AMBER language**
- **Verification using SPIN**
- **Application**
- **Conclusion**

The AMBER language

AMBER (*Architectural Modelling Box for Enterprise Redesign*):

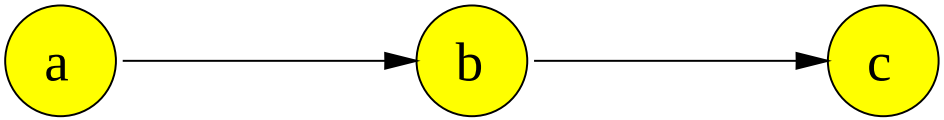
a specification language for business processes

Overview:

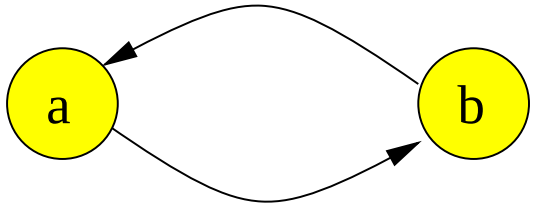
- graphical syntax
- causality-based semantics
- action enabling, phasing
- hierarchical composition, synchronization
- data entities

Actions and causality

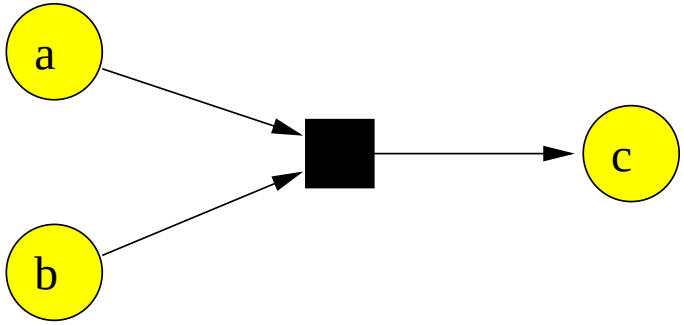
Action enabling:



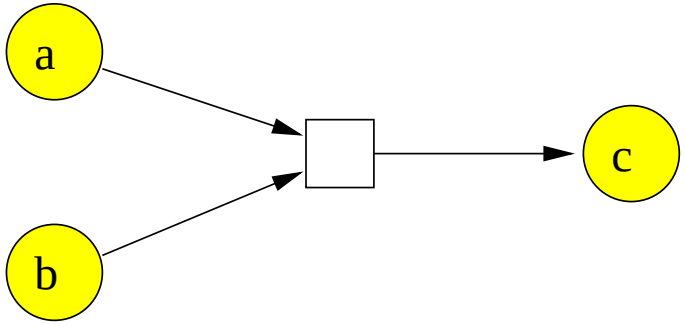
Deadlock:



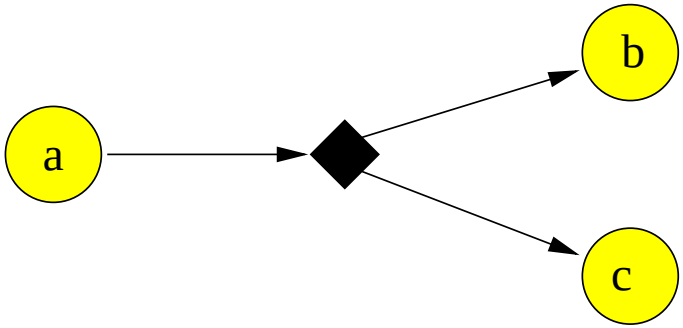
Splits and joins



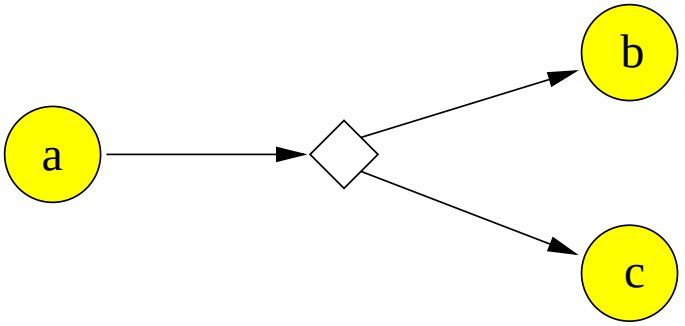
(a) AndJoin



(b) OrJoin



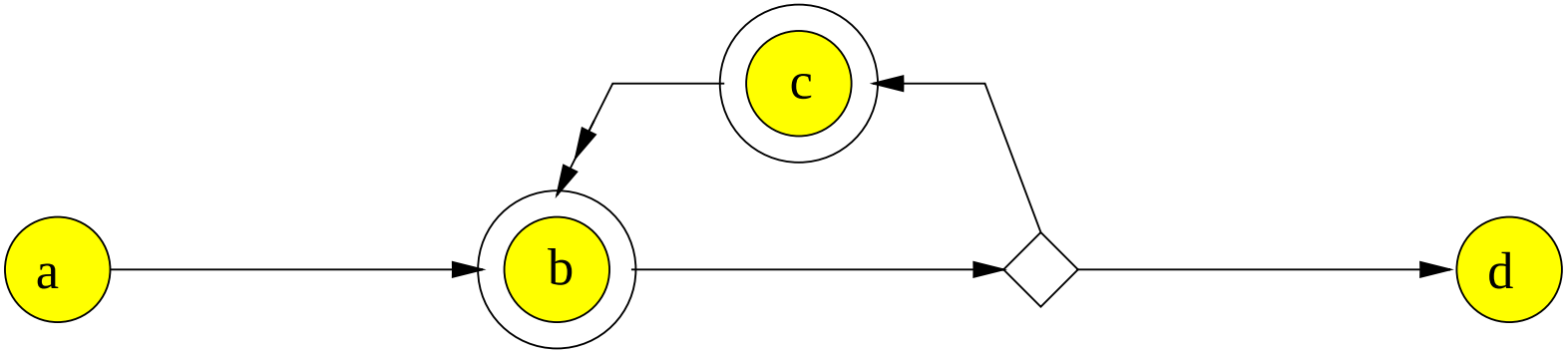
(c) AndSplit



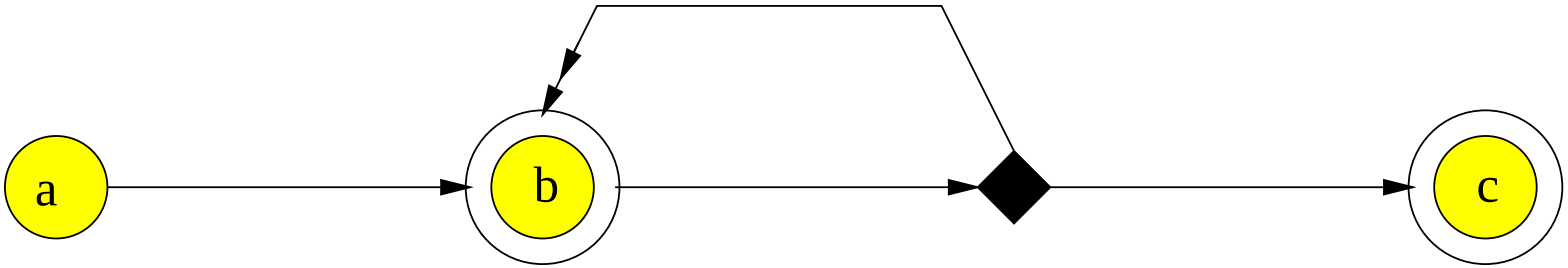
(d) OrSplit

Loops

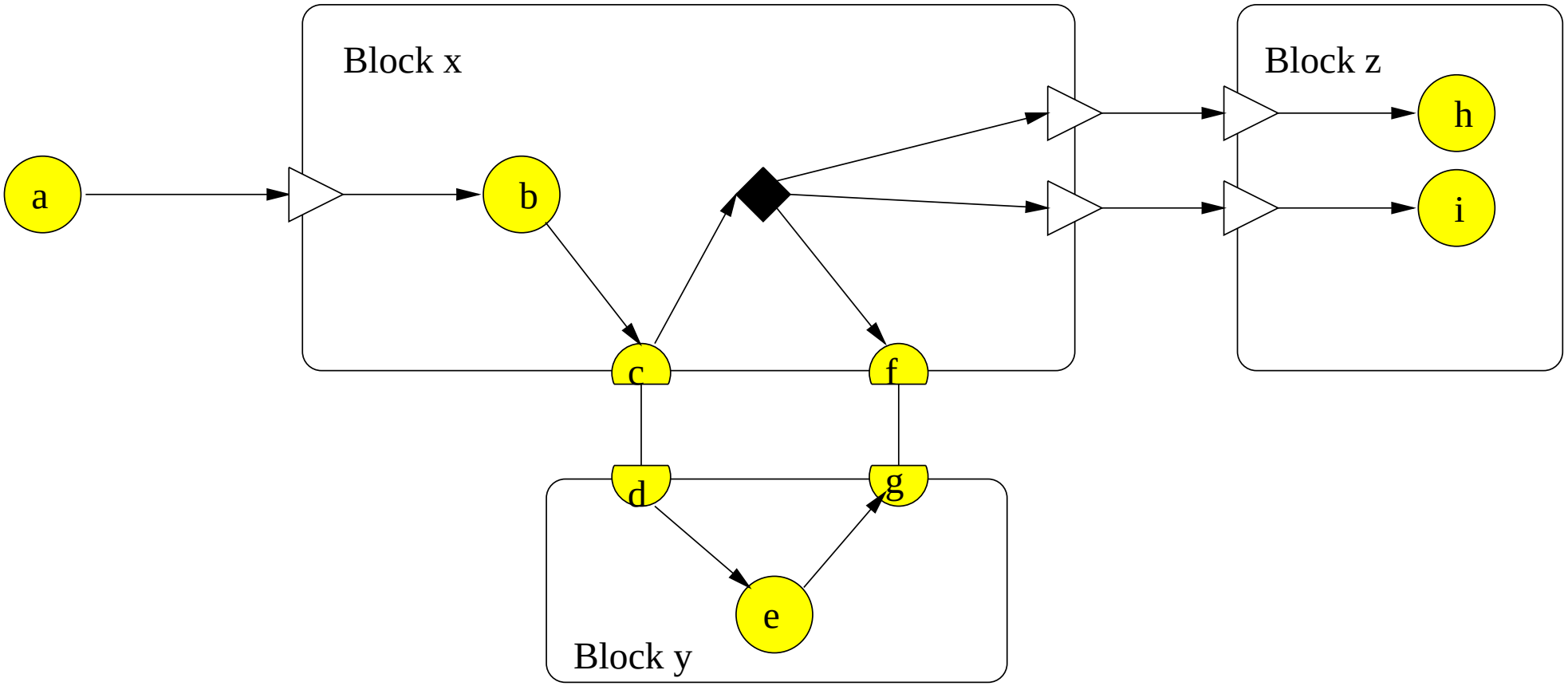
Finite-state loop:



Infinite-state loop:

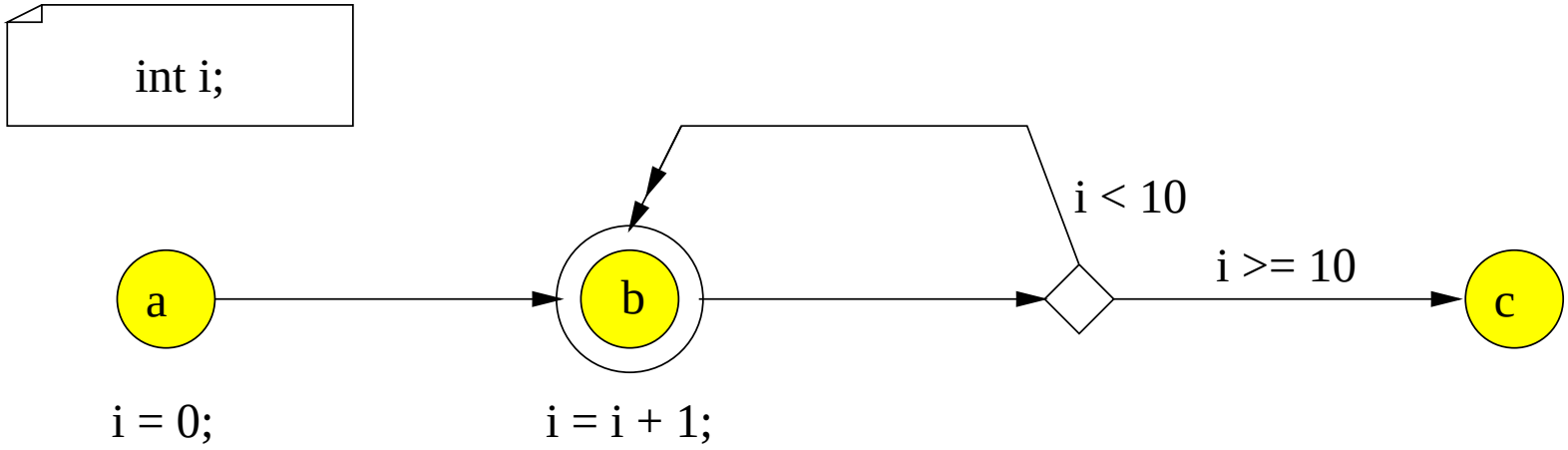


Blocks

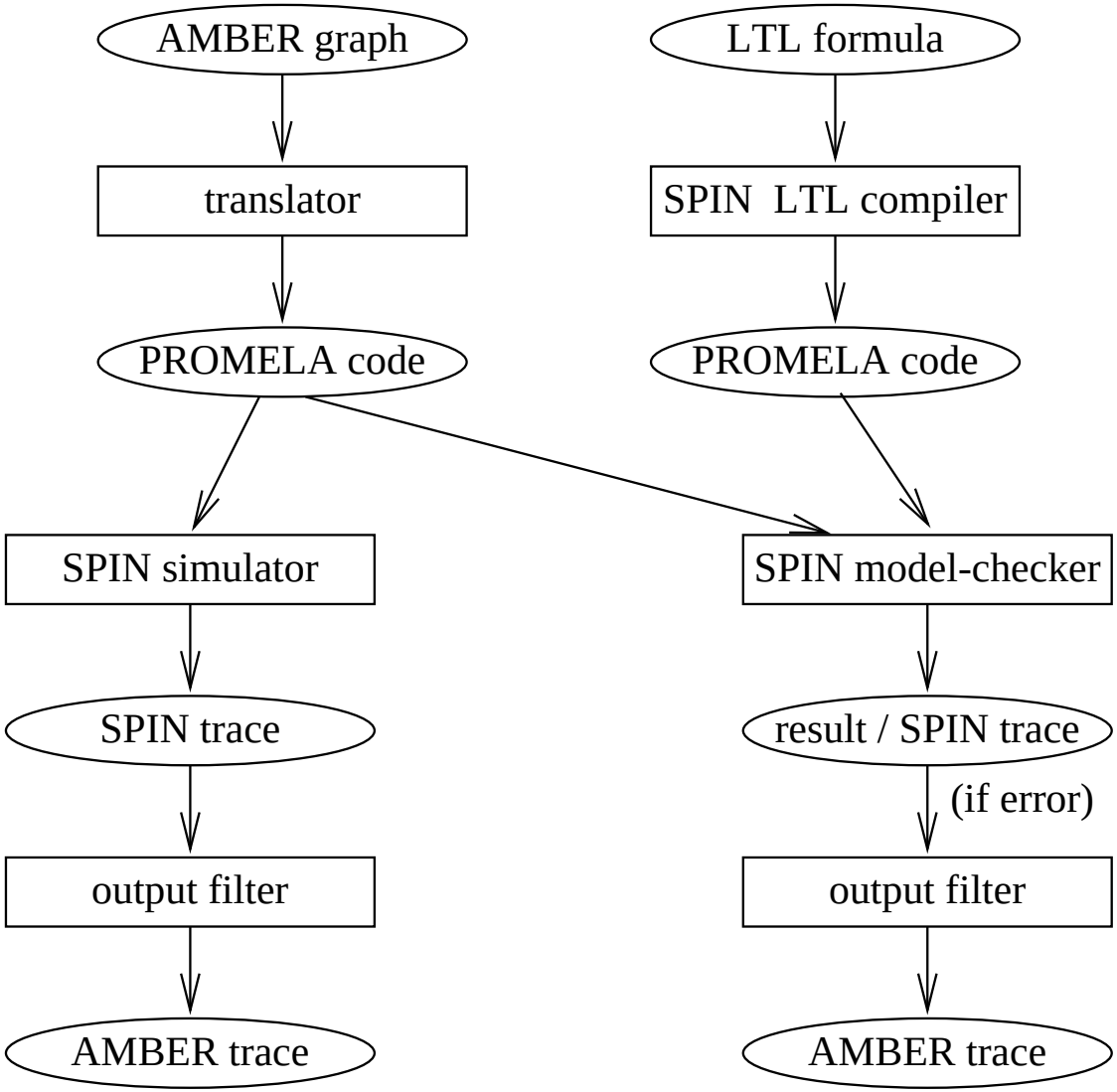


Data

Counter from 1 to 10:



Analysis methodology (Testbed Studio)



Translation from AMBER to Promela

Operational interleaving semantics for AMBER

state automaton

- **states** = node enabledness × data variables
- **transitions** = event execution

Promela model

preamble

- definitions of data and control variables
- definitions of LTL atomic propositions

process

- non-terminating `do-od` loop
- case distinction condition `->` event

Specification of temporal properties

LTL (*Linear Temporal Logic*)

- **behavioural properties:** execution of actions in the model

  ```
  <> executed_c
  ```

  where `executed_c = true` generated when action c is executed

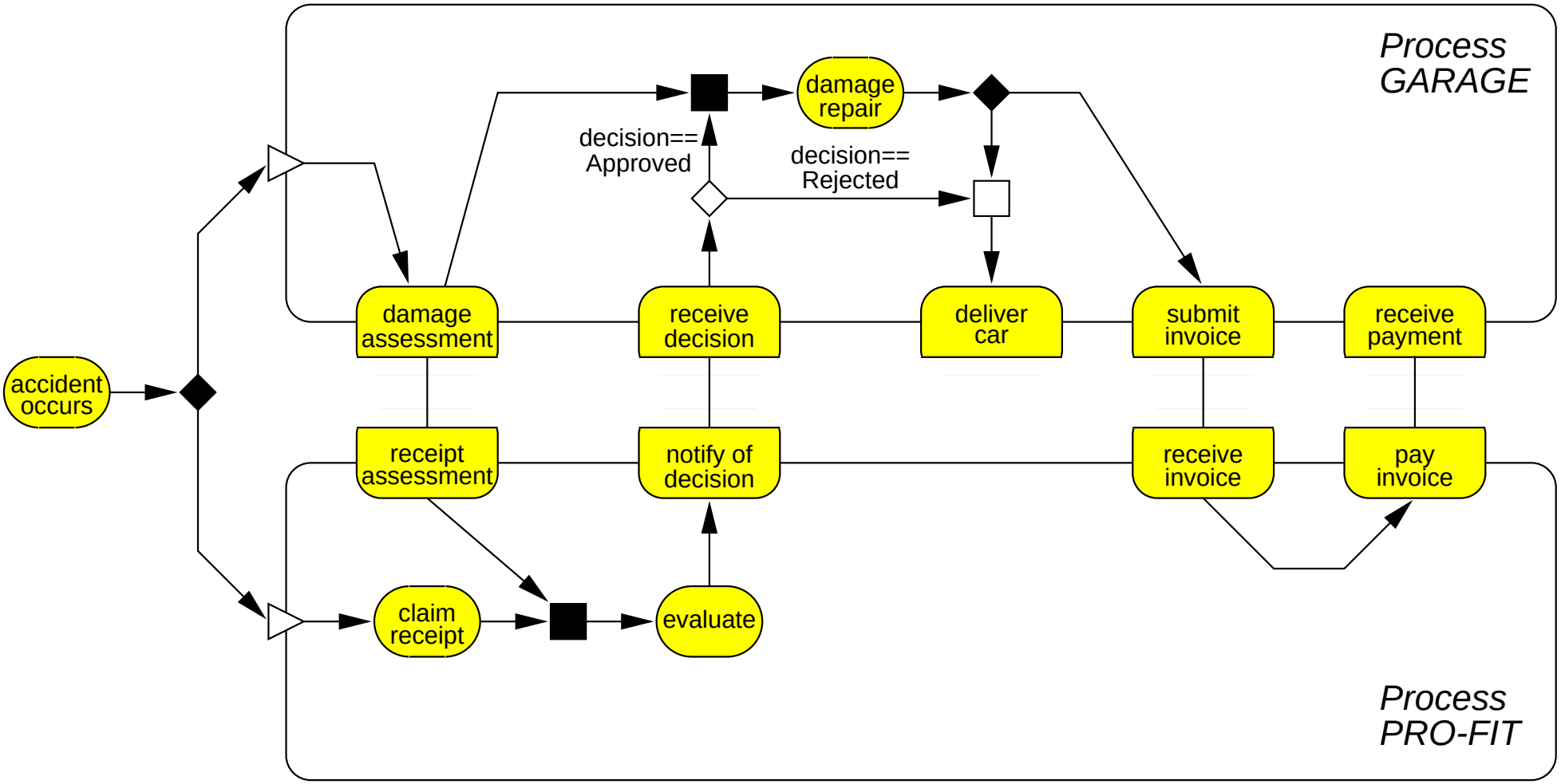
- **data-based properties:** evolution of data variables

  ```
  [] p
  ```

  where `#define p (i <= 10)` defined in the Promela model

Application

Example of AMBER specification:



Correctness properties

P1. *Is the car repaired only when the claim is approved?*

```
[] (!damage_repair W claim_approved)
```

P2. *Will every claim below 6000 be approved for customer 4?*

```
[] ((claim_below_6000 && customer_4) -> <> claim_approved)
```

P3. *Is the car always repaired when delivered?*

```
[] (!deliver_car W damage_repair)
```

P4. *Can the car be repaired when the claim is rejected?*

```
[] (claim_rejected -> [] !damage_repair)
```

P5. *Can the garage submit an invoice even if the claim is rejected?*

```
[] (claim_rejected -> [] !submit_invoice)
```

Conclusion

Results :

- translation from AMBER to Promela
- definition of an operational semantics for AMBER
- development of the Testbed Studio toolset
- application on medium-sized examples

Future work :

- graphical patterns for property specification
- syntactical checking of finite-state conditions
- handling of strong fairness properties
- application of Testbed Studio on real-size examples