Extending statecharts with process algebra operators

Marc Frappier,

B. Fraikin, R. St-Denis : Sherbrooke

F. Gervais, R. Laleau : Paris-Est

UNIVERSITÉ DE SHERBROOKE

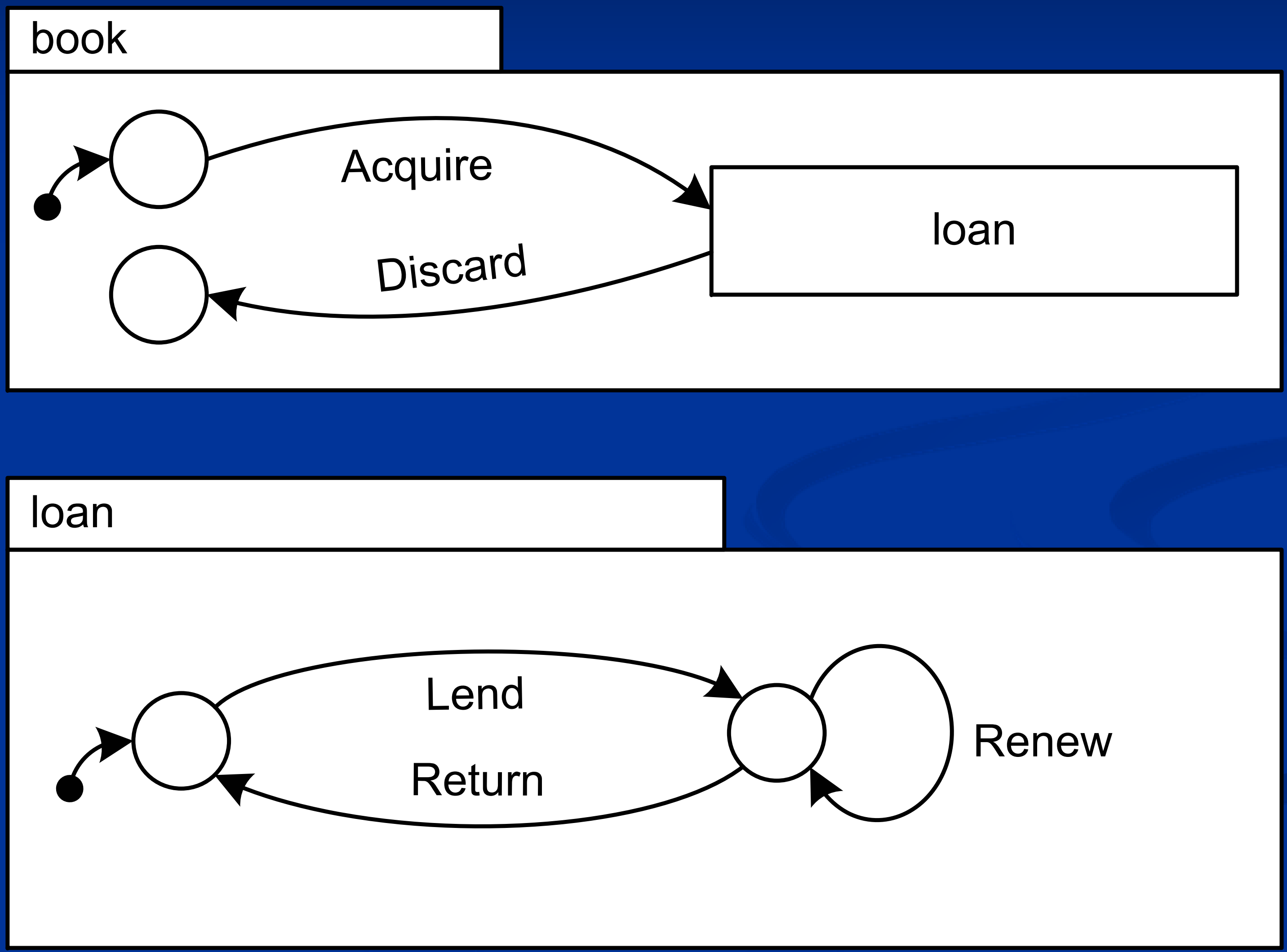
Plan

- Statecharts and Information systems specifications
- ASTD : Algebraic State Transition Diagrams
- Semantics of ASTD
- Conclusion

Statecharts

- graphical notation
- hierarchy + orthogonality
  - hierarchical states
  - AND states (parallel)
  - OR states (choice)
- nice for single instance behaviour
- provision for multiple instances in seminal paper (SCP 87)
- "*never*" implemented or formalised

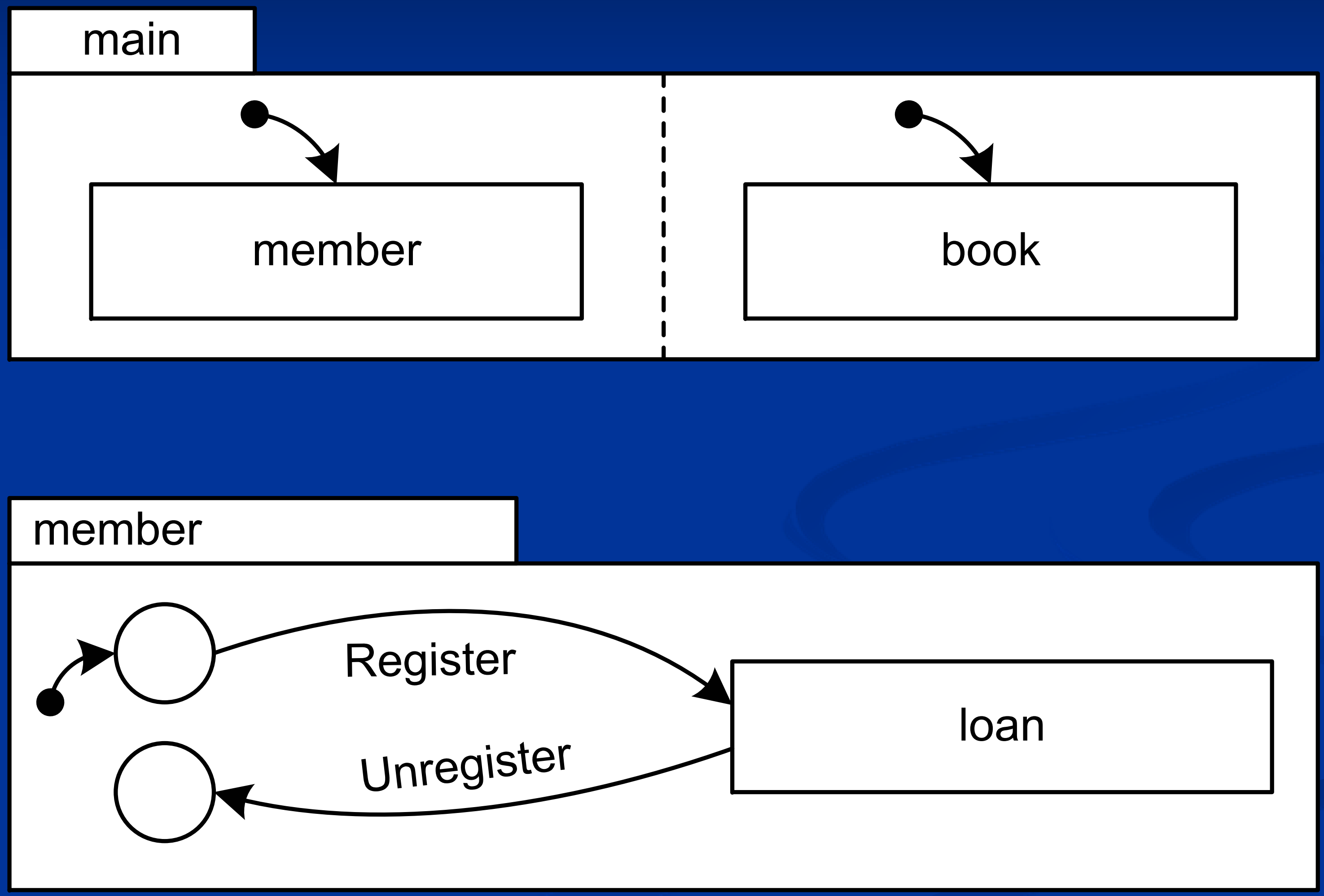
A library in statecharts



Problems

- need a guard to prevent discarding unreturned books
- behaviour of a single book
  - how to deal with several books?
  - put n copies of book in parallel

Adding members



Problems

- members can borrow several books at the same time
  - how to say that in statecharts?
- two copies of loan
  - one in member
  - one in book
  - how do they interact?
    - can broadcast communication help?
    - how can I say that member m1 borrows book b1?

Process algebra

- CCS, CSP, ACP, LOTOS, EB3, ...
- algebra
  - operators to combine process expressions
    - sequence, choice, interleave, synchronisation, guard, ...
    - quantification
  - operators is the essence of abstraction
    - combine small units to build large units

A Process expression for books

book(b : BookId) =

Acquire(b, _)

•

loan(_, b)*

•

Discard(b)

A process expression for loans

loan(bId:BookID, mId:MemberID) =

nbLoans(mId) < maxNbLoans(mId) $\Longrightarrow$
Lend(mId, bId)

•

(nbLoans(mId) < maxNbLoans(mId) $\Longrightarrow$
Renew(bId))*

•

Return(bId)

A process expression for members

member(m : MemberId) =

Register(m, _,_)

•

(||| b : BookId : loan(m, b)*)

•

Unregister(m)

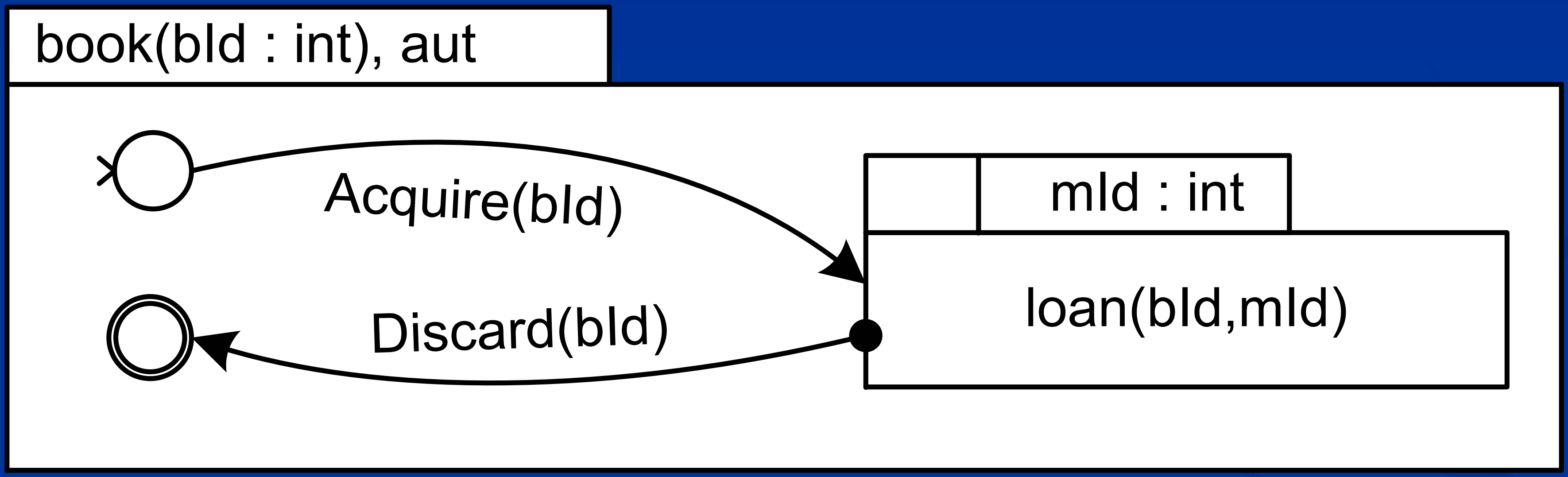
Main process expression

$$
\begin{aligned}
&\text{main} = \\
&\qquad (\ ||| \ b : \text{BookId} : \text{book}(b)^{\star}\ ) \\
&|| \\
&\qquad (\ ||| \ m : \text{MemberId} : \text{member}(m)^{\star}\ )
\end{aligned}
$$

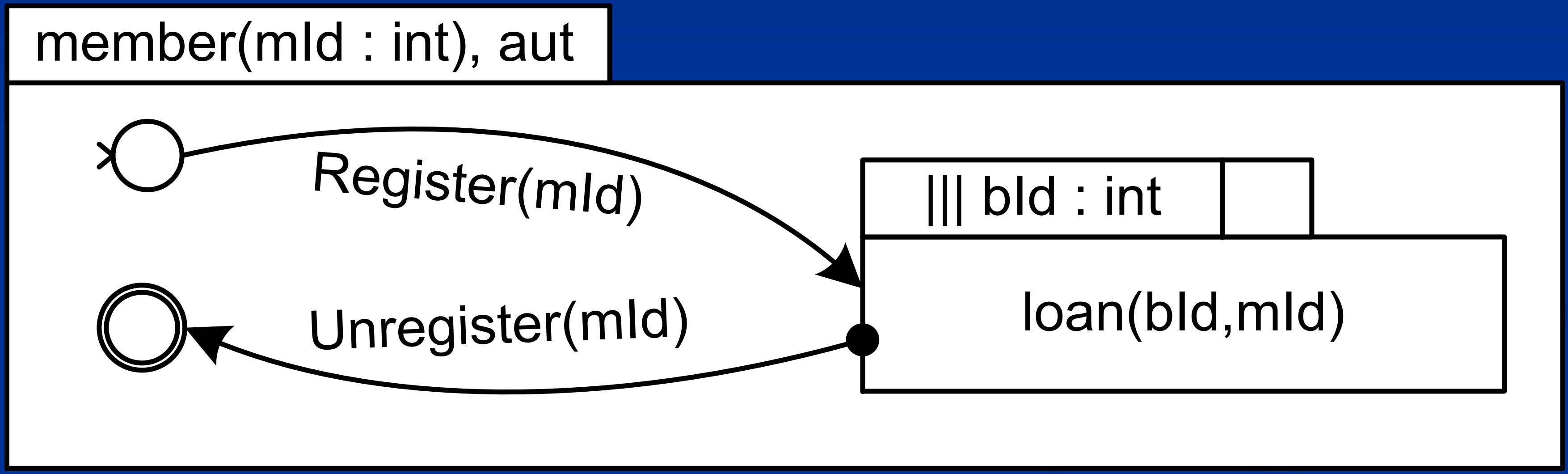
ASTD

- Algebraic State Transition Diagrams
- ASTD = statecharts + process algebra
  - graphical notation
  - power of abstraction
- statecharts become elementary process expressions
  - combine them using operators
- formal semantics
  - operational semantics

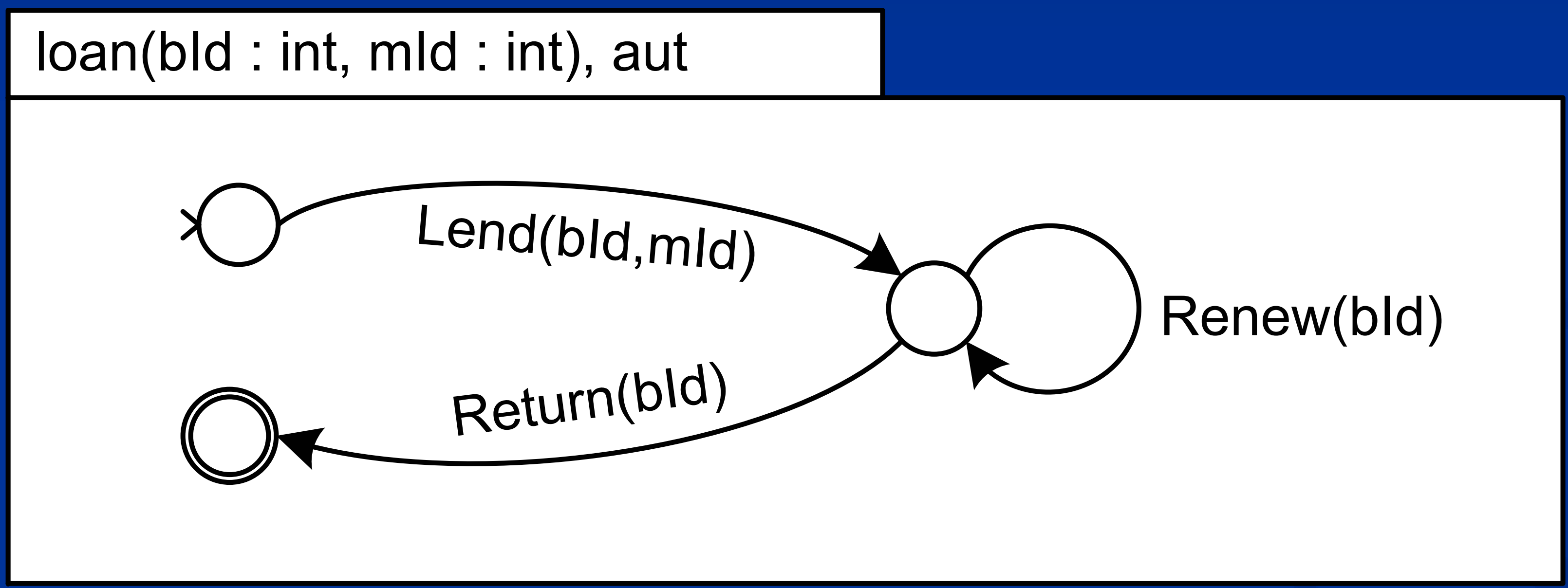
A book ASTD



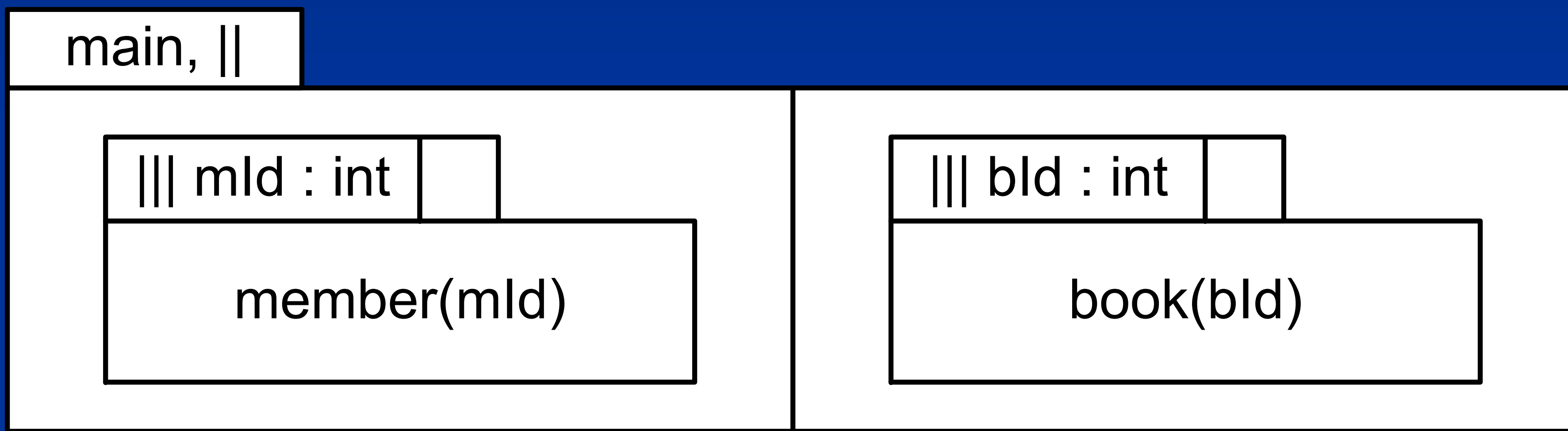
A member ASTD



A loan ASTD



The main ASTD



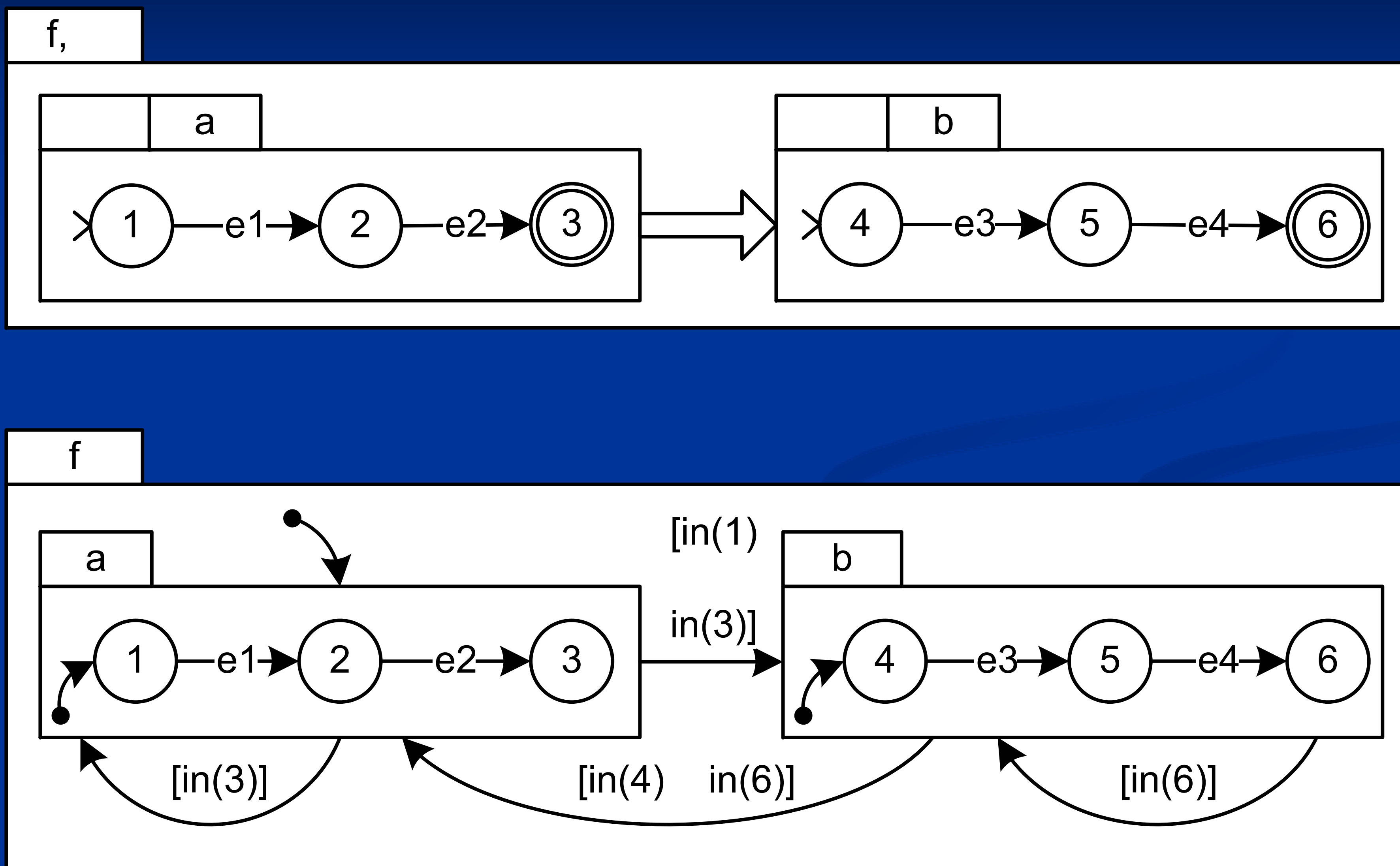
ASTD Operators

- ⇨ : sequence
- | : choice
  - |x : quantified choice
- |[ ]|: parallel composition with synchronisation
  - ||| interleave, || parallel composition
  - |||x, |[ ]|x : quantified version
- ⟹ : guard
- ★ : Kleene closure
- ASTD call : allows recursive calls

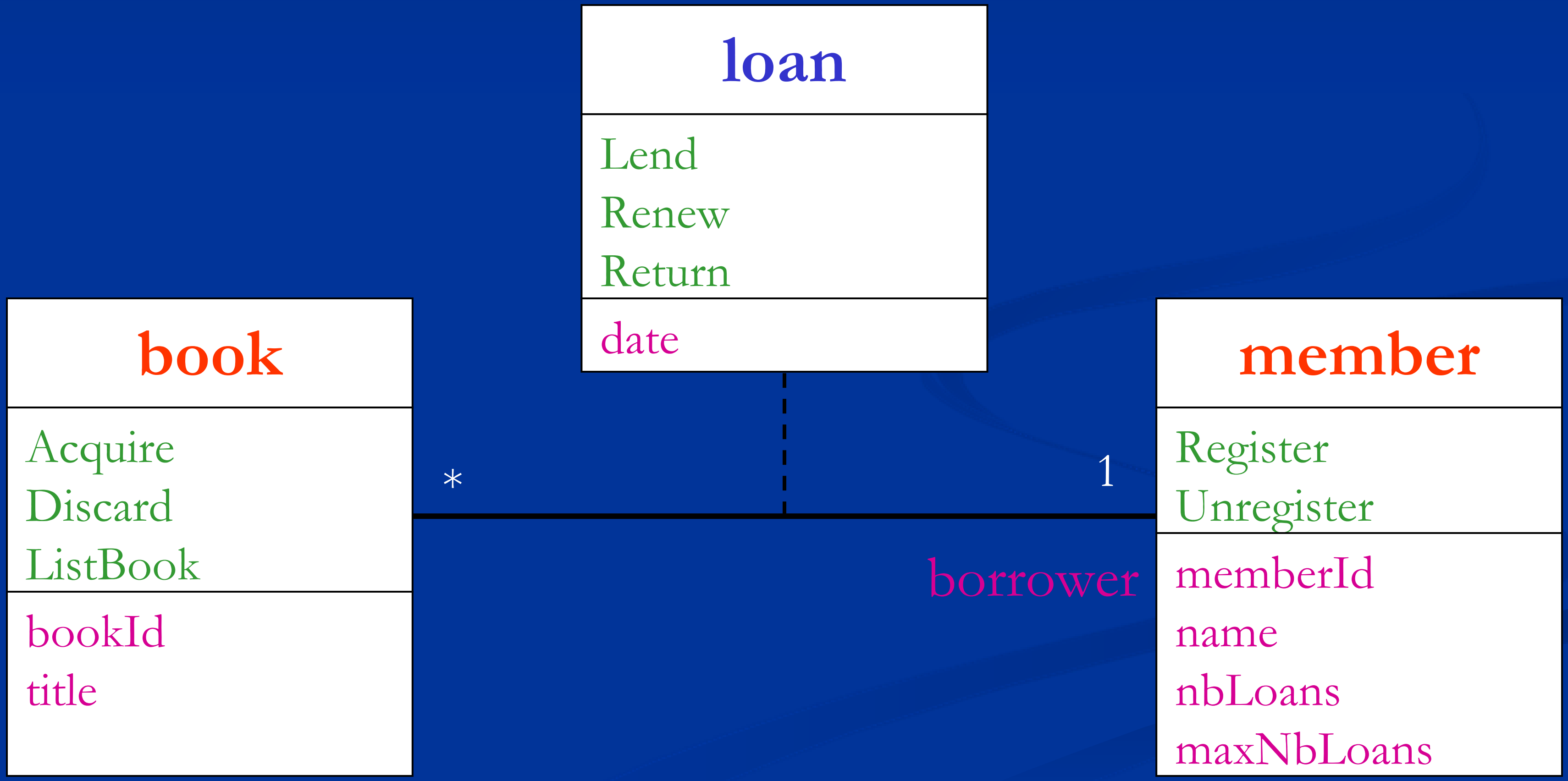
Power of abstraction

- suppose you have two statecharts, a and b
- you want to compose them as follows
  - execute a an arbitrary number of times
  - then execute b an arbitrary number of times
  - then start over again, an arbitrary number of times
- can’t do it in statecharts without peeking into a and b’s structure with guards
  - introduce a dependency between the compound and the components

Power of abstraction



Integration with the business class diagram



State variables

- system trace is the only state variable
- attributes are functions on the system trace
- can be used anywhere in ASTDs
  - guard, quantification sets, ...

nbLoans(mId : MemberId) =
    Register(mId, _)   : 0,
    Lend(mId, _)   : 1 + nbLoans(mId),
    Return(bId)   : **if** borrower(bId) = mId
                                  then nbLoans(mId) - 1,
    Unregister(mId, _)   : ⊥;

Operational semantics

- first used by Milner for CCS
- transitions

$$s \xrightarrow{\sigma}_{\mathsf{a}} s'$$

- ASTD a can execute σ from state s and move to state s’

Operational semantics

- transitions defined by a set of inference rules
- rules for each operator
- allows non-determinism
  - if several transitions can fire from s, then one is nondeterministically chosen
  - no priority

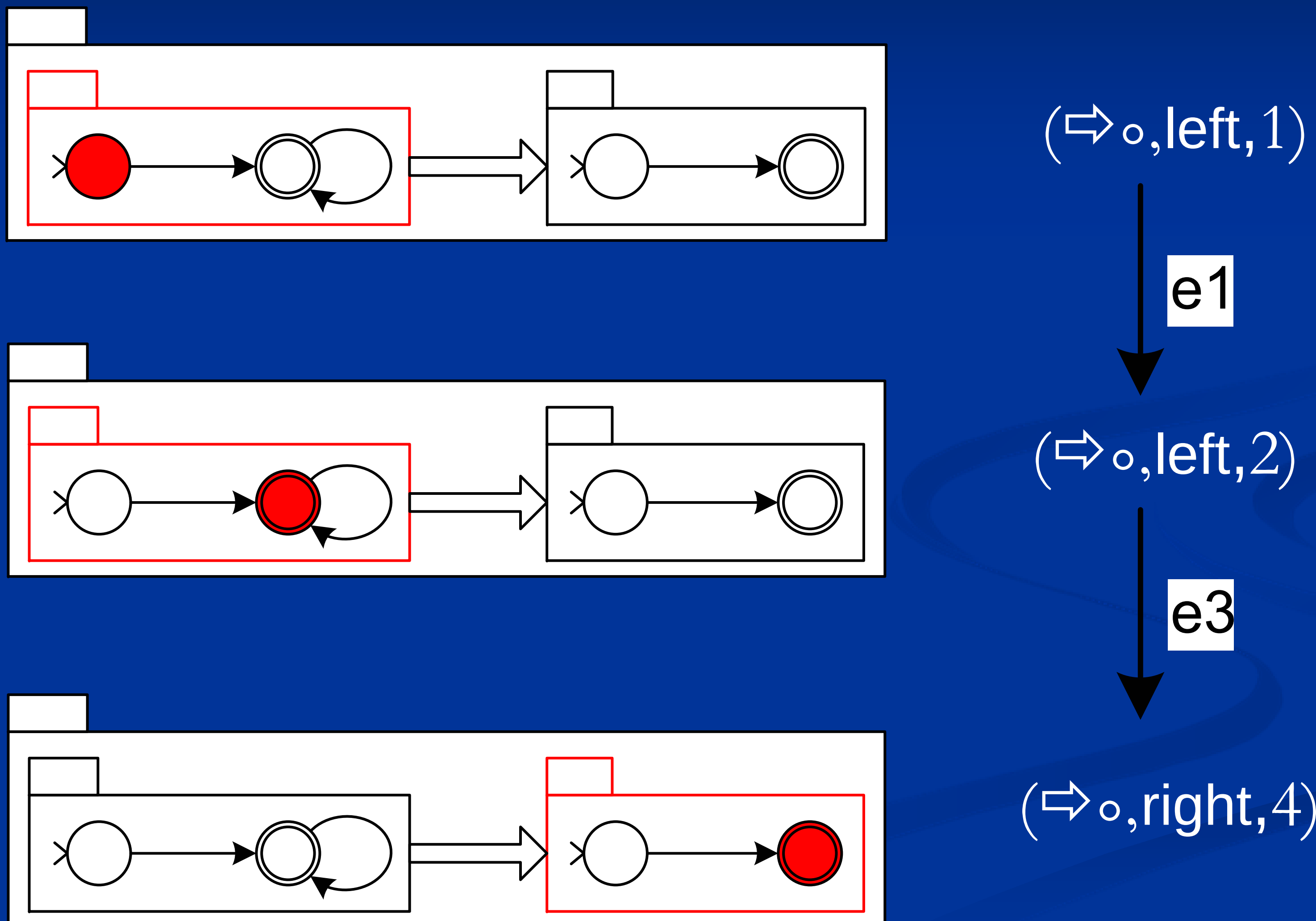
State

- elementary states
- compound states
  - one state type for each operator
  - eg, sequence

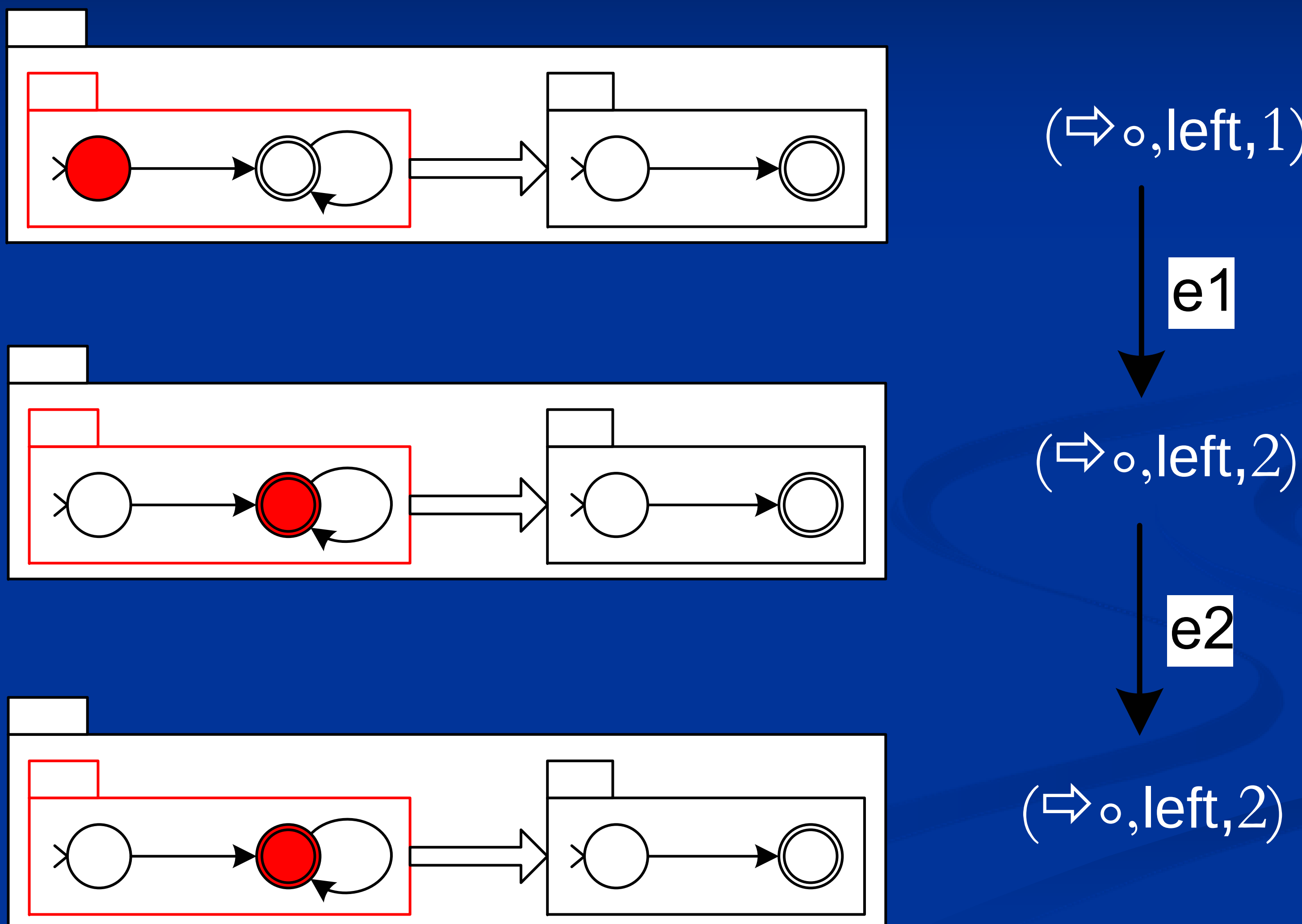
$$(\Rightarrow_\circ, \text{left}, s)$$

$$(\Rightarrow_\circ, \text{right}, s')$$

Sequence state transitions



Sequence state transitions



Sequence inference rules

$$\Rightarrow_1 \; \frac{s \xrightarrow{\sigma,\Gamma}_l s'}{(\Rightarrow_\circ, \mathsf{left}, s) \xrightarrow{\sigma,\Gamma} (\Rightarrow_\circ, \mathsf{left}, s')}$$

execute the left component

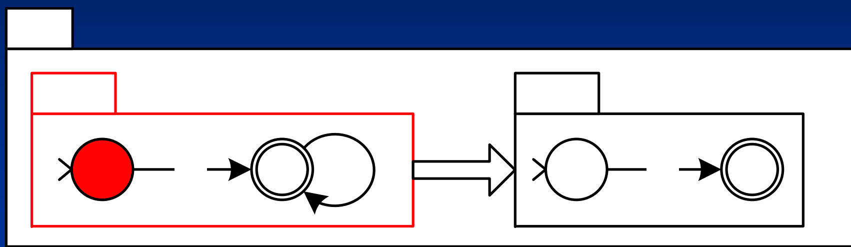
$$\Rightarrow_2 \; \frac{\mathit{final}_l(s)[\Gamma] \qquad \mathit{init}(r) \xrightarrow{\sigma,\Gamma}_r s'}{(\Rightarrow_\circ, \mathsf{left}, s) \xrightarrow{\sigma,\Gamma} (\Rightarrow_\circ, \mathsf{right}, s')}$$

when the left component is in a final state, the right component can execute

$$\Rightarrow_3 \; \frac{s \xrightarrow{\sigma,\Gamma}_r s'}{(\Rightarrow_\circ, \mathsf{right}, s) \xrightarrow{\sigma,\Gamma} (\Rightarrow_\circ, \mathsf{right}, s')}$$

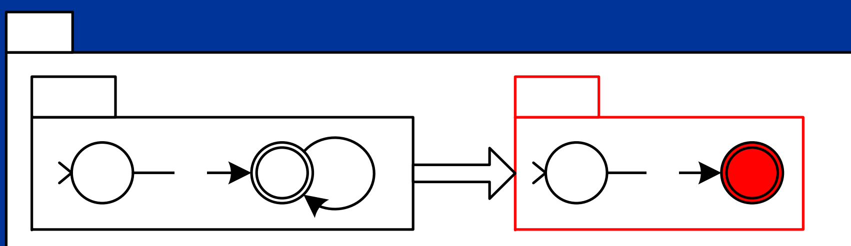
execute the right component

Closure state transitions



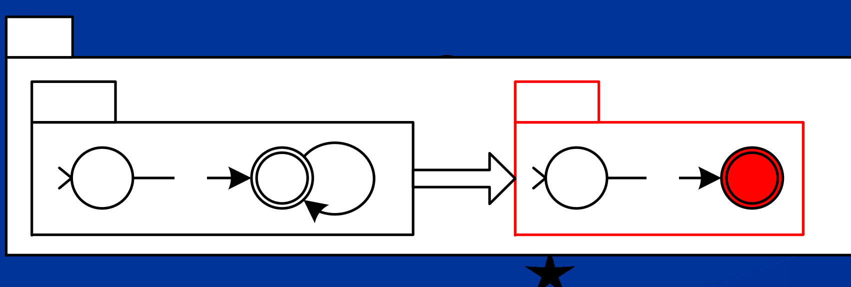
(⇨∘,left,(★,¬started, 1))

↓ ■



(⇨∘,right,(★,started, 2))

↓ ■



(⇨∘,right,(★,started, 2))

Closure inference rules

$$\star_1 \;\frac{(\mathit{final}_b(s)[\Gamma] \vee \neg \mathit{started?}) \qquad \mathit{init}(b) \xrightarrow{\sigma,\Gamma}_b s'}{(\star_\circ, \mathit{started?}, s) \xrightarrow{\sigma,\Gamma} (\star_\circ, \mathsf{true}, s')}$$

$$\star_2 \;\frac{s \xrightarrow{\sigma,\Gamma}_b s'}{(\star_\circ, \mathsf{true}, s) \xrightarrow{\sigma,\Gamma} (\star_\circ, \mathsf{true}, s')}$$

Initial and final states

- initial state is defined for each ASTD type
  - an ASTD type denotes the set of ASTDs that can be constructed for a given operator
  - recursively defined
- final state is a Boolean function which determines if a state is final

Initial and final states

sequence

$$\begin{aligned} init((\Rightarrow, l, r)) &\triangleq (\Rightarrow_\circ, \textsf{left}, init(l)) \\ final((\Rightarrow_\circ, \textsf{left}, s)) &\triangleq final_l(s) \wedge final_r(init(r)) \\ final((\Rightarrow_\circ, \textsf{right}, s)) &\triangleq final_r(s) \end{aligned}$$

Kleene closure

$$\begin{aligned} init((\star, b)) &\triangleq (\star_\circ, \textsf{false}, init(b)) \\ final((\star_\circ, started?, s)) &\triangleq final_b(s) \vee \neg started? \end{aligned}$$

Conclusion

- process algebra operators can improve the expressiveness of statecharts
- complete, precise models of information systems
  - not just single instance scenarios, but also multiple instance scenarios
- future work
  - tools for animation
  - model checking
  - code generation