Statecharts for the many: Algebraic State Transition Diagrams

Marc Frappier

GRIL – Groupe de recherche en ingénierie du logiciel

UNIVERSITÉ DE SHERBROOKE

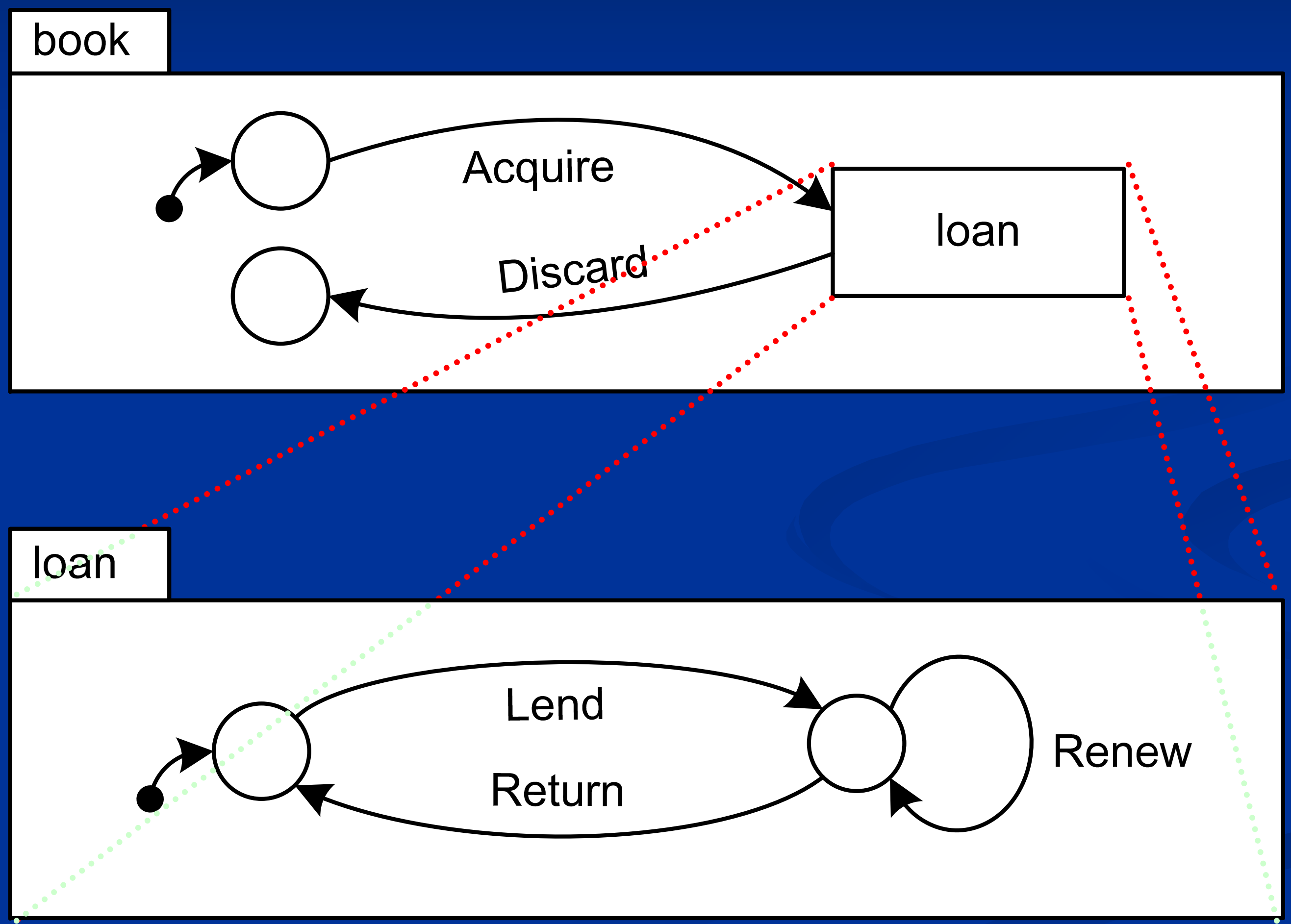
Plan

- Statecharts and information system 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
- parameterized states in Harel's seminal paper (SCP 87)
  - "*never*" implemented or formalised

A library in statecharts

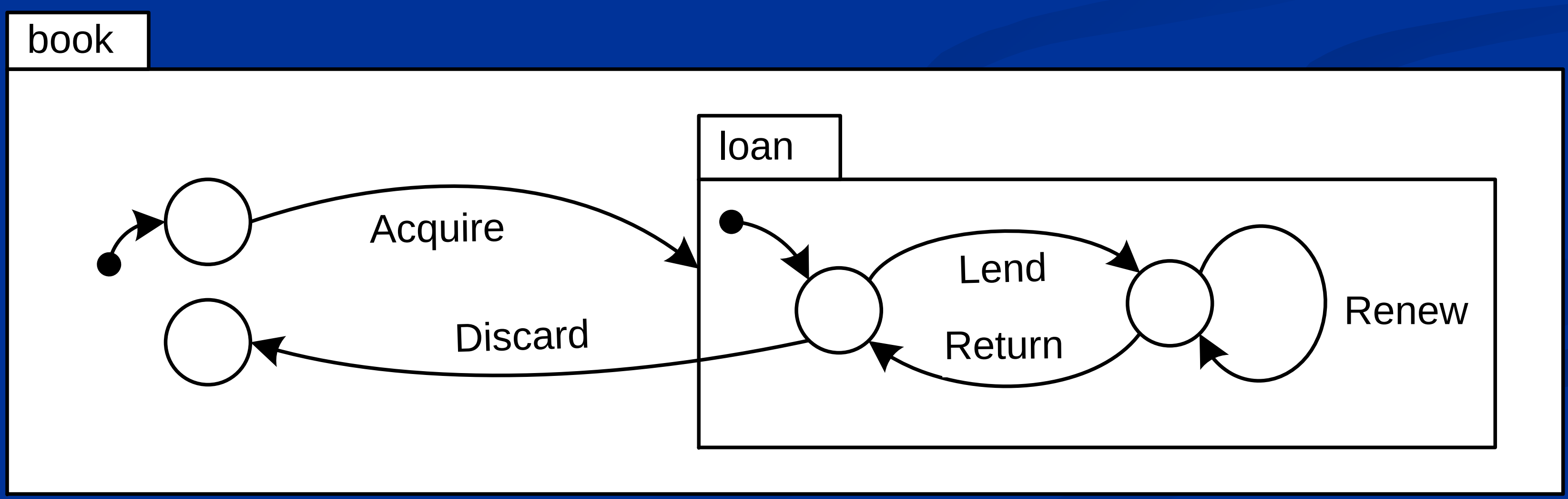


Problems

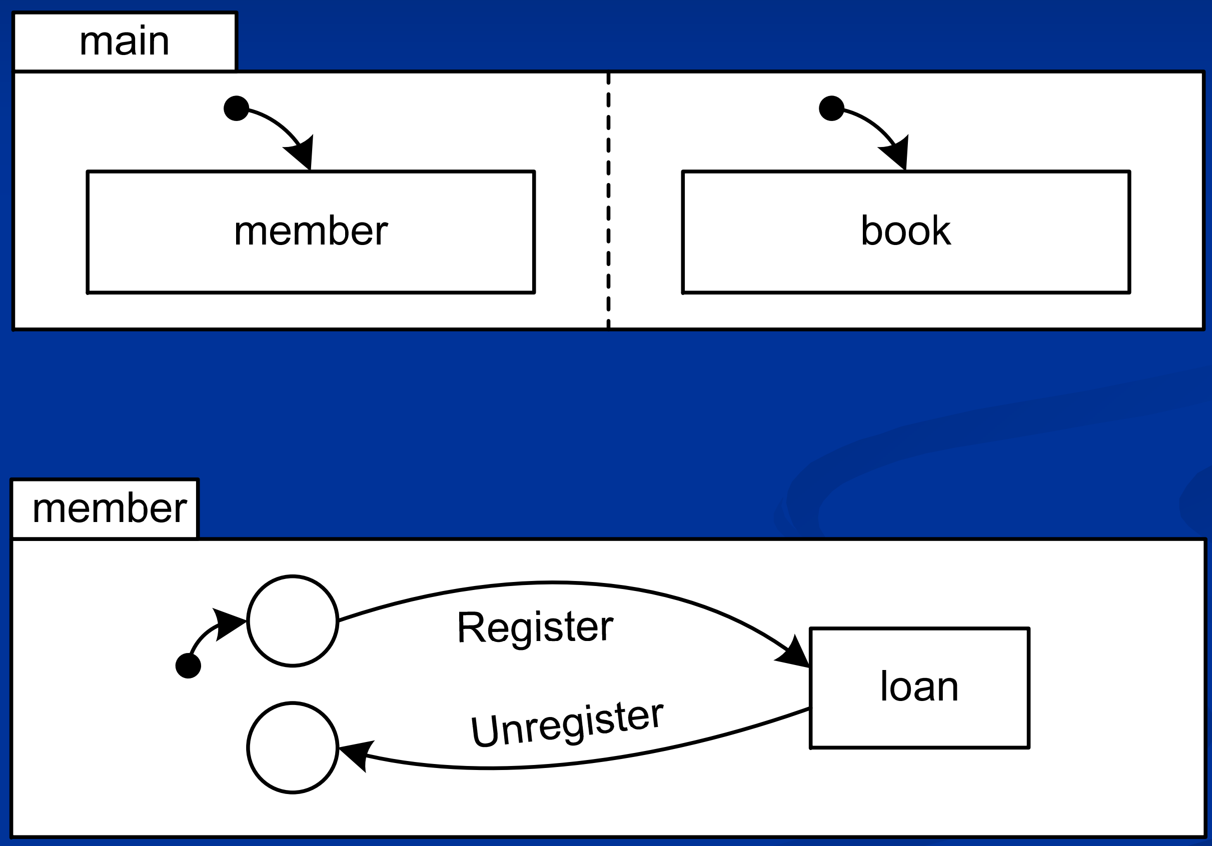
- only describes behaviour of a single book
  - how to deal with several books?
    - put n copies of book in parallel
      - not defined in statecharts or UML
      - available in ROSE RT, but it is not quite what we want here
- can discard an unreturned book
  - could add a guard to discard
    - unnecessary complexity
  - could make discard a transition from an inner state of loan
    - introduce coupling between book and loan

Potential solutions

- book knows about the structure of loan
  - makes loan less reusable
  - makes maintenance more difficult



Adding members



Problems

- a member can borrow several books in parallel
  - can't "easily" express that in statecharts or UML
    - State explosion
- two calls to loan
  - one in member, one in book
  - they both get the lend event
  - OK if only one member
  - KO if we have several members trying to borrow the same book
    - could remove loan from member
      - must add guard to Unregister to check for completed loan
      - loose visual ordering constraint

Potential solutions

- remove loan from member
  - loose visual ordering constraint between member and loan
    - replaced by a guard
  - need state variable

The single instance view: A weakness of statecharts

- both statecharts and UML state machines are designed to represent a single instance
  - eg, controller, object of a class, etc
- they offer no convenient means to express relationships between multiple instances
- in practice, designers only describe the single instance behaviour
  - leave it to the implementer to figure out the multiple instance case

A solution: Process algebra

- CCS, CSP, ACP, LOTOS, EB^3, ...
- algebra
  - operators to combine process expressions
    - sequence, choice, interleave, synchronisation, guard, ...
    - quantification
  - operators are the essence of abstraction
    - combine small units to build large units
    - operators foster abstraction by masking internal details

A Process expression for books

$$
\begin{aligned}
&\mathrm{book}(b : \mathrm{BookId}) = \\
&\quad \mathrm{Acquire}(b, \_) \\
&\quad \bullet \\
&\quad \mathrm{loan}(\_, b)^{\star} \\
&\quad \bullet \\
&\quad \mathrm{Discard}(b)
\end{aligned}
$$

A process expression for loans

loan(mId:Member, IDbId:BookID) =

guard

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

•

Renew(bId)*

•

Return(bId)

A process expression for members

member(m : MemberId) =

Register(m, _, _)

•

$(\boxed{|||\ b : BookId} : loan(m, b)^{\star})$

•

Unregister(m)

interleave quantification over all books

Interleave quantification

$$
\begin{array}{l}
\quad ||| \; x : \{1,2,3\} : P(x) \\
= \\
\quad P(1) \; ||| \; P(2) \; ||| \; P(3)
\end{array}
$$

Main process expression

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

Synchronisation over common actions

Synchronisation over common actions

$$a(1) \bullet b(1) \bullet c(1)$$

$$||$$

$$\boxed{|\, x : T :}\, a(x) \bullet b(x) \bullet c(2)$$

quantified choice

$$=$$

$$a(1) \bullet b(1) \bullet STOP$$

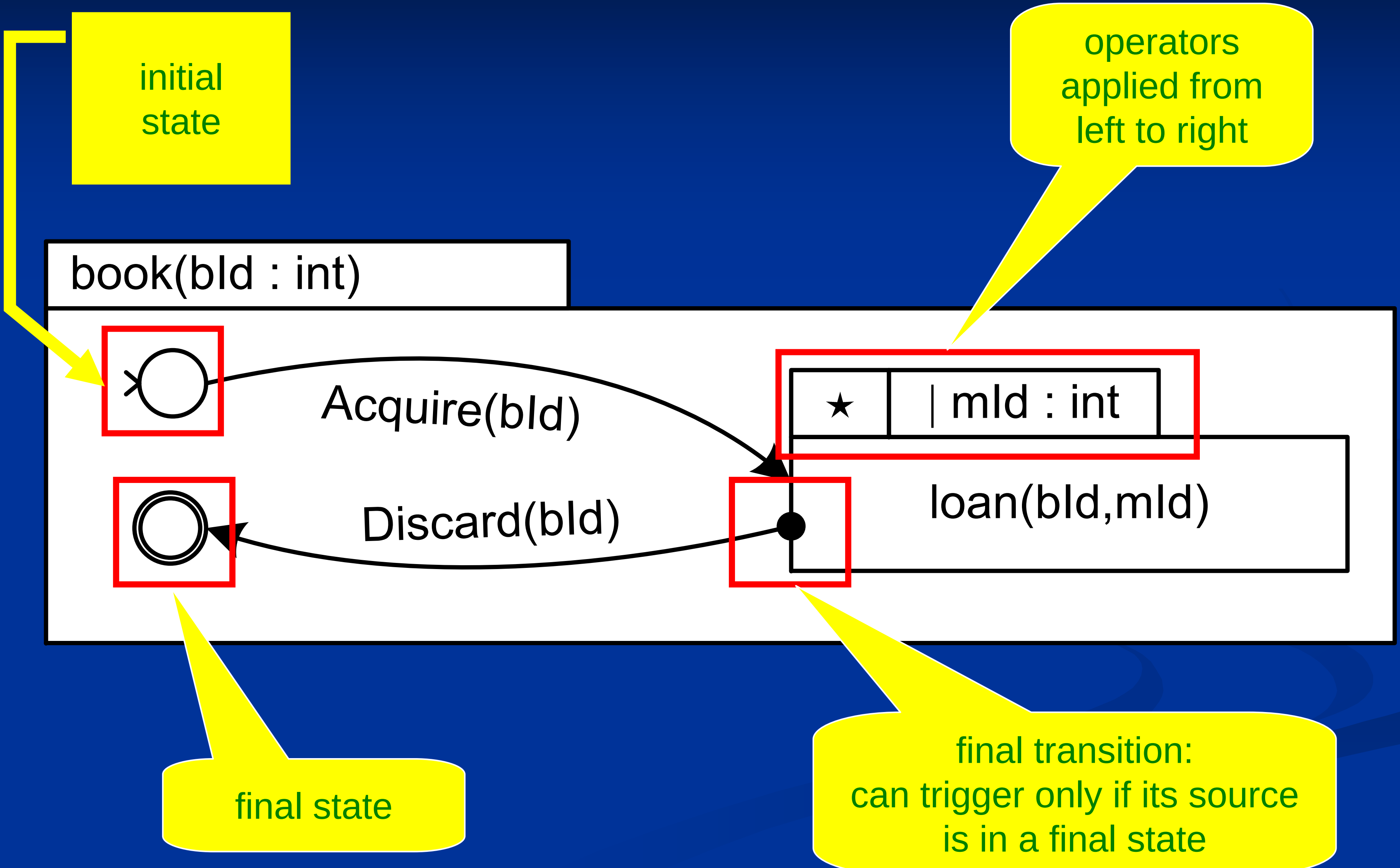
ASTD

- 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

ASTD Operators

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

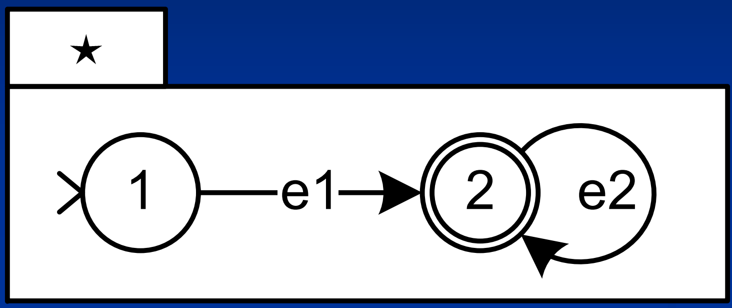
A book ASTD

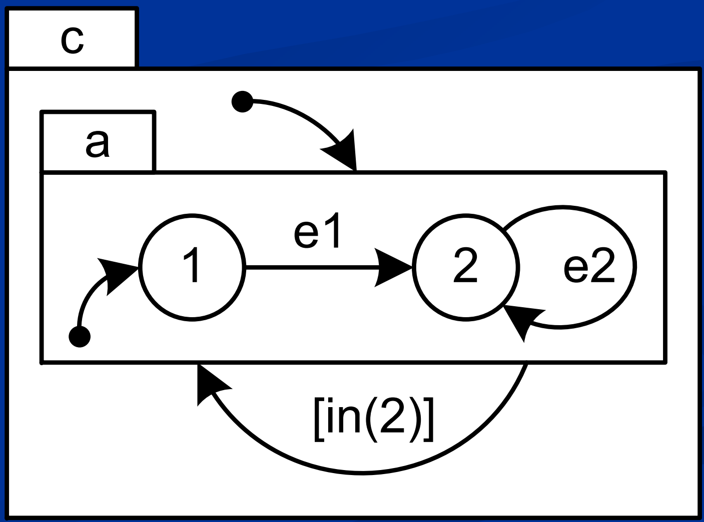


initial state
operators applied from left to right
book(bId : int)
Acquire(bId)
★
| mId : int
loan(bId,mId)
Discard(bId)
final state
final transition: can trigger only if its source is in a final state

Closure applied to an ASTD

- ★ means execute the ASTD an arbitrary number of times, including 0
  - when the ASTD is in a final state, it can start again from its initial state
- example traces are
  - empty trace
  - e1,e2,e2,...,e1,e1,e2, ...





The closure ASTD type

(★, body)

- ★ denotes the type constructor for a closure
- body is an ASTD (of any type)

The closure state type

- ★∘ is the closure state type constructor
- started? is a boolean value that indicates if its component has started its first iteration
- s is the state of its component

(★∘ ,started?, s)

States of a closure

function that defines the initial state of an ASTD

closure ASTD

closure initial state

- initial state
  - is the initial state of its component

$$\boxed{init}(\boxed{(\star, b)}) \quad \triangleq \quad \boxed{(\star_\circ, \mathsf{false}, init(b))}$$

- final states
  - its initial state
  - final states of its component

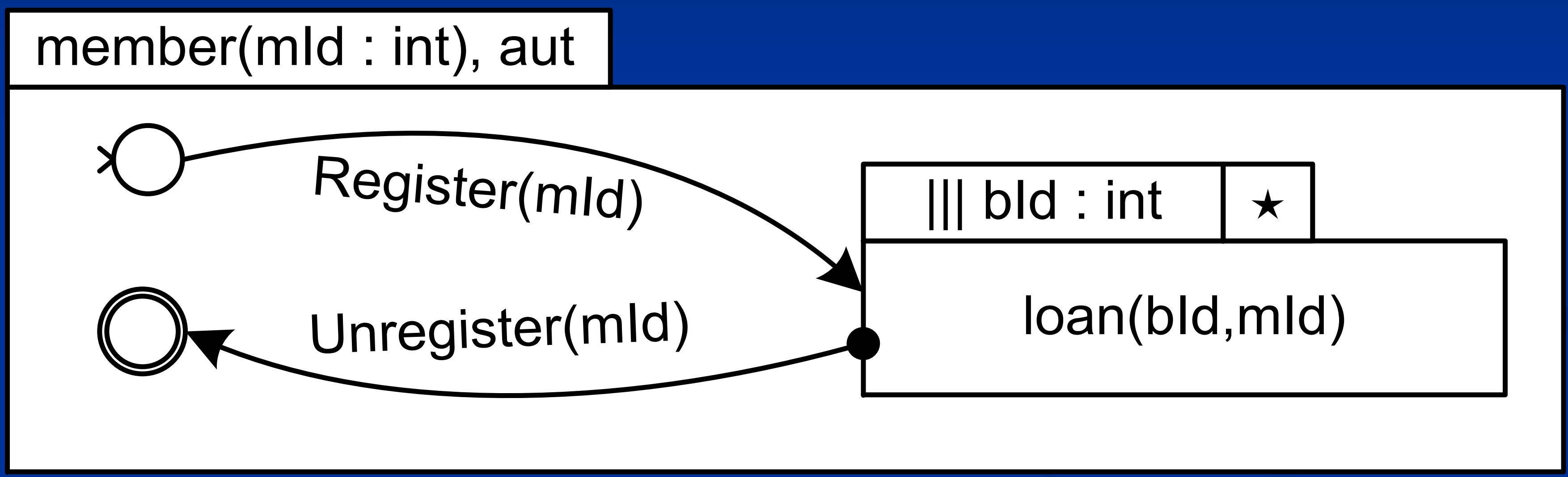
$$\boxed{final}((\star_\circ, started?, s)) \quad \triangleq \quad final_b(s) \vee \neg started?$$

function that determines if a state is final

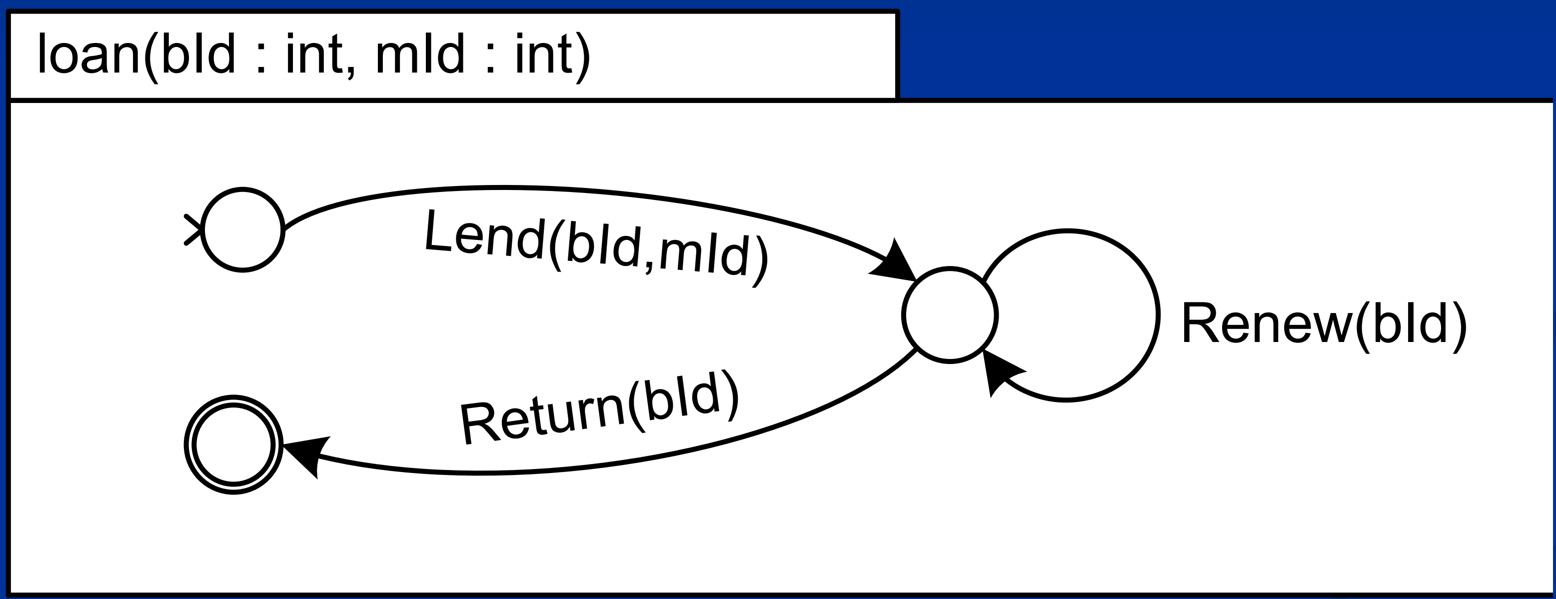
Final state

- an ASTD does not terminate when its current state is final
- a final state simply *enables* transitions of another ASTD within a
  - closure
  - sequence

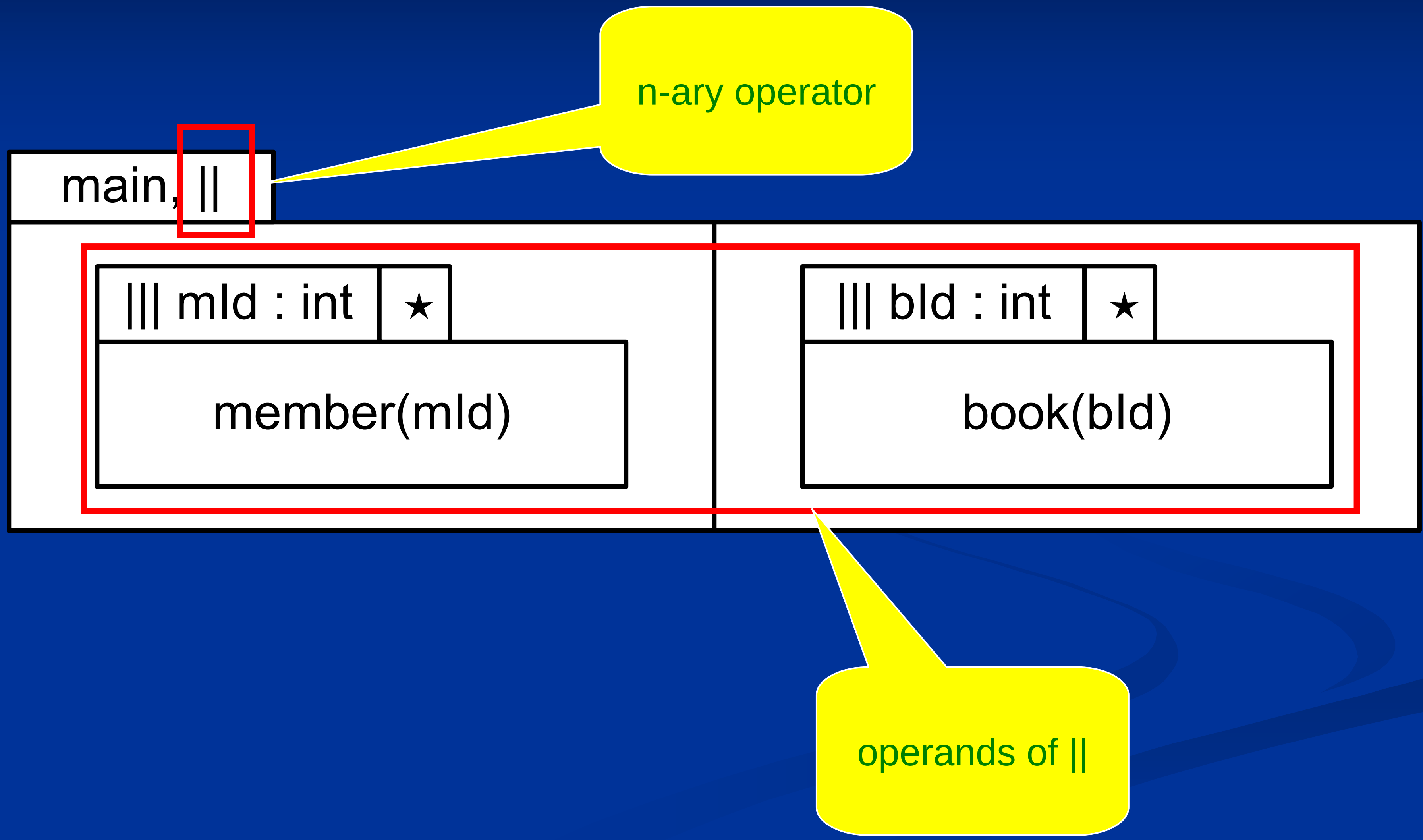
A member ASTD



A loan ASTD



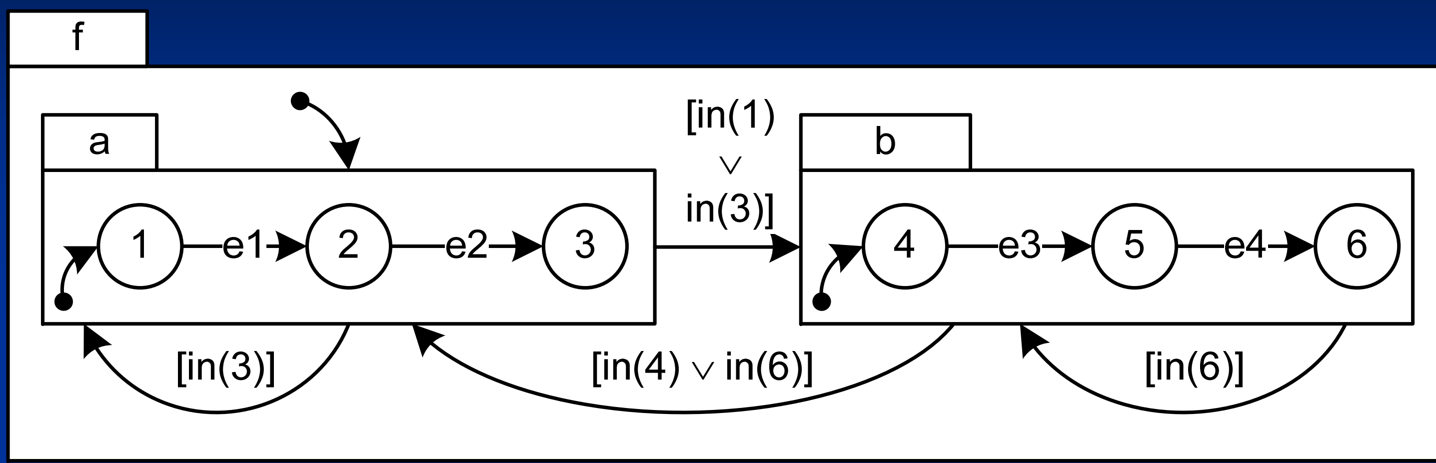
The main ASTD

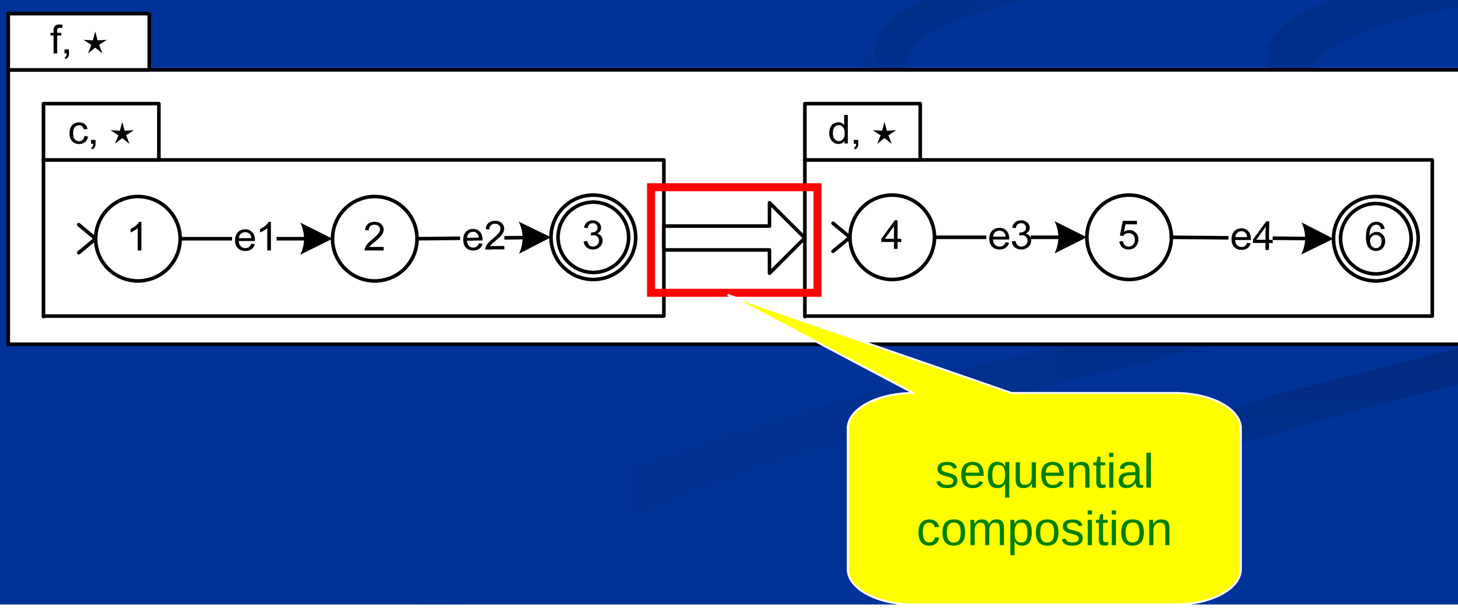


Power of abstraction

- 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





The sequence ASTD type

(⇨, left, right)

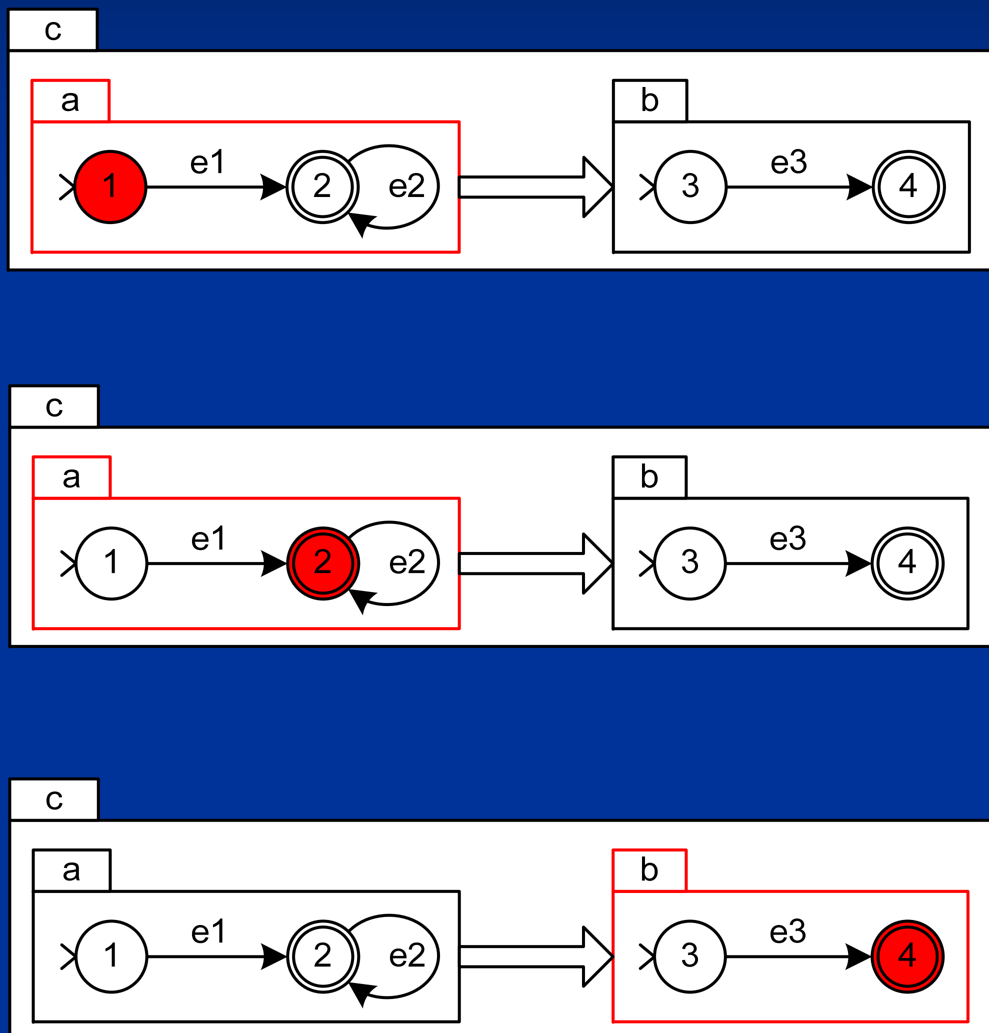
- ⇨ denotes the sequence ASTD type constructor
- left and right are ASTDs

The sequence state type

- ⇨∘ denotes the sequence state type constructor
- side denotes the current side of the sequence
  - left
  - right
- s denotes the state of the side component

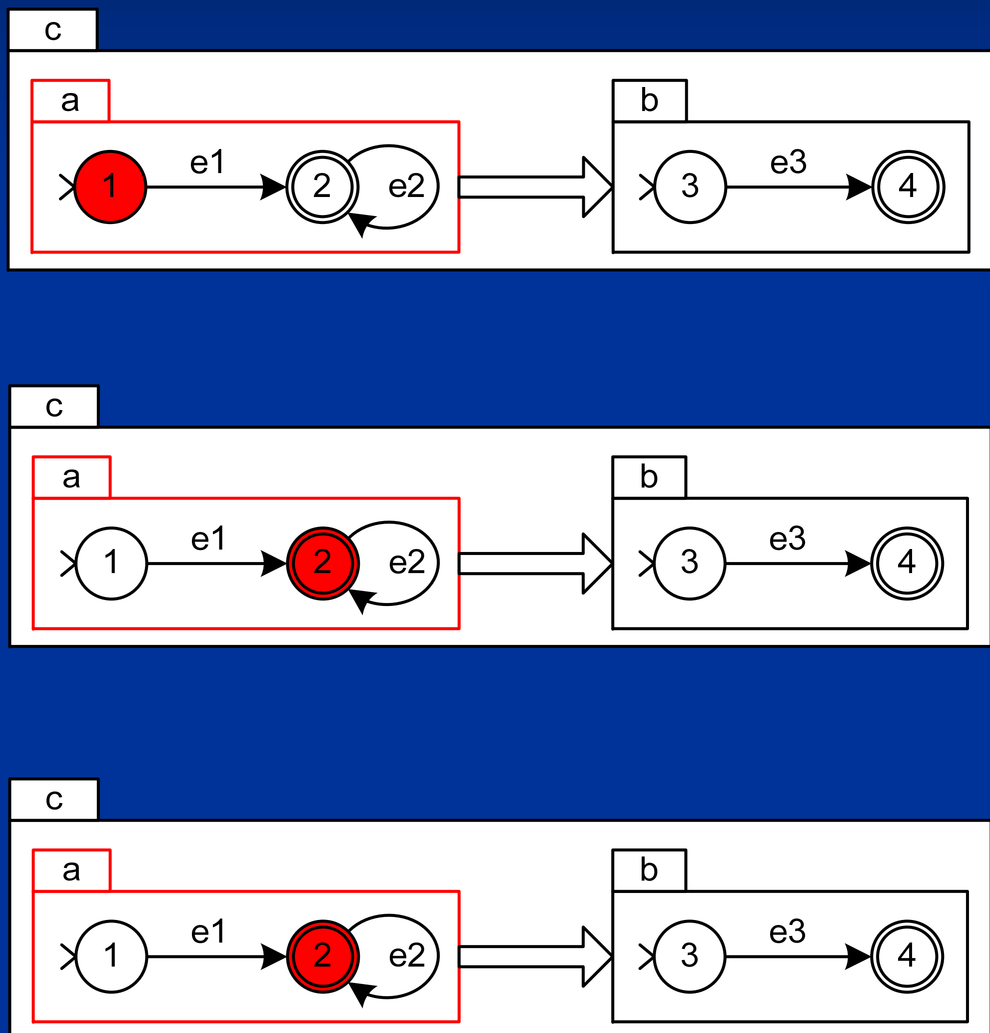
(⇨∘, side, s)

State transitions



$$(\Rightarrow\circ,\text{left},1) \xrightarrow{\text{e1}} (\Rightarrow\circ,\text{left},2) \xrightarrow{\text{e3}} (\Rightarrow\circ,\text{right},4)$$

State transitions



(⇨∘,left,1)

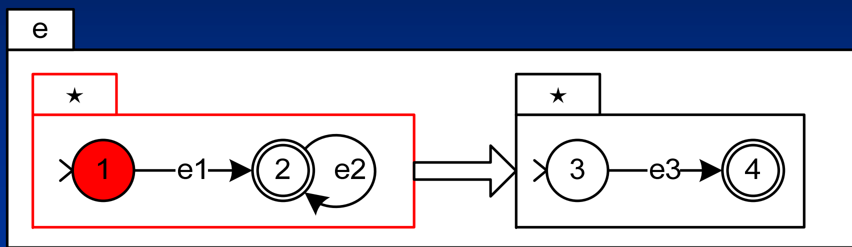
e1

(⇨∘,left,2)

e2

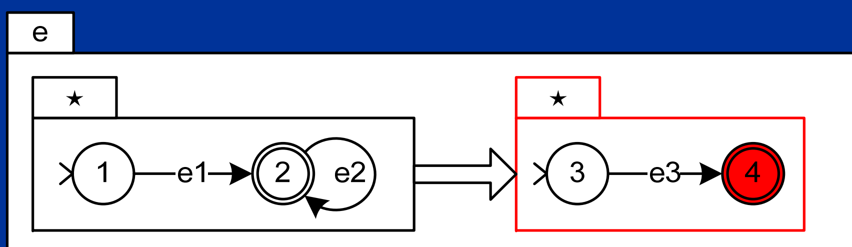
(⇨∘,left,2)

State transitions



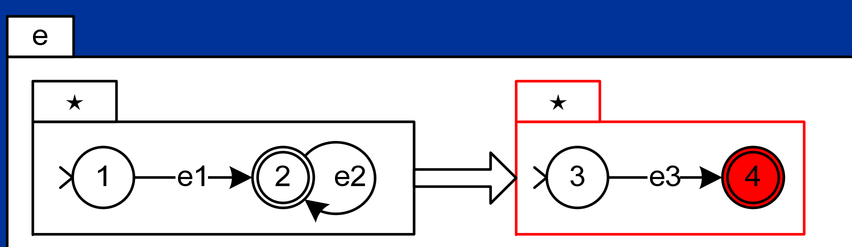
(⇨∘,left,(★∘,¬started,1))

$\Big\downarrow$ e3



(⇨∘,right,(★∘,started,4))

$\Big\downarrow$ e3



(⇨∘,right,(★,started,4))

Initial and final states of a sequence ASTD

$$
\begin{array}{rcl}
init((\Rightarrow, l, r)) & \triangleq & (\Rightarrow_{\circ}, \mathsf{left}, init(l)) \\
final((\Rightarrow_{\circ}, \mathsf{left}, s)) & \triangleq & final_{l}(s) \wedge final_{r}(init(r)) \\
final((\Rightarrow_{\circ}, \mathsf{right}, s)) & \triangleq & final_{r}(s)
\end{array}
$$

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

Inference rules

- first rules deals with environment, noted ([ ]), to manage variables introduced by
  - quantifications
  - process parameters

$$\mathsf{env}\ \frac{s \xrightarrow{\sigma, (\![\,]\!)} s'}{s \xrightarrow{\sigma} s'}$$

Auto … erence rules

similar to traditional
δ of an automaton

$$\mathsf{aut}_1 \quad \frac{\boxed{\delta((\mathsf{loc}, n_1, n_2), \sigma', g, \mathit{final?})} \qquad \boxed{\Psi}}{(\mathsf{aut}_\circ, n_1, h, s) \xrightarrow{\sigma,\Gamma} (\mathsf{aut}_\circ, n_2, h', \mathit{init}(\nu(n_2)))}$$

execute an automaton transition

$$\boxed{\begin{aligned}\Psi \triangleq \big((\mathit{final?} \Rightarrow \mathit{final}_{\nu(n_1)}(s)) \wedge {} \\ g \wedge \sigma' = \sigma \wedge h' = h \mathbin{\lhd\!\!\!+} \{n_1 \mapsto s\}\big)[\Gamma]\end{aligned}}$$

$$\mathsf{aut}_6 \quad \frac{s \xrightarrow{\sigma,\Gamma}_{\nu(n)} s'}{(\mathsf{aut}_\circ, n, h, s) \xrightarrow{\sigma,\Gamma} (\mathsf{aut}_\circ, n, h, s')}$$

execute a transition of the component

Closure inference rules

execute from the initial state of the component

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

execute the component when started

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

Sequence inference rules

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

execute on left

$$\Rightarrow_2 \frac{final_l(s)[\Gamma] \qquad init(r) \xrightarrow{\sigma,\Gamma}_r s'}{(\Rightarrow_\circ, \textsf{left}, s) \xrightarrow{\sigma,\Gamma} (\Rightarrow_\circ, \textsf{right}, s')}$$

execute on right when left is final

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

execute the right component

Choice: initial and final states

Choice state

$(|_\circ, \mathsf{side}, s)$

$$
\begin{array}{rcl}
init((|, l, r)) & \triangleq & (|_\circ, \bot, \bot) \\
final((|_\circ, \bot, \bot)) & \triangleq & final_l(init(l)) \vee final_r(init(r)) \\
final((|_\circ, \mathsf{fst}, s)) & \triangleq & final_l(s) \\
final((|_\circ, \mathsf{snd}, s)) & \triangleq & final_r(s)
\end{array}
$$

Choice inference rules

$$|_1 \; \frac{init(l) \xrightarrow{\sigma,\Gamma}_l s'}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathsf{fst}, s')}$$

execute the first component from its initial state

$$|_2 \; \frac{init(r) \xrightarrow{\sigma,\Gamma}_r s'}{(|_\circ, \bot, \bot) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathsf{snd}, s')}$$

execute the second component from its initial state

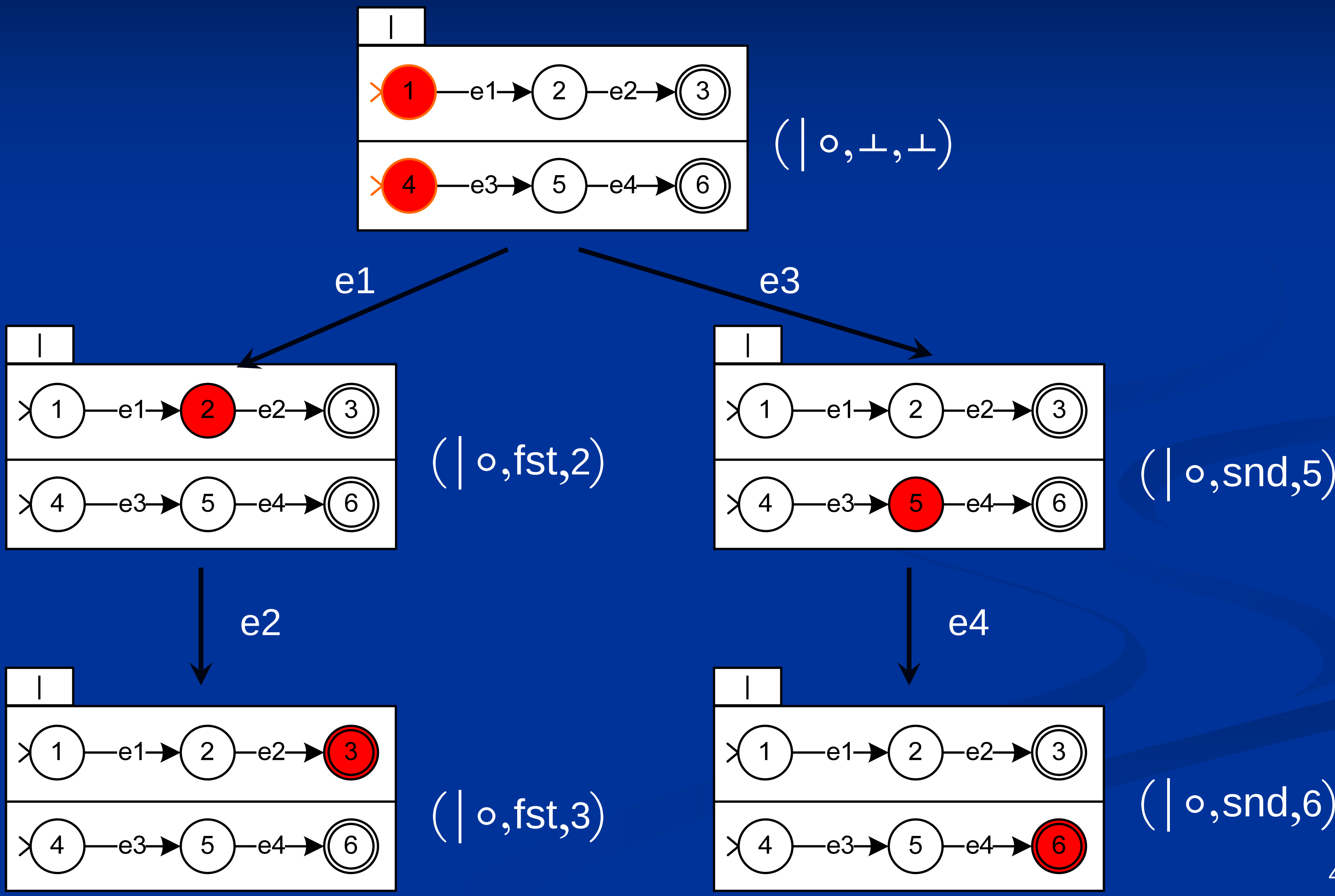
$$|_3 \; \frac{s \xrightarrow{\sigma,\Gamma}_l s'}{(|_\circ, \mathsf{fst}, s) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathsf{fst}, s')}$$

execute the first component when it has been selected

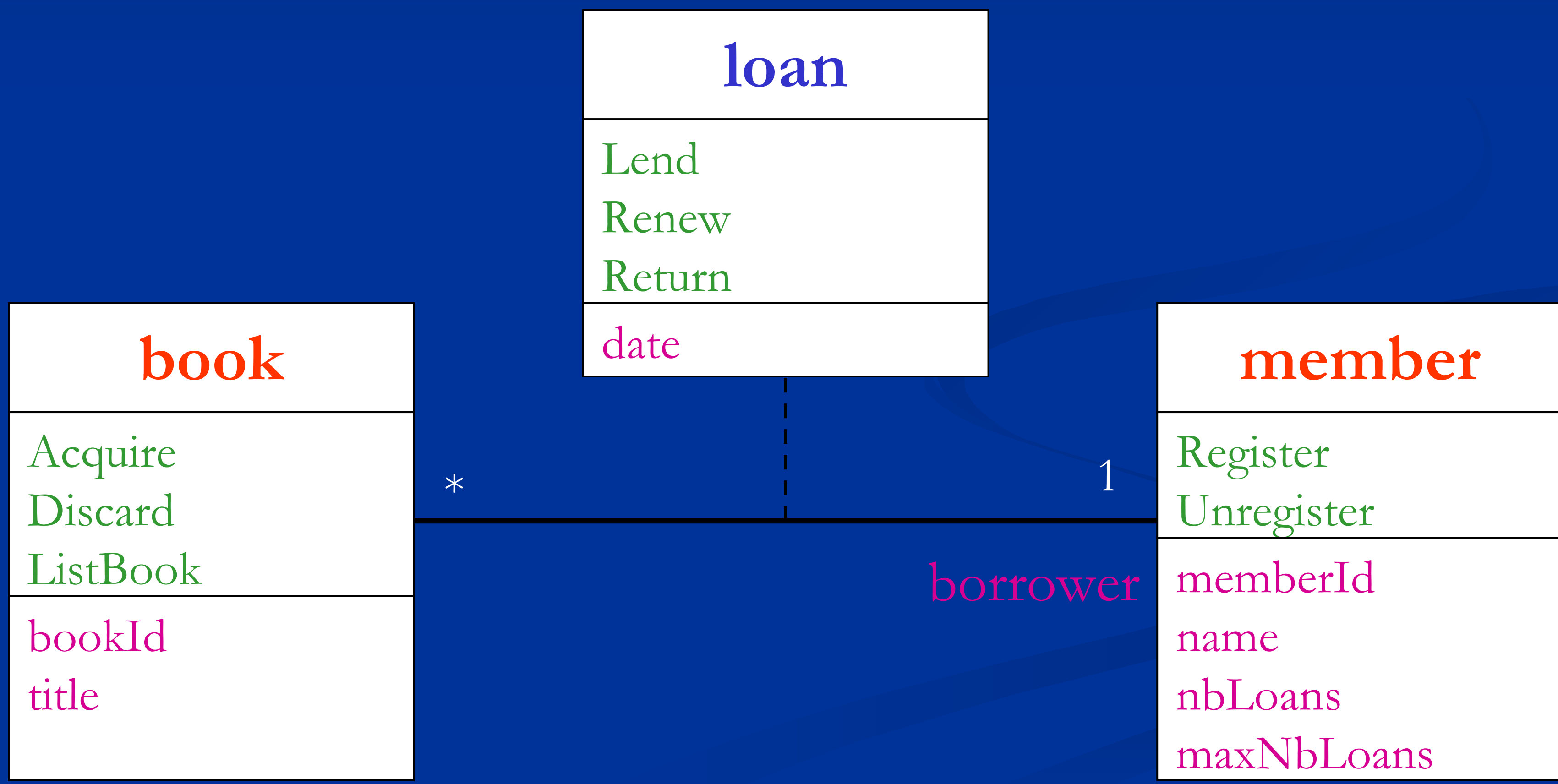
$$|_4 \; \frac{s \xrightarrow{\sigma,\Gamma}_r s'}{(|_\circ, \mathsf{snd}, s) \xrightarrow{\sigma,\Gamma} (|_\circ, \mathsf{snd}, s')}$$

execute the second component when it has been selected

Choice example



Integration with the business class diagram



State variables

- the system trace is the only state variable
- entity attributes are functions on this trace
- attributes 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, _)   : ⊥;

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