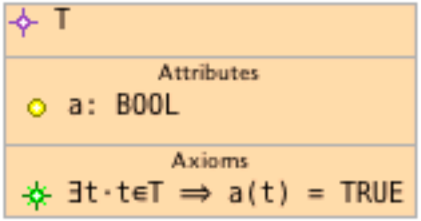
Refinement in UML-B

Extending Context Diagrams

Starting from a Context Diagram with a Class Type


T

Attributes

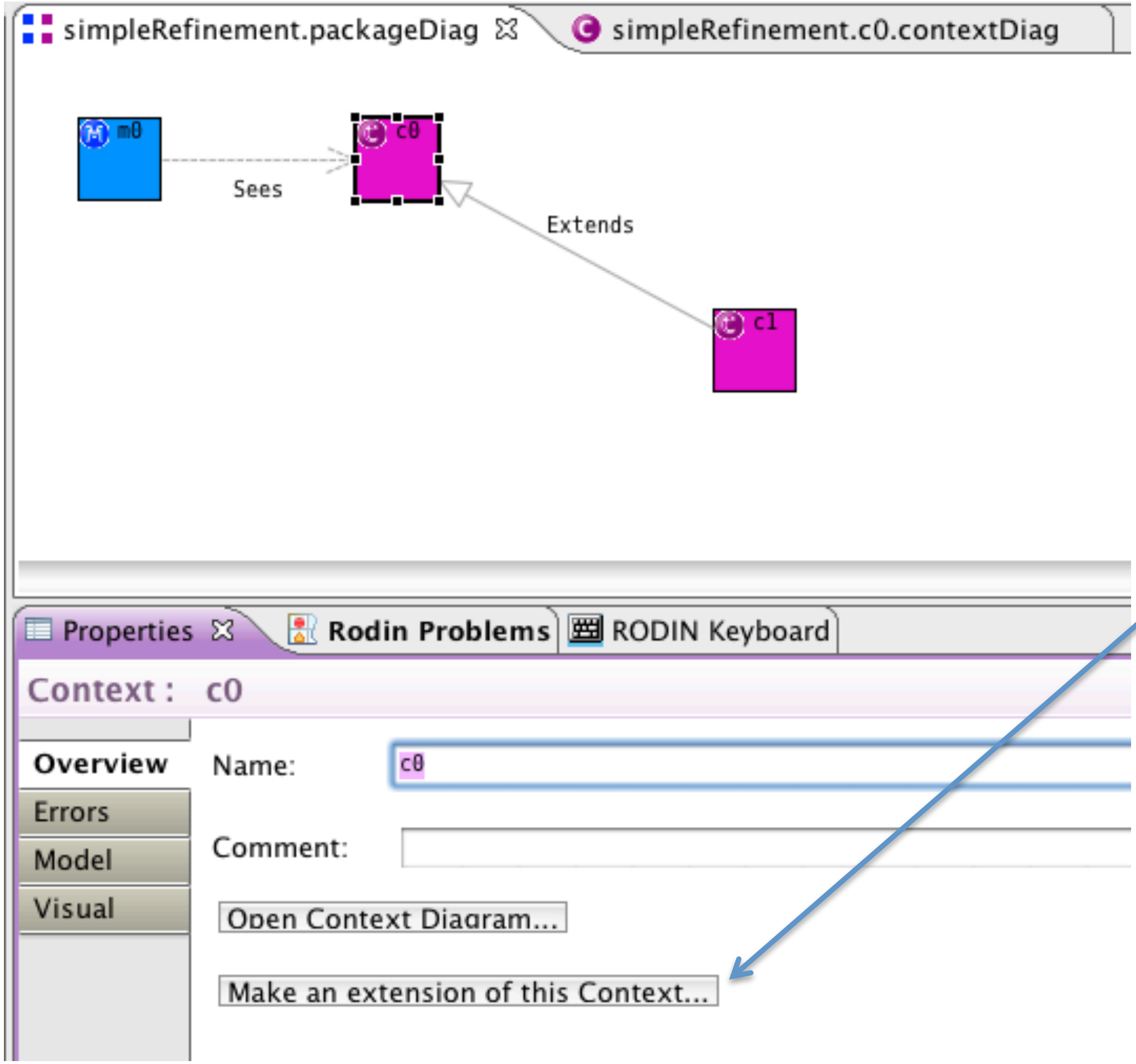
○ a: BOOL

Axioms

∃t·t∈T ⇒ a(t) = TRUE

```
CONTEXT
  c0

SETS
  T        //   ClassType

CONSTANTS
  a        //   attribute of T

AXIOMS
  a.type    :    a ∈ T ⟶ BOOL
  Axiom1    :    ∃t·t∈T ⇒ a(t) = TRUE

END
```

Make an Extension of this Context



Context : c0

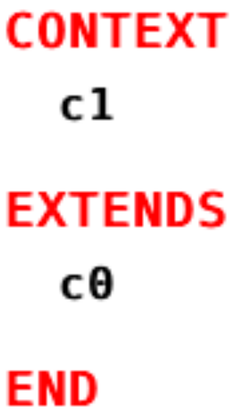
Name: c0

Comment:

1) Select context,
2) Click Button in Properties,
3) Makes a starting point
for extending

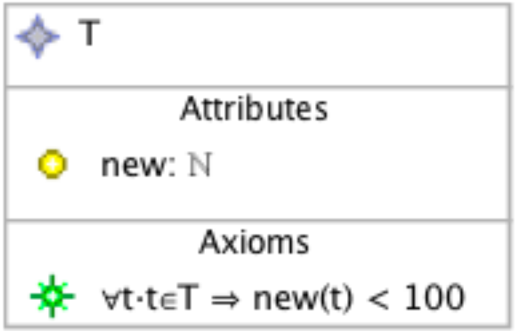
Provides a basis for extending classtypes

An empty ExtendedClassType



```
CONTEXT
   c1
EXTENDS
   c0
END
```

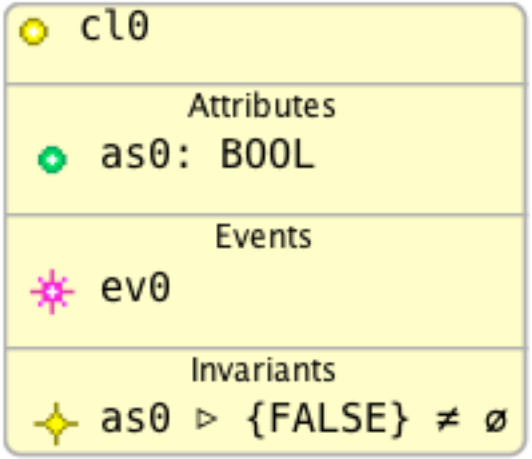
Add Attributes and Axioms to Extend ClassType



```
CONTEXT
   c1

EXTENDS
   c0

CONSTANTS
   new        //   attribute of T

AXIOMS
   new.type    :    new ∈ T → ℕ
   newAxiom    :    ∀t·t∈T ⇒ new(t) < 100

END
```

Refining Class Diagrams

Starting from a Class Diagram



cl0

Attributes

as0: BOOL

Events

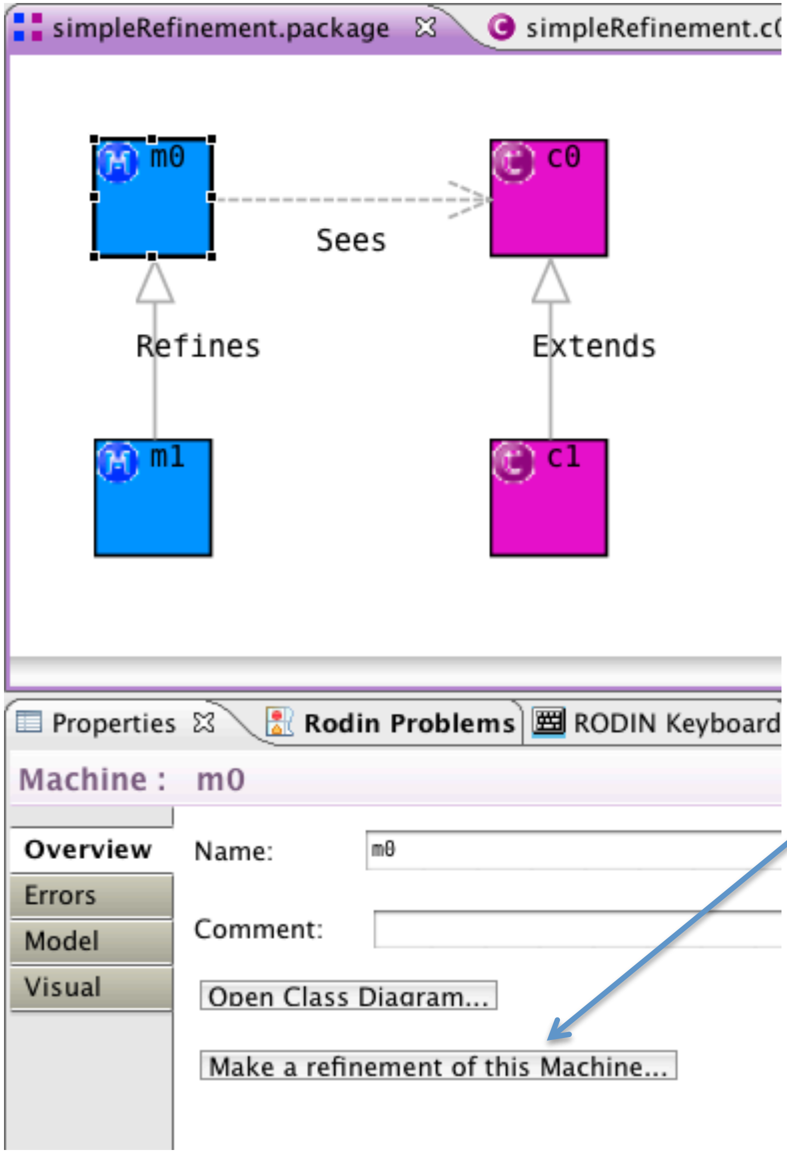
ev0

Invariants

as0 ▷ {FALSE} ≠ ∅

```
MACHINE
  m0
SEES
  m0_implicitContext
VARIABLES
  as0      //   attribute of cl0
INVARIANTS
  as0.type    :   as0 ∈ cl0 → BOOL
  Invariant1   :   as0 ▷ {FALSE} ≠ ∅
EVENTS

  INITIALISATION   ≙
  STATUS
    ordinary
  BEGIN
    as0.init    :   as0 ≔ cl0 × {FALSE}
  END

  ev0   ≙
  STATUS
    ordinary
  ANY
    thisCl0      //   contextual instance of class cl0
  WHERE
    thisCl0.type    :   thisCl0 ∈ cl0
    ev0.Guard1    :   (as0{thisCl0 ↦ TRUE}) ▷ {FALSE}≠∅
  THEN
    ev0.Action1    :   as0(thisCl0) ≔ TRUE
  END

END
```

“Make a refinement of this Machine”



Machine : m0

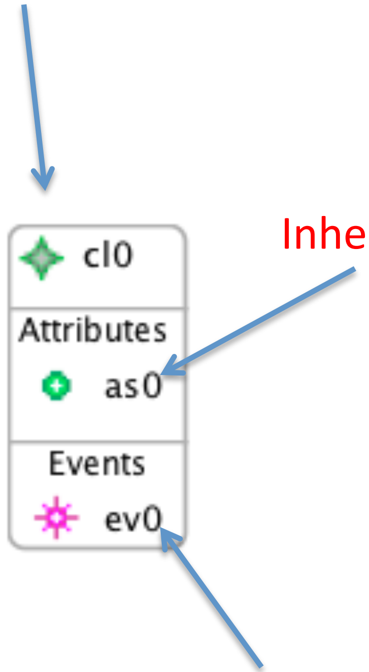
Name: m0

Comment:

Select Machine,
click Button in Properties,
Makes a starting point
for refinement

Basis for refinement

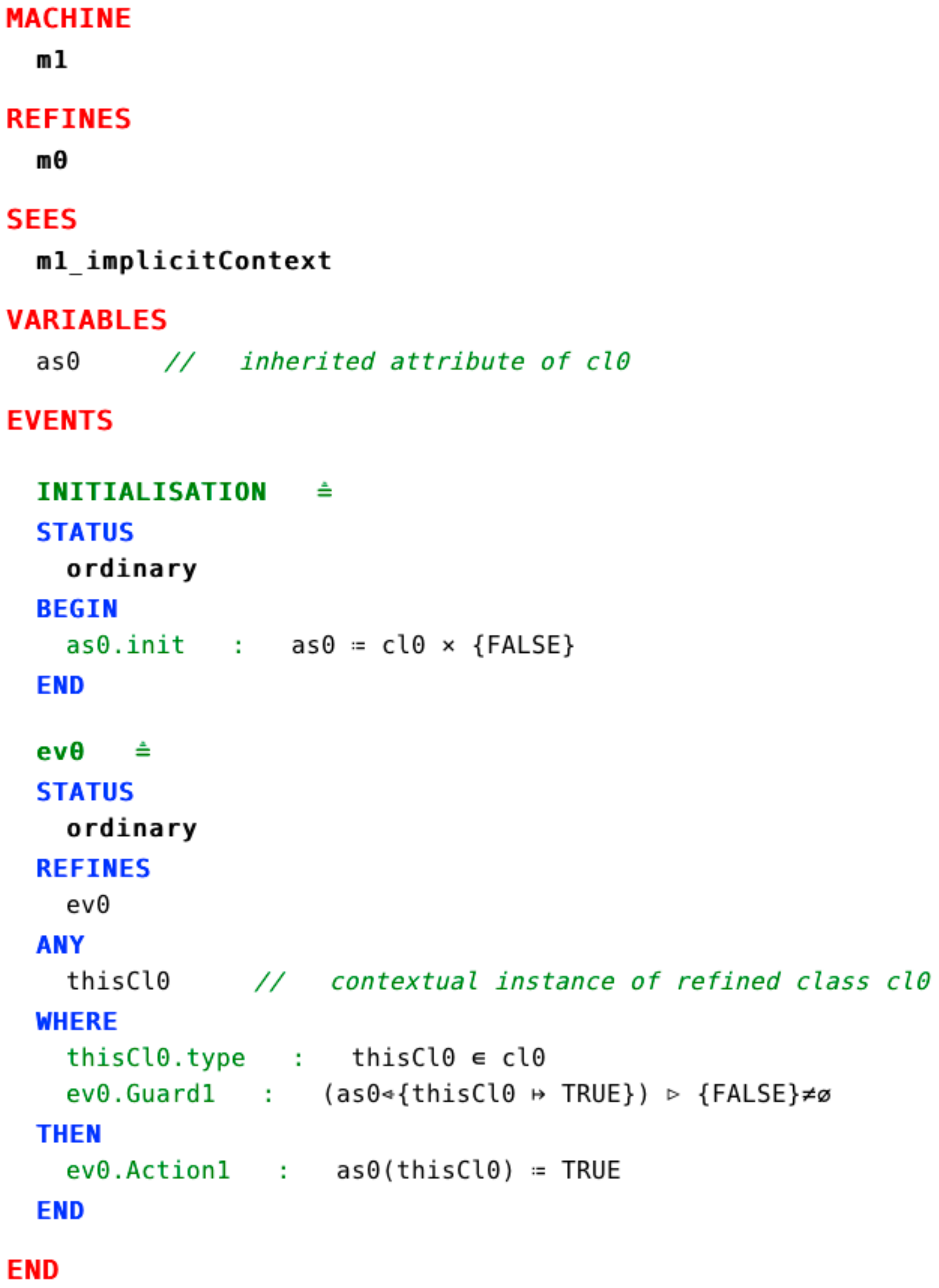
Refined Class



Inherited Attribute

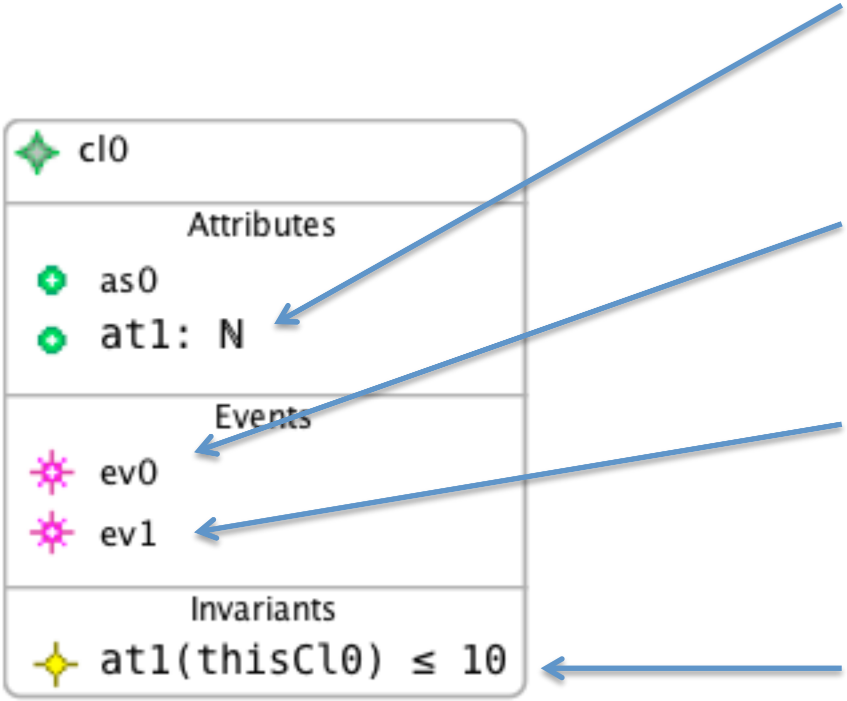
Copies of abstract Events are retained for refinement

Statemachines are copied
ready for refinement (not shown)



```
MACHINE
  m1
REFINES
  m0
SEES
  m1_implicitContext
VARIABLES
  as0     //   inherited attribute of cl0
EVENTS

  INITIALISATION   ≙
  STATUS
    ordinary
  BEGIN
    as0.init    :    as0 ≔ cl0 × {FALSE}
  END

  ev0   ≙
  STATUS
    ordinary
  REFINES
    ev0
  ANY
    thisCl0      //   contextual instance of refined class cl0
  WHERE
    thisCl0.type    :    thisCl0 ∈ cl0
    ev0.Guard1    :    (as0⩤{thisCl0 ↦ TRUE}) ▷ {FALSE}≠∅
  THEN
    ev0.Action1    :    as0(thisCl0) ≔ TRUE
  END

END
```

Refine a Class



add new attributes/associations

refine existing events
(strengthen guards, add actions,
split, merge)

add new events
(can only alter new variables)

add new invariants

Example – Bank Accounts and ATMs

ABSTRACT

Bank accounts have a balance which is zero when the account is opened.

Money may then be deposited in the account, increasing the balance by some amount,

or withdrawn, depleting the balance by some amount.

REFINEMENT

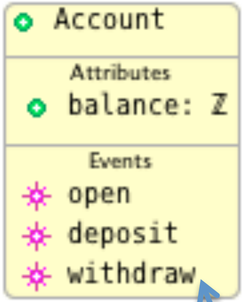
A card is associated with an account and withdrawls are made via an ATM machine.

The card is inserted into the ATM and either a successful withdrawl is completed and the card is ejected or the transaction fails.

(based on a case study by Mar Yah Said)

Example – Abstract

Note: for this example it is necessary to rename the contextual class instance from self to thisAccount so that it can be disambiguated when the event withdraw is moved to a different class.



Account

Attributes

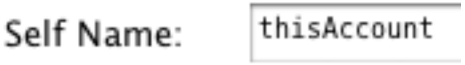
balance: ℤ

Events

open

deposit

withdraw

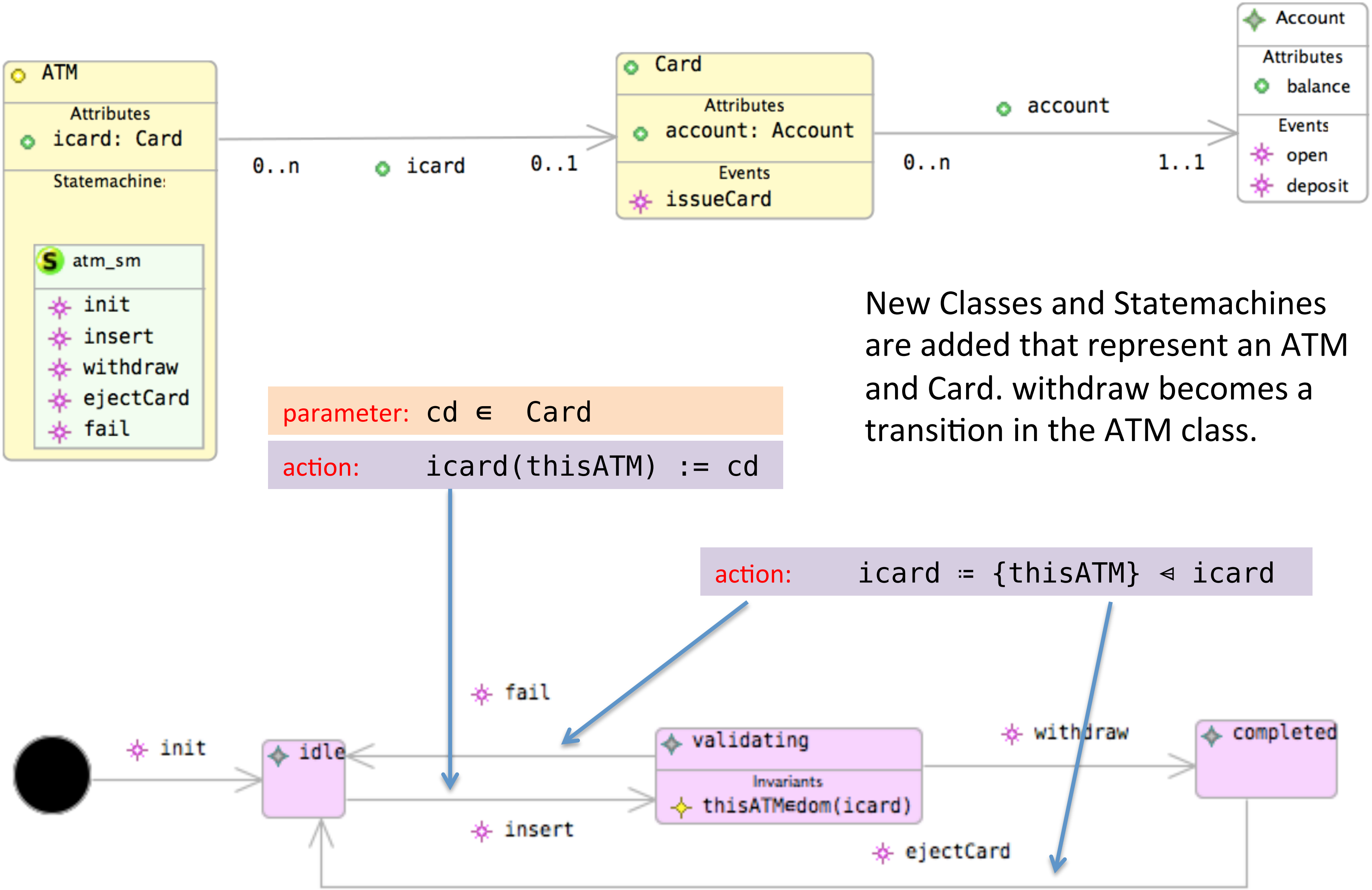


Self Name: thisAccount

parameter: amount ∈ ℕ1

action:
```
balance(thisAccount) ≔
          balance(thisAccount) − amount
```

Example – Refinement



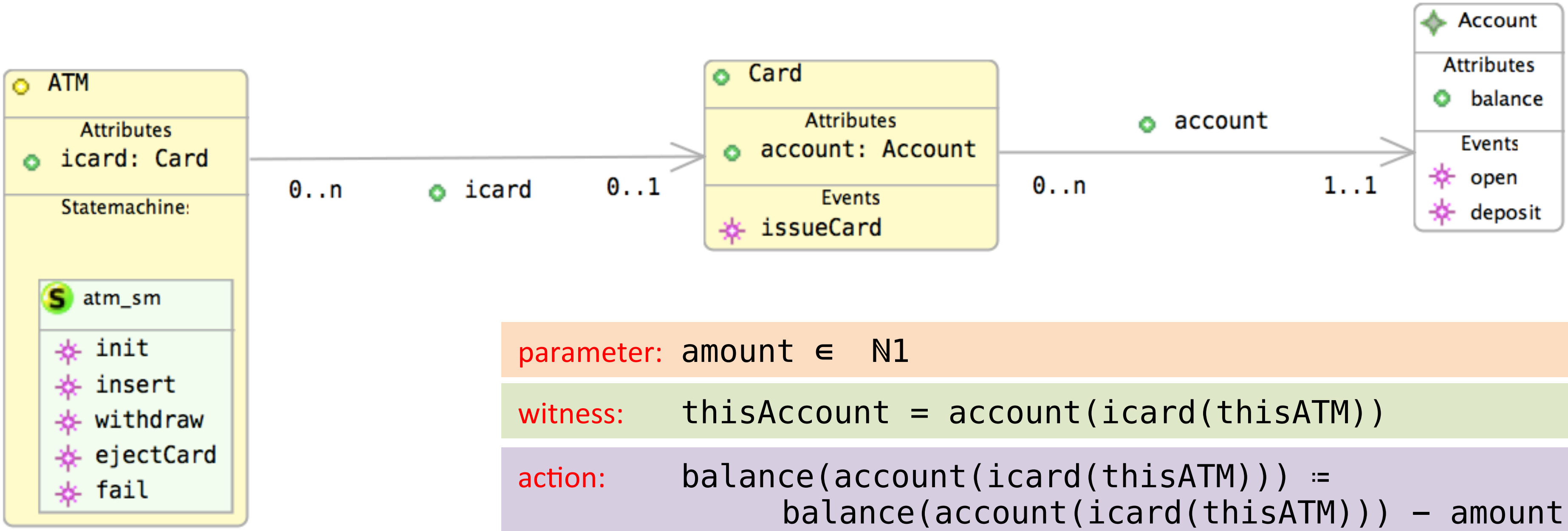
parameter: $cd \in Card$

action: $icard(thisATM) := cd$

action: $icard \coloneqq \{thisATM\} \lhd icard$

New Classes and Statemachines are added that represent an ATM and Card. withdraw becomes a transition in the ATM class.

Example – Refinement

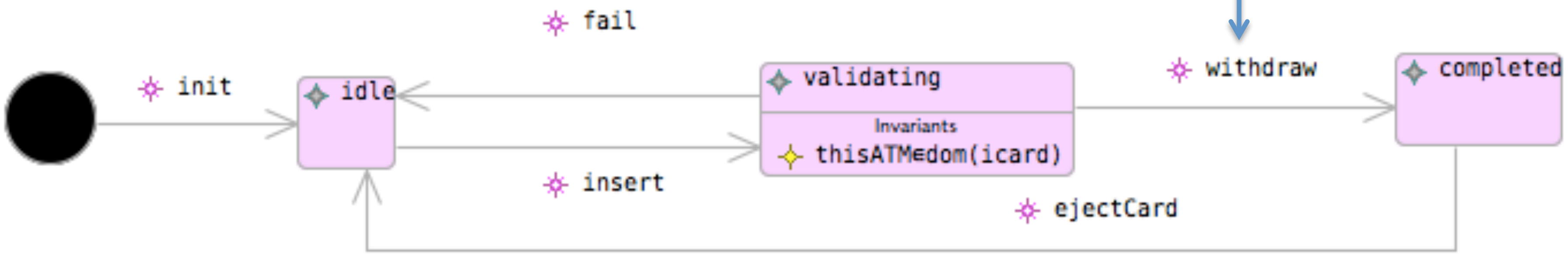


parameter: amount ∈ ℕ1

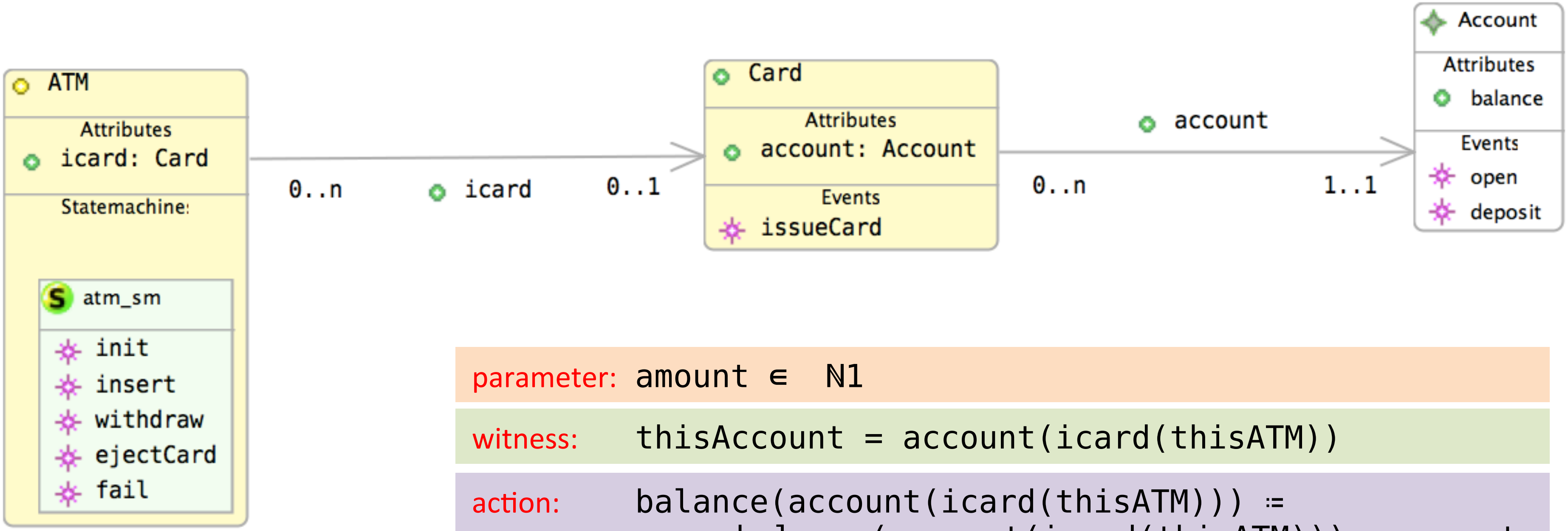
witness: thisAccount = account(icard(thisATM))

action: balance(account(icard(thisATM))) ≔ balance(account(icard(thisATM))) − amount

A witness shows the relationship with the old class instance



Example – Refinement

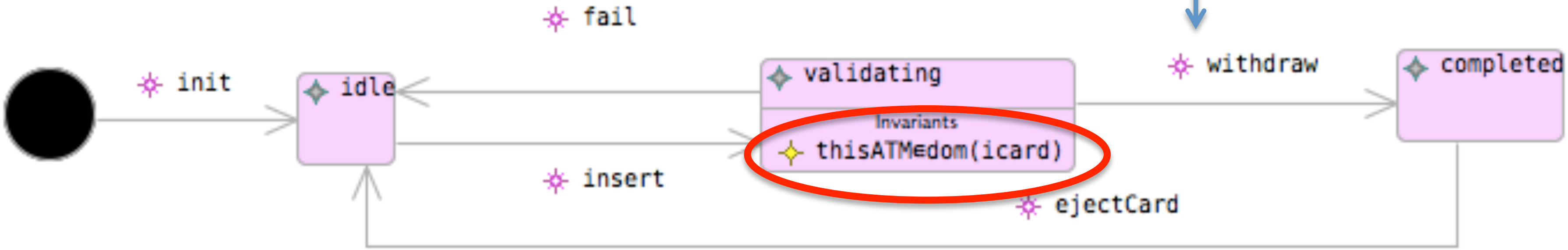


parameter: amount ∈ ℕ1

witness: thisAccount = account(icard(thisATM))

action: balance(account(icard(thisATM))) ≔ balance(account(icard(thisATM))) − amount

A state invariant is needed to help the well-definedness proof of the use of icard in withdraw (because it's a partial function)

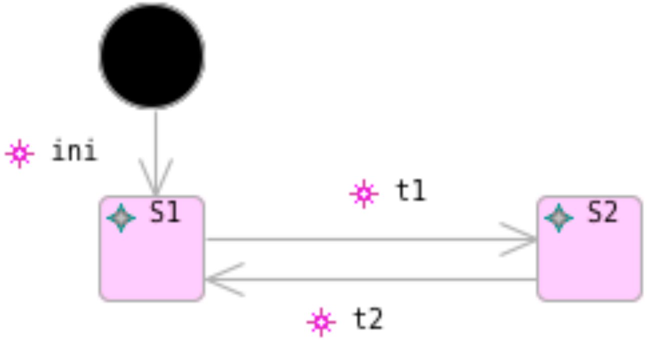


Refining Statemachines

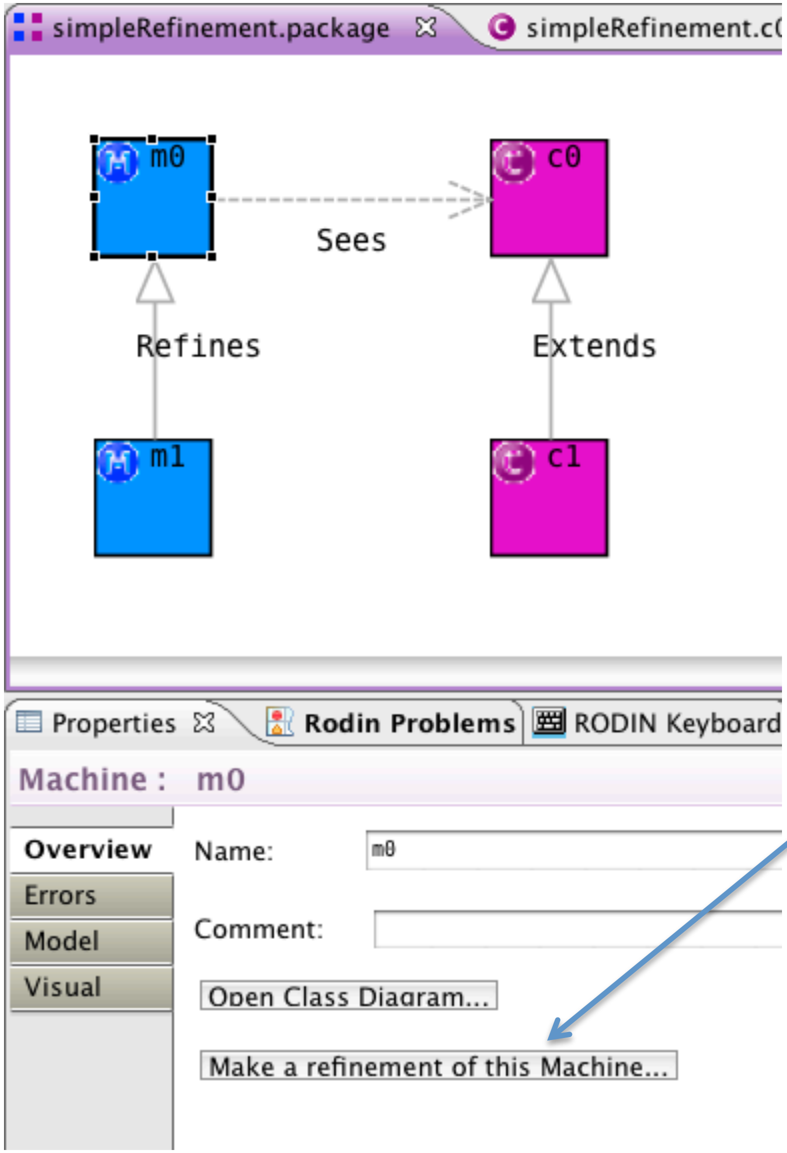
Slides show state sets translation.

Also works using state function translation

Starting from a Simple Statemachine



“Make a refinement of this Machine”



Machine : m0

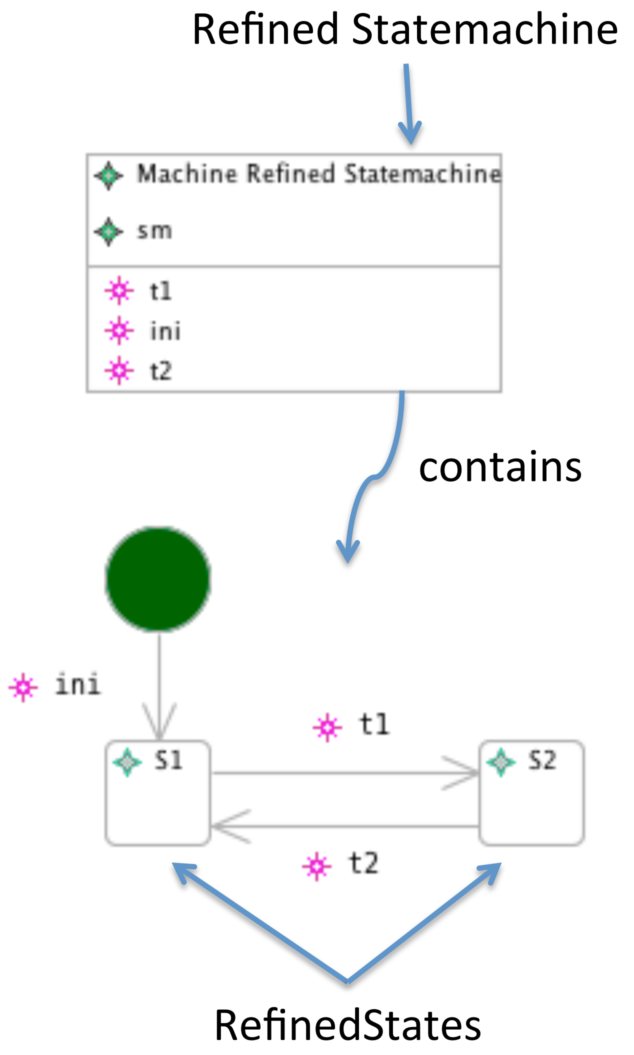
Name: m0

Comment:

Same as for Class Refinement

Select Machine,
click Button in Properties,
Makes a starting point
 for refinement

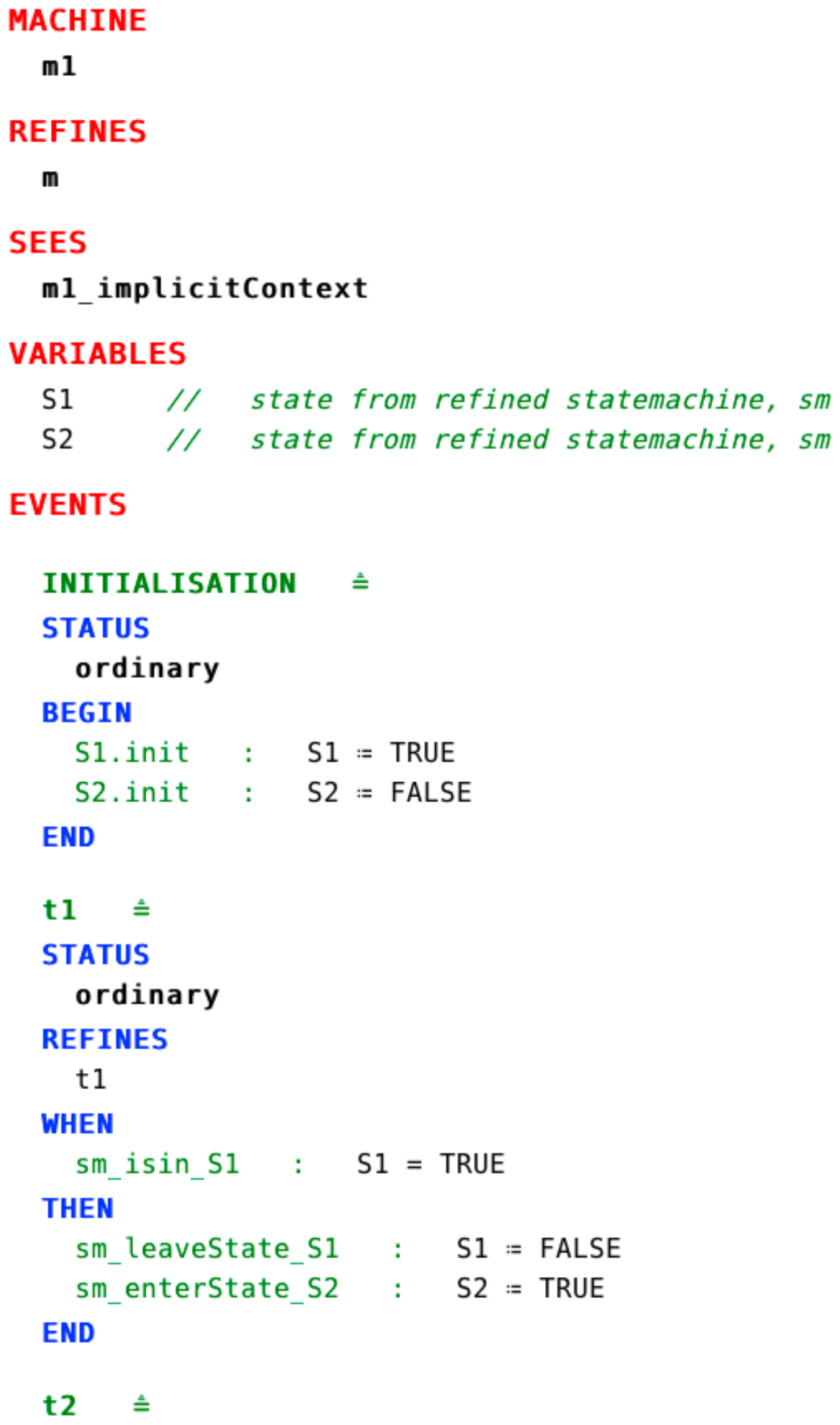
gives us..



Refined Statemachine

contains

RefinedStates



```
MACHINE
  m1
REFINES
  m
SEES
  m1_implicitContext
VARIABLES
  S1      //   state from refined statemachine, sm
  S2      //   state from refined statemachine, sm
EVENTS

  INITIALISATION    ≙
  STATUS
    ordinary
  BEGIN
    S1.init    :    S1 ≔ TRUE
    S2.init    :    S2 ≔ FALSE
  END

  t1   ≙
  STATUS
    ordinary
  REFINES
    t1
  WHEN
    sm_isin_S1    :    S1 = TRUE
  THEN
    sm_leaveState_S1    :    S1 ≔ FALSE
    sm_enterState_S2    :    S2 ≔ TRUE
  END

  t2   ≙
```

What can we do?

Refine the existing transitions

- strengthen guards
- add actions to alter any new variables
- split transitions (as long as they have same source and target state)

Can add things to a state

- Invariants
- Nested State-machines

What we can’t do (when refining statemachines)

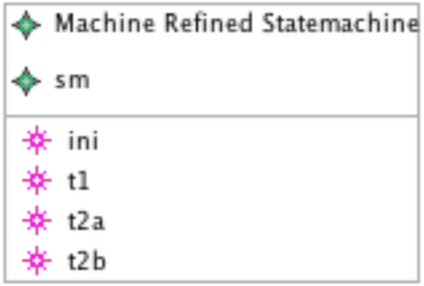
Cannot add new states

- state sets – would contradict the existing partition invariant
- state function – would alter the exisiting enumerated type

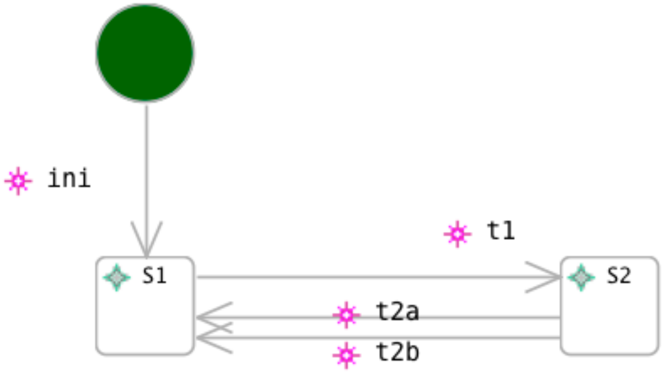
Cannot add completely new transitions

- new events must not alter old variables (e.g. state change would)

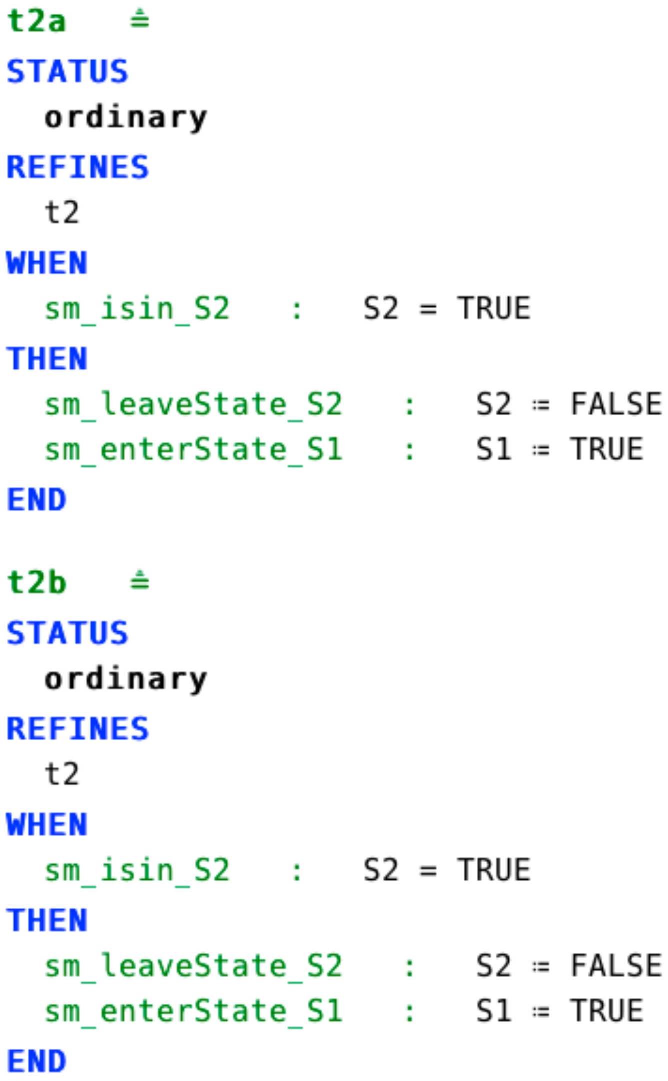
Transition Splitting – Preparing for a Nested Statemachine



Machine Refined Statemachine
- sm
  - ini
  - t1
  - t2a
  - t2b

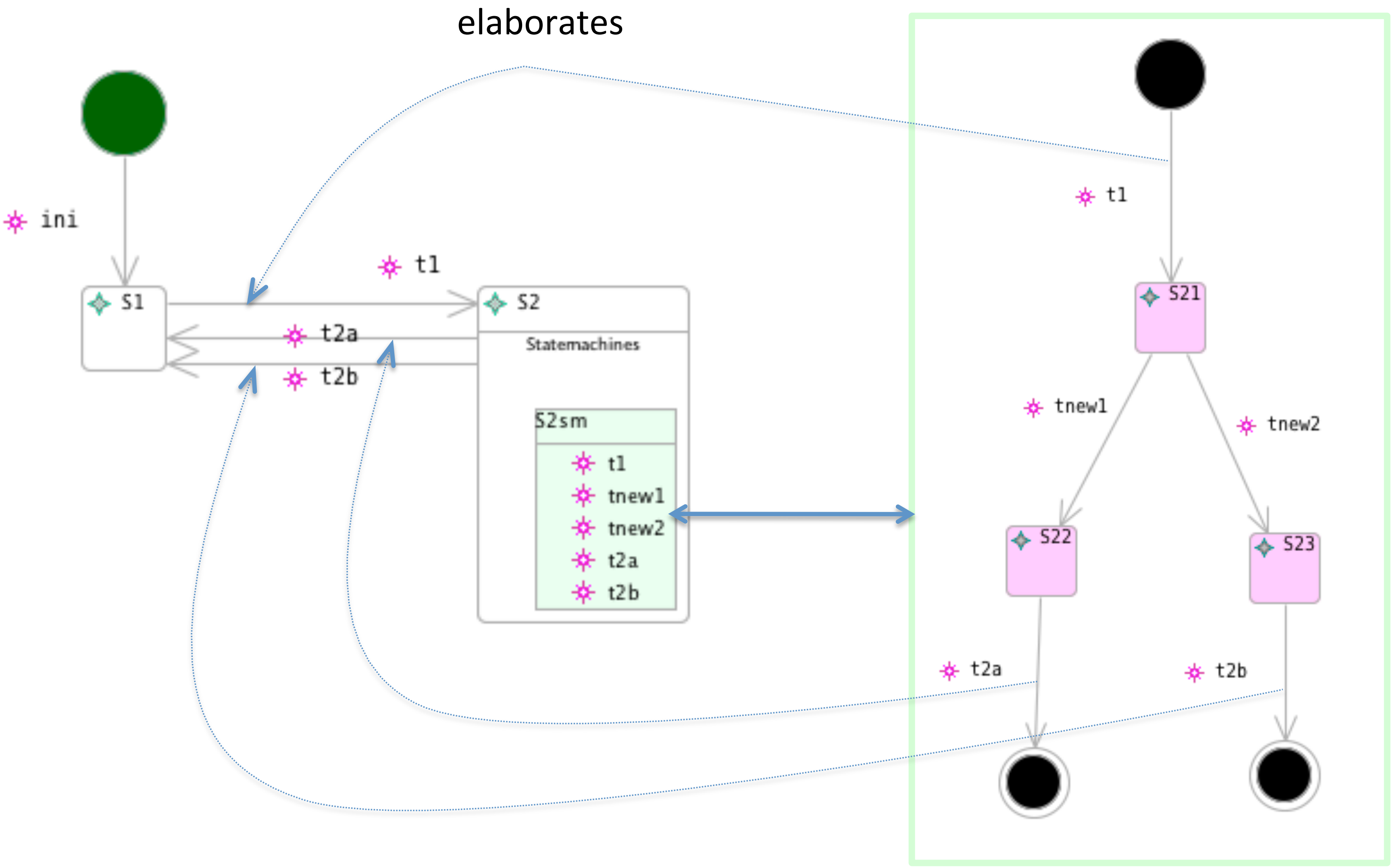


t2 has been split into 2 ‘cases’
(both refine t2)

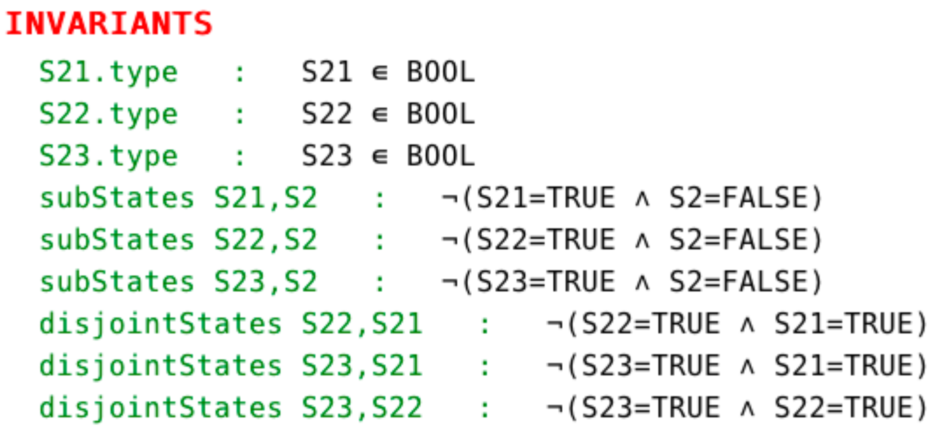


```
t2a  ≙
STATUS
  ordinary
REFINES
  t2
WHEN
  sm_isin_S2    :    S2 = TRUE
THEN
  sm_leaveState_S2    :    S2 ≔ FALSE
  sm_enterState_S1    :    S1 ≔ TRUE
END

t2b  ≙
STATUS
  ordinary
REFINES
  t2
WHEN
  sm_isin_S2    :    S2 = TRUE
THEN
  sm_leaveState_S2    :    S2 ≔ FALSE
  sm_enterState_S1    :    S1 ≔ TRUE
END
```

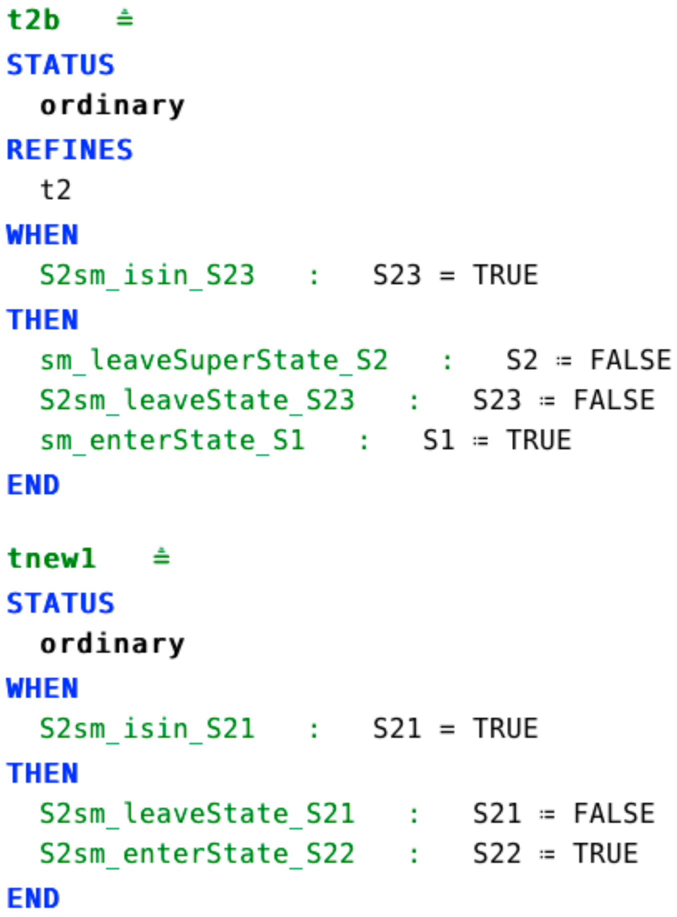
Adding a Nested Statemachine



Translation of Refinement



```
INVARIANTS
  S21.type    :    S21 ∈ BOOL
  S22.type    :    S22 ∈ BOOL
  S23.type    :    S23 ∈ BOOL
  subStates S21,S2    :    ¬(S21=TRUE ∧ S2=FALSE)
  subStates S22,S2    :    ¬(S22=TRUE ∧ S2=FALSE)
  subStates S23,S2    :    ¬(S23=TRUE ∧ S2=FALSE)
  disjointStates S22,S21    :    ¬(S22=TRUE ∧ S21=TRUE)
  disjointStates S23,S21    :    ¬(S23=TRUE ∧ S21=TRUE)
  disjointStates S23,S22    :    ¬(S23=TRUE ∧ S22=TRUE)
```



```
t2b   ≙
STATUS
  ordinary
REFINES
  t2
WHEN
  S2sm_isin_S23    :    S23 = TRUE
THEN
  sm_leaveSuperState_S2    :    S2 ≔ FALSE
  S2sm_leaveState_S23    :    S23 ≔ FALSE
  sm_enterState_S1    :    S1 ≔ TRUE
END

tnew1   ≙
STATUS
  ordinary
WHEN
  S2sm_isin_S21    :    S21 = TRUE
THEN
  S2sm_leaveState_S21    :    S21 ≔ FALSE
  S2sm_enterState_S22    :    S22 ≔ TRUE
END
```

Example – Card Validation by PIN

In the ATM example, add a refinement to explain how card validation works.

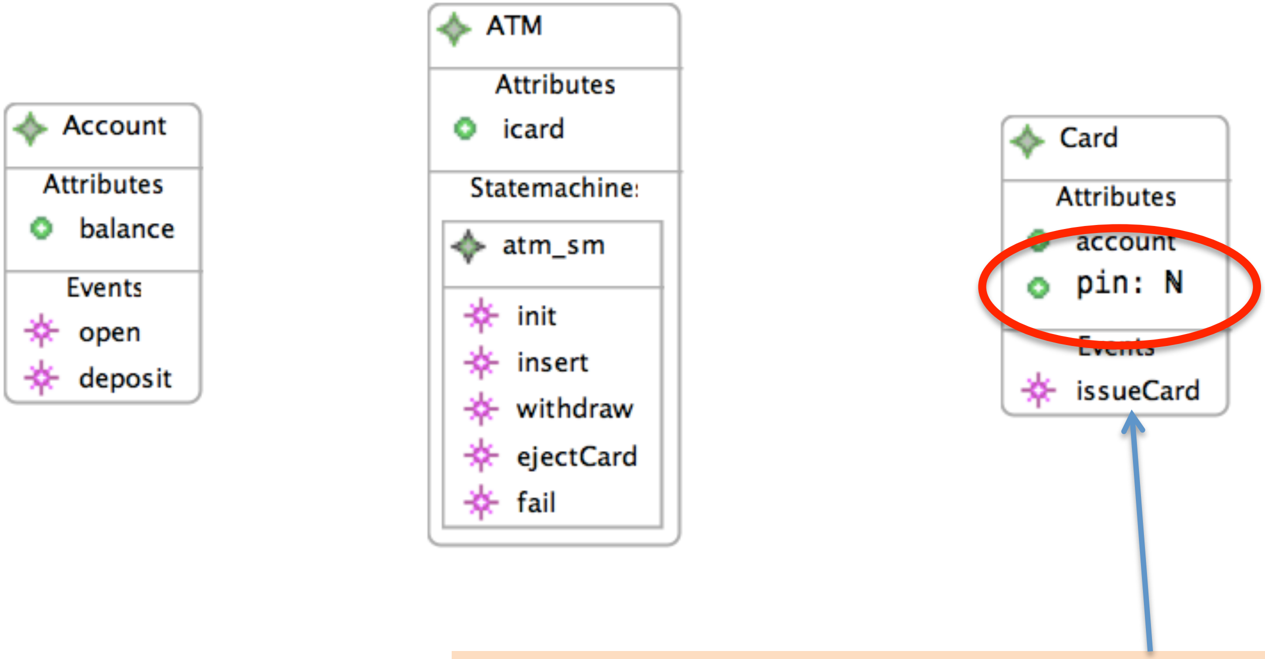
There is a PIN number associated with a card.

A number is entered at the ATM.

If the number matches the inserted cards PIN the validation succeeds.

If the number doesn't match the PIN the validation fails.

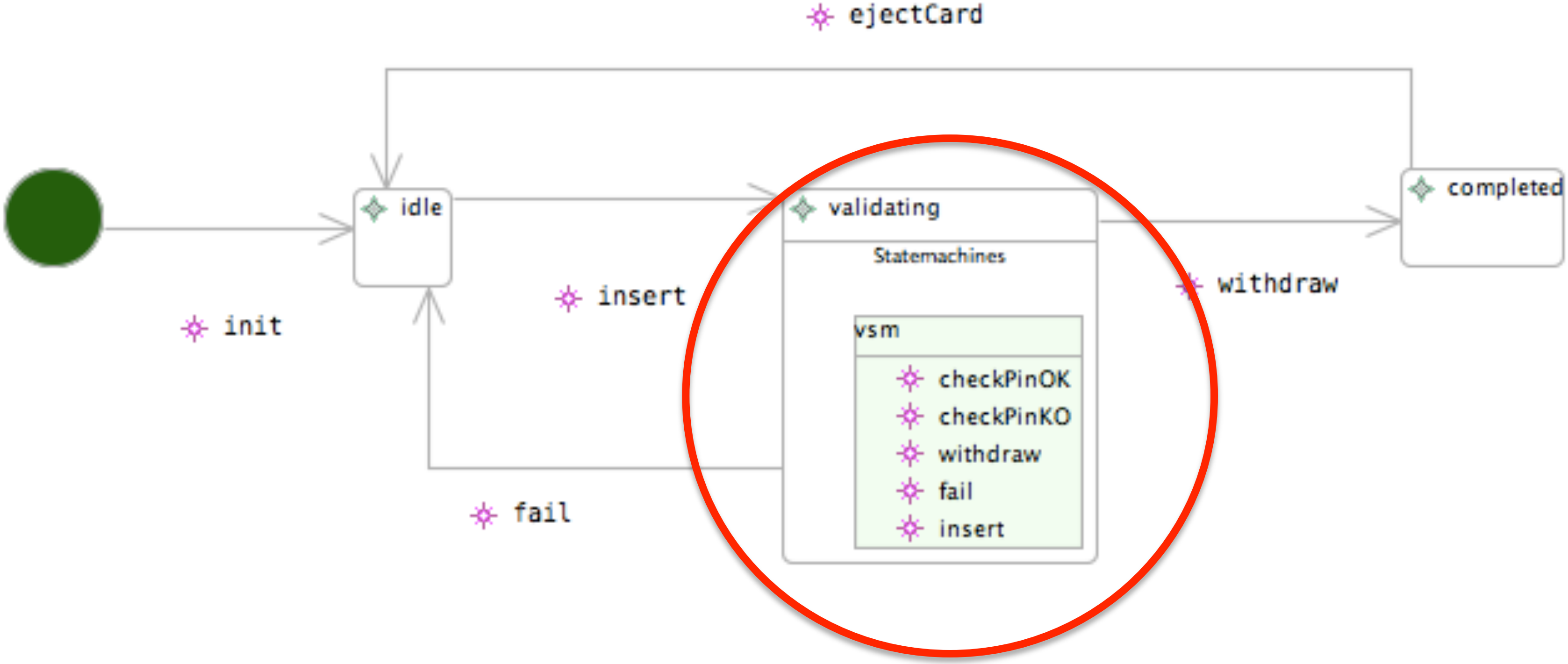
Refined Class Diagram



parameter: $n \in \mathbb{N}$

action: pin(thisCard) := n

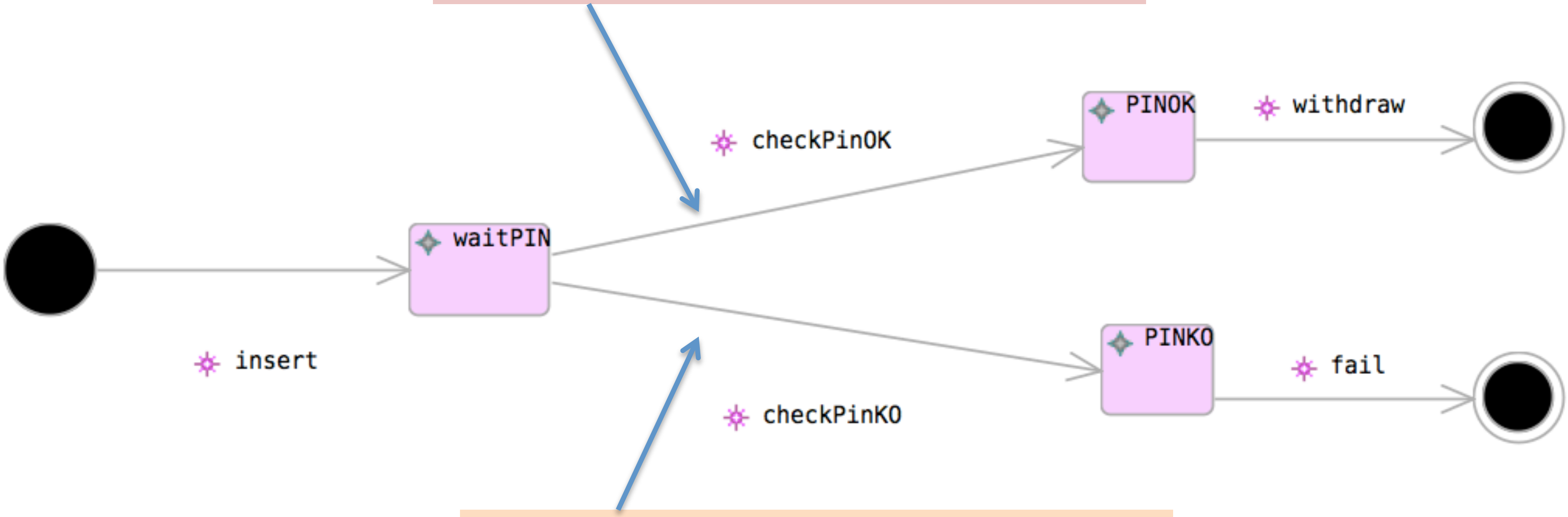
Refined Statemachine



New nested statemachine

parameter: n ∈ ℕ

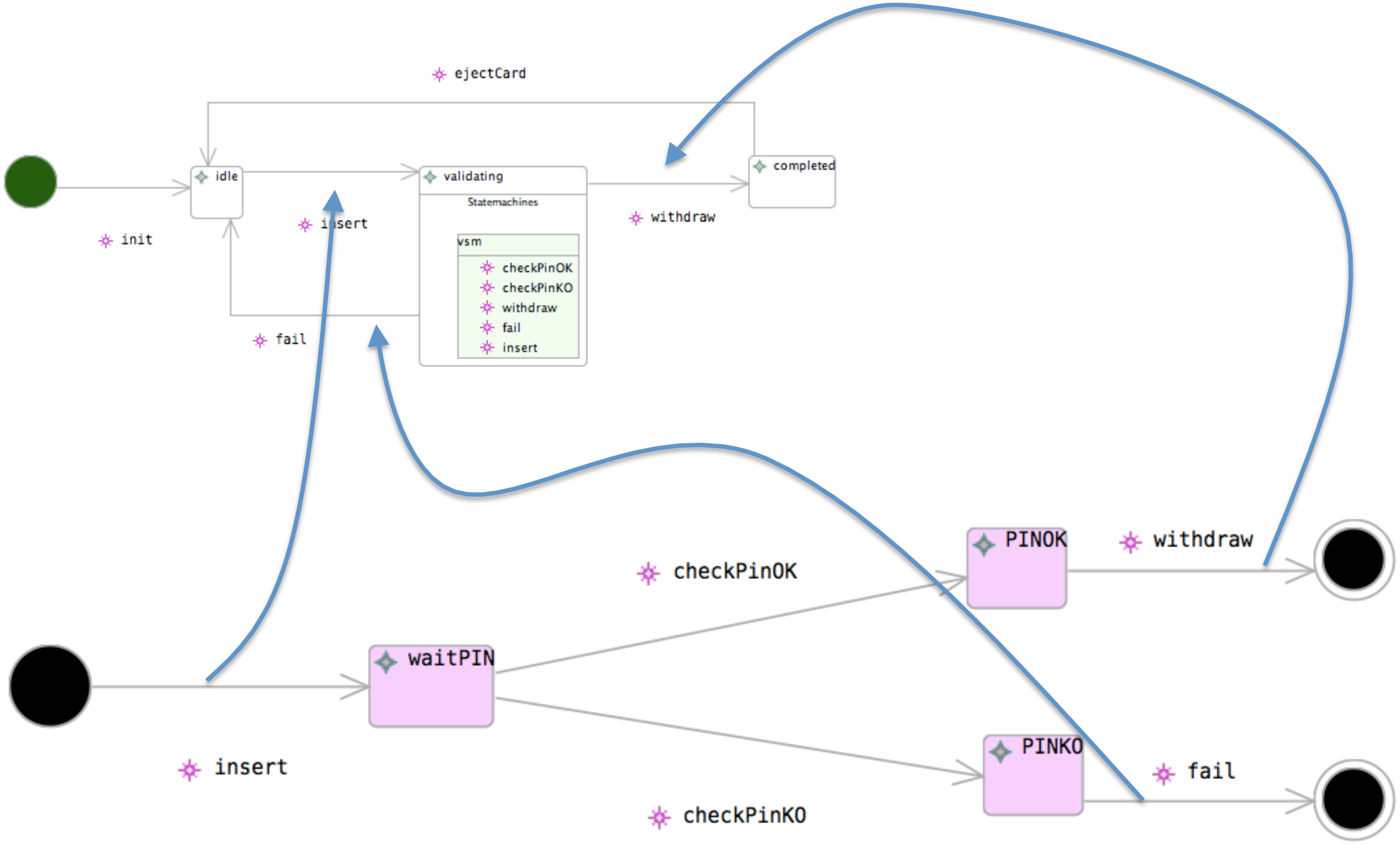
guard: n = pin(icard(thisATM))



parameter: n ∈ ℕ

guard: n ≠ pin (icard(thisATM))

New nested statemachine – transition elaboration



Summary

Extended Class Types

Refined Classes & Inherited Attributes

Moving events between classes

Statemachine refinement

- transition splitting
- nested statemachines