Invariant Contracts for Modules in Java

Andreas Roth

3rd KeYWorkshop, Königswinter, June 7th

Overview

- Why is verification currently not modular in KeY?

Overview

- Why is verification currently not modular in KeY?
- What is the goal of modular verification?

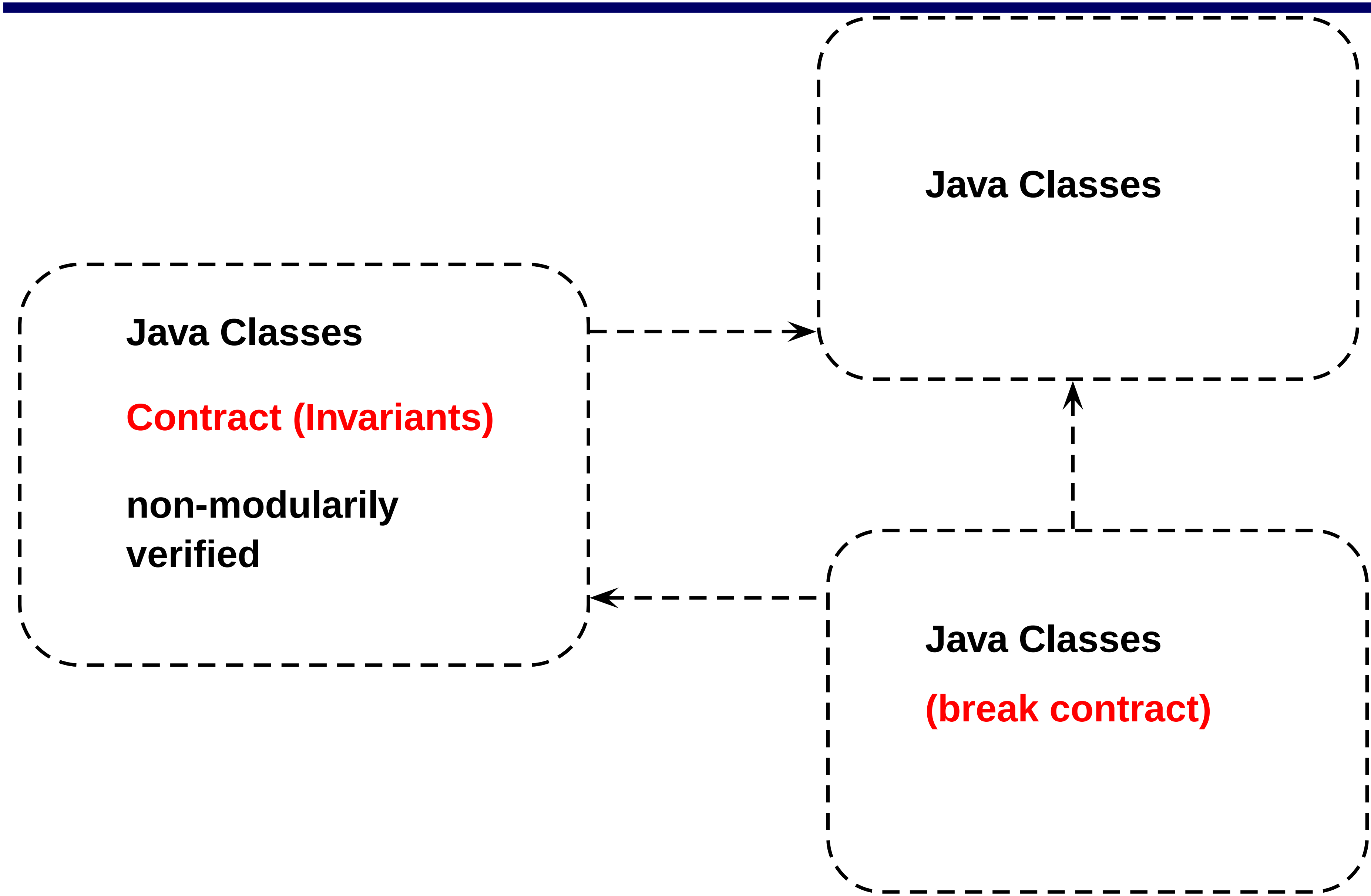
  Notions: module, module contracts, depends-clause, module-protected attributes

Overview

- Why is verification currently not modular in KeY?
- What is the goal of modular verification?

  Notions: module, module contracts, depends-clause, module-protected attributes
- How can the goal be achieved with KeY? What changes in KeY?

The Problem



Java Classes
Java Classes
Contract (Invariants)
non-modularily
verified
Java Classes
(break contract)

The Problem

```
public class Date {
  int month, year;
  //. . .
}
```

```
self.start.year<self.end.year or
(self.start.year=self.end.year and
self.start.month<=self.end.month)

public class Period {
  Date start, end;
  //. . .
}
```

```
public class Main {
  public Period myPeriod(){
    Date jan=new Date(1,2004);
    Date sep=new Date(9,2004);
    Period p=new Period(jan, sep);
    jan.setYear(2005);
    return p;   } }
```

Some Observations and Questions

- **Recent Approaches**: Existing techniques like Ownership Types are imposed on the problem. Problem solved by restricting allowed programs. Is the restriction minimal? Is Ownership needed?

Some Observations and Questions

- **Recent Approaches**: Existing techniques like Ownership Types are imposed on the problem. Problem solved by restricting allowed programs. Is the restriction minimal? Is Ownership needed?
- **Preferred Approach**:

Some Observations and Questions

- **Recent Approaches**: Existing techniques like Ownership Types are imposed on the problem. Problem solved by restricting allowed programs. Is the restriction minimal? Is Ownership needed?
- **Preferred Approach**:
  1. Make requirements explicit: Modules, Contracts, local and global (modular) Correctness.

Some Observations and Questions

- **Recent Approaches**: Existing techniques like Ownership Types are imposed on the problem. Problem solved by restricting allowed programs. Is the restriction minimal? Is Ownership needed?
- **Preferred Approach**:
  1. Make requirements explicit: Modules, Contracts, local and global (modular) Correctness.
  2. Define abstract theoretical criterion which satisfies requirements.

Some Observations and Questions

- **Recent Approaches**: Existing techniques like Ownership Types are imposed on the problem. Problem solved by restricting allowed programs. Is the restriction minimal? Is Ownership needed?
- **Preferred Approach**:
  1. Make requirements explicit: Modules, Contracts, local and global (modular) Correctness.
  2. Define abstract theoretical criterion which satisfies requirements.
  3. Find (efficient) methods to fulfil criterion.

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Module contract of m (simplified): Invariant contracts for classes C_m.

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Module contract of m (simplified): Invariant contracts for classes C_m.

Module contract ct_m fulfilled . . .

- in a set of classes T iff for all methods p of T:
  p preserves the invariants of ct_m.

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Module contract of m (simplified): Invariant contracts for classes C_m.

Module contract ct_m **fulfilled ...**

- in a set of classes T iff for all methods p of T:
  p preserves the invariants of ct_m.
- locally iff fulfilled in C_m.

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Module contract of m (simplified): Invariant contracts for classes C_m.

Module contract ct_m fulfilled . . .

- in a set of classes T iff for all methods p of T: p preserves the invariants of ct_m.
- locally iff fulfilled in C_m.

Goal: for all T

Explicit Notion of Modules and Their Contracts

Definition (Module): Given classes C_m and E_m with $E_m \subseteq C_m$.

$(C_m, E_m, \emptyset)$ is a module. If I_m are modules, then (C_m, E_m, I_m) is a module.

All modules m, m' must satisfy: for all usages of types c' of m' in m:

$$m' \in I_m \text{ and } c' \in E_{m'}.$$

Module contract of m (simplified): Invariant contracts for classes C_m.

Module contract ct_m **fulfilled . . .**

- in a set of classes T iff for all methods p of T: Goal: for all T

  p preserves the invariants of ct_m.
- locally iff fulfilled in C_m. Mastered (KeY)

Depends Clauses

Notion for the analysis of invariant contracts:

Definition (Depends Clause): Set D_ϕ of attribute chains $a_1.\cdots.a_n$.

Depends Clauses

Notion for the analysis of invariant contracts:

Definition (Depends Clause): Set D_ϕ of attribute chains $a_1.\cdots.a_n$. For all states s_1 and s_2:

For all $d \in D_\phi$, objects e of appropriate type $e.d_{s_1} = e.d_{s_2}$

implies

$$(s_1 \models \phi \text{ iff } s_2 \models \phi)$$

Depends Clauses

Notion for the analysis of invariant contracts:

Definition (Depends Clause): Set D_ϕ of attribute chains $a_1.\cdots.a_n$. For all states s_1 and s_2:

$$\text{For all } d \in D_\phi\text{, objects } e \text{ of appropriate type } e.d_{s_1} = e.d_{s_2}$$

$$\text{implies}$$

$$\left(s_1 \models \phi \text{ iff } s_2 \models \phi\right)$$

In the example:

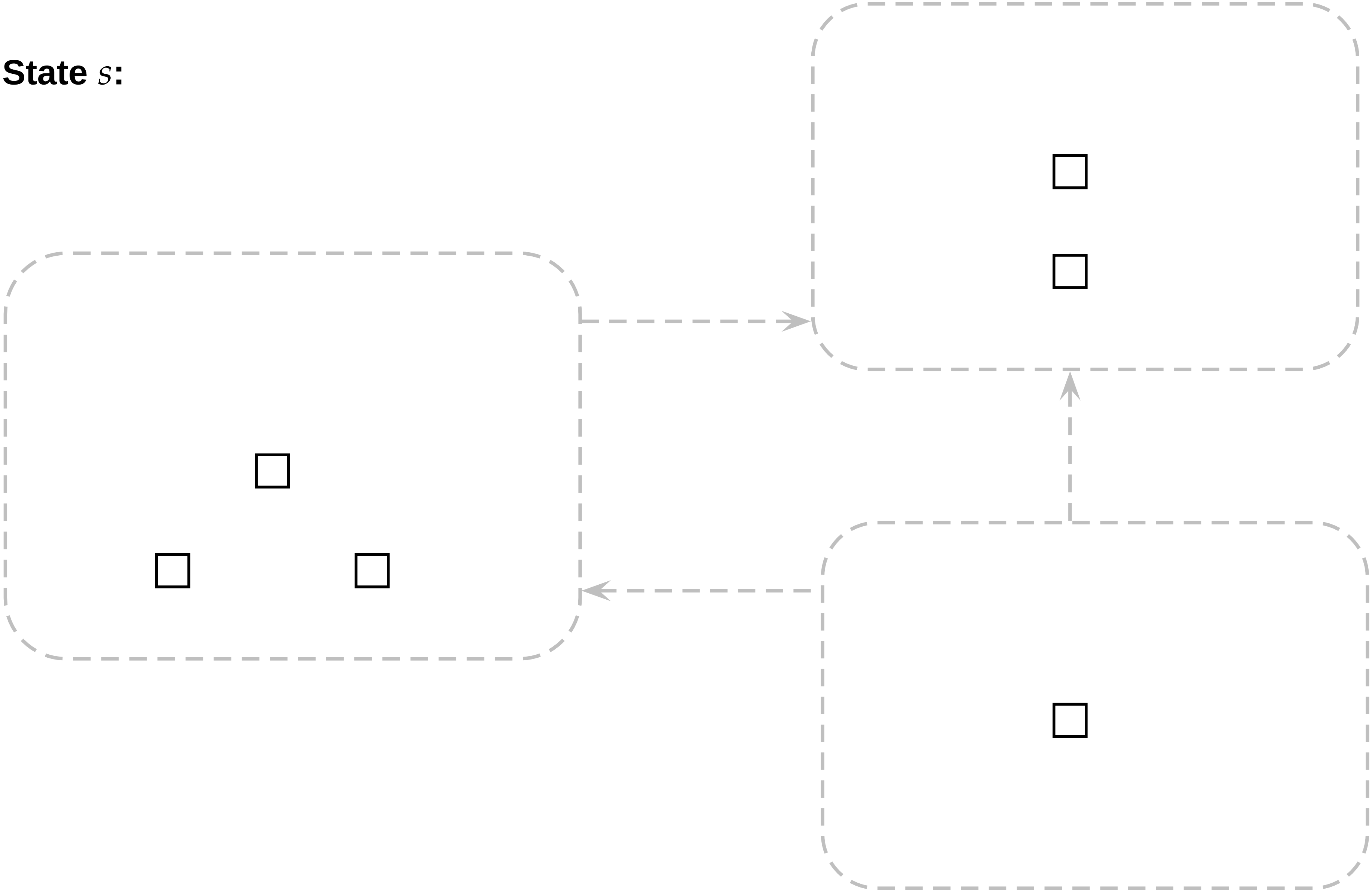
self.start.year$<$self.end.year or
(self.start.year=self.end.year and
self.start.month$<$=self.end.month)

depends on

{ start.year, end.year,
start.month, end.month }

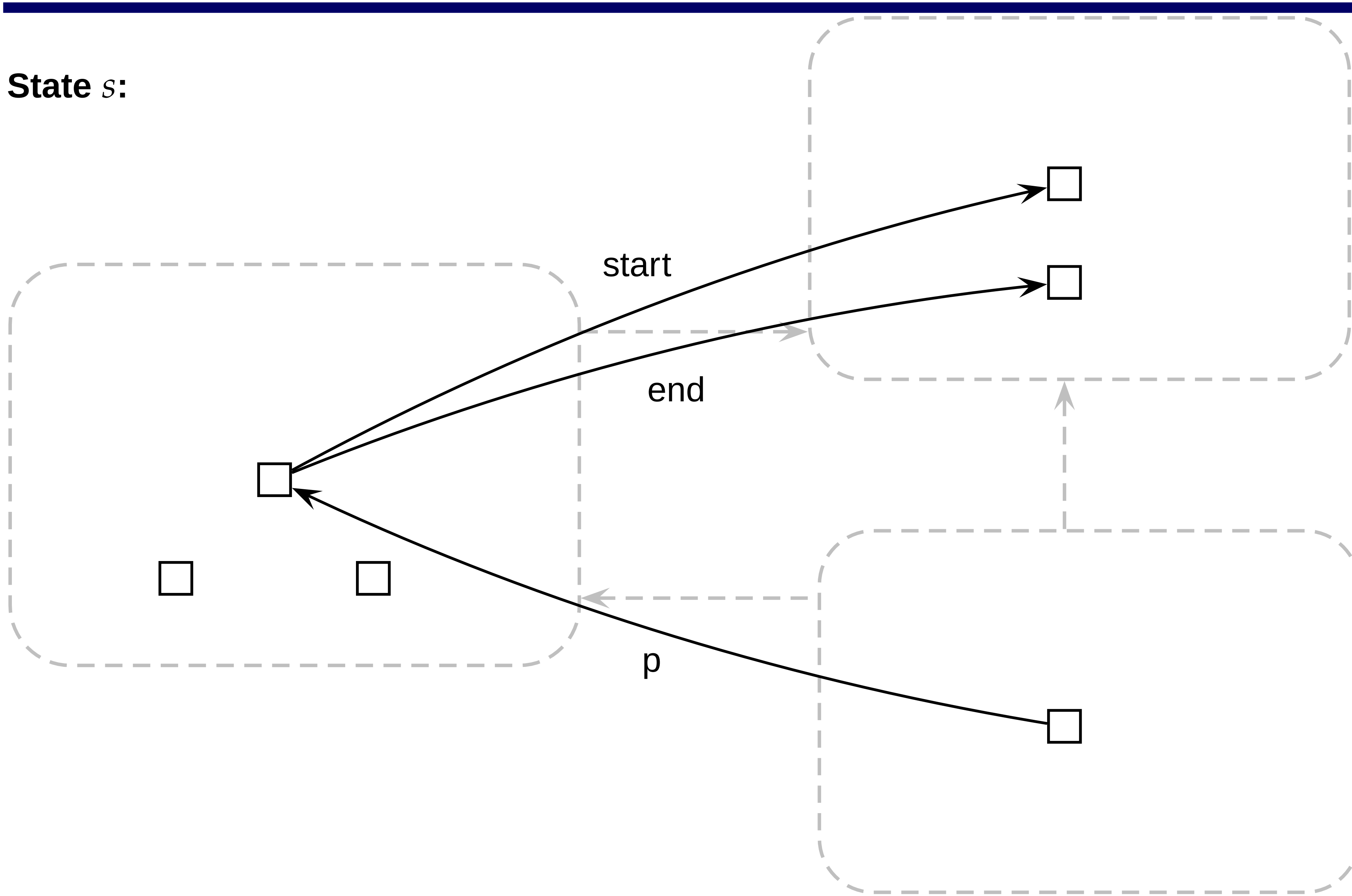
Module-Protection I

State s:



Module-Protection I

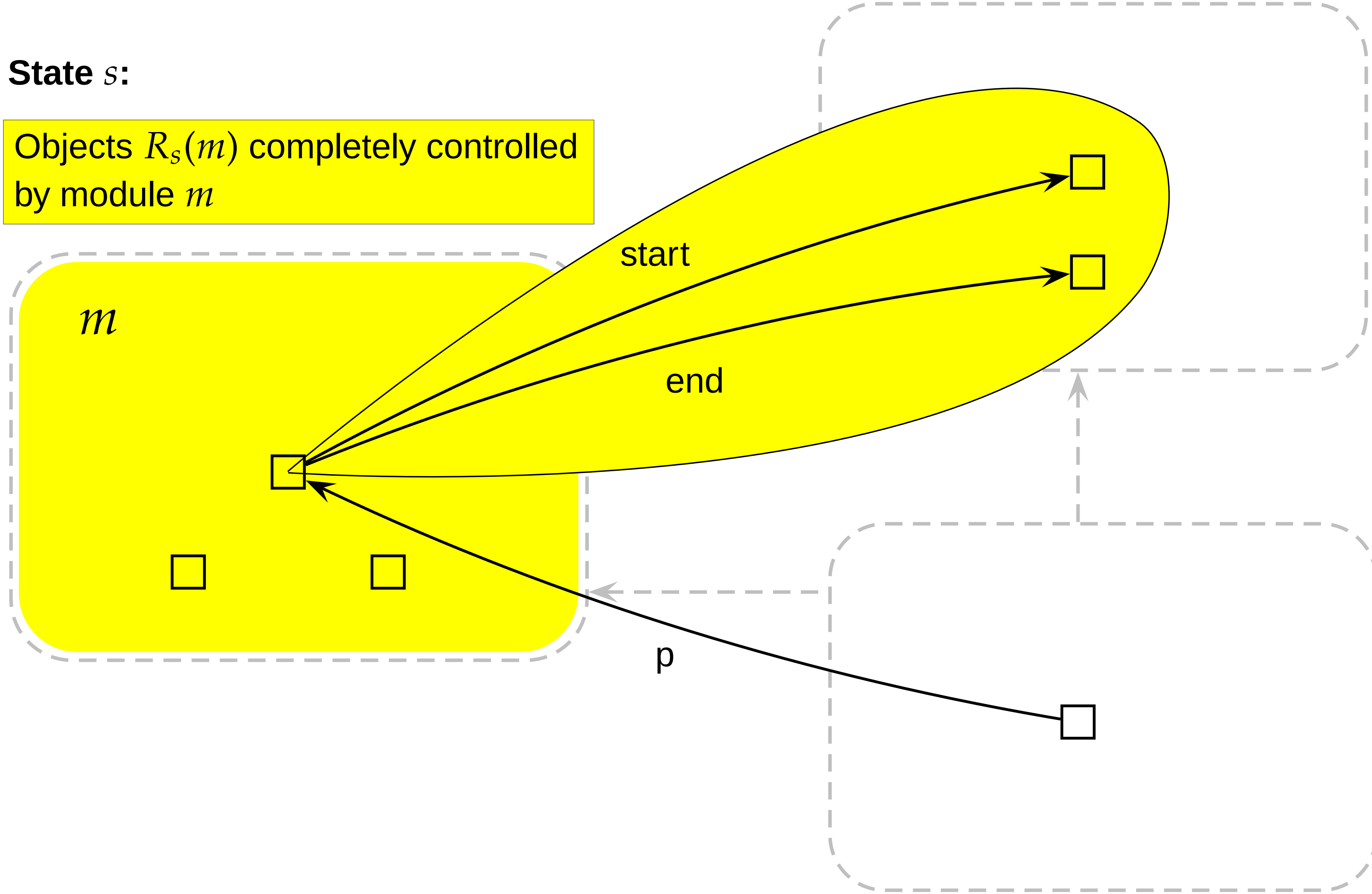
State s:



Module-Protection I

State s:

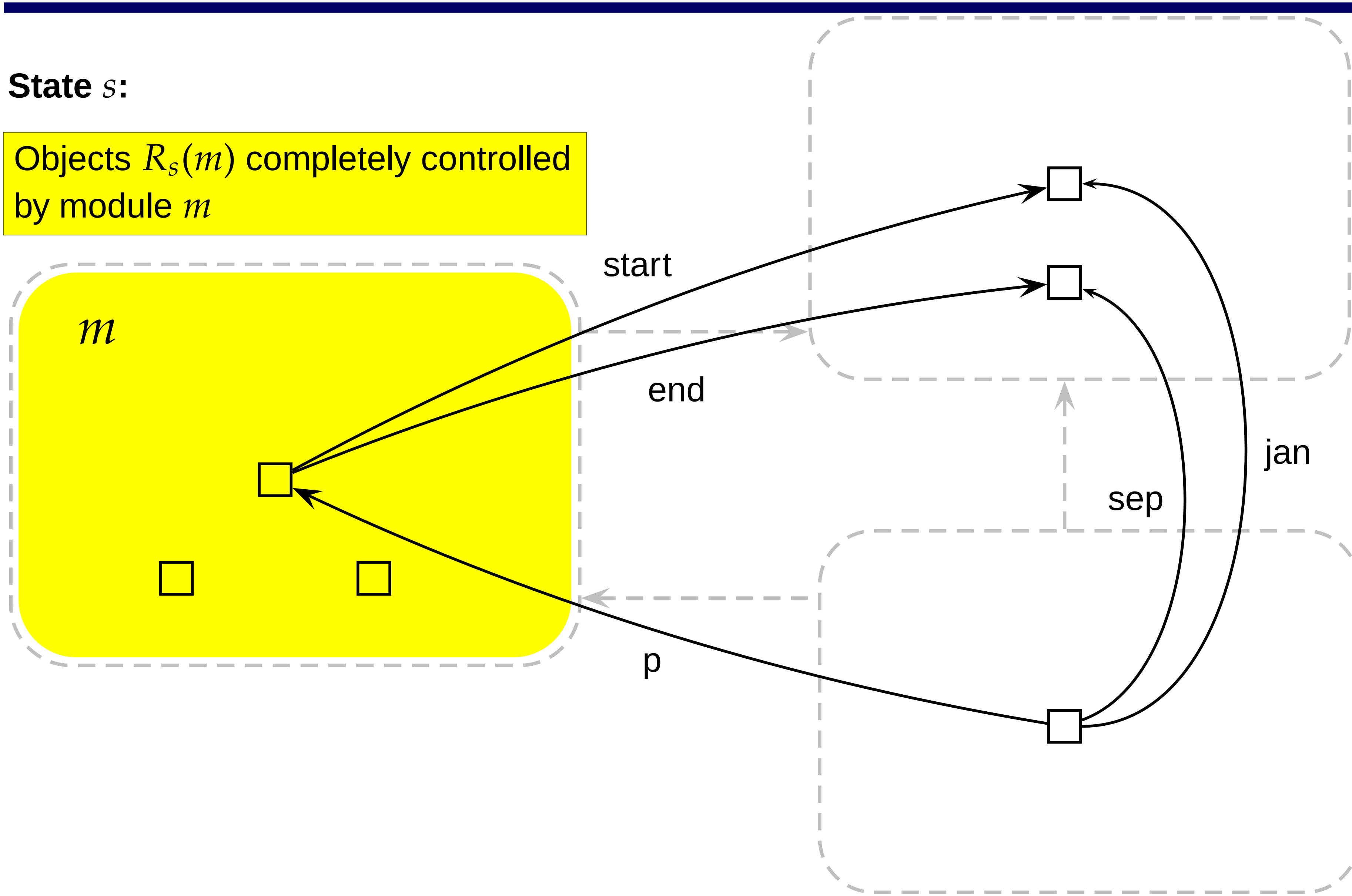
Objects $R_s(m)$ completely controlled by module m



Module-Protection I

State s:

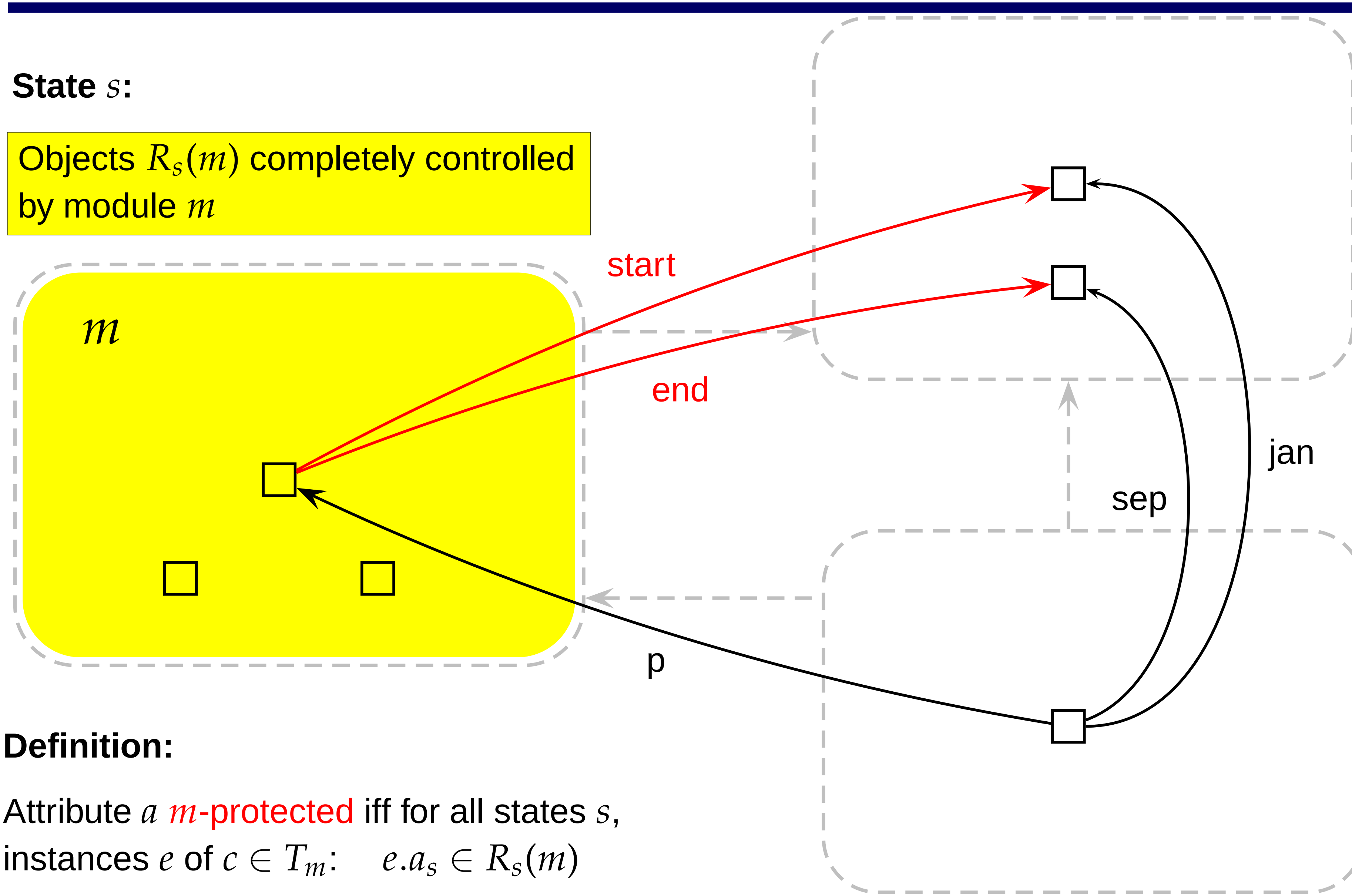
Objects $R_s(m)$ completely controlled by module m



Module-Protection I

State s:

Objects $R_s(m)$ completely controlled by module m



Definition:

Attribute a m-protected iff for all states s, instances e of $c \in T_m$: $\quad e.a_s \in R_s(m)$

Module-Protection II

Consider now only private attributes!

Theorem:

Module m, module contract ct_m fulfilled locally, D union of depends clauses of invariants from ct_m, T classes of modules that (transitively) import m.

If for all $a_1. \cdots .a_n \in D$, $n = 1$

Then: ct_m fulfilled in T.

Module-Protection II

Consider now only private attributes!

Theorem:

Module m, module contract ct_m fulfilled locally, D union of depends clauses of invariants from ct_m, T classes of modules that (transitively) import m.

If for all $a_1 . \cdots . a_n \in D$, $n = 1$

Then: ct_m fulfilled in T.

Module-Protection II

Consider now only private attributes!

Theorem:

Module m, module contract ct_m fulfilled locally, D union of depends clauses of invariants from ct_m, T classes of modules that (transitively) import m.

If for all $a_1.\cdots.a_n \in D$, $n = 1$ or for all $i = 1, \ldots, n-1$:

- a_i defined in m, m-protected

Then: ct_m fulfilled in T.

Module-Protection II

Consider now only private attributes!

Theorem:

Module m, module contract ct_m fulfilled locally, D union of depends clauses of invariants from ct_m, T classes of modules that (transitively) import m.

If for all $a_1.\cdots.a_n \in D$, $n = 1$ or for all $i = 1, \ldots, n-1$:

- a_i defined in m, m-protected , or
- a_i defined in $m' \neq m$, strictly m'-protected

Then: ct_m fulfilled in T.

Establishing Module-Protection

So far: Theoretical Criterion

Needed: Method to prove module-protection

No ideal solution (yet). But:

- Known patterns to solve problems are subsumed.

  (Partial) Immutability by final attributes, unique pointers, ownership (by type systems), confined types

- Proof by DL proof obligation possible (?)

(Possible / Needed) Extensions

- Protection of whole attribute chains instead of single attributes.

(Possible / Needed) Extensions

- Protection of whole attribute chains instead of single attributes.
- Treatment of subtypes and protected attributes. Introduction of subtype contracts.

(Possible / Needed) Extensions

- Protection of whole attribute chains instead of single attributes.
- Treatment of subtypes and protected attributes. Introduction of subtype contracts.
- Treatment of arrays.

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of invariant $\phi_{\texttt{C}}$ of class `C`

$$\phi_{\texttt{C}} \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \phi_{\texttt{C}}$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of invariant $\phi_{\mathtt{C}}$ of class `C`

$$\phi_{\mathtt{C}} \land pre_m \rightarrow \langle \mathtt{self.C::m(p)} \rangle \phi_{\mathtt{C}}$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of invariant $\phi_{\texttt{C}}$ of class `C`

$$\forall \texttt{self:C}\ .\phi_{\texttt{C}} \land pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \forall \texttt{self:C}\ .\phi_{\texttt{C}}$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of invariant $\phi_{\texttt{C}}$ of class `C`

$$\forall \texttt{self:C} \,.\phi_{\texttt{C}} \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \forall \texttt{self:C} \,.\phi_{\texttt{C}}$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of all invariants $\phi_{c'}$ of all classes c'

$$\Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self:} c' \,.\, \phi_{c'}\Big) \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self:} c' \,.\, \phi_{c'}\Big)$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

Preservation of all invariants $\phi_{c'}$ of all classes c'

$$\Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self}{:}c' \,.\, \phi_{c'} \Big) \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self}{:}c' \,.\, \phi_{c'} \Big)$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account
Preservation only shown for methods of `C`.	Show for methods of all classes

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification.

For all methods: Preservation of all invariants $\phi_{c'}$ of all classes c'

$$\Big(\bigwedge_{\text{all classes } c'} \forall \mathtt{self}{:}c' \,.\phi_{c'} \Big) \land pre_m \rightarrow \langle \mathtt{self.C::m(p)} \rangle \Big(\bigwedge_{\text{all classes } c'} \forall \mathtt{self}{:}c' \,.\phi_{c'} \Big)$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account
Preservation only shown for methods of `C`.	Show for methods of all classes

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification. **Now:** Modular verification

For all methods: Preservation of all invariants $\phi_{c'}$ of all classes c'

$$\Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self:}c' \,.\phi_{c'} \Big) \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \Big(\bigwedge_{\text{all classes } c'} \forall \texttt{self:}c' \,.\phi_{c'} \Big)$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account
Preservation only shown for methods of `C`.	Show for methods of all classes

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification. **Now:** Modular verification

For all methods: Preservation of all invariants $\phi_{c'}$ of all classes c' of module

$$\Big(\bigwedge_{\substack{\text{all classes } c' \\ \text{of module}}} \forall \texttt{self:} c' \, .\phi_{c'} \Big) \wedge pre_m \rightarrow \langle \texttt{self.C::m(p)} \rangle \Big(\bigwedge_{\substack{\text{all classes } c' \\ \text{of module}}} \forall \texttt{self:} c' \, .\phi_{c'} \Big)$$

Problems	Changes
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account
Preservation only shown for methods of `C`.	Show for methods of all classes

Implications for KeY Proof Obligations

Currently in KeY: Non-modular verification. **Now:** Modular verification

For all methods: Preservation of all invariants $\phi_{c'}$ of all classes c' of module

$$\Big(\bigwedge_{\substack{\text{all classes } c' \\ \text{of module}}} \forall \mathtt{self}{:}c'.\phi_{c'}\Big) \land pre_m \rightarrow \langle \mathtt{self.C::m(p)} \rangle \Big(\bigwedge_{\substack{\text{all classes } c' \\ \text{of module}}} \forall \mathtt{self}{:}c'.\phi_{c'}\Big)$$

Problems	**Changes**
Preservation only shown for `self`.	Consider other `C` objects.
Only `C`'s invariant	Take all other classes into account
Preservation only shown for methods of `C`.	Show for methods of all classes
	Show module protectedness

Conclusions

- Explicit notion for modules in Java

Conclusions

- Explicit notion for modules in Java
- Definition of local and modular correctness of invariants

Conclusions

- Explicit notion for modules in Java
- Definition of local and modular correctness of invariants
- Criterion for modular correctness as conditions on attributes

Conclusions

- Explicit notion for modules in Java
- Definition of local and modular correctness of invariants
- Criterion for modular correctness as conditions on attributes
- Criterion in the line of current approaches for alias control

Conclusions

- Explicit notion for modules in Java
- Definition of local and modular correctness of invariants
- Criterion for modular correctness as conditions on attributes
- Criterion in the line of current approaches for alias control
- KeY proof obligations will change for modular proofs