Managing Inconsistencies in Software Design Models using Artificial Intelligence Techniques

Jorge Pinna Puissant

May 2009

EVOLUMONS
Université de Mons
www.evolumons.be

Research Field

Introduction (1)

- **Use of Models:**
  - **Raises** the level of abstraction and **hides** the accidental complexity.
  - Opens new possibilities for **creating**, **analysing**, **manipulating** and **formally reasoning** about **systems**.
- **Software Evolves**
  - Software used in the real-world must be **continuously evolved** and **adapted**.
  - If they don't evolve, they become **obsolete**, due to changes in the operational environment or user requirements.

Research Field

Introduction (2)

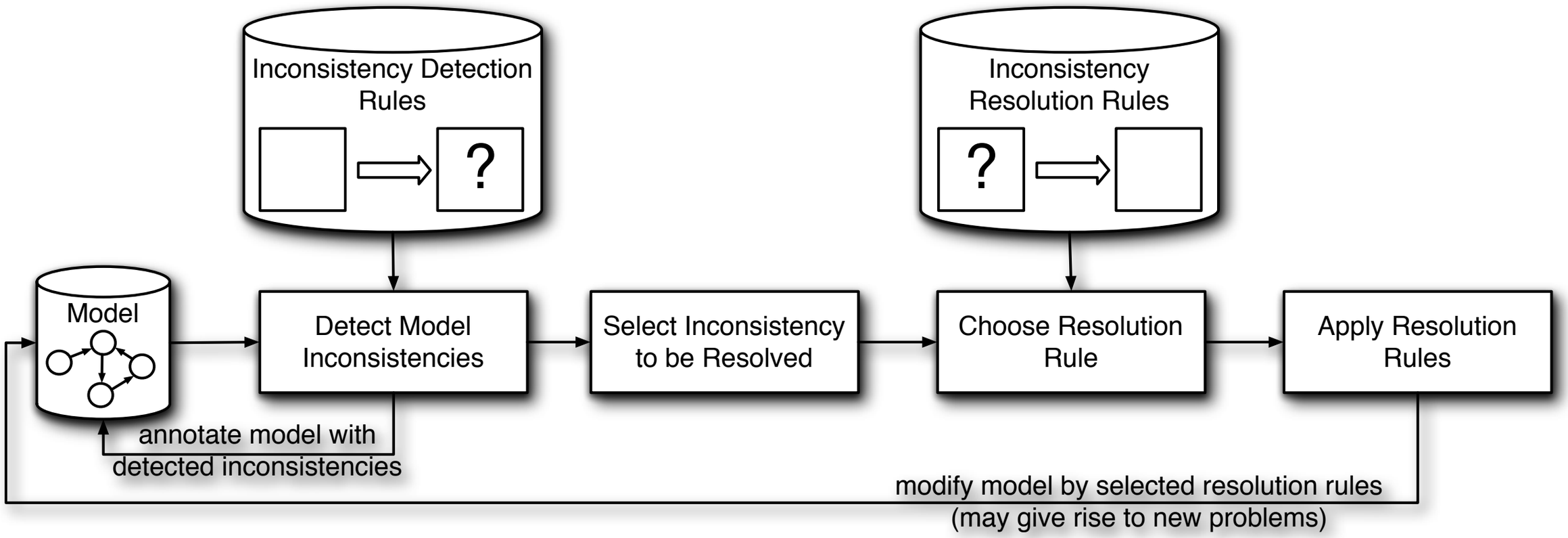
- **Quality Criteria**
  - All Software systems need to satisfy **quality criteria** related to performance, correctness, security, safety, reliability, soundness and completeness.
- **Models Evolve**
  - Support for software evolution at the level of models is crucial.
  - We aim to explore the **interaction** between model **evolution**, model **quality** and model **consistency**.

Research Field

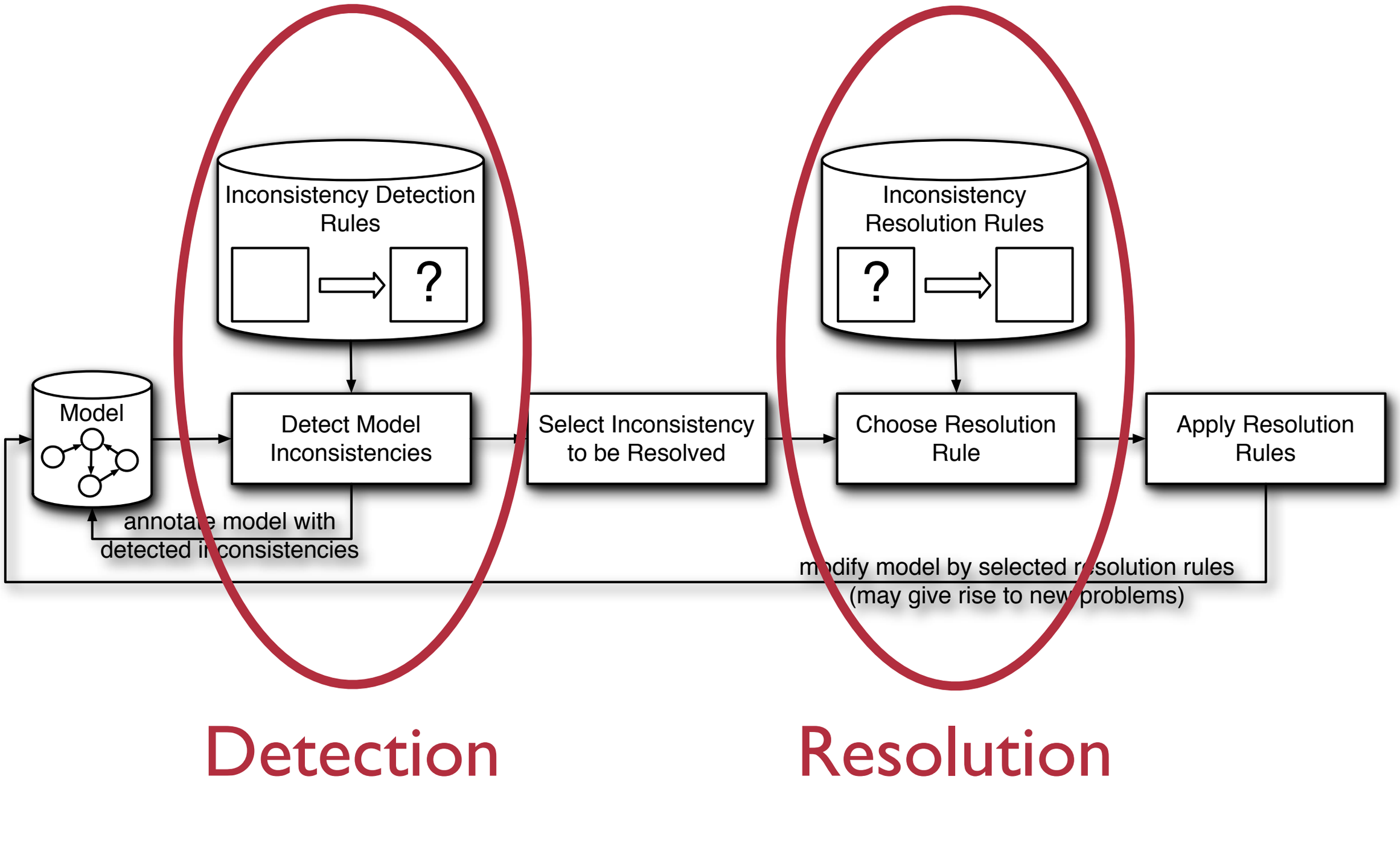
Introduction (3)

- **Model Inconsistency**
  - In a model-driven development approach, **inconsistencies inevitably arise.**
  - Need to **formally define** the various types of model **inconsistencies** in a uniform **framework**.
  - **Techniques** to **detect** and **resolve** model inconsistencies are **required**.
- **Logic-based Formalism**
  - A formal method has to be **hidden** (black box) and **lightweight** (fast)
  - **Various formalisms** may be suited to specify and reason about model **quality** and model **consistency** in the context of evolution.
  - **Logic** is a good choice if you want to **represent models** and **reason about the behaviour** of a software.

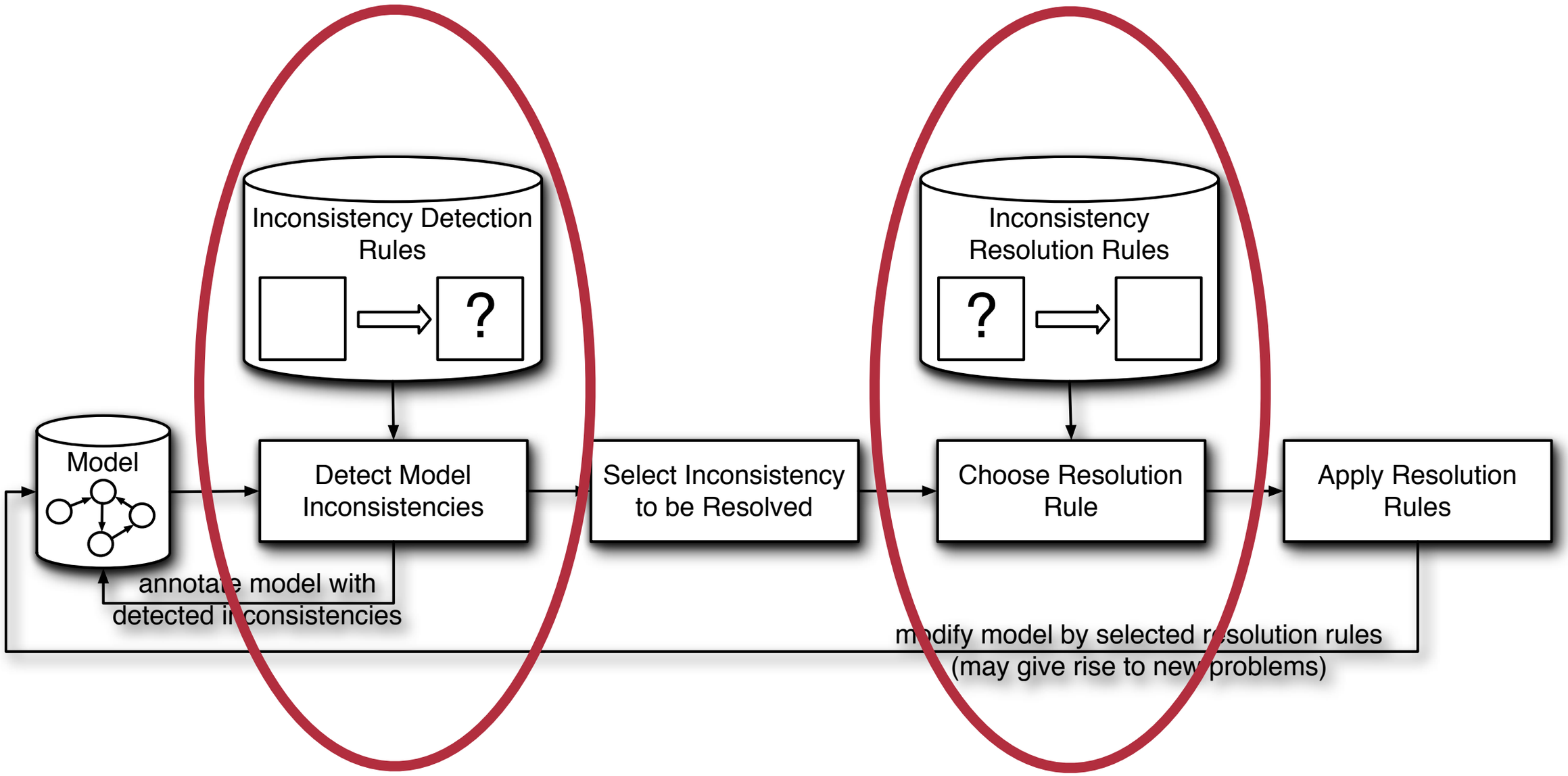
Model Inconsistency Management



Model Inconsistency Management



Model Inconsistency Management



Rule-based System

Automated Planning and Scheduling

Rule-based System

- A **rule-based** system is a **declarative** program that uses **rules** to derive **conclusions** from **facts**.

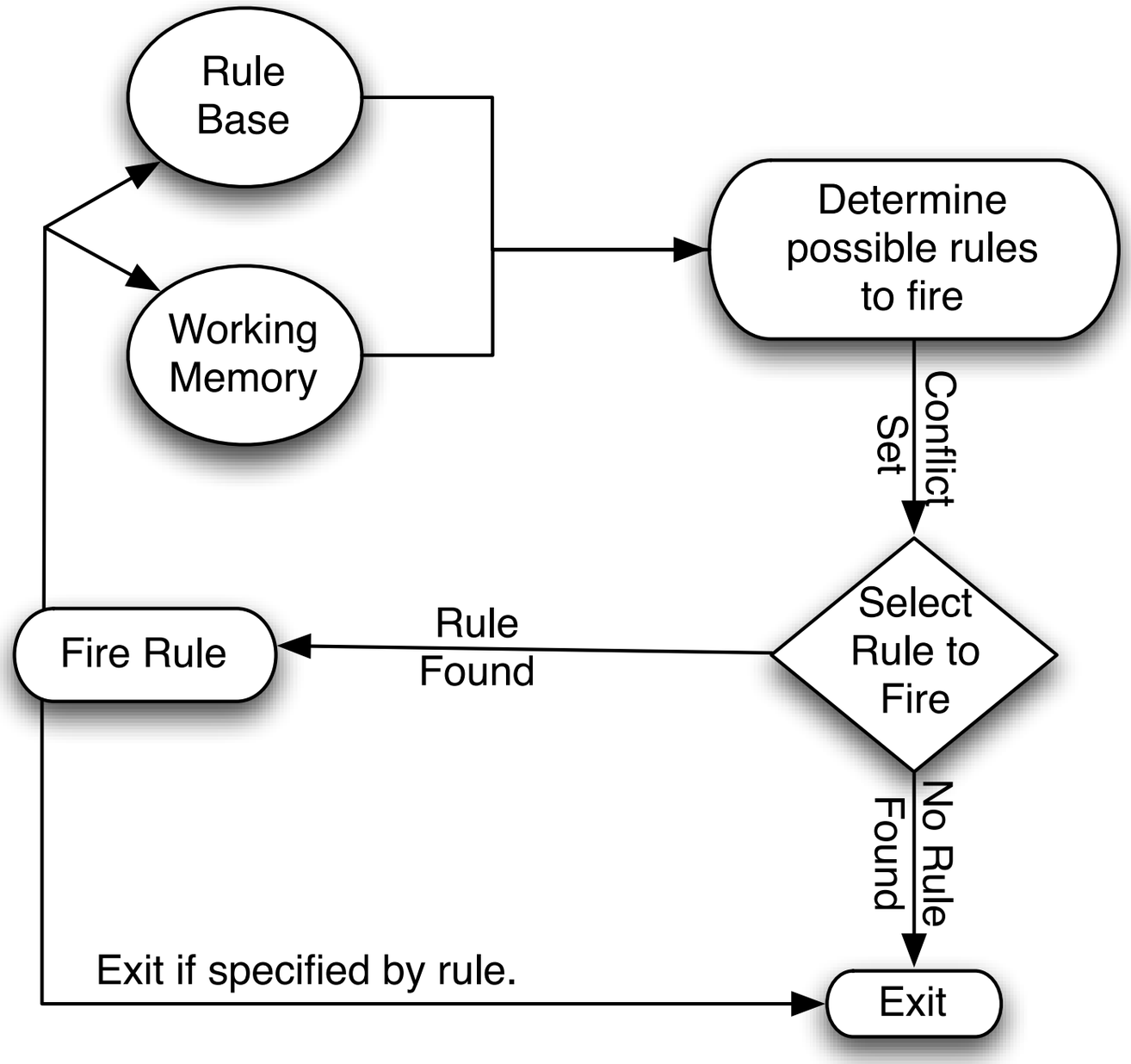
- In a rule-based program **only** the **individual** rules are needed. The rule engine, determines **which** rules apply at any **given** time and **executes** them as appropriate.

Rule-based System

- A well-known application of **rule-based** systems are **Expert Systems.**
- **Expert Systems** were popular in **1970s** and **1980s**.
- **Expert systems** were built successfully around rules for **medical diagnosis**, **engineering**, **chemistry** and **computer sales**. (ex. **MYCIN** a program for diagnosing bacterial infections of the blood; **ELIZA** primitive natural language processing.)

Rule-based System

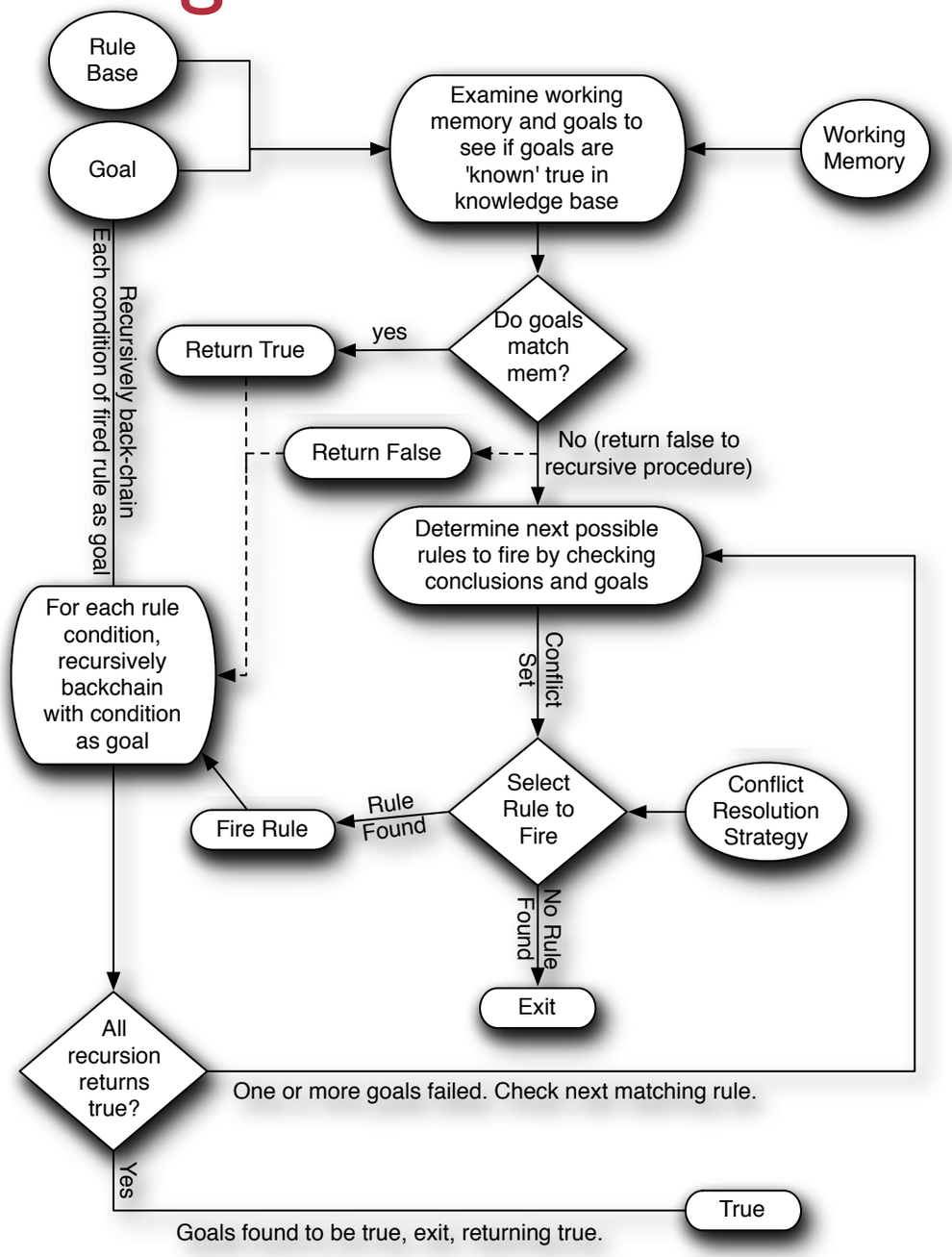
Forward Chaining



Jess normally used **Forward Chaining**.

Rule-based System

Backward Chaining



Prolog normally used **Backward Chaining**.

Rule-based System

- Example using forward chaining:
  - **Rule 1**: If it's a **rainy day**, advise people to bring umbrellas when they go out.
  - **Rule 2**: If the **road is slippery**, warn people to drive more carefully.
  - **Rule 3**: If it's a **rainy day**, **road is slippery**.
  - **Fact 1**: It's a **rainy day**.
- Solution:
  - **Rule 1 Fired** ⟶ Advise: "use umbrellas"
  - **Rule 3 Fired** ⟶ Insert Fact: Road is slippery.
  - **Rule 2 Fired** ⟶ Advise: "Drive more carefully"
- **Jess** is a **rule engine** and scripting language developed at Sandia National Laboratories in Livermore, California. It is written in Java.

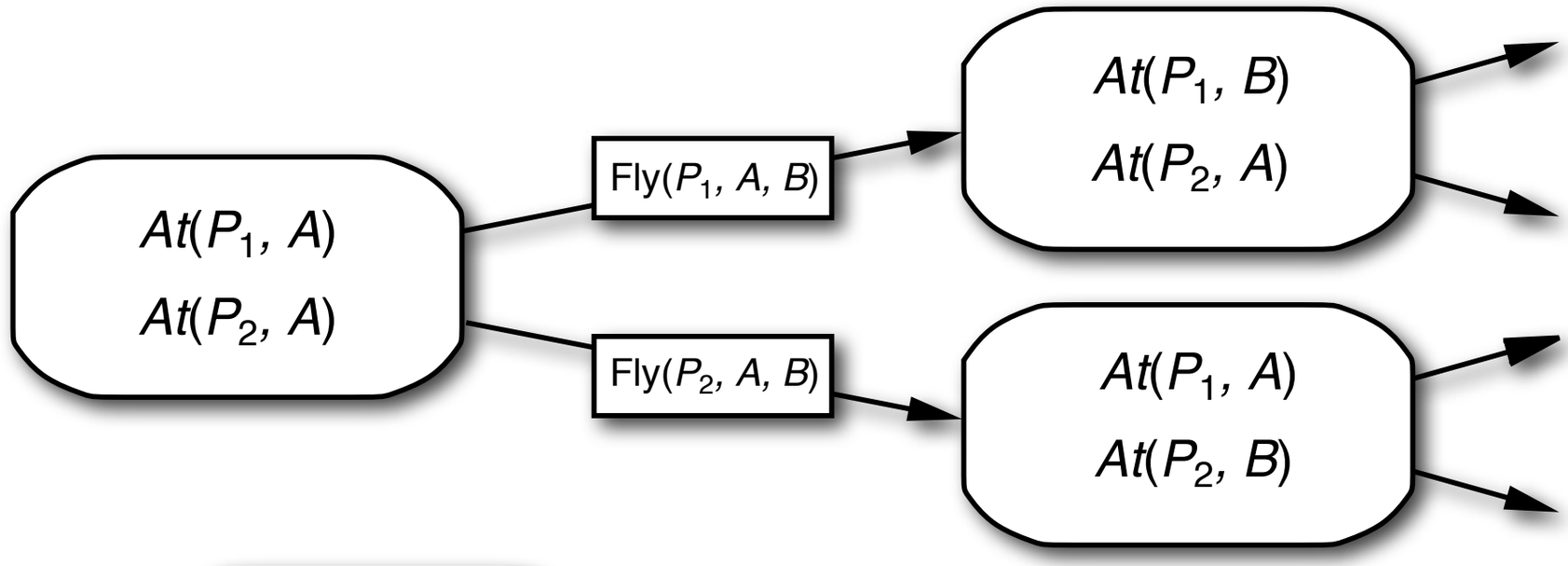
Automated Planning and Scheduling

- **Planning** is the task of coming up with a sequence of **actions** that will achieve a **goal**.
- **Planners decompose** the world into logical conditions.
- **States**: a conjunctions of **positive** and **negative literals**.
- **Goals**: **partially** specified **state** represented as a **conjunction** and **disjunction** of positive an negative ground literals.
- **Actions**: are specified in terms of **preconditions** and the **effect**.

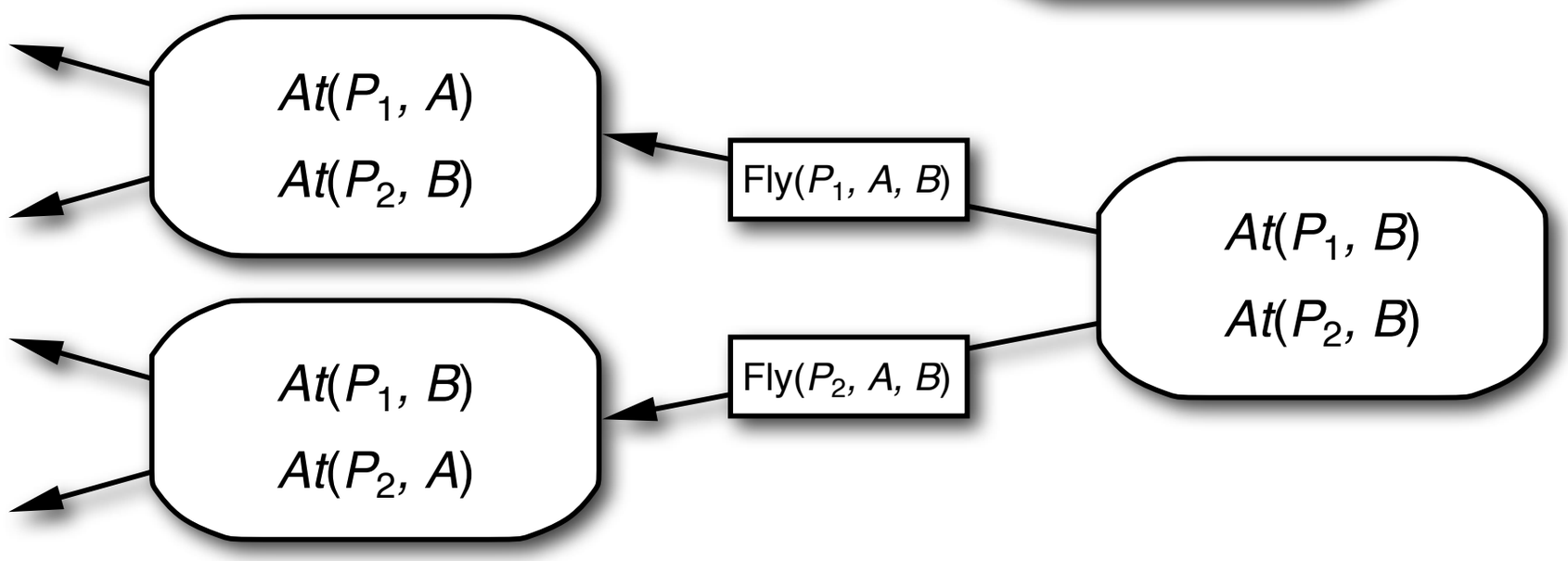
$$Action(Fly(p, from, to),$$
$$\text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT:} \neg At(p, from) \wedge At(p, to))$$

Automated Planning and Scheduling

Forward (progression) state-space



Backward (regression) state-space

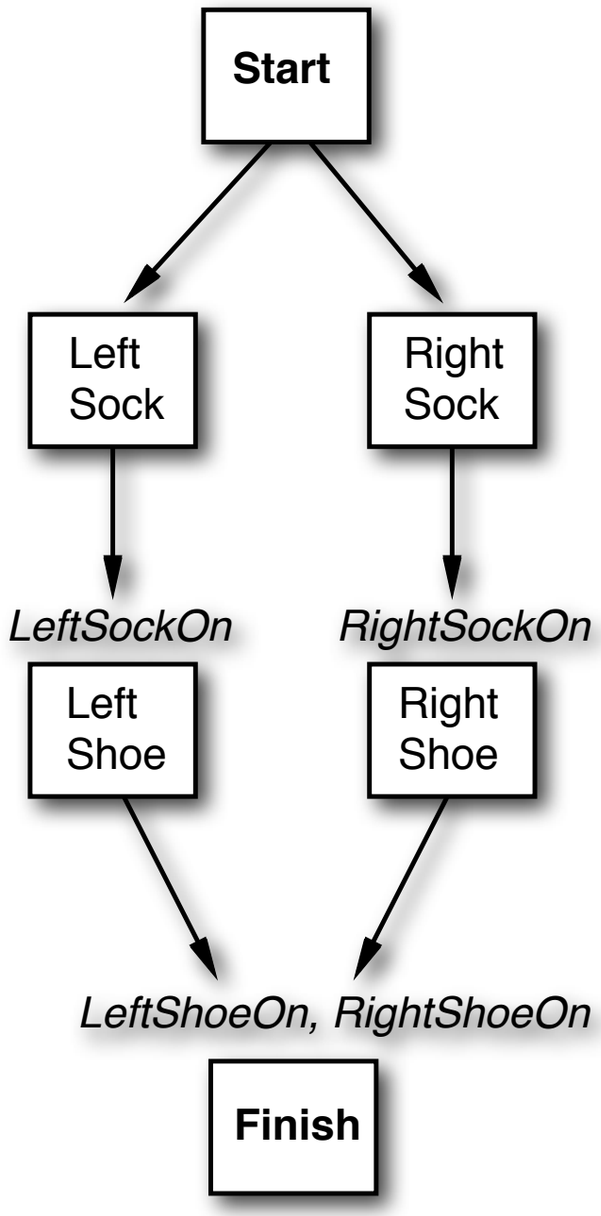


Automated Planning and Scheduling

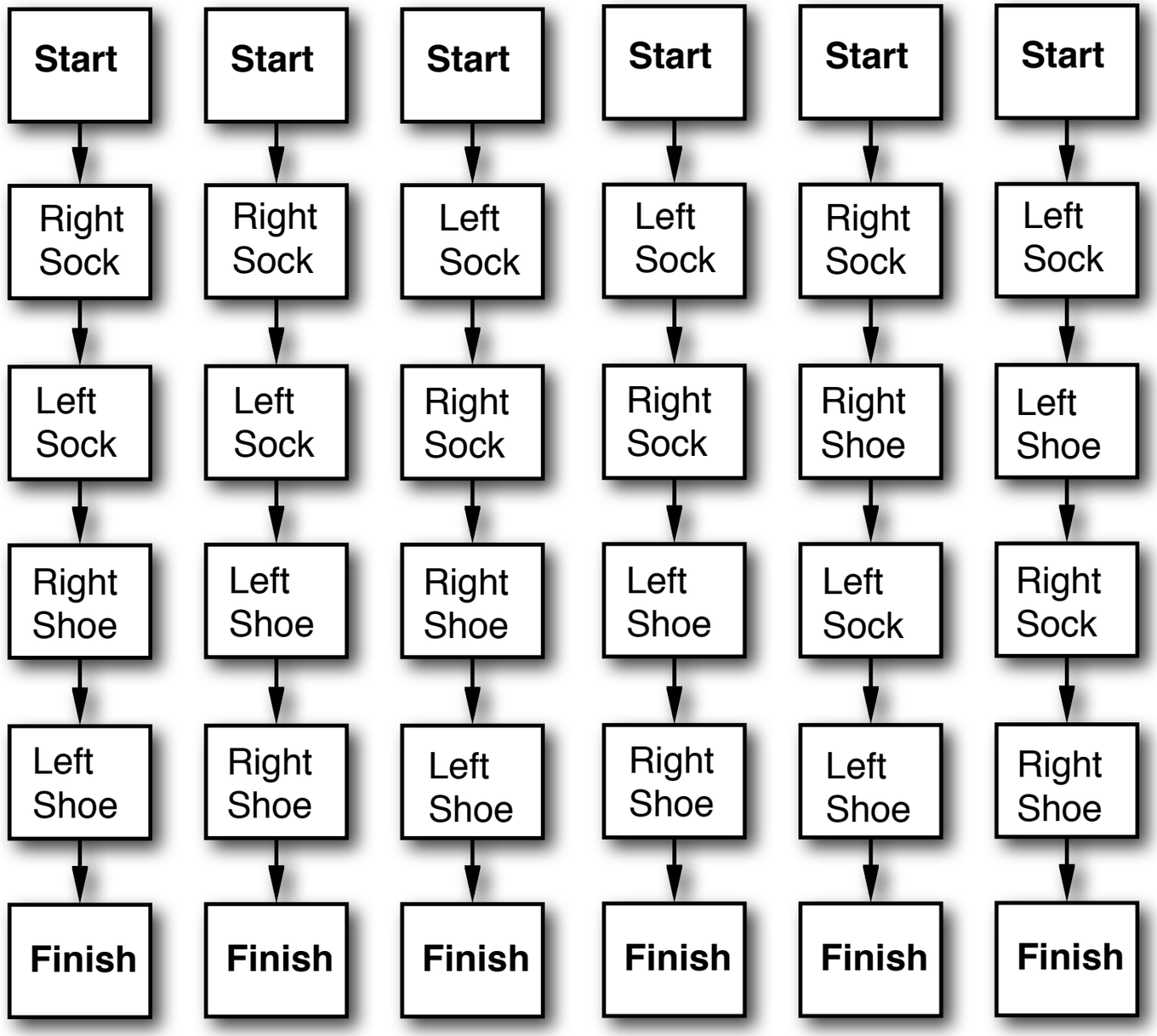
- **Total ordered plan:** explore only strictly **linear sequences** of actions directly connected to the **start** or **goal**.
- **Forward** and **Backward** state-space search are particular forms of **totally ordered** plan search.
- **Partial-Order Plan:** can place **two actions** into a plan **without** specifying **which comes first**.

Automated Planning and Scheduling

Partial-Order Plan:



Total-Order Plans:



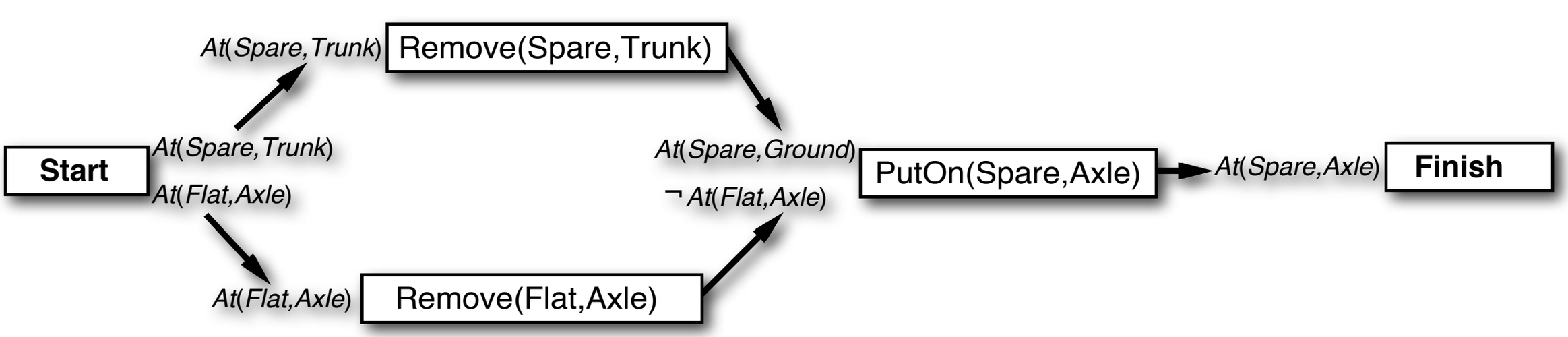
Automated Planning and Scheduling

- Example : The spare tire problem.

$Init(At(Flat, Axle) \ \wedge \ At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg\, At(Spare, Trunk) \ \wedge \ At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg\, At(Flat, Axle) \ \wedge \ At(Flat, Ground))$
$Action(PutOn(Spare, \ Axle),$
PRECOND: $At(Spare, Ground) \ \wedge \ \neg\, At(Flat, Axle)$
EFFECT: $\neg\, At(Spare, Ground) \ \wedge \ At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg\, At(Spare, Ground) \ \wedge \ \neg\, At(Spare, Axle) \ \wedge \ \neg\, At(Spare, Trunk)$
$\wedge \neg\, At(Flat, Ground) \ \wedge \ \neg\, At(Flat, Axle))$

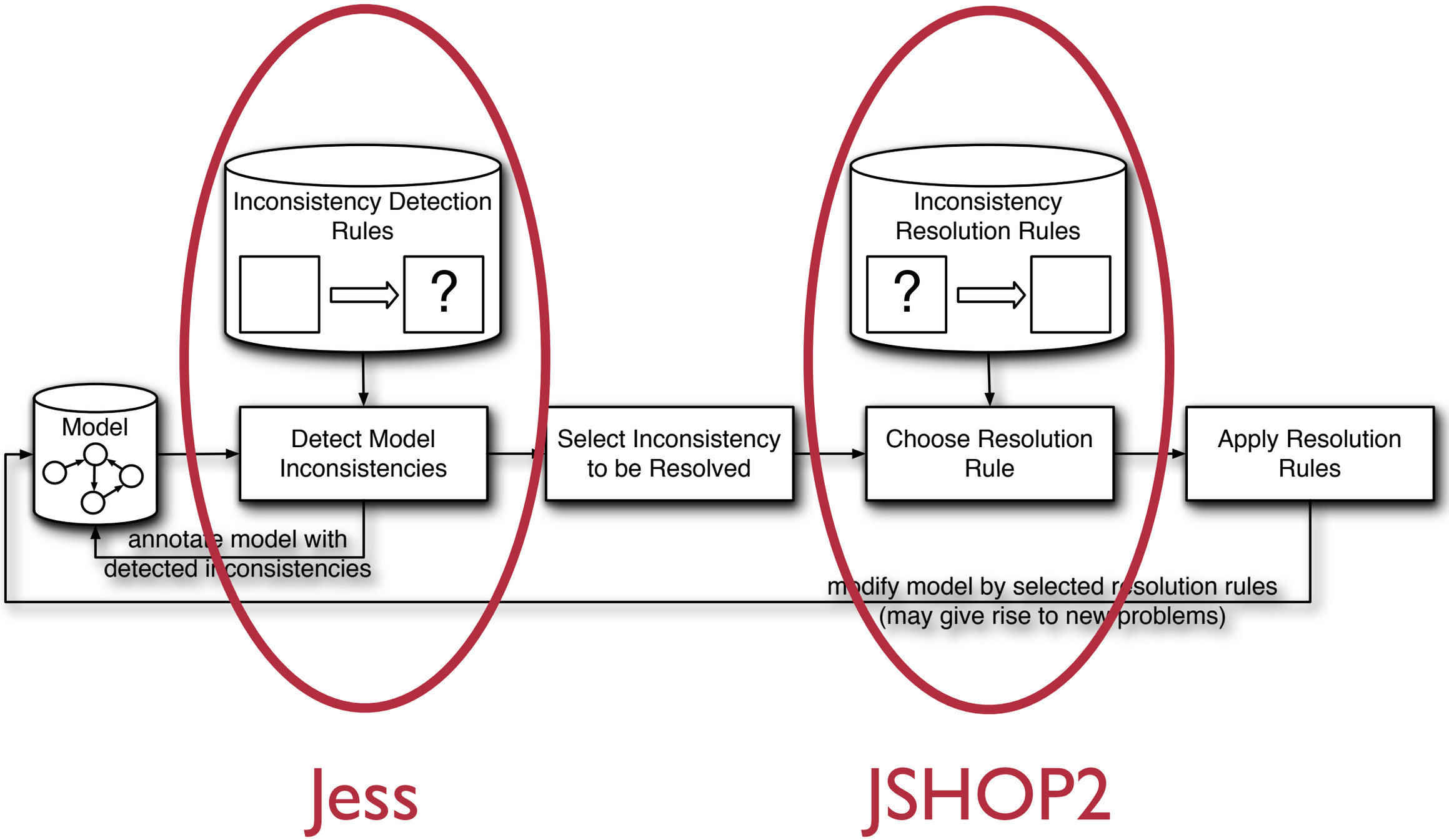
Automated Planning and Scheduling

- Solution in Partial-Order Planning



Current Research

Model inconsistency management



Current Research

- Examples of rules for the detection of inconsistencies using Jess

```
(defrule verify_source_relation
   "Whith this rule we verify that the source of the relation is a class"
    (relation (id ?id) (source ?source) (target ?target))
    (class (id ?target) (name ?class_name))
    (not (class (id ?source)))
   =>
    (printout t "INCONSISTENCY:-" "The relation " ?id " Don't have a
class as a source." crlf "The class with problem is " ?class_name crlf))

(defrule verify_target_relation
   "Whith this rule we verify that the target of the relation is a class"
    (relation (id ?id) (source ?source) (target ?target))
    (class (id ?source) (name ?class_name))
    (not (class (id ?target)))
   =>
    (printout t "INCONSISTENCY:-" "The relation " ?id " Don't have a
class as a target." crlf "The class with problem is " ?class_name crlf))
```

Acknowledgement

Project: Model-Driven Software Evolution

WebSite: www.evolumons.be/ARC/

This research is **funded** by the ***Actions de Recherche Concertées*** - *Ministère de la Communauté française - Direction générale de l'Enseignement non obligatoire et de la Recherche scientifique.*