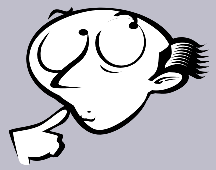
Risk-Driven Engineering of Requirements for Dependable Systems

Axel van Lamsweerde
University of Louvain
B-1348 Louvain-la-Neuve (Belgium)

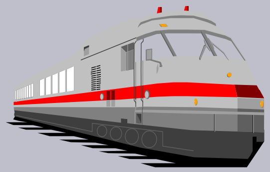
Marktoberdorf Summer School
August 2012



Requirements engineering (RE), roughly ...

- Identify & analyze problems with an existing system (system-**as-is**),
- Identify & evaluate objectives, opportunities, options for new system (system-**to-be**),
- Identify & define functionalities of, constraints on, responsibilities in system-to-be,
- Specify & organize all of these in a **requirements document** to be maintained throughout system development & evolution

System = software + environment
(people, devices, existing software)



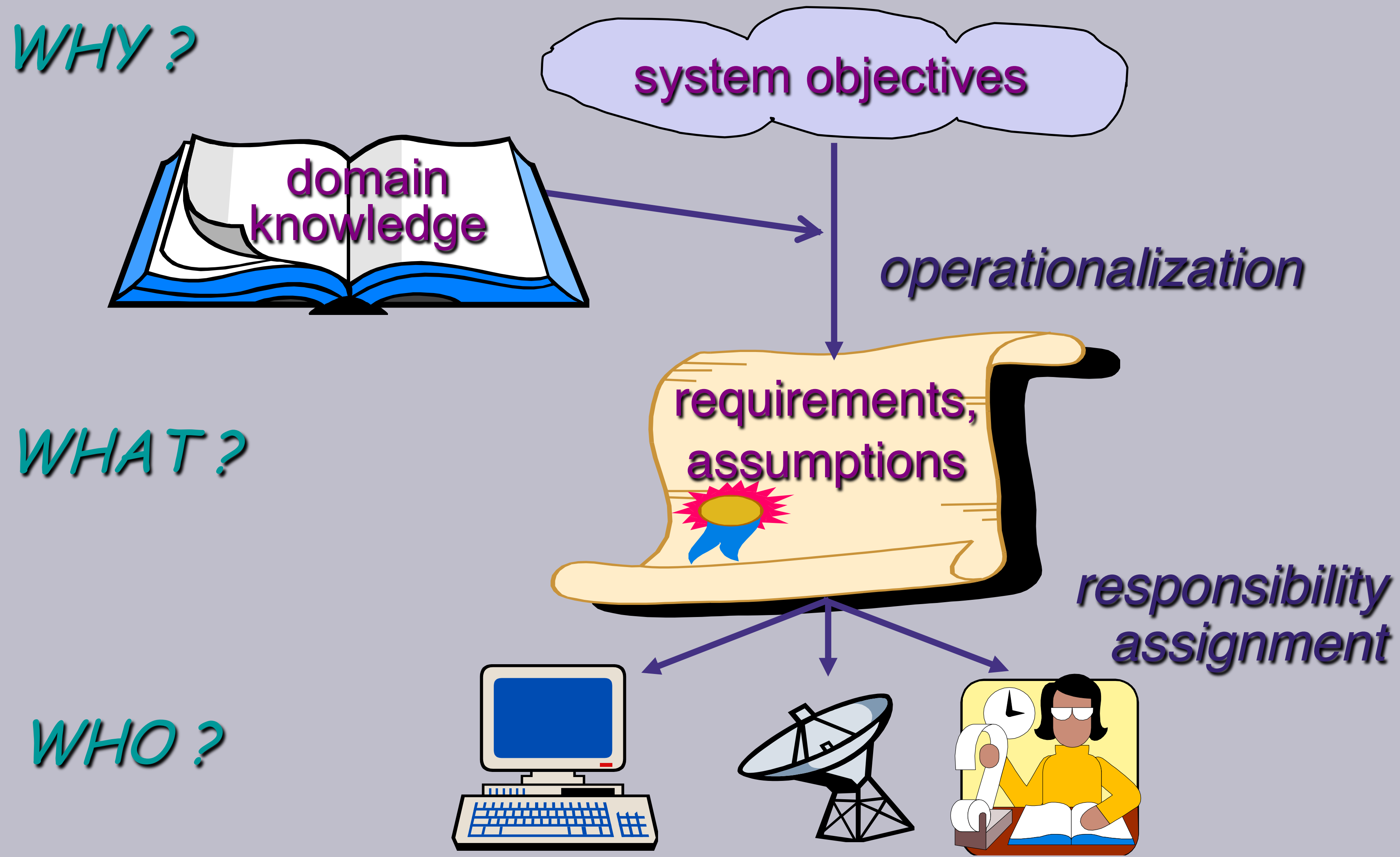
Example:
transportation between airport terminals

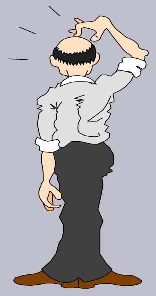
- Problem (system-**as-is**):
  - passengers frequently missing flight connections among terminals; slow & inconvenient bus transportation
  - number of passengers increasing regularly
- Objectives, options (system-**to-be**):
  - support high-frequency trains between terminals
  - with **or** without train drivers ?
- Functionalities, constraints:
  - software-based control of train accelerations, of doors opening etc. to achieve *prompt* and *safe* transportation
- RE deliverable: requirements document for system-to-be

The scope of RE is broad

- Composite system: software-to-be + environment
- Multiple system versions: as-is, to-be, to-be-next
- Multiple options (evaluation, selection)
- Multiple stakeholders to be involved
- Multiple dimensions: WHY, WHAT, WHO

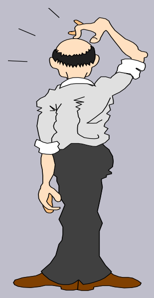
The scope of RE: WHY, WHAT, WHO





RE is hard:
multiple transitions to handle

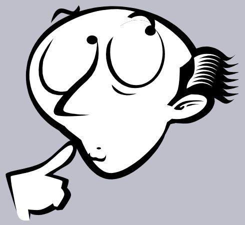
- Informal problem world → formal machine world
- High-level, strategic → low-level, technical
- Imprecise, unstructured → precise, structured



RE is hard: difficult transitions to handle

- Informal problem world → formal machine world
- High-level, strategic → low-level, technical
- Imprecise, unstructured → precise, structured
- Implicit, hidden → explicit, adequate
- Conflicting → consistent
- Partial → sufficiently complete

- **Intended, ideal → unexpected, realistic**
  (hazards, threats)



RE is critical

- Major cause of software failure

  Requirements-related errors are the most
  numerous, persistent, expensive, dangerous

- Severe consequences

  accidents, environmental degradations
  cost overruns, delivery delays, dissatisfaction

- Multiple impact

  legal, social, economical, technical

- Certification issues

Requirements **completeness** is a major challenge

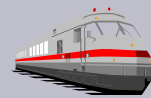
- Missing requirements = major cause of software failure
- Often result from **poor risk analysis**
  - lack of anticipation of what could go wrong

    => over-ideal system,

    no requirements on handling adverse events

Risks must be anticipated at RE time

- Risk = uncertain factor whose occurrence may result in **loss of satisfaction** of a corresponding **objective**

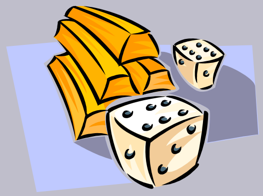
  e.g. a passenger forcing doors opening while train moving 

  a meeting participant not checking email regularly 

- A risk has...
  - a **likelihood** of occurrence,
  - one or more undesirable **consequences**

  e.g. passengers falling out of train moving with doors open 

- Each risk consequence has ...
  - a **likelihood** of occurrence if the risk occurs
  - a **severity**: degree of loss of satisfaction of objective



Risk management at RE time

what system-specific risks?

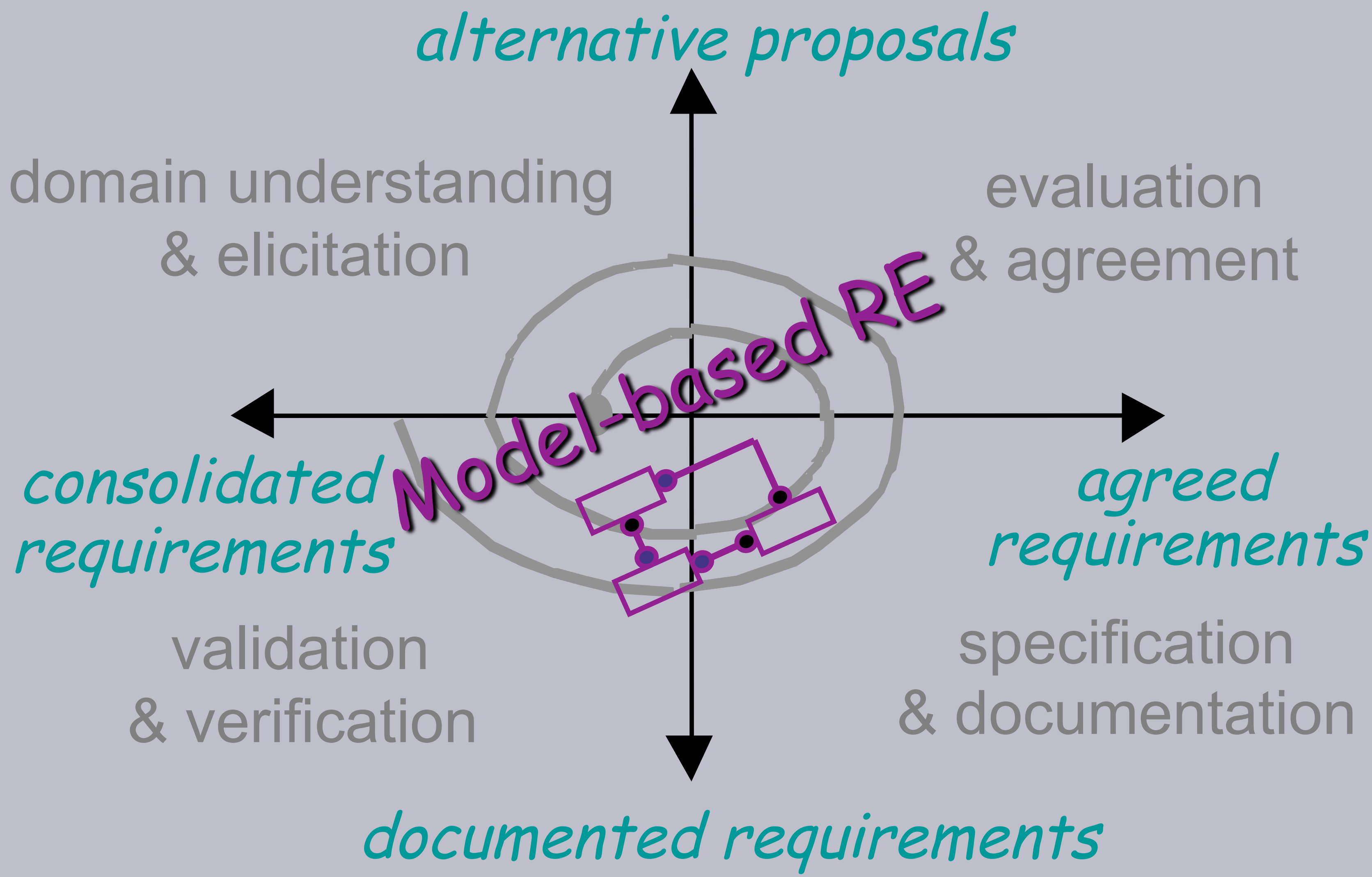
likely?
severe, likely consequences?

countermeasures as
new requirements

Course outline

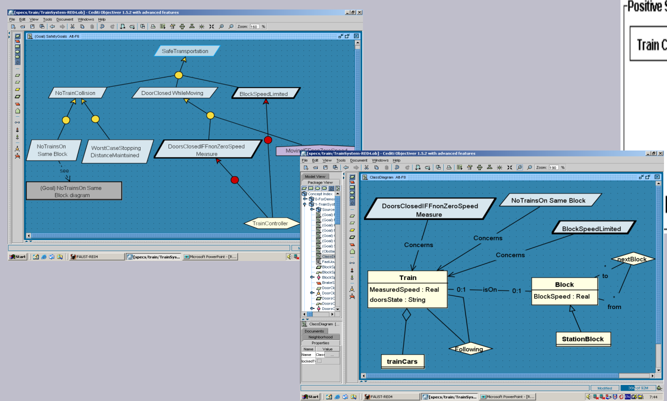
- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model items
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

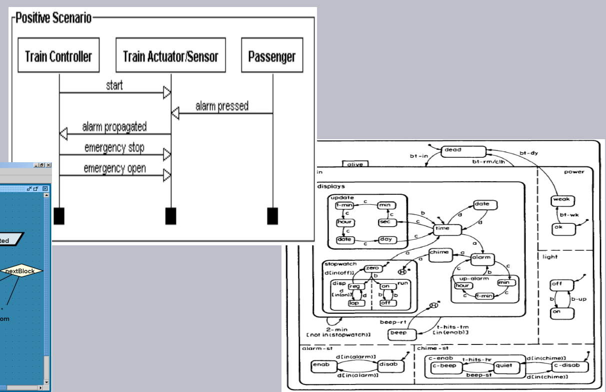
Models are interface among RE tasks



Model-based RE

- To focus on **key aspects**
- To support early **analysis** & fix of critical errors
- To support explanation to stakeholders
- To support decisions among multiple options

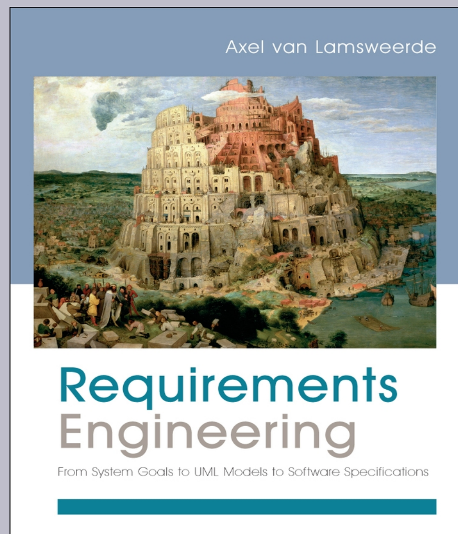




Model building at RE time should be **goal-oriented**

To enable ...

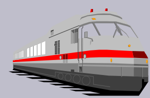
- satisfaction arguments   *Specs, Assumptions* $\vdash$ *Goal*
- completeness & pertinence of the model
- early, incremental analysis
- model refinement & synthesis (deductive, inductive)
- reasoning about alternative options
- validation by stakeholders
- backward traceability
- generation of ...
  - requirements document
  - architectural fragments
  - runtime reqirements monitors



Declarative abstractions for system modeling at RE time

- **Goal**
  - **prescriptive** statement of intent about *system*

    "Trains shall stop at stop signals"



- **Domain property**
  - **descriptive** statement about *environment*

    "Train doors are either open or closed"

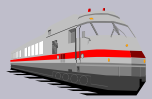
- Both used for model building
  - Goals may be refined, negotiated, weakened, prioritized ... ***unlike*** domain properties

Goals are formulated at different levels of abstraction

- **Higher-level** goals

  strategic, coarse-grained

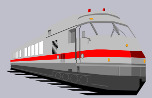
  "50% increase of transportation capacity”



- **Lower-level** goals

  technical, fine-grained

  ”Acceleration command sent every 3 secs"



Goal satisfaction requires agent cooperation

- **Agent**: active component, controls behaviors

  software-to-be, device, human role, existing sw

  TrainController, Passenger, SpeedSensor, TrackingSystem

The more fine-grained a goal,

the fewer agents required for its satisfaction

SafeTransportation vs. DoorsClosedWhileMoving

Goal satisfaction requires agent cooperation

- **Agent**: active component, controls behaviors

  software-to-be, device, human role, existing sw

  TrainController, Passenger, SpeedSensor, TrackingSystem

  The more fine-grained a goal,

  the fewer agents required for its satisfaction

  SafeTransportation vs. DoorsClosedWhileMoving

- **Requirement**: goal assigned to single software agent

  Train.measuredSpeed $\neq$ 0 $\rightarrow$ Train.DoorsState = "closed"

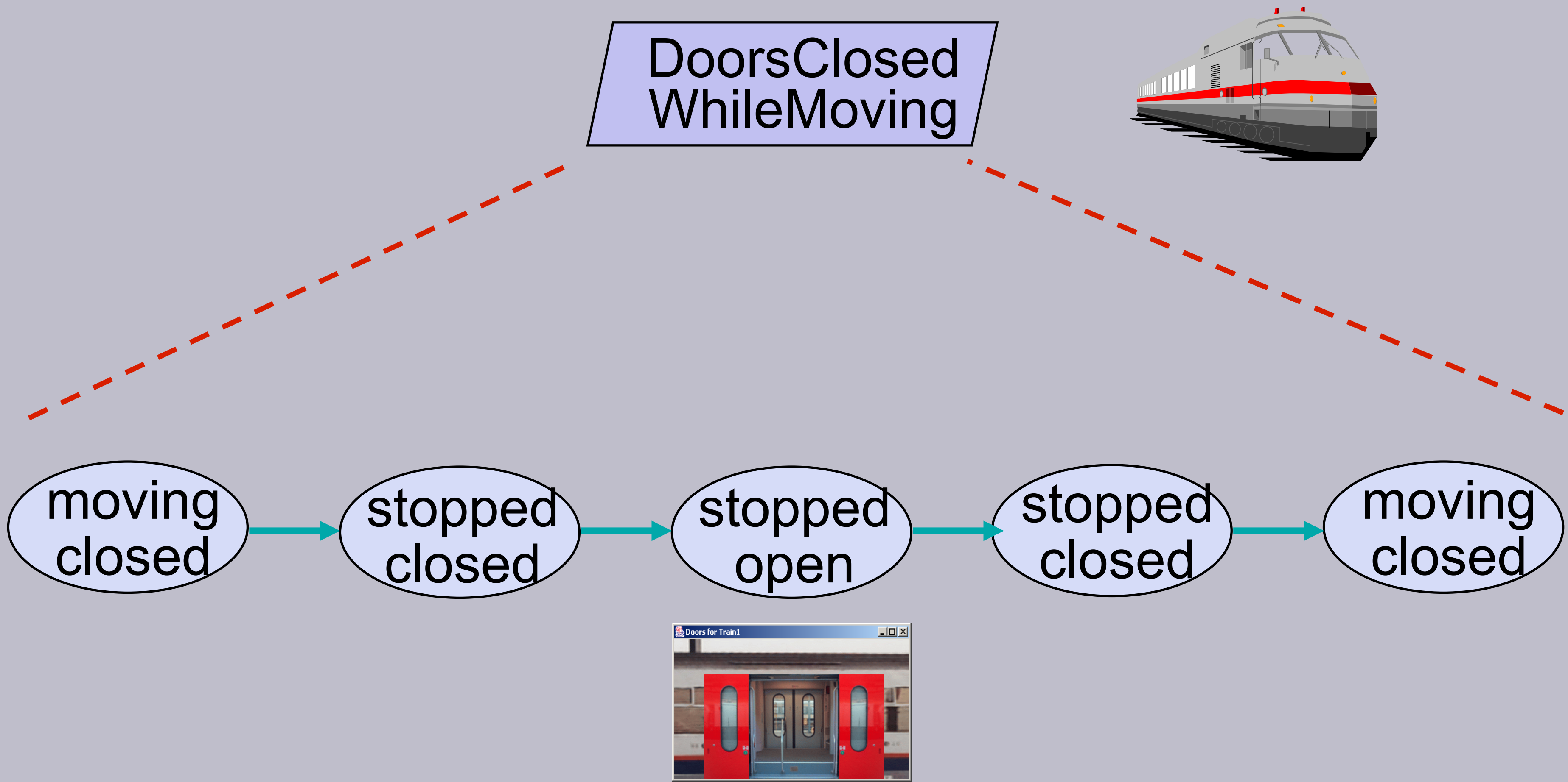
- **Expectation**: goal assigned to single environment agent (prescriptive assumption)

  Train.measuredSpeed $\neq$ 0 iff Train.Speed $\neq$ 0

Goal types & categories

- Two *types* of goals
  - **Behavioral goals:** prescribe intended behaviors
    can be satisfied in clear-cut sense
    used for deriving operational models & risk analysis
  - Soft goals prescribe preferred behaviors
- Two categories
  - functional, non-functional goals

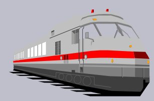
Behavioral goals prescribe sets of behaviors declaratively



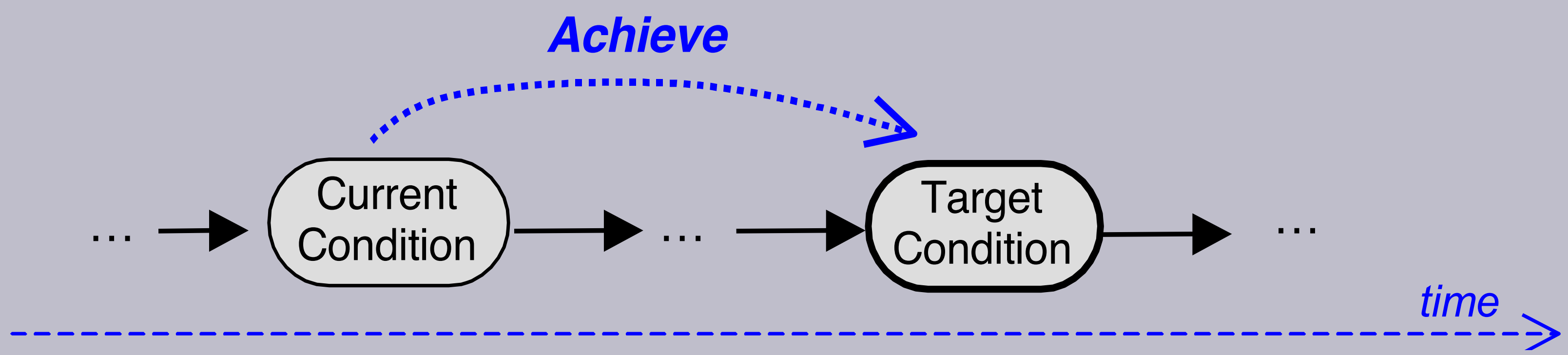
Behavioral goals: subtypes and specification patterns

- ***Achieve*** [TargetCondition]:

  [if CurrentCondition **then] sooner-or-later** TargetCondition

Achieve [FastJourney]: 

if train is at some platform **then within 5 minutes** it is at next platform



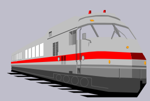
Behavioral goals: subtypes and specification patterns (2)

- ***Maintain*** [GoodCondition]:

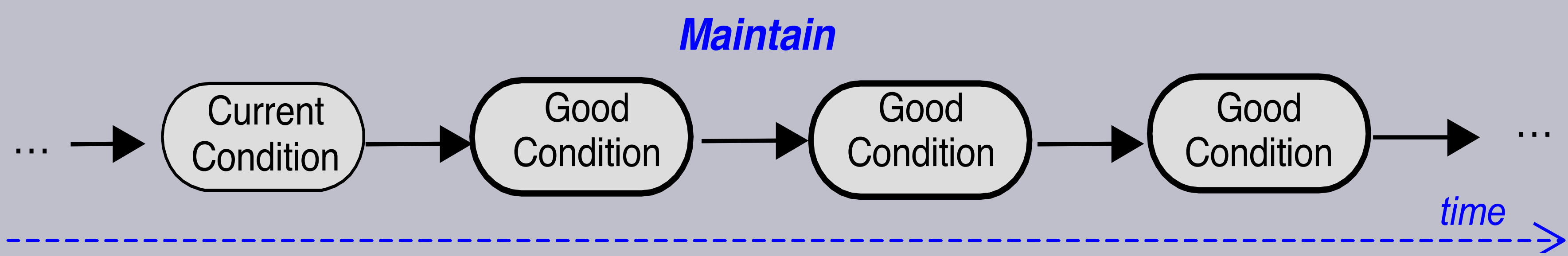
  [if CurrentCondition **then] always** GoodCondition

  always (**if** CurrentCondition **then** GoodCondition)

Maintain *[DoorsClosedWhileMoving]:*



always (**if** a train is moving **then** its doors are closed)



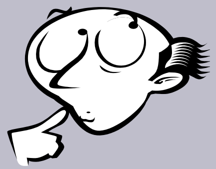
Goal types & categories

- Two *types* of goals
  - Behavioral goals: prescribe intended behaviors
    can be satisfied in clear-cut sense
    used for deriving operational models & risk analysis
  - **Soft goals:** prescribe preferred behaviors
    can<u>not</u> be satisfied in clear-cut sense
    used for comparing alternative options

“Stress conditions of air traffic controllers shall be reduced”

Goal types & categories

- Two *types* of goals
  - Behavioral goals: prescribe intended behaviors
    can be satisfied in clear-cut sense
    used for deriving operational models
  - Soft goals: prescribe preferred behaviors
    cannot be satisfied in clear-cut sense
    used for comparing alternative options
- Two *categories* of goals
  - **functional**: underlying operation, feature, service, task
  - **non-functional**: quality goals e.g. security, accuracy, ...
    architectural goals, development goals,...

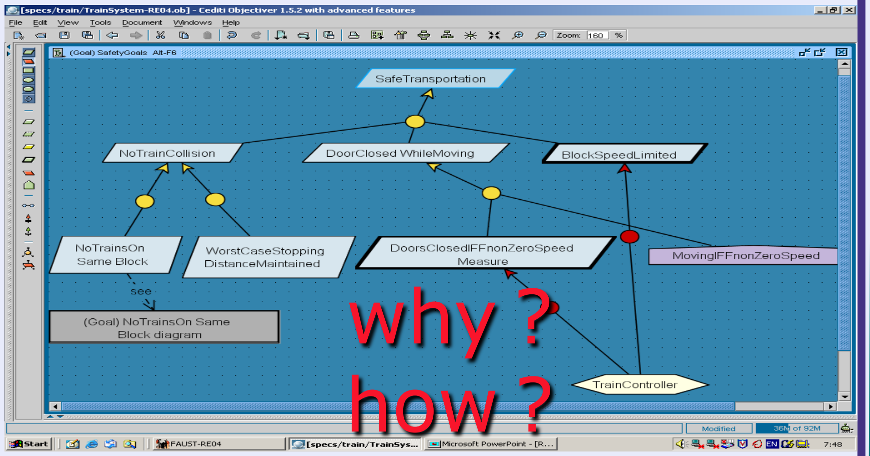


What kind of system model for RE ?

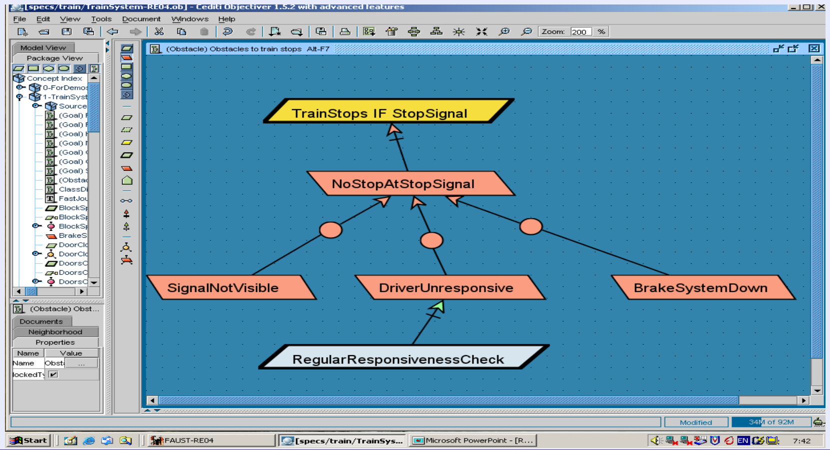
- **Multi-view**
  - complementary facets, for model comprehensiveness
    intentional, structural, responsibility, operational, behavioral
  - inter-view rules for structural consistency
- **Multi-formalism**
  - Diagrammatic
    Goal AND/OR refinement graphs
    UML subset: class, sequence, state diagrams
    Event-based behaviors: Labeled Transition Systems (LTS)
  - Formal (*when* & *where* needed): real-time temporal logic
  - Quantitative: propagation equations

What models for RE ?

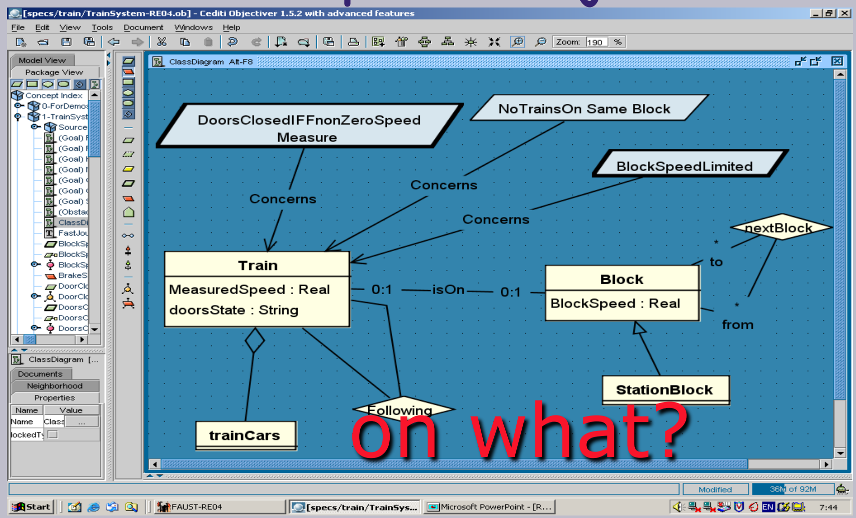
Goals



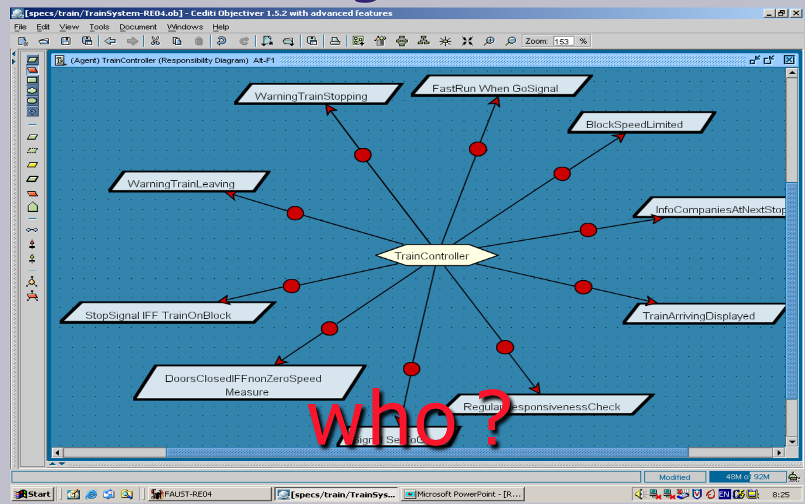
Risks



Conceptual objects

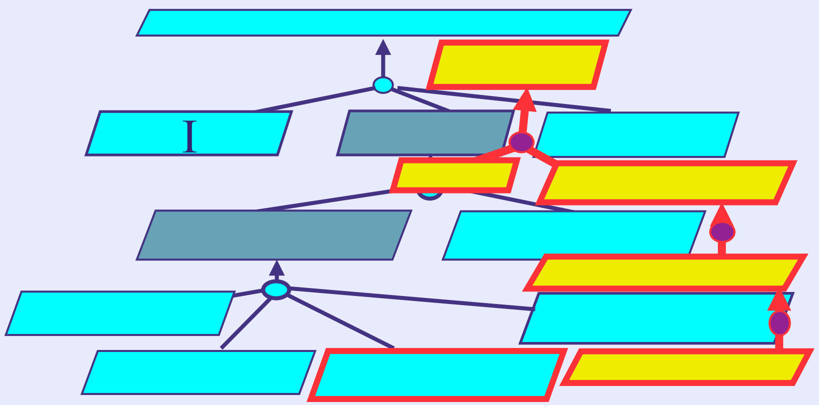


Agents

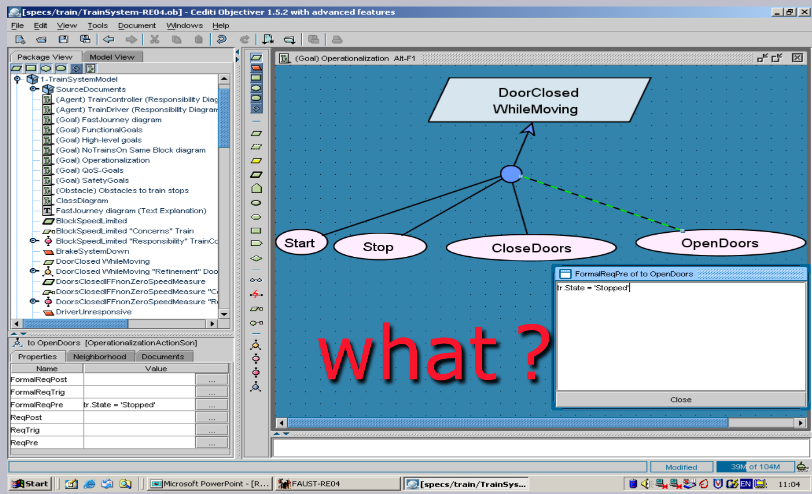


What models for RE ?

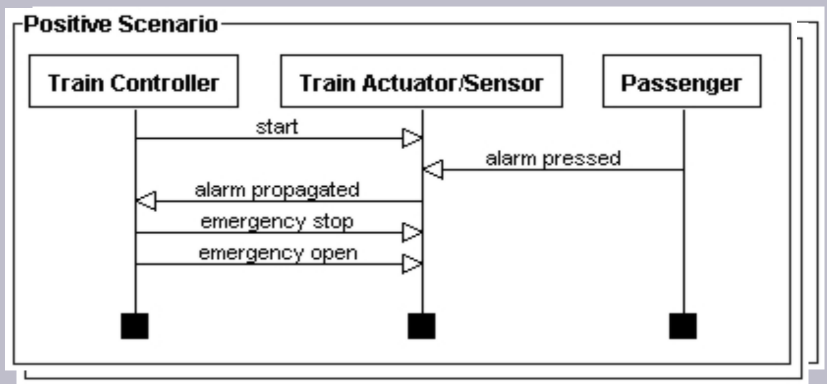
Threats



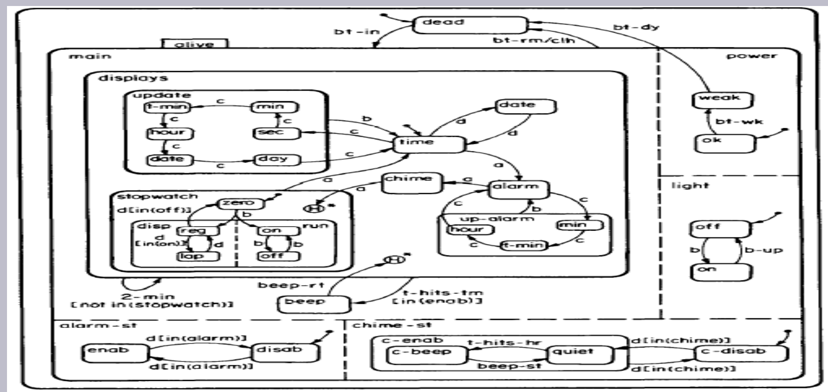
Operations



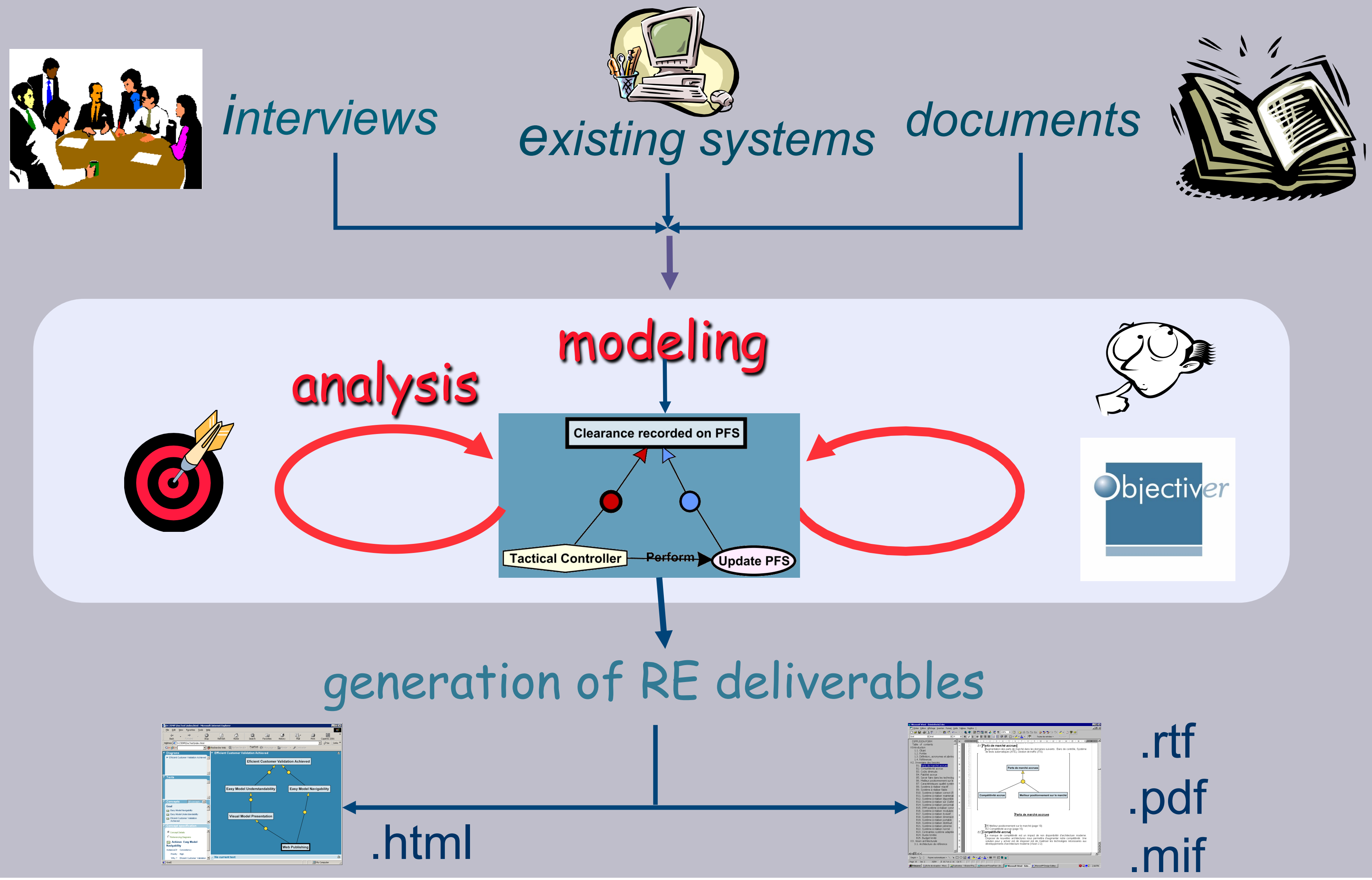
Behaviors - Scenarios



Behaviors - State machines

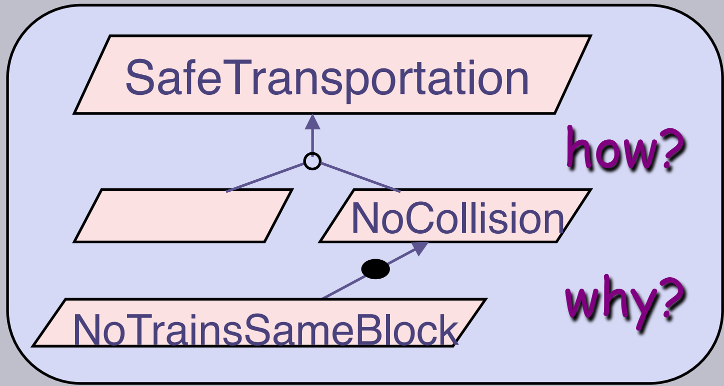


The focus here is on **model building & analysis** at RE time

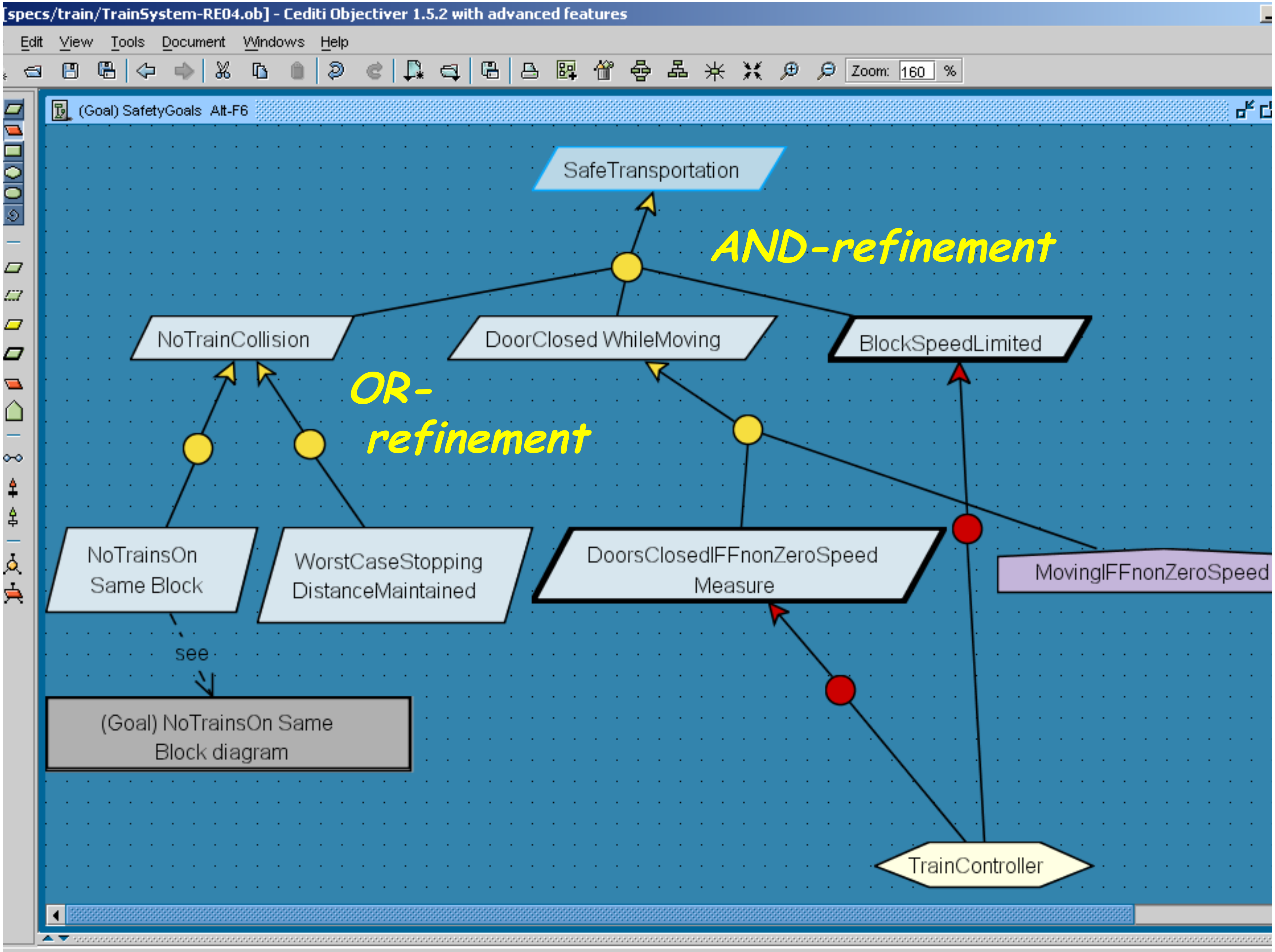


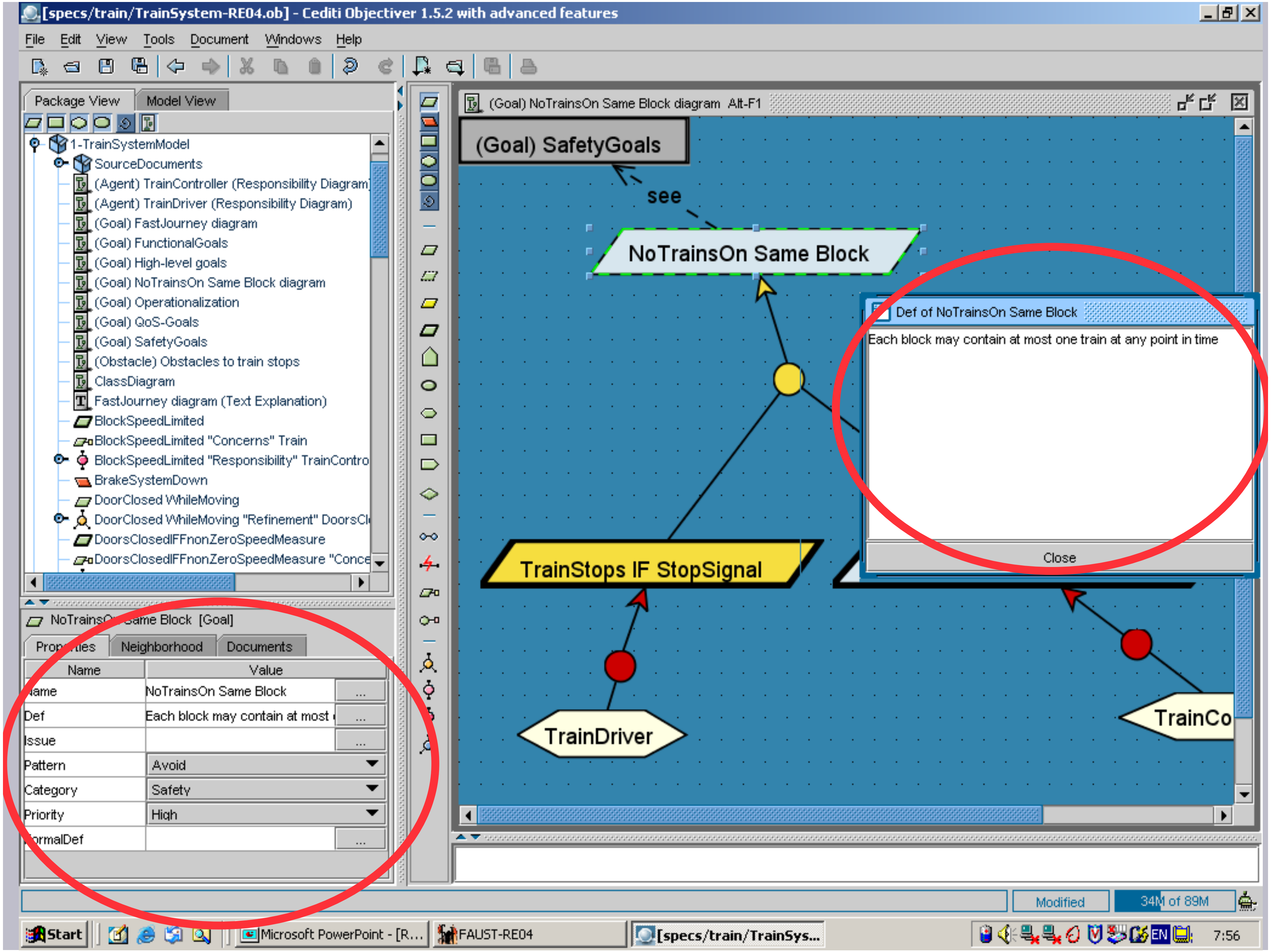
Goal-oriented model building

1. Domain analysis: refine/abstract goals

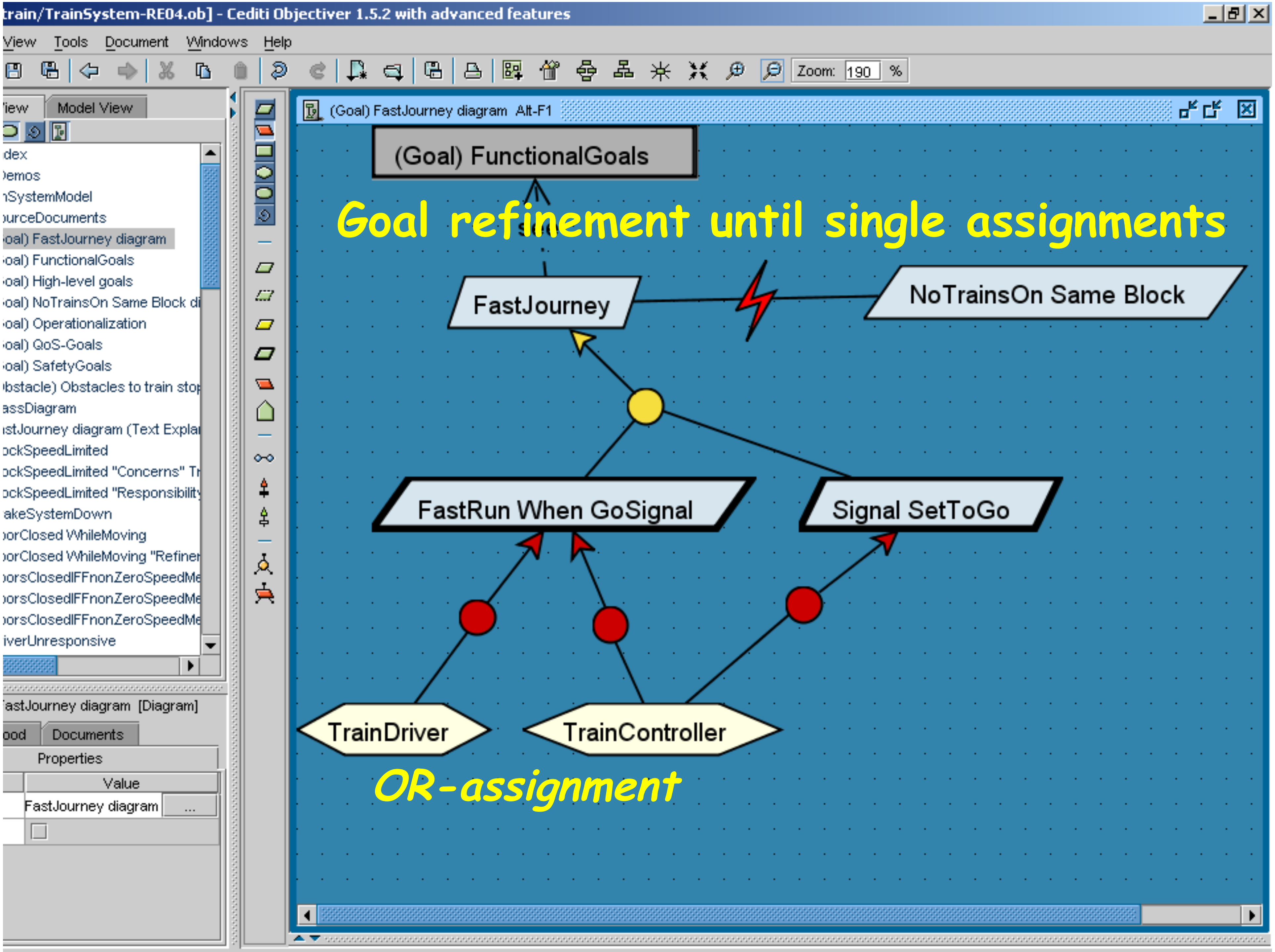


system-as-is


[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
Edit View Tools Document Windows Help
Zoom: 160 %
(Goal) SafetyGoals Alt-F6
SafeTransportation
AND-refinement
NoTrainCollision
DoorClosed WhileMoving
BlockSpeedLimited
OR-refinement
NoTrainsOn Same Block
WorstCaseStopping DistanceMaintained
DoorsClosedIFFnonZeroSpeed Measure
MovingIFFnonZeroSpeed
see
(Goal) NoTrainsOn Same Block diagram
TrainController



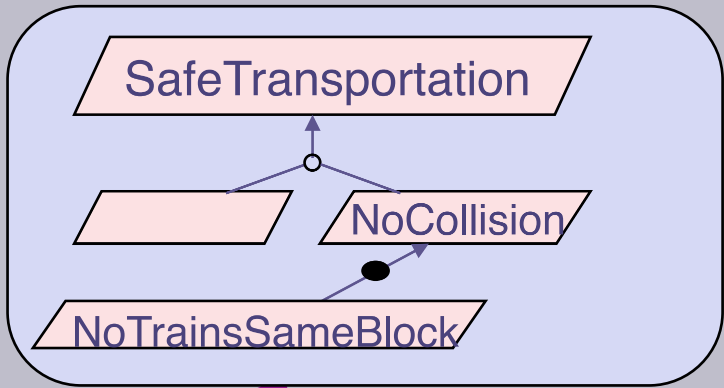
[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
1-TrainSystemModel
SourceDocuments
(Agent) TrainController (Responsibility Diagram)
(Agent) TrainDriver (Responsibility Diagram)
(Goal) FastJourney diagram
(Goal) FunctionalGoals
(Goal) High-level goals
(Goal) NoTrainsOn Same Block diagram
(Goal) Operationalization
(Goal) QoS-Goals
(Goal) SafetyGoals
(Obstacle) Obstacles to train stops
ClassDiagram
FastJourney diagram (Text Explanation)
BlockSpeedLimited
BlockSpeedLimited "Concerns" Train
BlockSpeedLimited "Responsibility" TrainContro
BrakeSystemDown
DoorClosed WhileMoving
DoorClosed WhileMoving "Refinement" DoorsCl
DoorsClosedIFFnonZeroSpeedMeasure
DoorsClosedIFFnonZeroSpeedMeasure "Conce
NoTrainsOn Same Block [Goal]
Properties
Neighborhood
Documents
Name
Value
Name
NoTrainsOn Same Block
Def
Each block may contain at most
Issue
Pattern
Avoid
Category
Safety
Priority
High
FormalDef
(Goal) NoTrainsOn Same Block diagram Alt-F1
(Goal) SafetyGoals
see
NoTrainsOn Same Block
Def of NoTrainsOn Same Block
Each block may contain at most one train at any point in time
Close
TrainStops IF StopSignal
TrainDriver
TrainCo
Modified
34M of 89M
Start
Microsoft PowerPoint - [R...
FAUST-RE04
[specs/train/TrainSys...
7:56



train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
View Tools Document Windows Help
Zoom: 190 %
Model View
(Goal) FastJourney diagram Alt-F1
(Goal) FunctionalGoals
Goal refinement until single assignments
FastJourney
NoTrainsOn Same Block
FastRun When GoSignal
Signal SetToGo
TrainDriver
TrainController
OR-assignment
FastJourney diagram [Diagram]
Documents
Properties
Value
FastJourney diagram

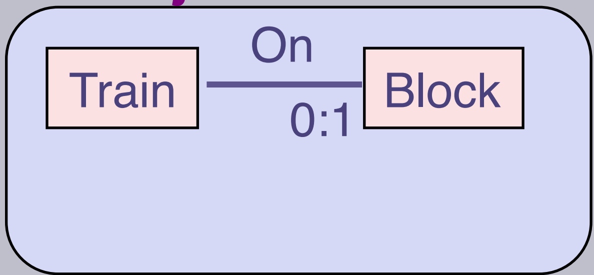
Goal-oriented model building

1. Domain analysis: refine/abstract goals

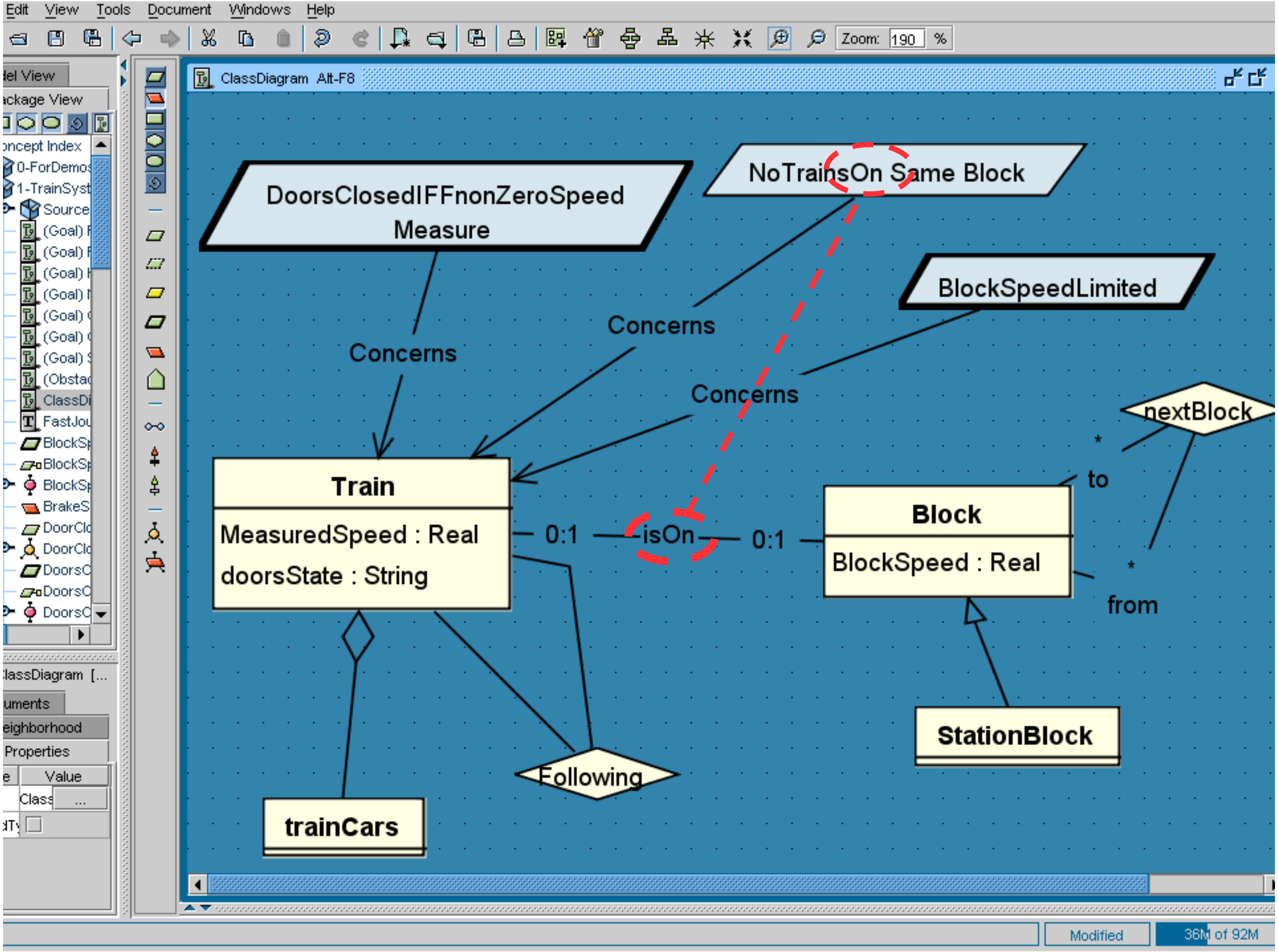


system-as-is

2. Domain analysis: derive/structure objects

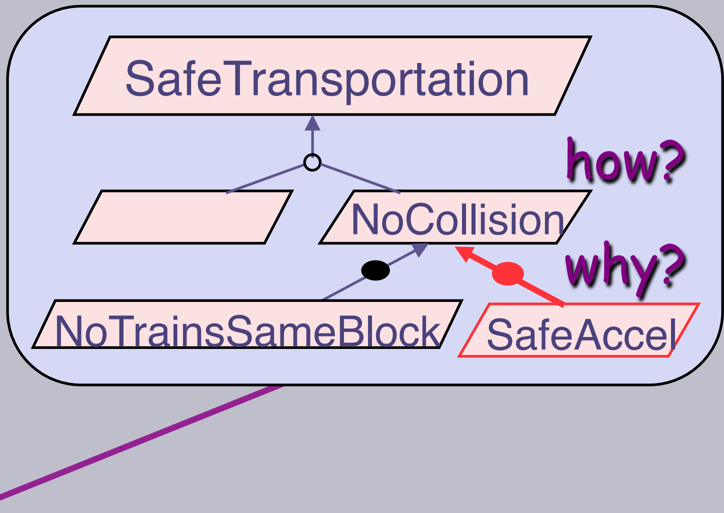


system-as-is


Edit View Tools Document Windows Help
Zoom: 190 %
ClassDiagram Alt-F8
DoorsClosedIFFnonZeroSpeed Measure
NoTrainsOn Same Block
BlockSpeedLimited
Concerns
Concerns
Concerns
nextBlock
Train
MeasuredSpeed : Real
doorsState : String
0:1
isOn
0:1
Block
BlockSpeed : Real
to
from
StationBlock
Following
trainCars
Modified
36M of 92M

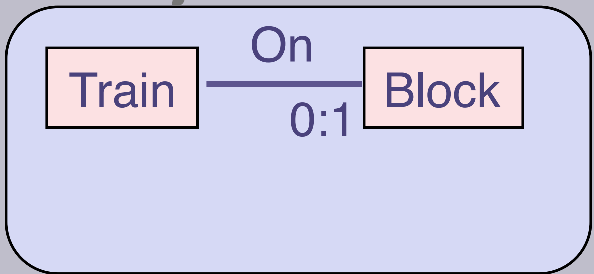
Goal-oriented model building

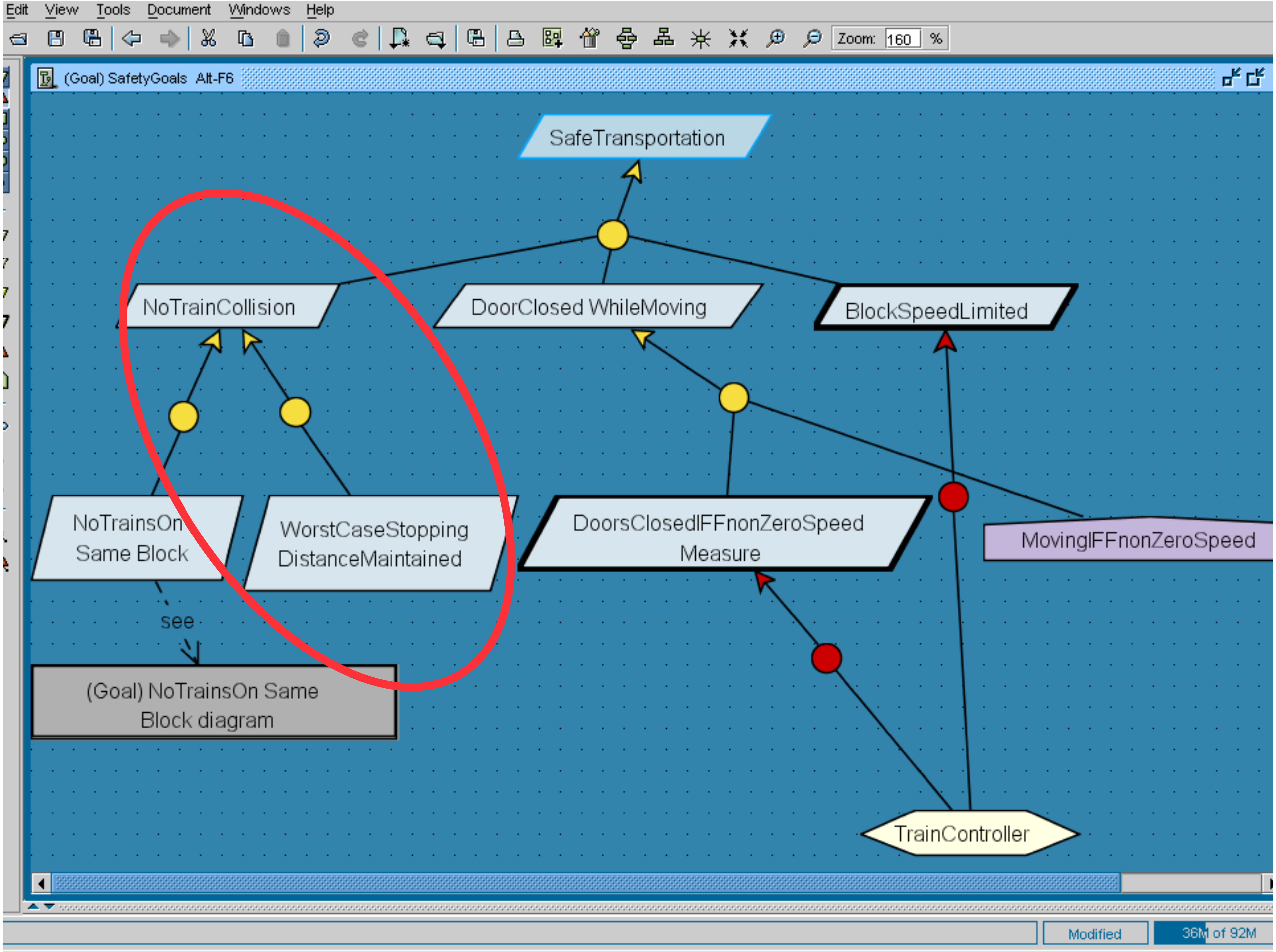
1. Domain analysis: refine/abstract goals



3. System-to-be: enriched goals (alternatives)

2. Domain analysis: derive/structure objects

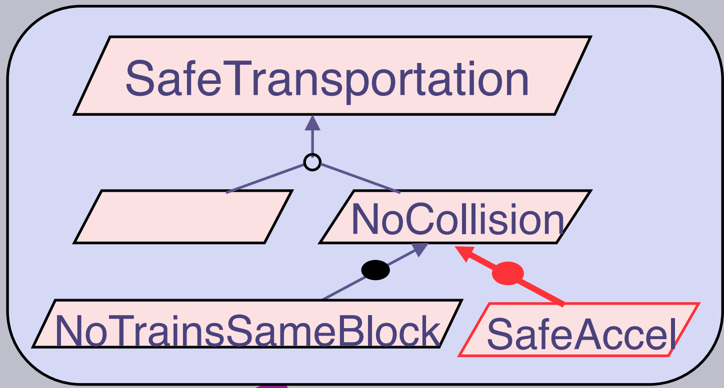




Edit View Tools Document Windows Help
Zoom: 160 %
(Goal) SafetyGoals Alt-F6
SafeTransportation
NoTrainCollision
DoorClosed WhileMoving
BlockSpeedLimited
NoTrainsOn Same Block
WorstCaseStopping DistanceMaintained
DoorsClosedIFFnonZeroSpeed Measure
MovingIFFnonZeroSpeed
see
(Goal) NoTrainsOn Same Block diagram
TrainController
Modified
36M of 92M

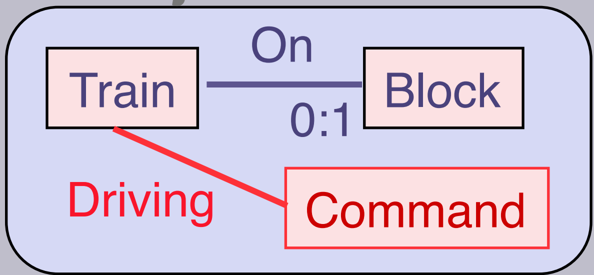
Goal-oriented model building

1. Domain analysis: refine/abstract goals



3. S2B analysis: enriched goals (alternatives)

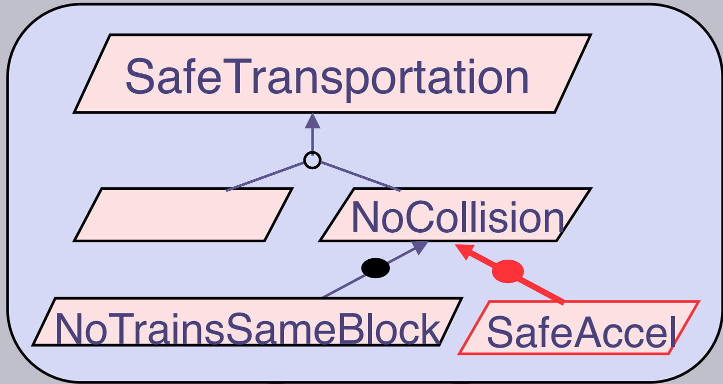
2. Domain analysis: derive/structure objects



4. System-to-be: enriched objects from new goals

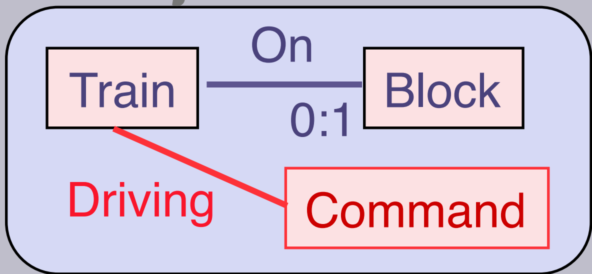
Goal-oriented model building

1. Domain analysis: refine/abstract goals

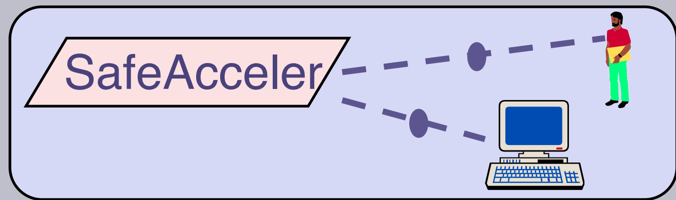


3. S2B analysis: enriched goals (alternatives)

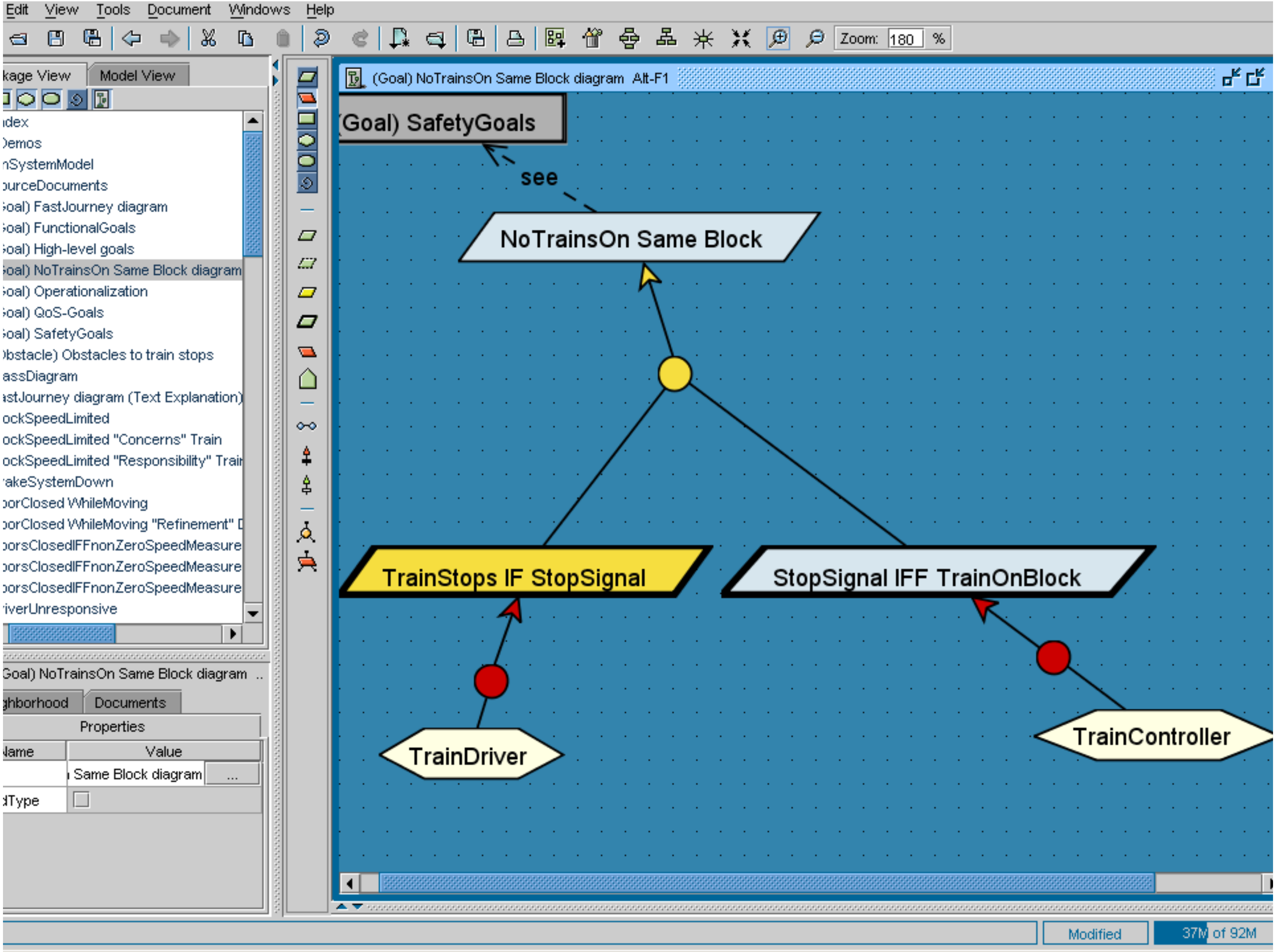
2. Domain analysis: derive/structure objects



4. S2B analysis: enriched objects from new goals



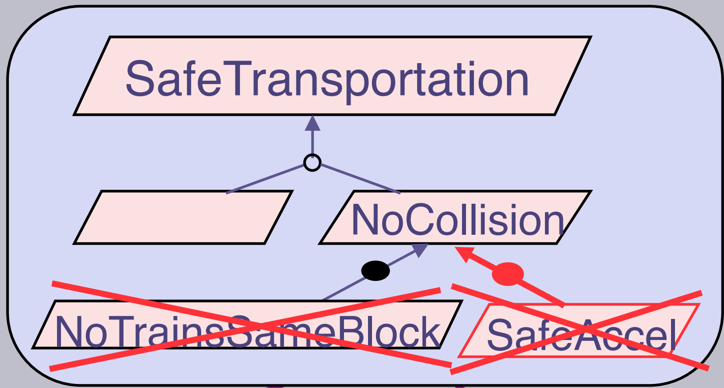
5. Responsibility analysis: agent assignment



Edit View Tools Document Windows Help
Zoom: 180 %
Model View
(Goal) NoTrainsOn Same Block diagram Alt-F1
(Goal) SafetyGoals
see
NoTrainsOn Same Block
TrainStops IF StopSignal
StopSignal IFF TrainOnBlock
TrainDriver
TrainController
Documents
Properties
Value
Same Block diagram
Modified
37M of 92M

Goal-oriented model building

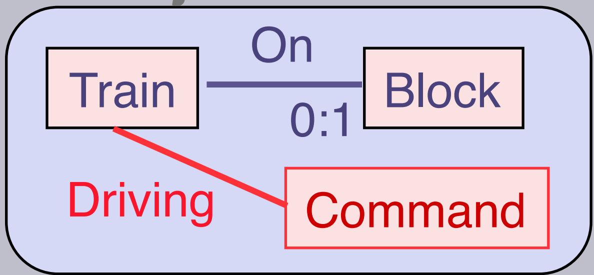
1. Domain analysis: refine/abstract goals



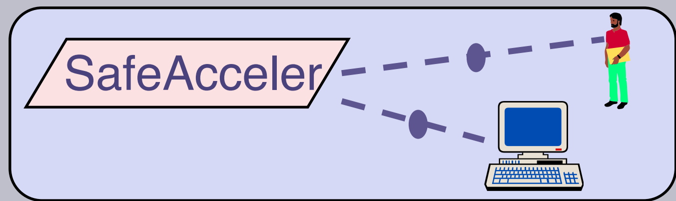
3. S2B analysis: enriched goals (alternatives)

2. Domain analysis: derive/structure objects

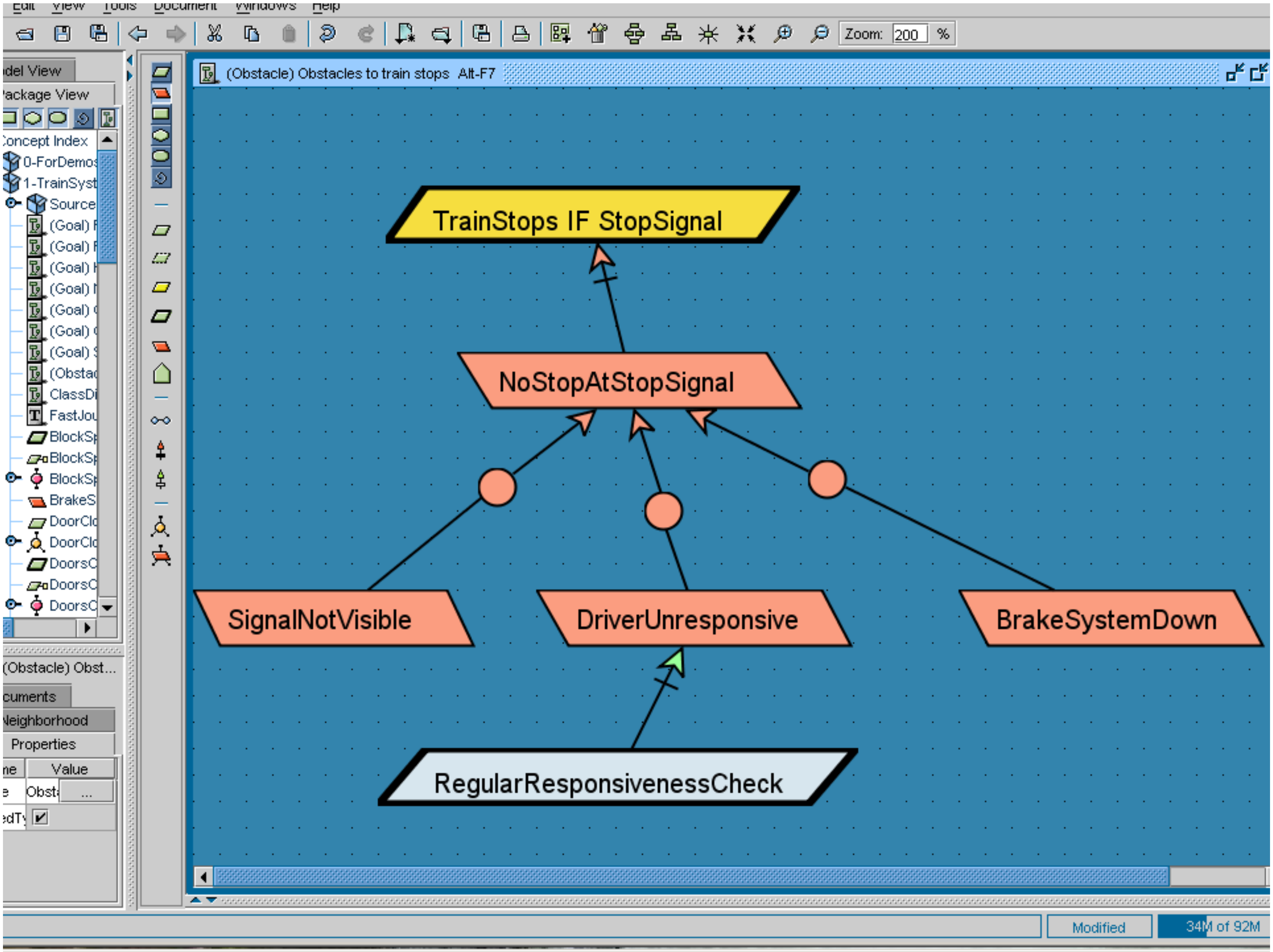
1-5 // Risk & conflict analysis



4. S2B analysis: enriched objects from new goals

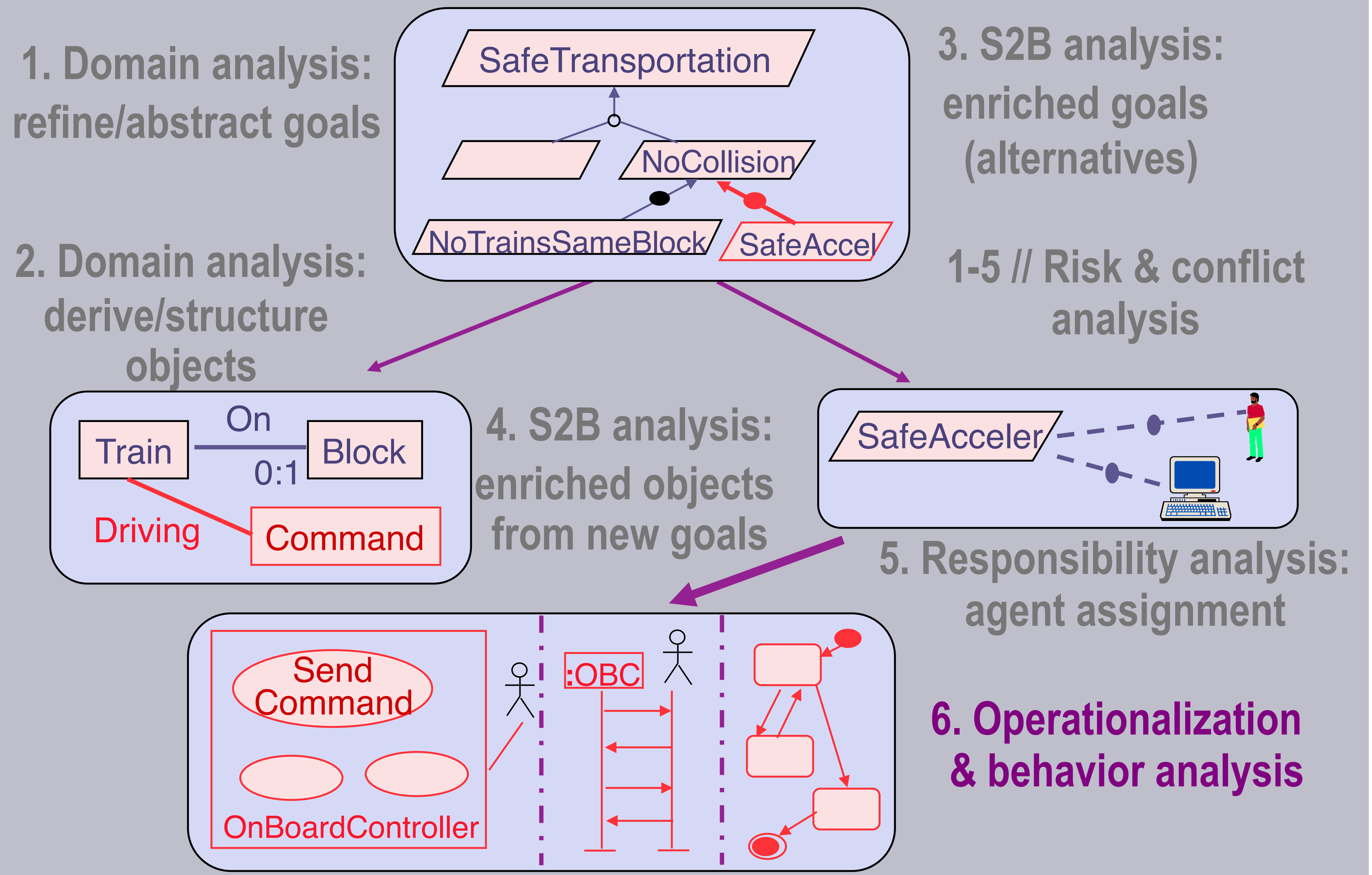


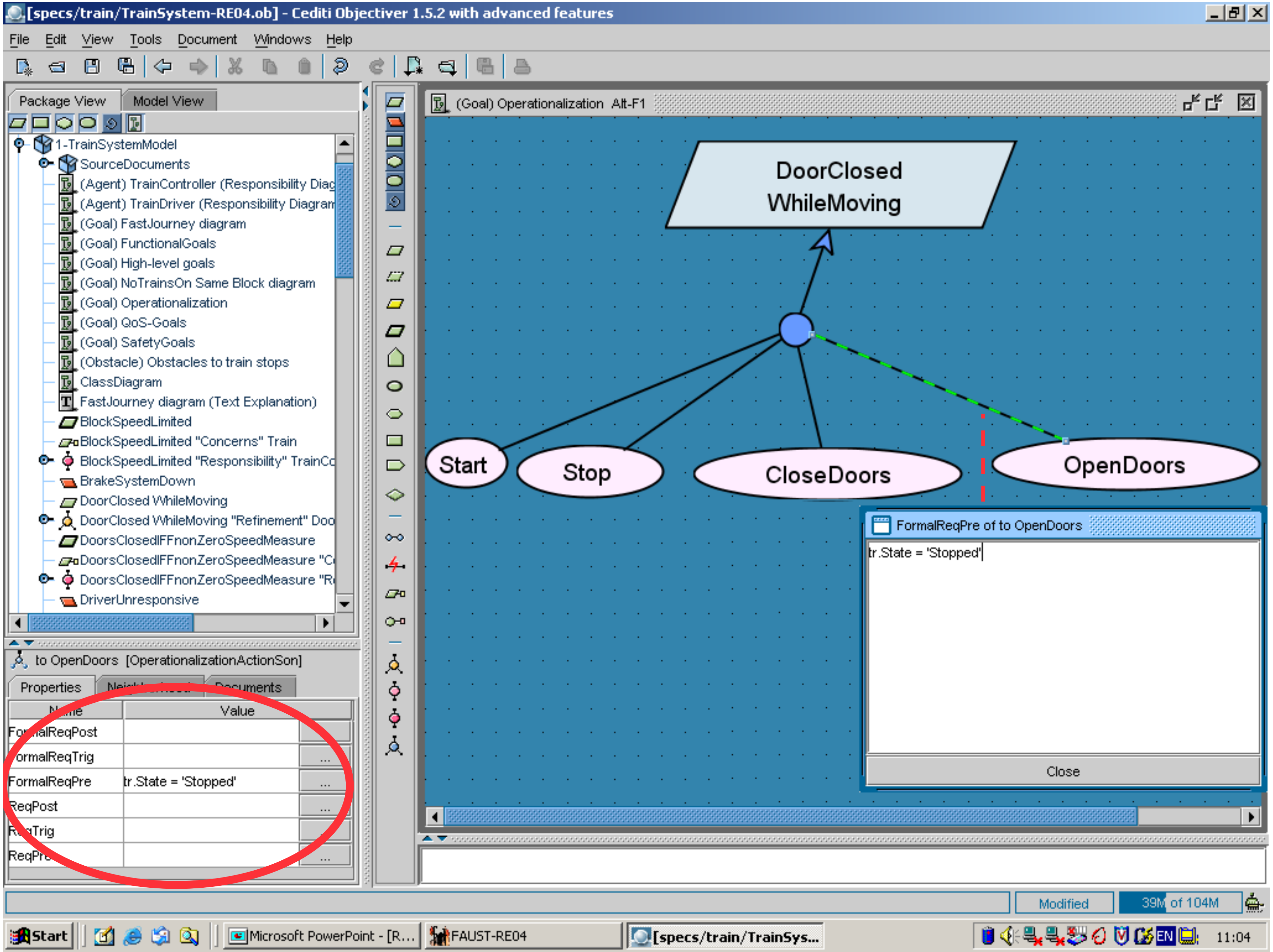
5. Responsibility analysis: agent assignment



(Obstacle) Obstacles to train stops Alt-F7
Zoom: 200 %
TrainStops IF StopSignal
NoStopAtStopSignal
SignalNotVisible
DriverUnresponsive
BrakeSystemDown
RegularResponsivenessCheck
Modified
34M of 92M

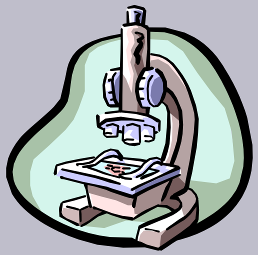
Goal-oriented model building




[specs/train/TrainSystem-RE04.ob] - Cediti Objectiver 1.5.2 with advanced features
File Edit View Tools Document Windows Help
Package View
Model View
1-TrainSystemModel
SourceDocuments
(Agent) TrainController (Responsibility Diag
(Agent) TrainDriver (Responsibility Diagram
(Goal) FastJourney diagram
(Goal) FunctionalGoals
(Goal) High-level goals
(Goal) NoTrainsOn Same Block diagram
(Goal) Operationalization
(Goal) QoS-Goals
(Goal) SafetyGoals
(Obstacle) Obstacles to train stops
ClassDiagram
FastJourney diagram (Text Explanation)
BlockSpeedLimited
BlockSpeedLimited "Concerns" Train
BlockSpeedLimited "Responsibility" TrainCo
BrakeSystemDown
DoorClosed WhileMoving
DoorClosed WhileMoving "Refinement" Doo
DoorsClosedIFFnonZeroSpeedMeasure
DoorsClosedIFFnonZeroSpeedMeasure "C
DoorsClosedIFFnonZeroSpeedMeasure "R
DriverUnresponsive
to OpenDoors [OperationalizationActionSon]
Properties
Documents
Value
FormalReqPre
tr.State = 'Stopped'
ReqPost
(Goal) Operationalization Alt-F1
DoorClosed WhileMoving
Start
Stop
CloseDoors
OpenDoors
FormalReqPre of to OpenDoors
tr.State = 'Stopped'
Close
Modified
of 104M
Start
Microsoft PowerPoint - [R...
FAUST-RE04
[specs/train/TrainSys...
11:04

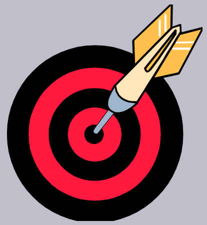
Outline

- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model items
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis



Specifying model items formally

- To support more accurate analysis & derivations
- Optional "button”: only **when** and **where** needed
- Declarative formalism for goals & domain properties
  - real-time temporal logic
- More operational formalism for operations
  - goal-oriented pre-/postconditions



Specifying goals

DoorsClosedWhileMoving

goal

annotation

Goal *Maintain* [DoorsClosedWhileMoving]

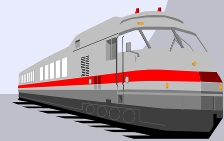
Def *All train doors shall be kept closed at any time when the train is moving*

FormalSpec $\forall$ tr: Train
tr.Speed $\neq$ 0 $\Rightarrow$ tr.DoorState = 'closed'

[Category Safety]

[Priority Highest]

[Source From interview with railway engineer X ...]



Some bits of real-time linear temporal logic

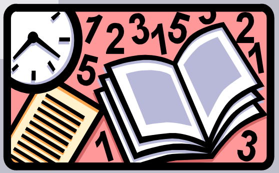
o P: P shall hold in the **immediately next** state

◊ P: P shall hold in **some future** state

□ P: P shall hold in **every future** state

P ***U*** N: P shall hold in every future state ***until*** N holds

P ***W*** N: P shall hold in every future state ***unless*** N holds



Some bits of real-time linear temporal logic (2)

Propositional connectives

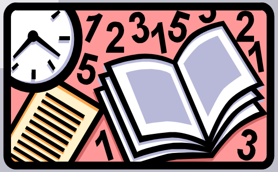
$\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$

First-order language

quantifiers on *object instance* variables $\forall$, $\exists$

$P \Rightarrow Q$: $\Box\,(P \rightarrow Q)$

$P \Leftrightarrow Q$: $\Box\,(P \leftrightarrow Q)$



Some bits of real-time linear temporal logic (3)

Real-time constructs:

$\Box_{\leq T}$ P: P shall hold in every future state **up to *T* time units**

$\Diamond_{\leq T}$ P: P shall hold **within T time units**

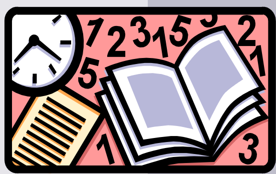
Operators on past:

• P: P did hold in the previous state (right before)

♦ P, ■ P, P ***S*** O, P ***B*** O: always *P* **since/back to** *O*

$♦_{\leq T}$ P, $■_{\leq T}$ P, etc

@ P = • (¬ P) ∧ P



Interpretation over historical state sequences

H: historical sequence of states (behavior)

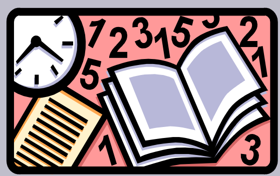
i: time position (time is isomorphic to naturals)

$(H, i) \models \circ P$ iff $(H, next(i)) \models P$

smallest time unit

$(H, i) \models \Diamond P$ iff $(H, j) \models P$ for some $j \geq i$

$(H, i) \models \Box P$ iff $(H, j) \models P$ for all $j \geq i$

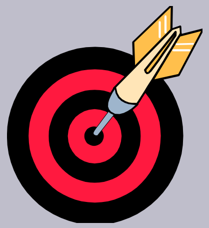


Interpretation over historical state sequences (2)

$(H, i) \models P\ \mathcal{U}\ N$ iff $(H, j) \models N$ for some $j \geq i$
and $(H, k) \models P$ for all k: $i \leq k < j$

$(H, i) \models P\ \mathcal{W}\ N$ iff $(H, i) \models P\ \mathcal{U}\ N$ or $(H, i) \models \Box P$

$(H, i) \models \Diamond_{\leq T} P$ iff $(H, j) \models P$ for some $j \geq i$
with dist $(i,j) \leq T$



Specifying goals: examples

DoorsClosedBetweenPlatforms

goal

annotation

Goal *Maintain* [DoorsClosedBetweenPlatforms]

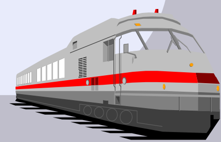
Def *All train doors shall be kept closed at any time between two successive platforms*

FormalSpec .... **?**

[Category Safety]

[Priority Highest]

[Source From interview with railway engineer X ...]



Specifying goals: examples

DoorsClosedBetweenPlatforms *goal*

Goal *Maintain* [DoorsClosedBetweenPlatforms]

Def *All train doors shall be kept closed at any time between two successive platforms*

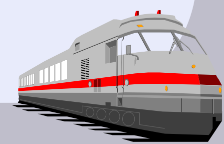
FormalSpec $\forall$ tr: Train, pl: Platform

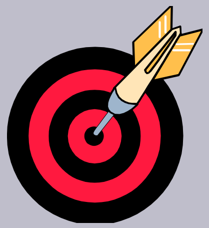
$$At\ (tr, pl) \land \circ \neg At\ (tr, pl) \Rightarrow tr.Doors = \text{"closed"}\ \mathcal{W}\ At\ (tr, next(pl))$$

[Category Safety]

[Priority Highest]

[Source From interview with railway engineer X ...]





Specifying goals: examples

Achieve [FastJourneyBetweenPlatforms]

annotation

Goal *Achieve* [FastJourneyBetweenPlatforms]

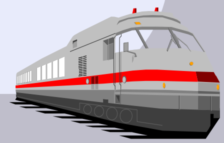
Def *A train shall reach the next platform from the current one within T time units*

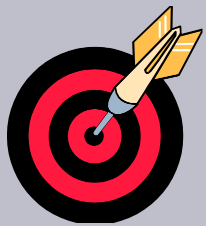
FormalSpec …. **?**

[Category …]

[Priority …]

[Source …]





Specifying goals: examples

Achieve [FastJourneyBetweenPlatforms]

Goal *Achieve* [FastJourneyBetweenPlatforms]

Def *A train shall reach the next platform from the current one within T time units*

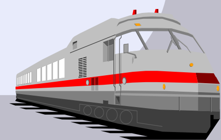
FormalSpec $\forall$ tr: Train, pl: Platform

At (tr, pl) $\Rightarrow$ $\Diamond_{\leq T}$ At (tr, next (pl))

[Category Safety]

[Priority Highest]

[Source From interview with railway engineer X ...]

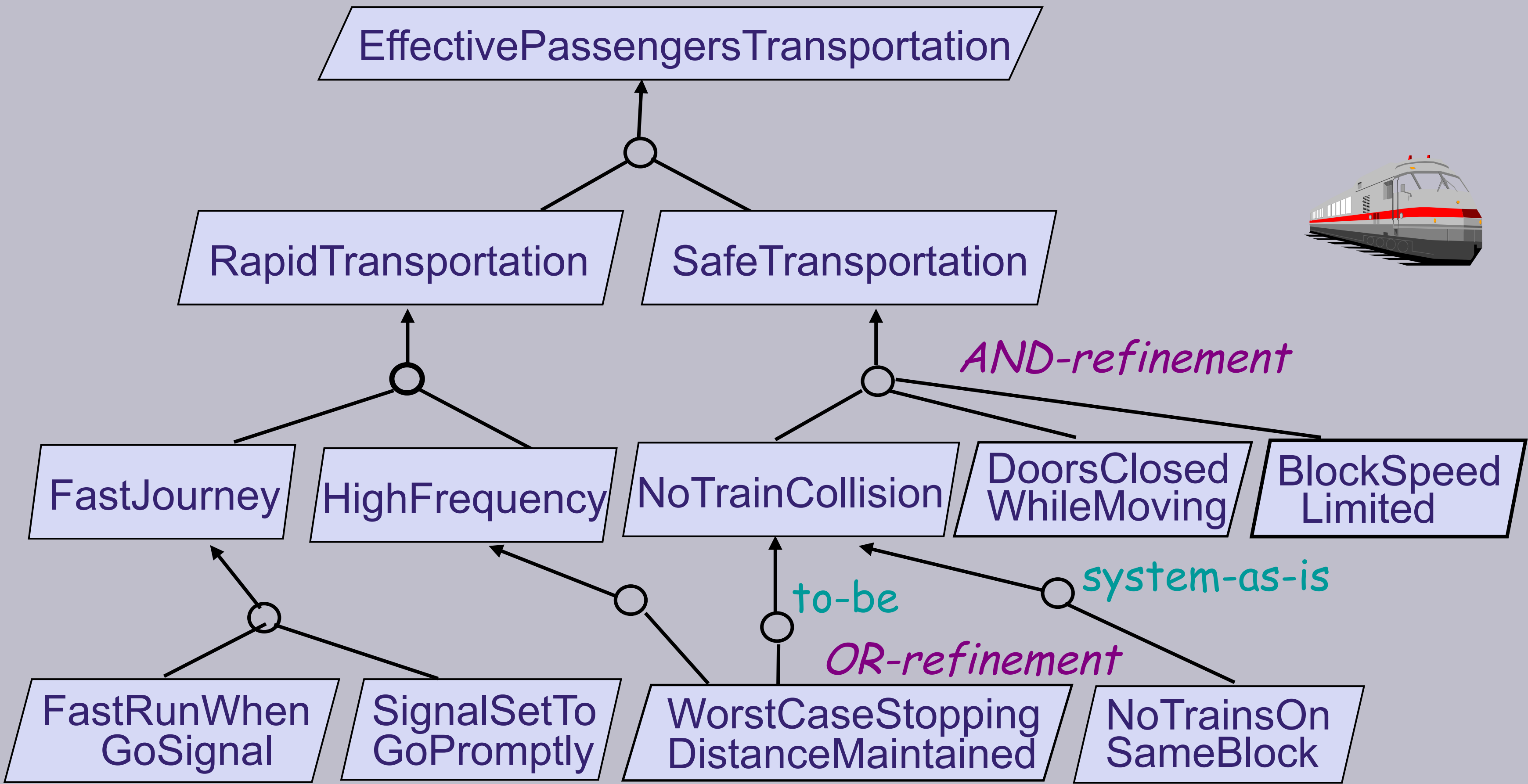


Outline

- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - *Assessing the likelihood & severity of obstacles*
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

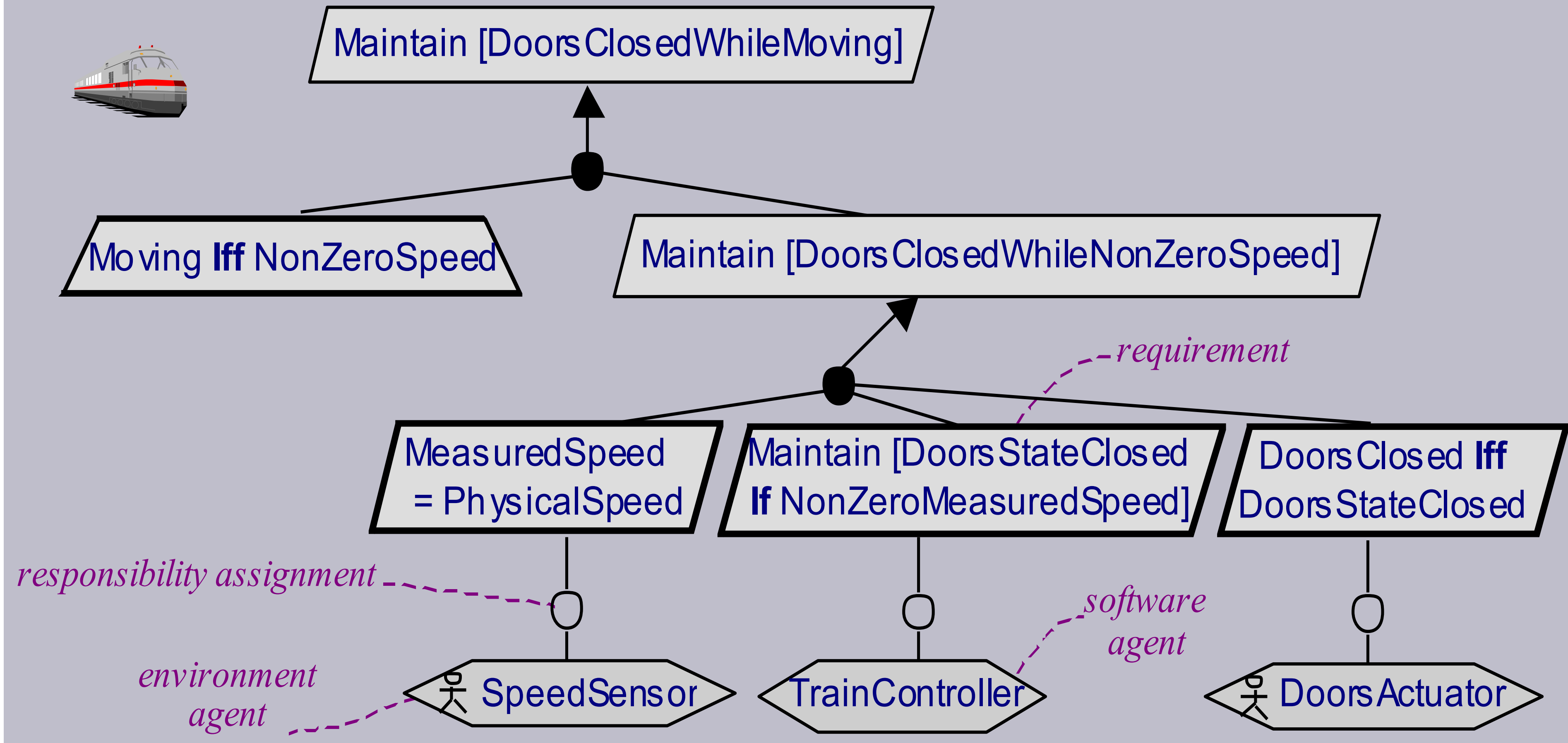
A goal model is an AND/OR graph

- Goals are recursively refined/abstracted



A goal model is an AND/OR graph (2)

- Leaf nodes = goals assignable to single system agents



AND-refinements

- **AND-refinement** of goal G into subgoals $SG_1, \ldots, SG_n$ means:

  G can be satisfied by satisfying $SG_1, \ldots, SG_n$

- AND-refinements should be ...
  - **complete**: $\{SG_1, \ldots, SG_n, \text{Dom}\} \models G$

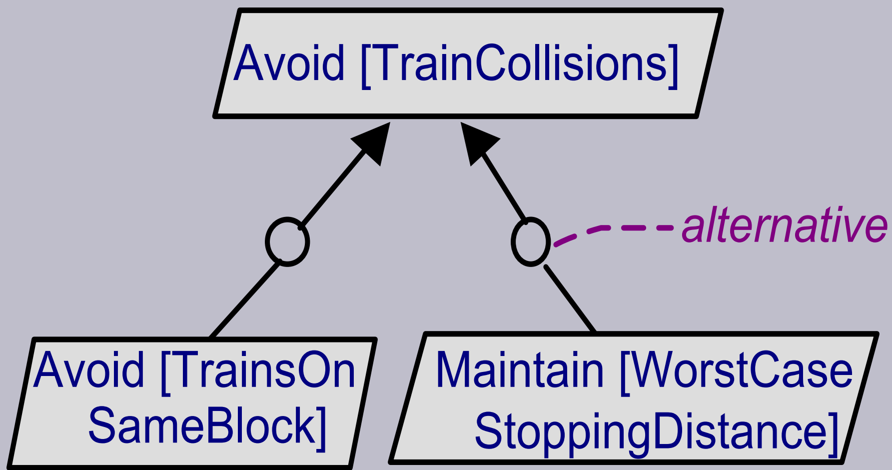
    essential for requirements completeness
  - **consistent**: $\{SG_1, \ldots, SG_n, \text{Dom}\} \not\models \mathit{false}$
  - **minimal**: $\{SG_1, \ldots, SG_{j-1}, SG_{j+1}, \ldots, SG_n, \text{Dom}\} \not\models G$

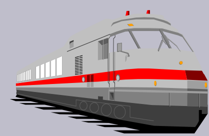
    to avoid unnecessarily restrictive requirements/expectations

OR-refinements

- **OR-refinement** of goal *G* into refinements $R_1, ..., R_m$ means:
  G can be satisfied by satisfying all subgoals from
  any of the alternative refinements R_i

- Alternative goal refinements yield different options
  (system variants)
  - pros/cons to be evaluated against soft goals for selection





Checking goal refinements

- Aim: show that refinements are correct and complete

  Subgoals, Assumptions, DomainProps $\vdash$ ParentGoal

- (Approach 1: use theorem prover)

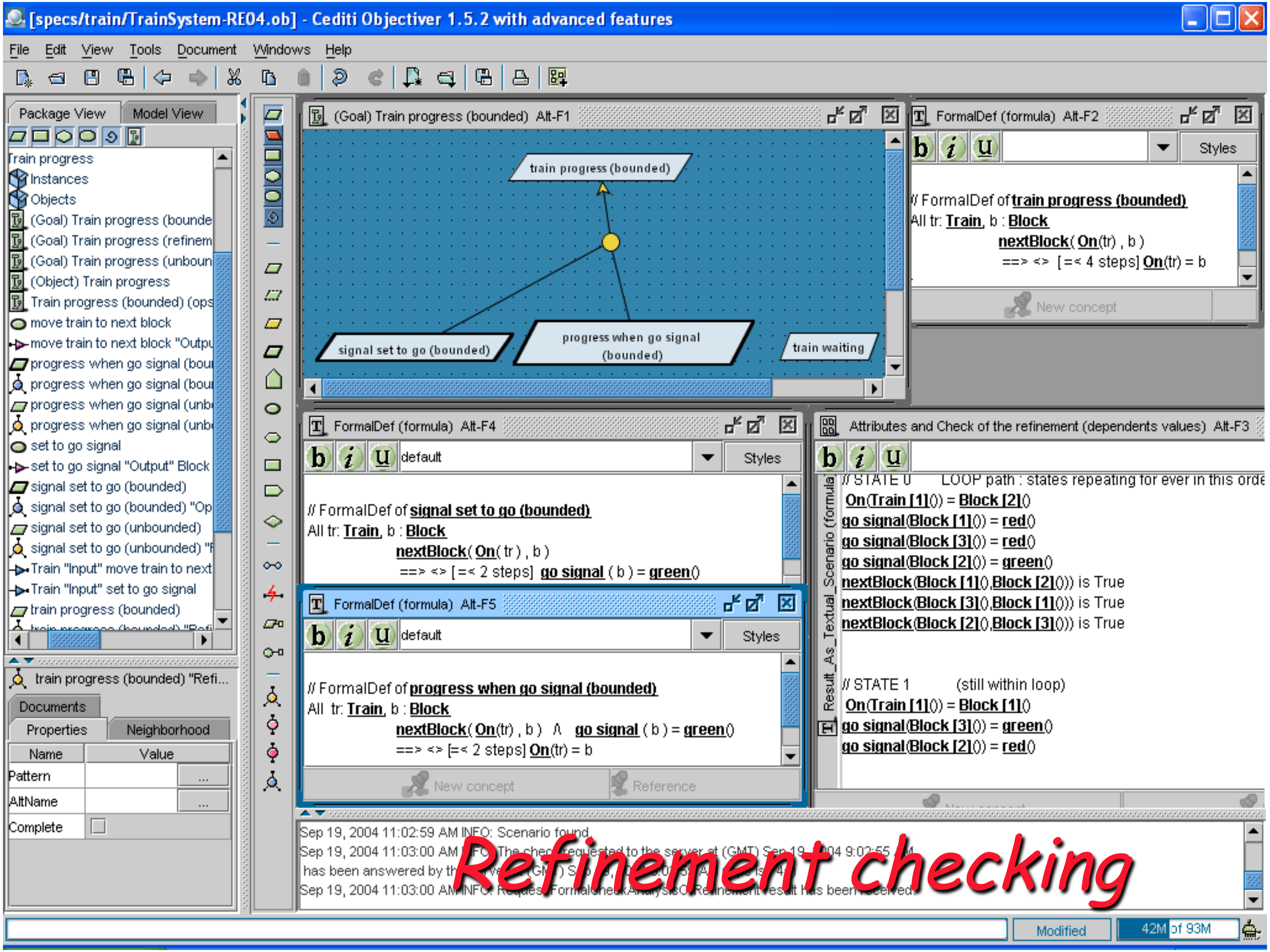
  heavyweight, non-constructive

- Approach 2: front end to bounded SAT solver
  - incremental check/debug of goal model fragments
  - on selected object instances (propositionalization)

  Input: $SubG_1 \wedge \ldots \wedge SubG_n \wedge Dom \wedge \neg\, ParentGoal$

  Output: OK

  KO + counter-example scenario

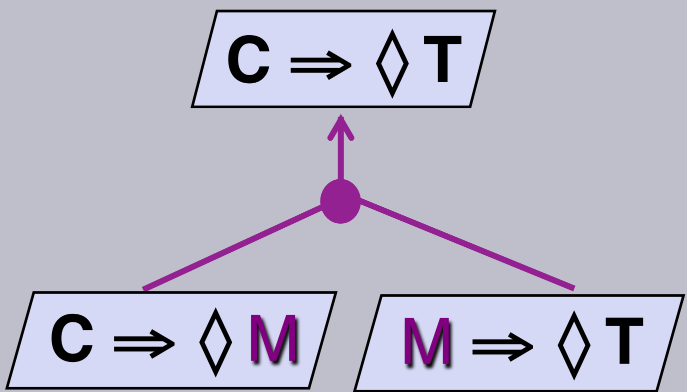


Refinement checking

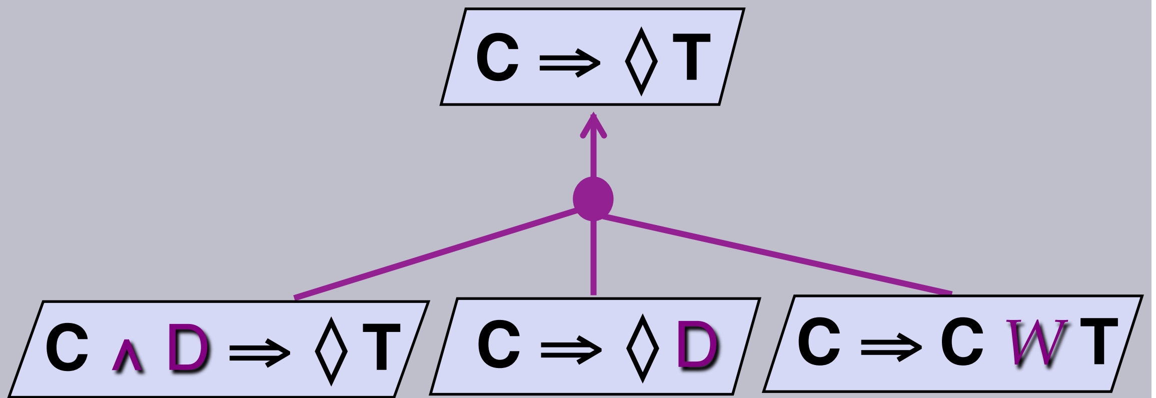
Approach 3: reuse refinement patterns

- Catalogue of patterns encoding *refinement tactics*
- Generic refinements proved formally, once for all
- Reuse through instantiation, in matching situation

Can be used informally (natural language templates)

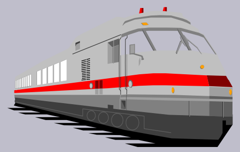


milestone-driven



guard introduction

Checking goal refinements with patterns



Achieve [TrainProgress]

$On(tr, b) \Rightarrow \Diamond\, On(tr, next(b))$

missing subgoal !!
detectable automatically

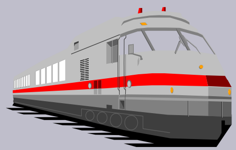
Achieve [ProgressWhenGo]

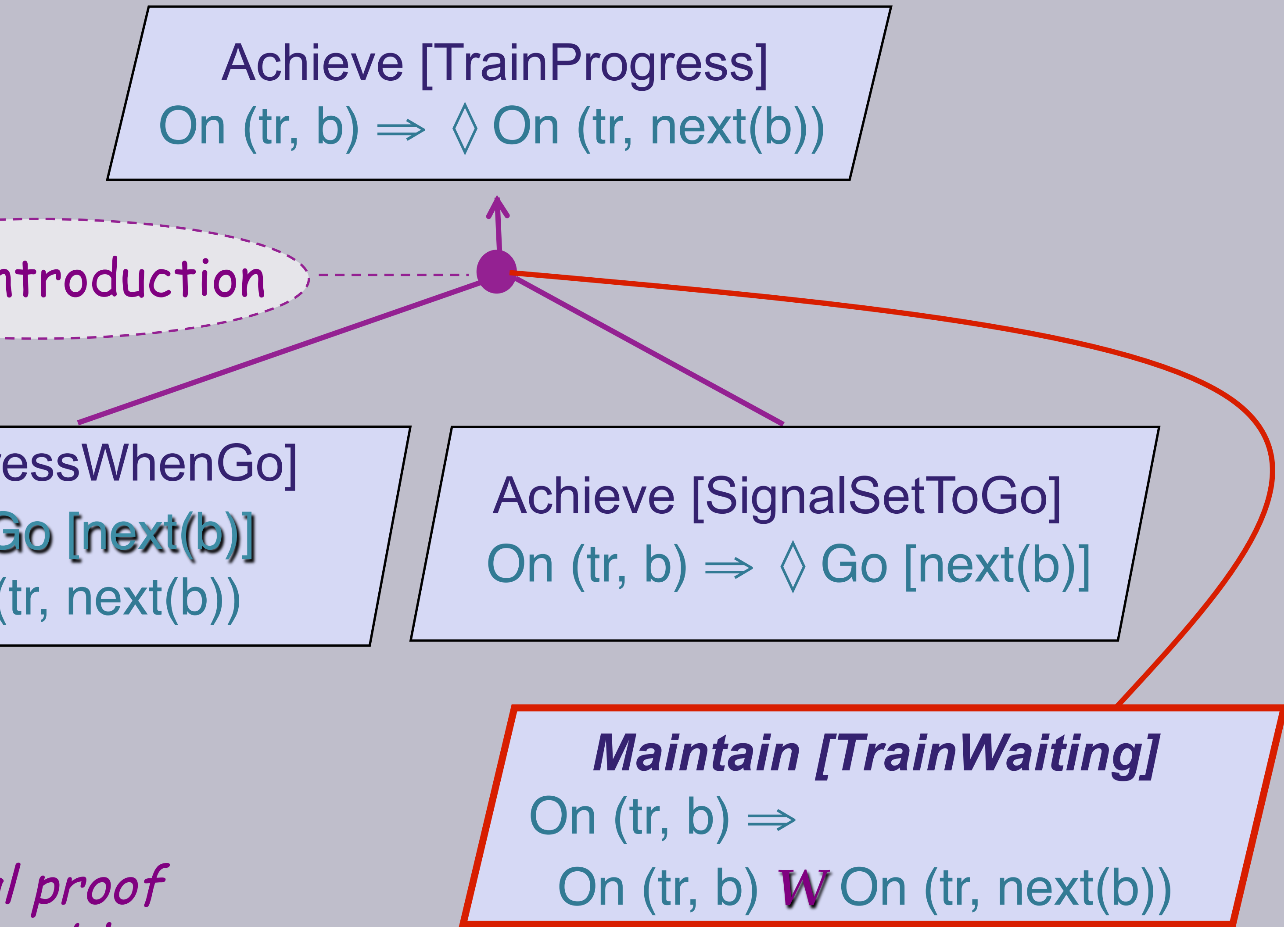
$On(tr, b) \wedge Go[next(b)] \Rightarrow \Diamond\, On(tr, next(b))$

Achieve [SignalSetToGo]

$On(tr, b) \Rightarrow \Diamond\, Go[next(b)]$

Checking goal refinements with patterns

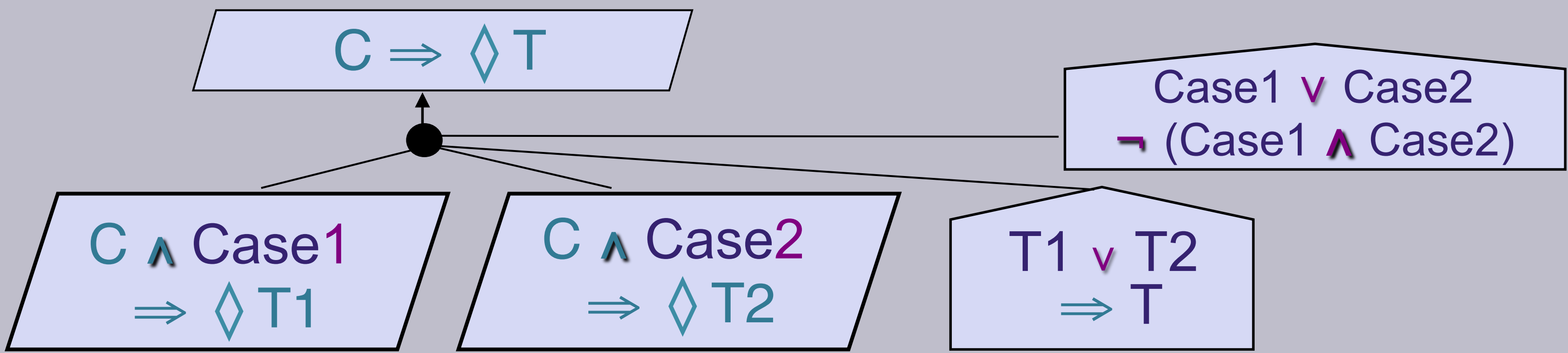




mathematical proof
hidden, reusable

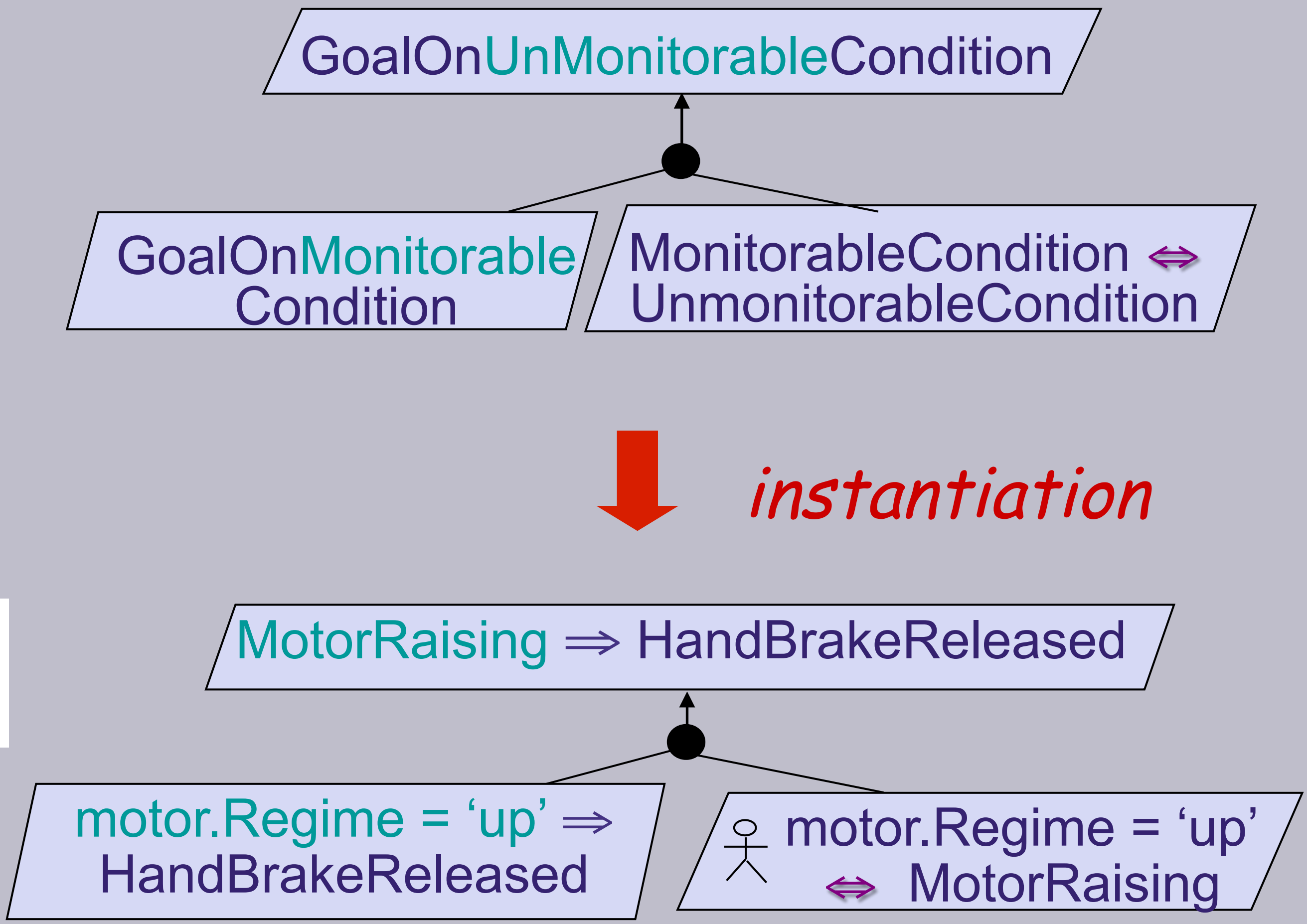
Some other frequent patterns

- Refinement **by case**
  - applicable when the goal satisfaction space can be partitioned into cases (disjoint, covering all possibilities)

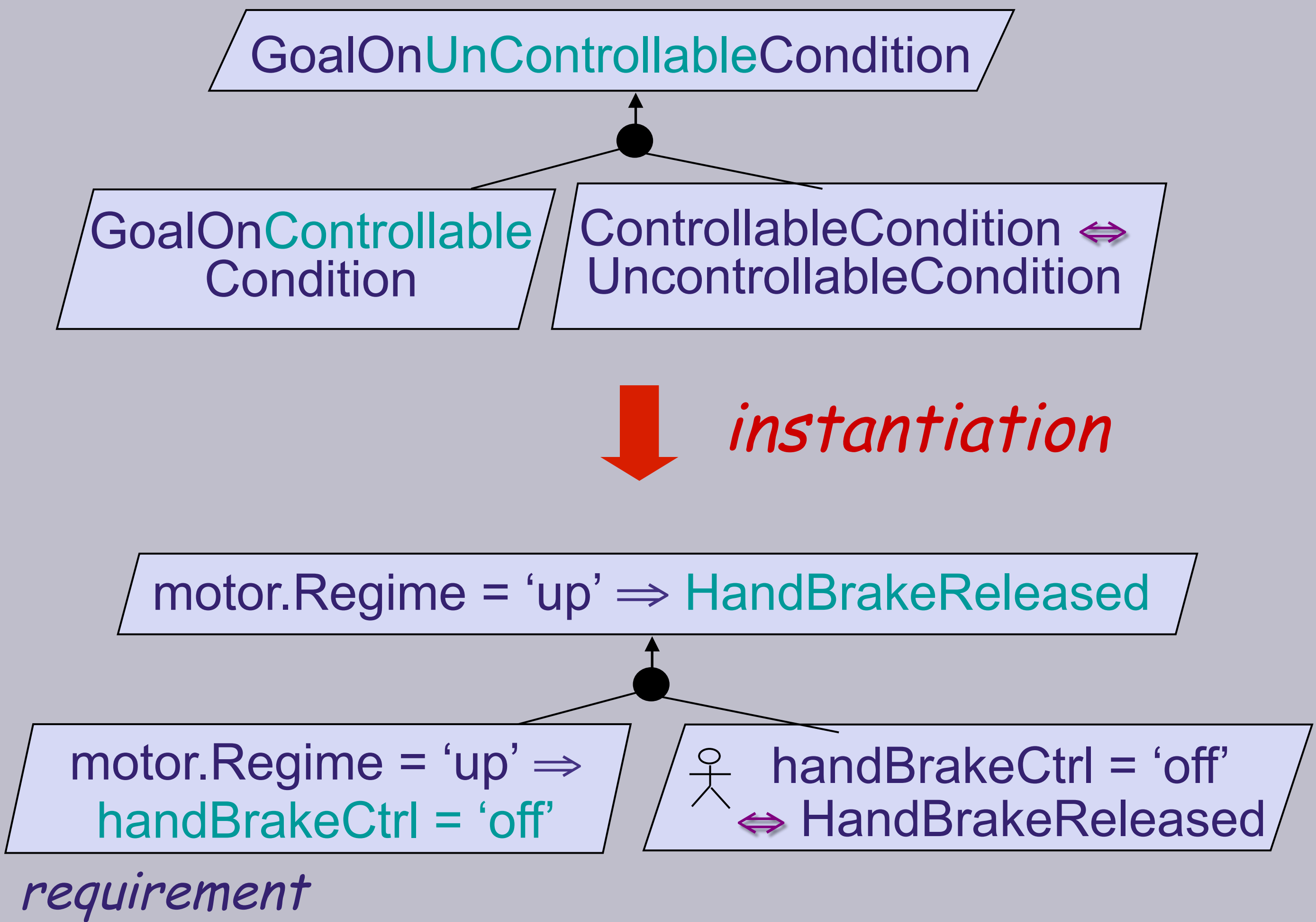


(Similar pattern for *Maintain* goals)

Other frequent patterns … (2)



Other frequent patterns … (3)



Patterns can be used for operationalization

proved correct once for all

G: $C \Rightarrow \circ T$

Operation Op1
- **DomPre** $\neg$ T
- **DomPost** T
- **ReqTrig for** *G:* C

Operation Op2
- **DomPre** T
- **DomPost** $\neg$ T
- **ReqPre for** *G:* $\neg$ C

Operationalization pattern: example

HighWaterSignal = 'On' ⇒ o PumpSwitch = 'On'

C: HighWaterSignal = 'On'
T: PumpSwitch = 'On'

Operation Op1
DomPre ¬ **T**
DomPost **T**
ReqTrig for *G*: *C*

Operation Op2
DomPre **T**
DomPost ¬ **T**
ReqPre for *G*: ¬ *C*

Operationalization pattern: example

HighWaterSignal = 'On' $\Rightarrow \circ$ PumpSwitch = 'On'

C: HighWaterSignal = 'On'
T: PumpSwitch = 'On'

Operation SwitchPump*On*
DomPre PumpSwitch ≠ On
DomPost PumpSwitch = On
ReqTrig for *RootGoal*
HighWaterSignal = 'On'

Operation SwitchPump*Off*
DomPre PumpSwitch = On
DomPost PumpSwitch ≠ On
ReqPre for *RootGoal*
HighWaterSignal ≠ 'On'

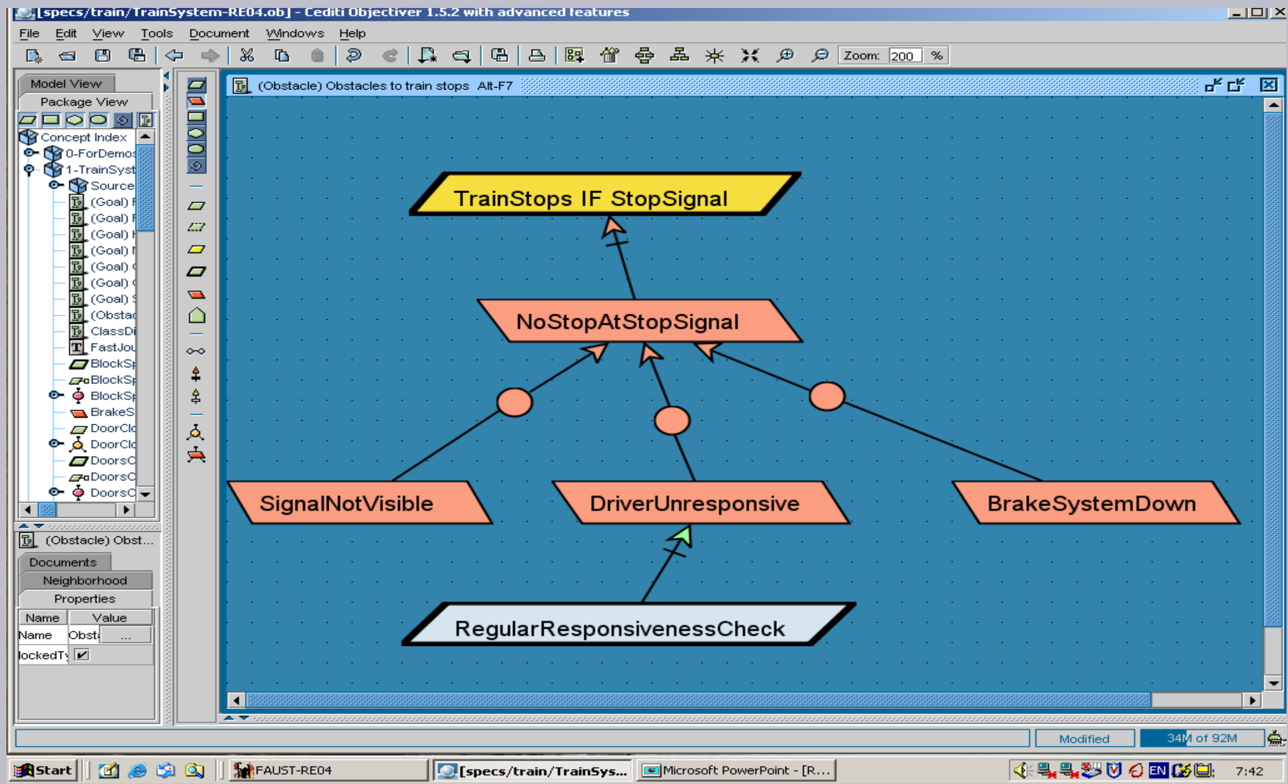
Outline

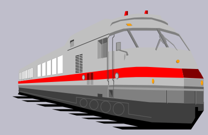
- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

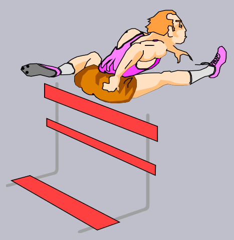
Obstacle analysis for risk-driven RE

- Motivation: goals in refinement graph are often too ideal,
  likely to be violated under abnormal conditions
  (unintentional *or* intentional agent behaviors)

- Risk analysis can be anchored on goal models



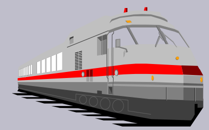




What are obstacles ?

- **Obstacle** to goal = condition on system for goal violation
  - {*O*, *Dom* } |= ¬ *G*   *obstruction*
  - {*O*, *Dom* } |≠ **false**   *domain consistency, obstacle satisfiability*

e.g. *G:* StopSignal ⇒ TrainStopsAtBlockSignal



Dom: TrainStopsAtStopSignal ⇒ DriverResponsive

O: ◊ (StopSignal ∧ ¬ DriverResponsive)

- For behavioral goal: existential property capturing unadmissible behavior (**negative** scenario)

[van Lamsweerde & Letier, TSE'2000]

Completeness of a set of obstacles

- Ideally, a set of obstacles to G should be complete

  $\{\neg O_1, \ldots, \neg O_n, Dom\} \models G$ *domain completeness*

  e.g.

  DriverResponsive ∧ ¬ *BrakeSystemDown* ∧ *SignalVisible* ∧ StopSignal

  ⇒ TrainStopsAtBlockSignal ???

- Completeness is highly desirable for mission-critical goals
  - but bounded by what we know about the domain

- Obstacle analysis may help elicit relevant domain properties

Obstacle categories for heuristic identification

Correspond to goal categories & their refinement ...

- **Hazard** obstacles obstruct **Safety** goals
- **Threat** obstacles obstruct **Security** goals
  - Disclosure, Corruption, DenialOfService, ...

- **Inaccuracy** obstacles obstruct **Accuracy** goals
- **Misinformation** obstacles obstruct **Information** goals
  - NonInformation, WrongInformation, TooLateInformation, ...
- **Dissatisfaction** obstacles obstruct **Satisfaction** goals
  - NonSatisfaction, PartialSatisfaction, TooLateSatisfaction, ...
- **Unusability** obstacles obstruct **Usability** goals
- ...

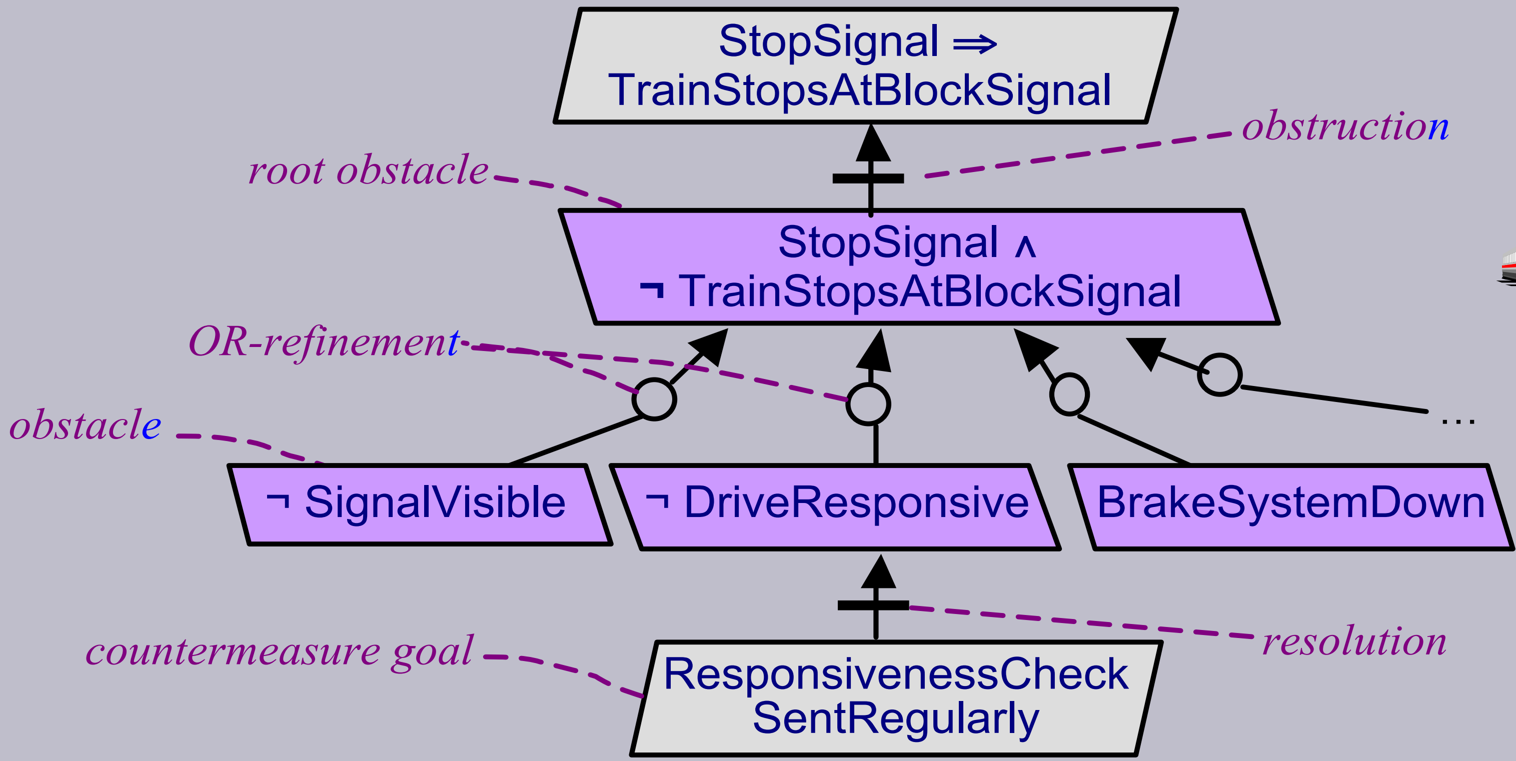
Obstacle refinement

- **AND**-refinement of obstacle *O* should be ...
  - complete: $\{subO_1, ..., subO_n, Dom\} \models O$
  - consistent: $\{subO_1, ..., subO_n, Dom\} \not\models \textbf{false}$
  - minimal: $\{subO_1, ..., subO_{j-1}, subO_{j+1}, ..., subO_n, Dom\} \models O$
- **OR**-refinement of obstacle *O* should be ...
  - entailments: $\{subO_i, Dom\} \models O$
  - domain-consistent: $\{subO_i, Dom\} \not\models \textbf{false}$
  - domain-complete: $\{\neg subO_1, ..., \neg subO_n, Dom\} \models \neg O$
  - disjoint: $\{subO_i, subO_j, Dom\} \models \textit{false}$
- **If** $subO_i$ OR-refines *O* **and** *O* obstructs *G*

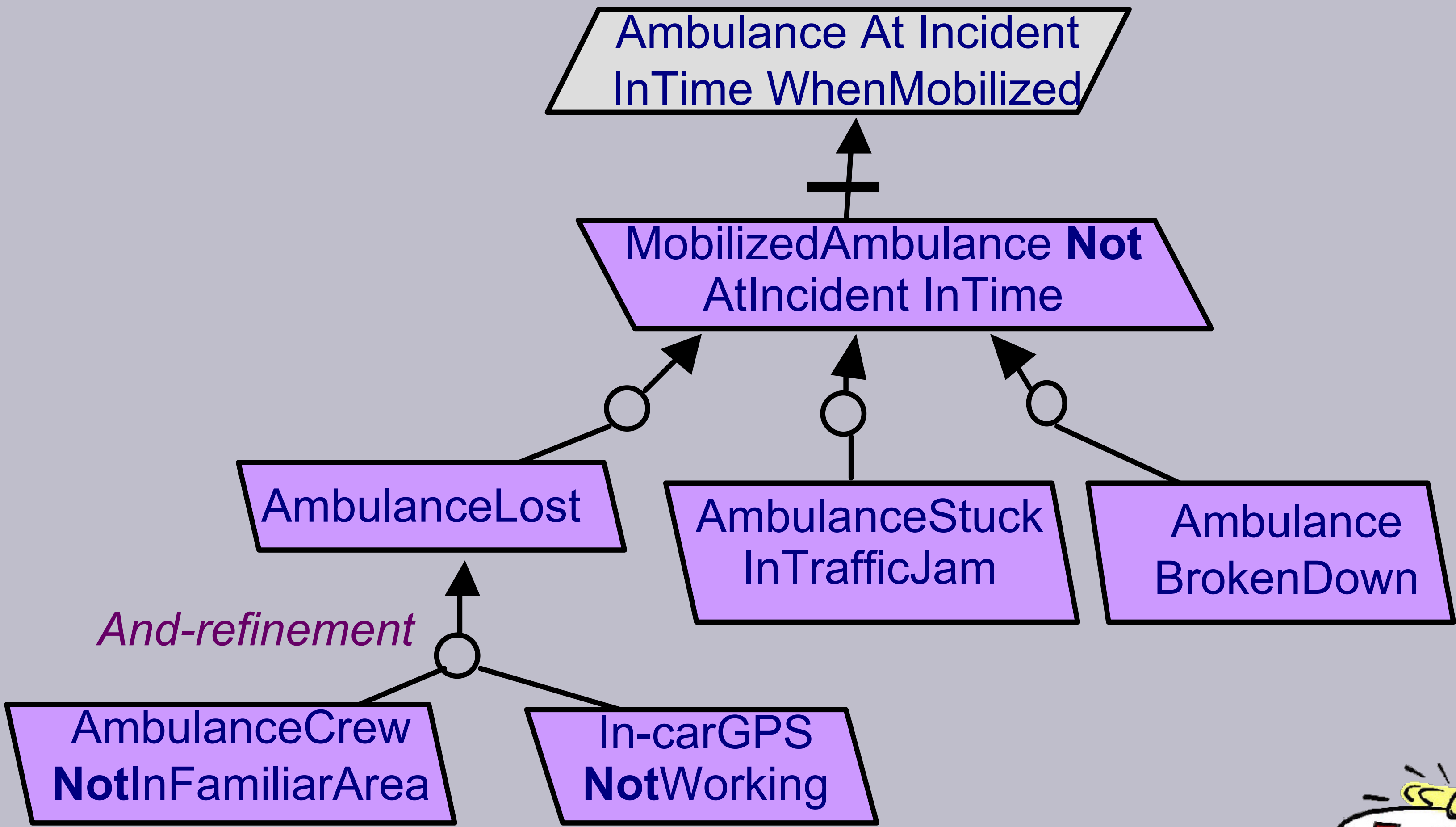
  then $subO_i$ obstructs *G*

Obstacle diagrams as AND/OR refinement trees

- Anchored on leafgoals in goal model
  - **root**: ¬ *G*
  - obstacle **AND/OR**-refinement: same semantics as goals
  - **leaf** obstacles: feasibility, likelihood, resolution easier to determine



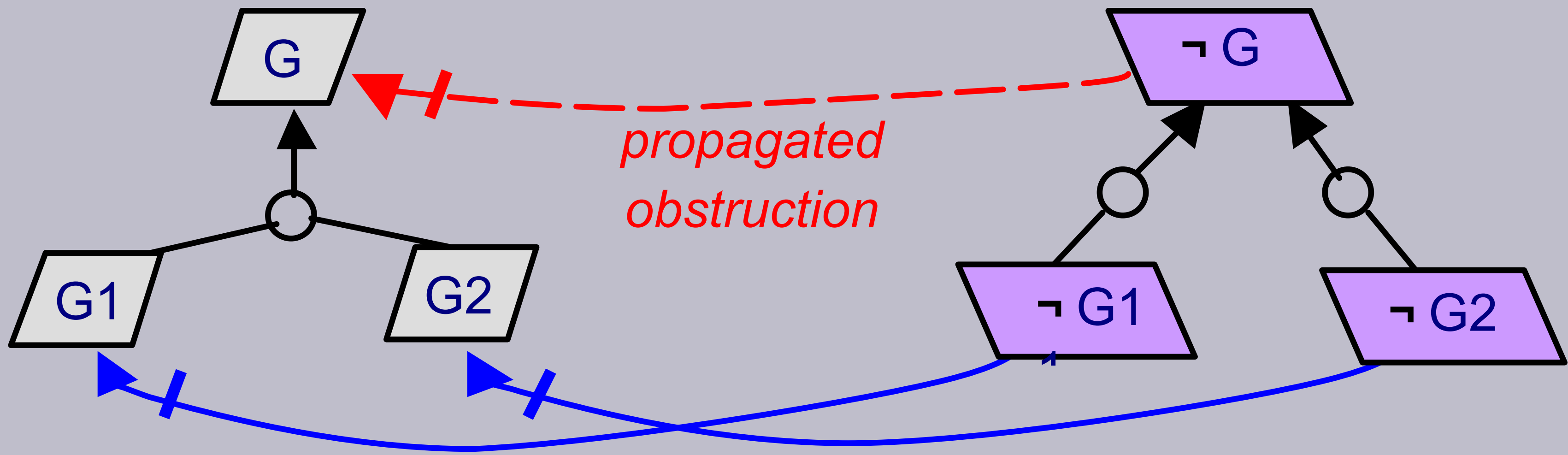
Obstacle diagrams as AND/OR refinement trees (2)



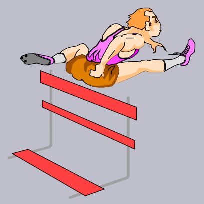
can be used informally

Obstructions propagate bottom-up in goal AND-refinement trees

- Cf. De Morgan's law: ¬ (G1 ∧ G2) equivalent to ¬ G1 ∨ ¬ G2



=> *Severity of* ***consequences*** *of an obstacle can be assessed in terms of higher-level goals obstructed*



Obstacle analysis for increased system robustness

- Anticipate obstacles ...
  - ⇒ more realistic goals,
    new goals as countermeasures to abnormal conditions
  - ⇒ more complete, realistic goal model

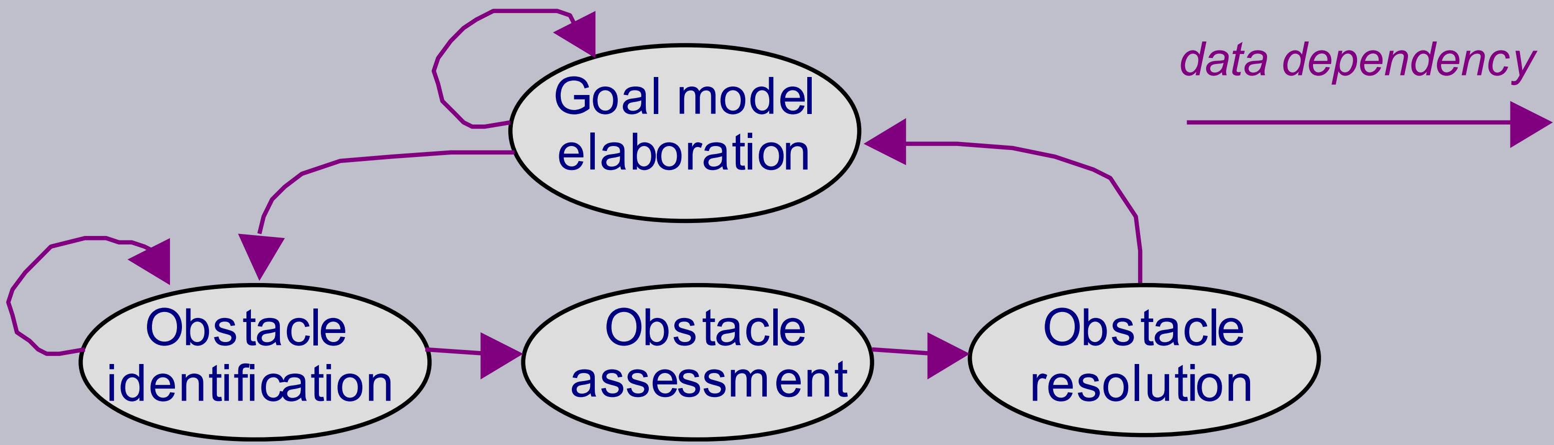
- **Obstacle analysis:**

  For selected goals in the goal model ...

  - identify as many obstacles to it as possible;
  - assess their likelihood & severity;
  - resolve them according to likelihood & severity

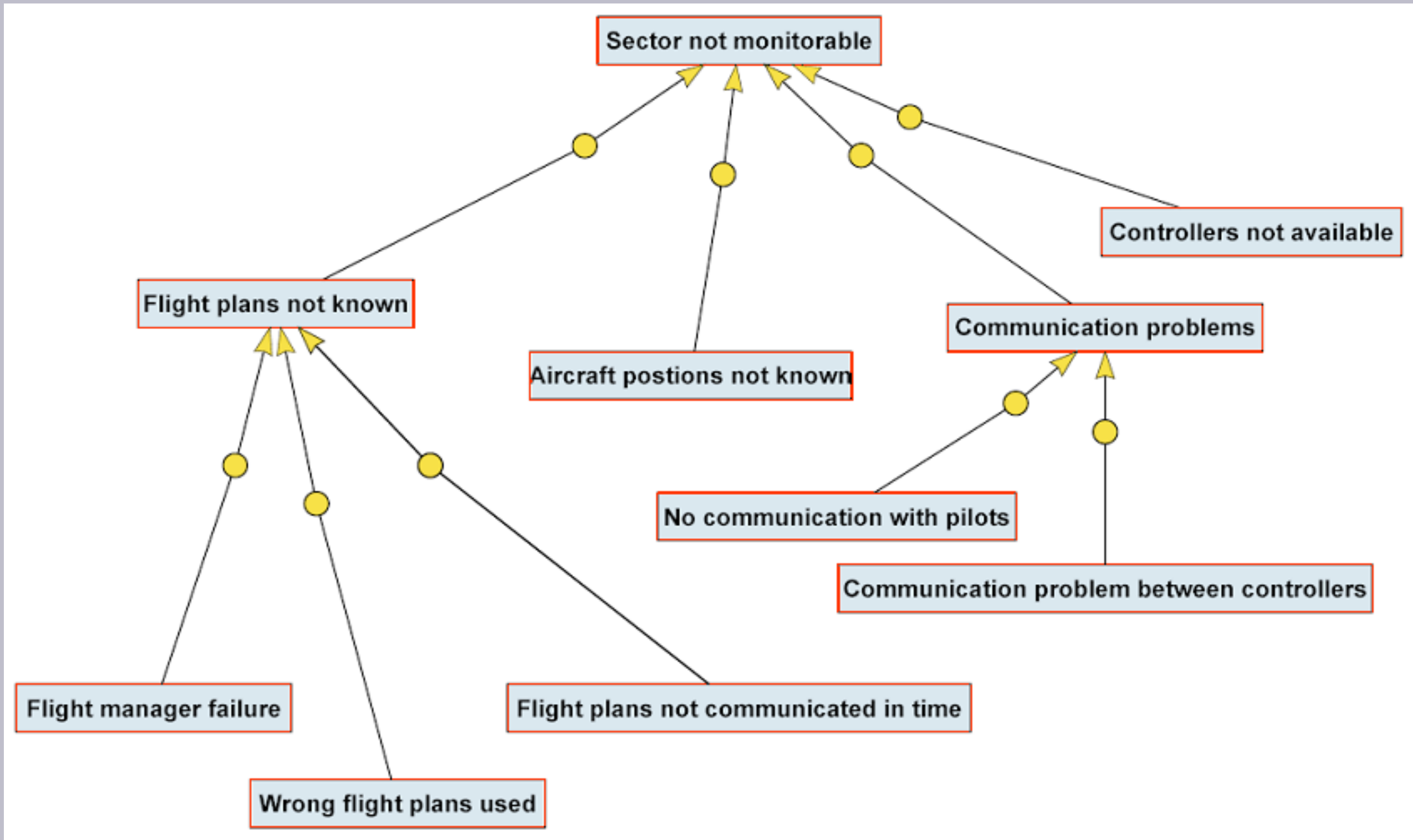
=> *new goals as countermeasures in the goal model*

Obstacle analysis & goal model elaboration are intertwined



- Goal-obstacle analysis loop terminates when remaining obstacles can be tolerated
  - unlikely or acceptable consequences
- Which goals to consider in the goal model?
  - **leafgoals** (requirements or expectations): easier to find how to break finer-grained goals
  - mission-critical goals

Obstacle analysis : a motivating example



Real air traffic control project, CEDITI, completed **March 2002**

Uberlingen
mid-air collision, **July 2002**

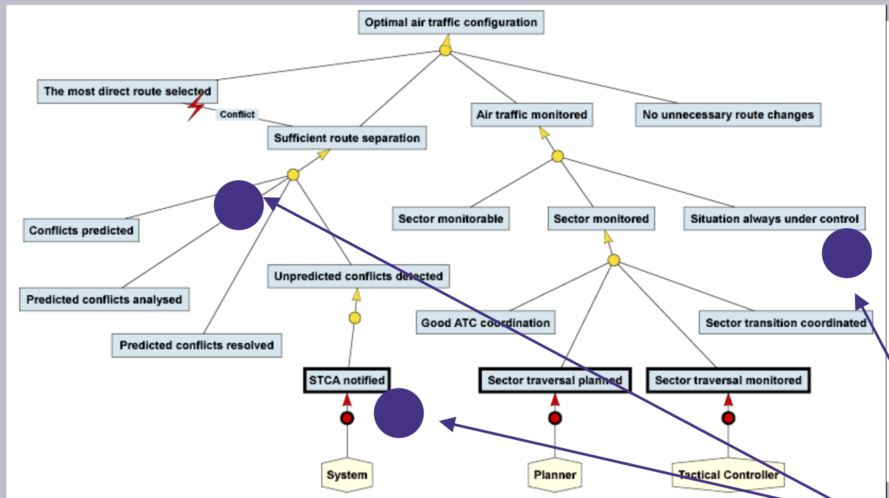
Facts

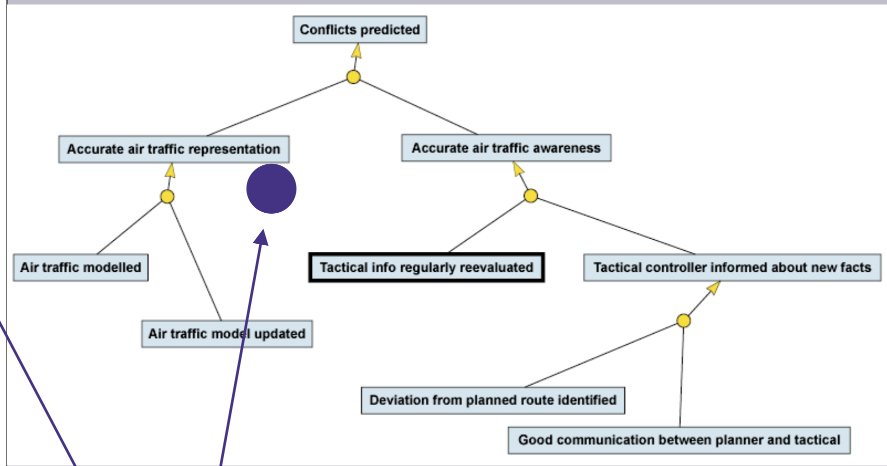
- July 1st 2002, southern Germany
- DHL Boeing 757 x Russian Tu-154
- 71 people killed, incl. 52 children

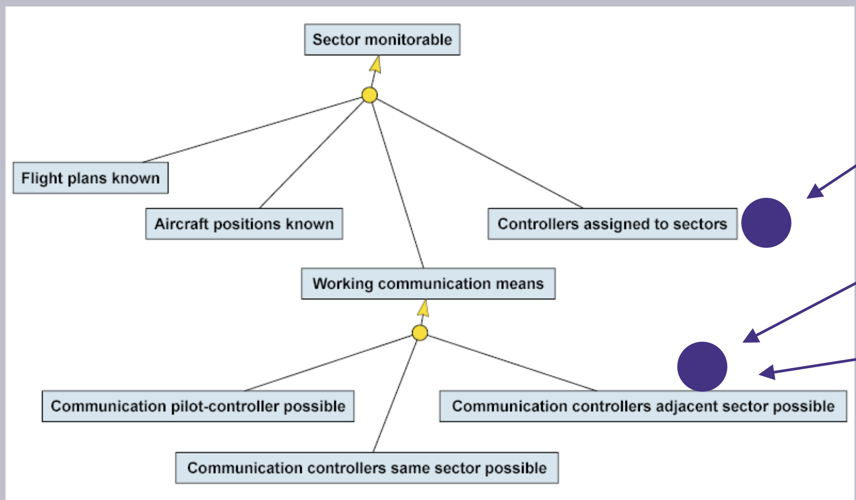
Preliminary analysis shows:

- STCA out of order at Swiss ATC
- Only 1 controller on duty at crash time (the other one was taking a break) → controller overloaded
- Problem between air traffic handover between Switzerland and Germany for another flight landing
- German ATC failed to call Swiss ATC
- Conflict between Tu's TCAS embedded system and tower's order
- Pilot choice: Tower's order prior to TCAS
- Discrepancies between screen displays and radar traces

Obstacle analysis : a motivating example







- STCA out of order at Swiss ATC
- Only 1 controller on duty at crash time (the other one was taking a break) → controller overloaded
- Problem between air traffic handover between Switzerland and Germany for another flight
- German ATC failed to call Swiss ATC
- Conflict between Tu's TCAS embedded system and tower's order
- Pilot choice: Tower's order prior to TCAS
- Discrepancies between screen displays and radar traces

Outline

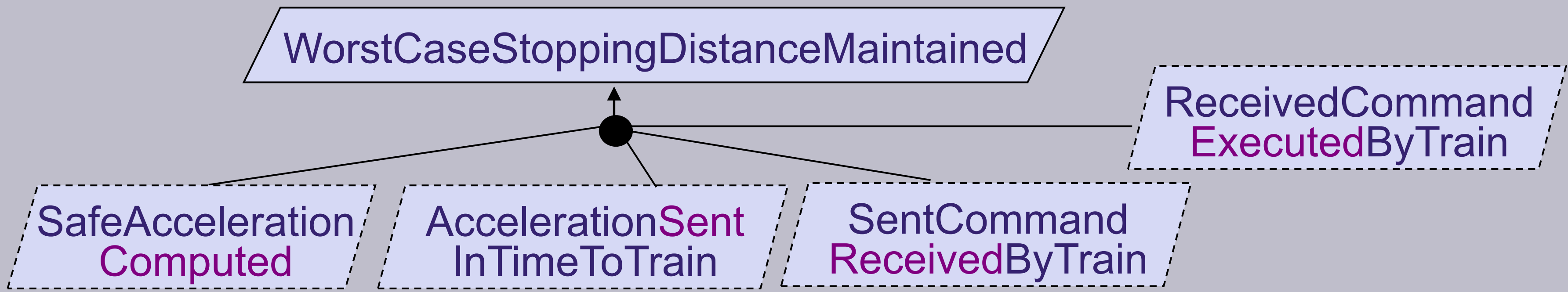
- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

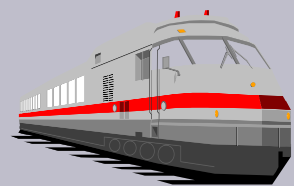
Obstacle identification

- For obstacle to goal *G* ...
  - negate *G*;
  - find as many AND/OR refinements of ¬ *G* as possible in view of domain properties ...
  - ... until reaching obstruction preconditions
    - that are *feasible* by the environment of the agents assigned to *G*
    - whose likelihood & severity is easy to assess

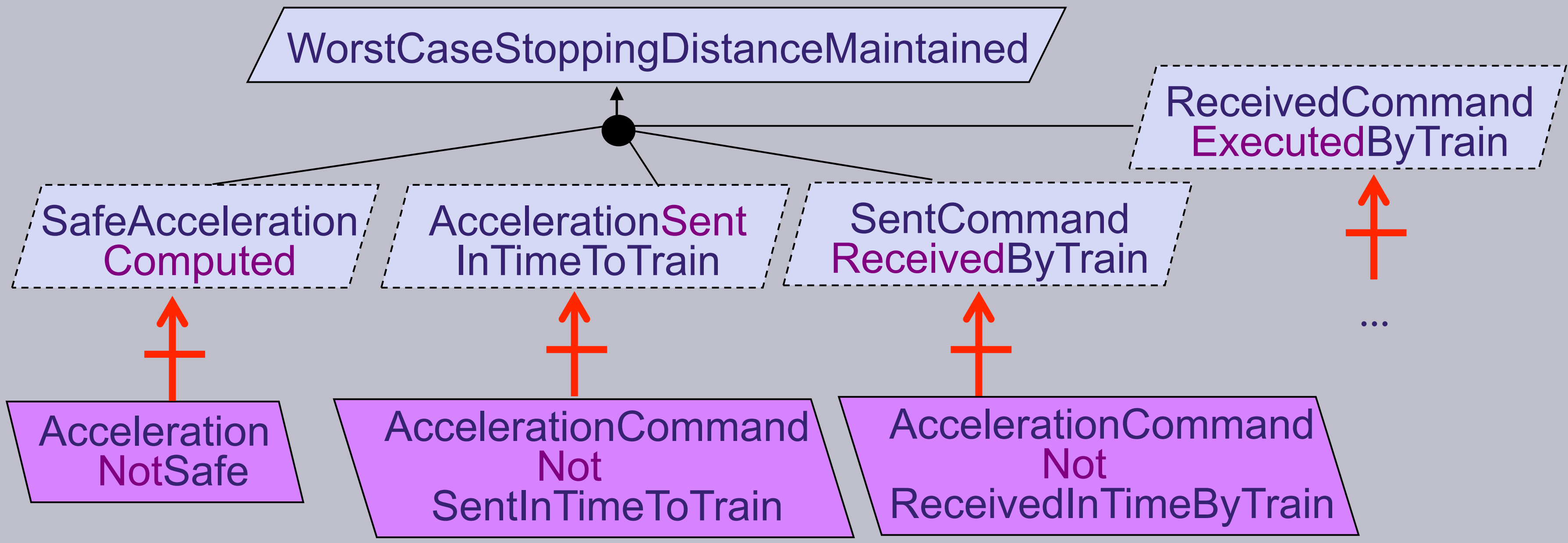
= goal-anchored construction of **fault-tree**

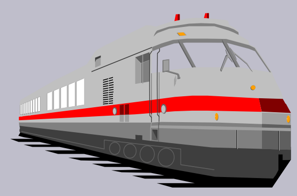
Obstacle identification: informal example



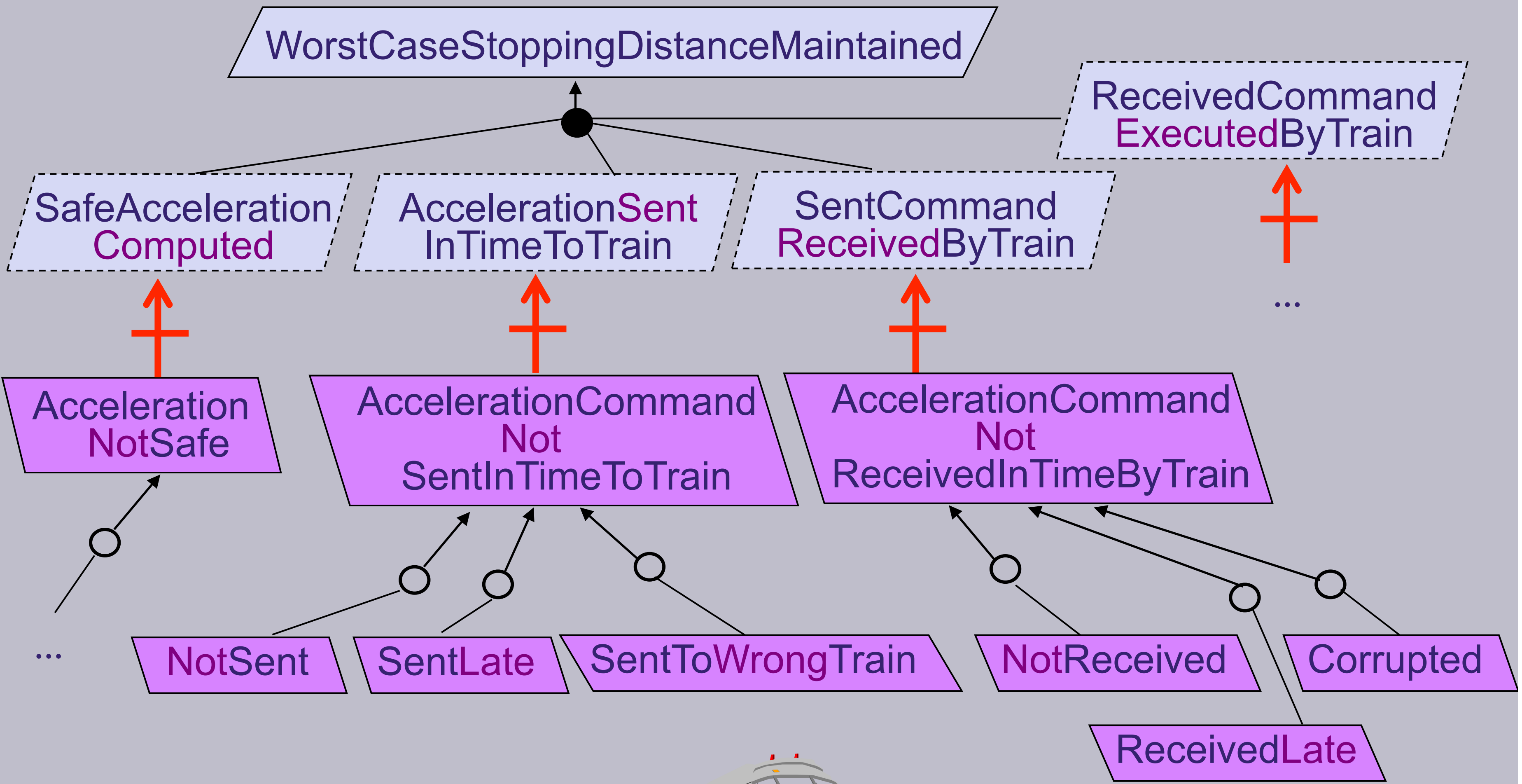


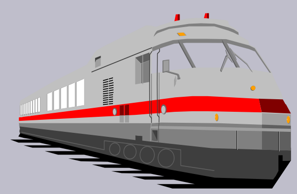
Obstacle identification: informal example





Obstacle identification: informal example





Can we identify obstacles systematically?

- The problem: generate obstacles O such that

$$O,\ Dom \vdash \neg G$$

$$Dom \nvDash \neg O$$

- Various techniques available ...
  - tautology-based refinement from $\neg G$
  - regression of $\neg G$ through Dom
  - reuse of formal obstruction patterns
  - combine model checking and inductive learning

Generating obstacles: tautology-based refinement

- Take goal negation as root
- Use tautologies to drive refinements

e.g.

$\neg (A \wedge B)$ *equiv* $\neg A \vee \neg B$

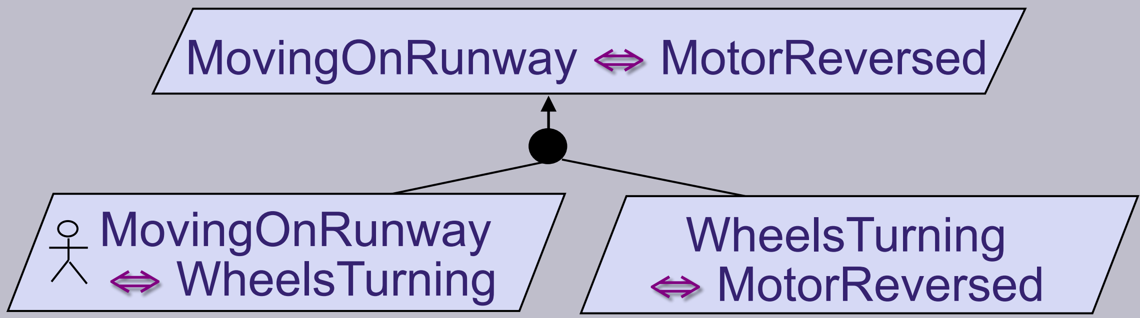
$\neg (A \vee B)$ *equiv* $\neg A \wedge \neg B$

$\neg (A \Rightarrow B)$ *equiv* $A \wedge \neg B$

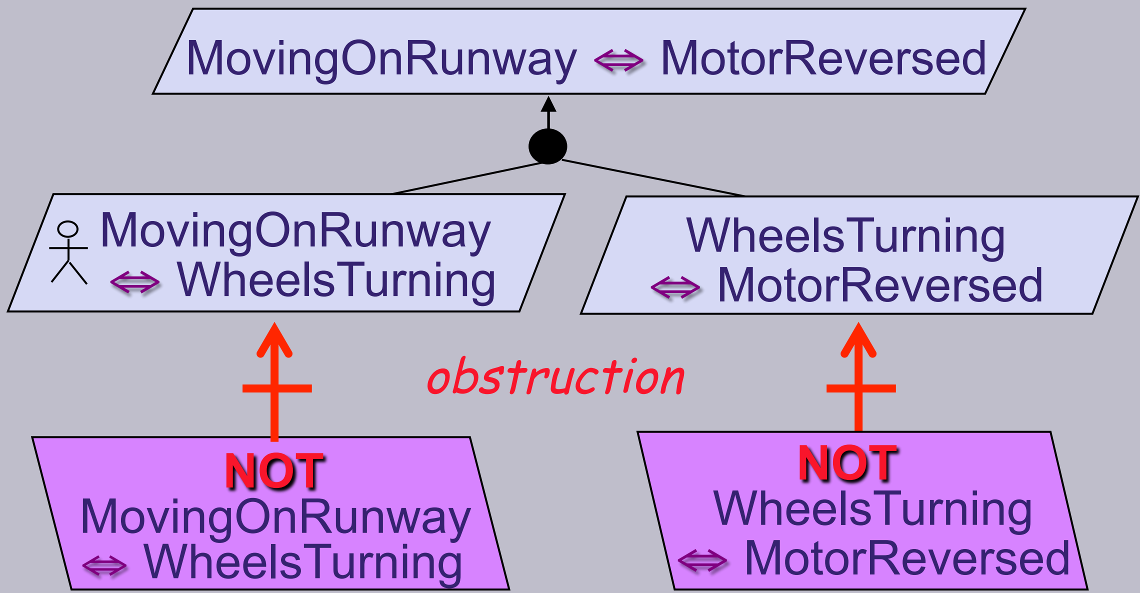
$\neg (A \Leftrightarrow B)$ *equiv* $(A \wedge \neg B) \vee (\neg A \wedge B)$

=> complete OR-refinements when **v**-connective gets in

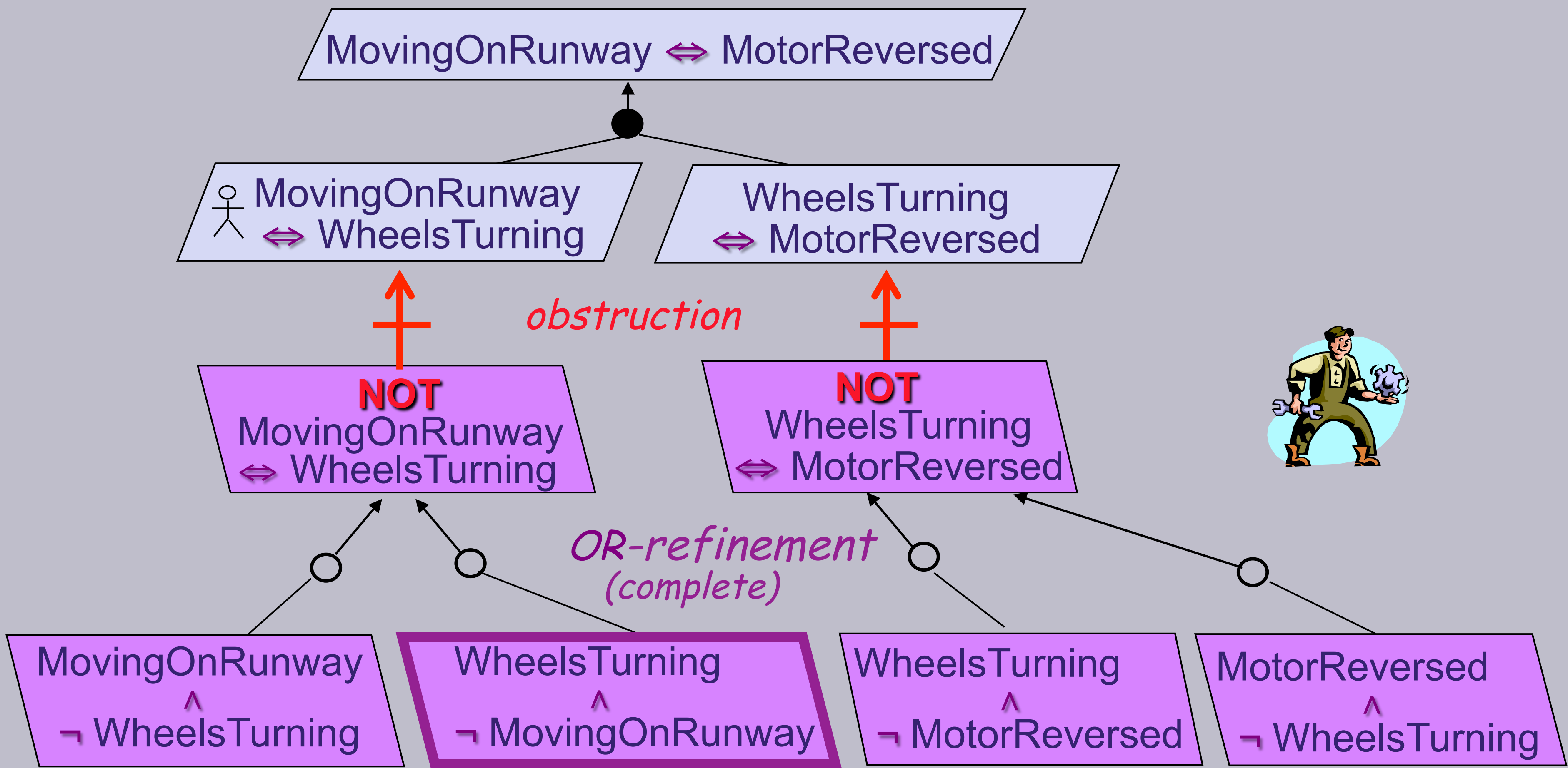
Tautology-based refinement:
A320 braking logic example

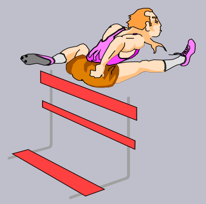


Tautology-based refinement:
A320 braking logic example



Tautology-based refinement: A320 braking logic example





Recall: obstacle analysis for increased system robustness

- Obstacle = feasible precondition for goal obstruction
- Anticipate obstacles ...
  - ⇒ new goals as countermeasures to abnormal conditions
  - ⇒ more complete goal model
- **Obstacle analysis:**

  For selected goals in the goal model ...
  - **identify** as many obstacles to it as possible;
  - assess their likelihood & severity;
  - resolve them according to likelihood & severity

Can we identify obstacles systematically?

- The problem: generate obstacles *O* such that

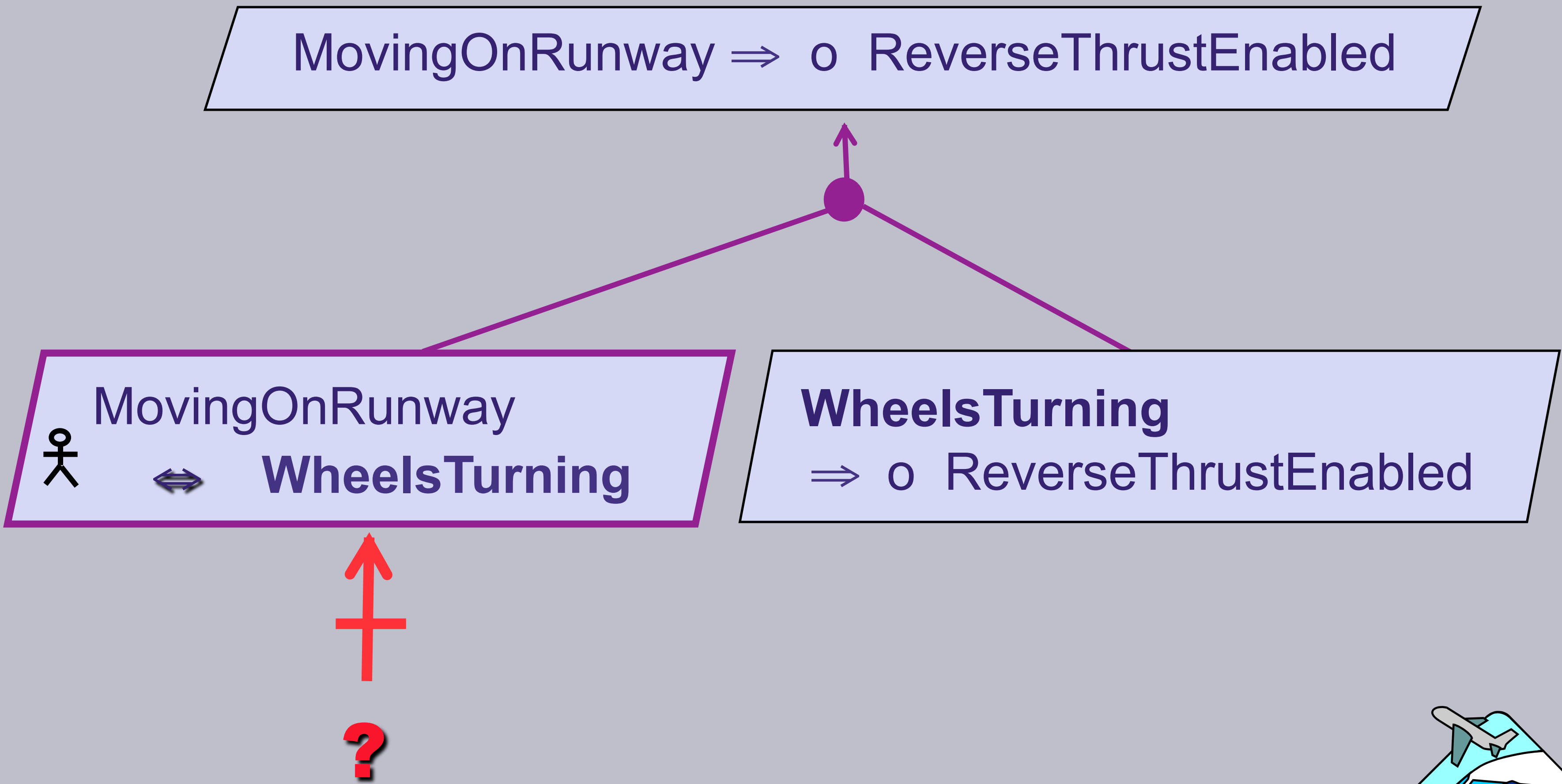
$$O, Dom \vdash \neg G$$

$$Dom \not\models \neg O$$

- Various techniques available …

  - tautology-based refinement from $\neg G$
  - regression of $\neg G$ through *Dom*
  - reuse of formal obstruction patterns
  - combine model checking and inductive learning

Generating obstacles: regressing goal negations



Original A 320 braking logic

Generating obstacles: regressing goal negations

Find precondition for obstruction of ...

MovingOnRunway ⇒ WheelsTurning

Warsow obstacle

Generating obstacles: regressing goal negations

Find precondition for obstruction of ...

MovingOnRunway ⇒ WheelsTurning

→ goal negation:

◊ MovingOnRunway ∧ ¬ WheelsTurning

Warsaw obstacle

Generating obstacles: regressing goal negations

Find precondition for obstruction of ...

MovingOnRunway ⇒ **WheelsTurning**

→ goal negation:

◊ MovingOnRunway ∧ ¬ WheelsTurning

→ regress through domain properties:

? *necessary* conditions for *wheels turning* ?

→

Warsaw obstacle

Generating obstacles: regressing goal negations

Find precondition for obstruction of ...

MovingOnRunway ⇒ **WheelsTurning**

→ goal negation:

◊ MovingOnRunway ∧ ¬ WheelsTurning

→ regress through domain properties:

? *necessary* conditions for *wheels turning* ?

WheelsTurning ⇒ ¬ Aquaplaning

i.e. Aquaplaning ⇒ ¬ WheelsTurning

Generating obstacles: regressing goal negations

Find precondition for obstruction of ...

MovingOnRunway ⇒ WheelsTurning

→ goal negation:

◊ MovingOnRunway ∧ ¬ WheelsTurning

→ regress through domain properties:

? *necessary* conditions for *wheels turning* ?

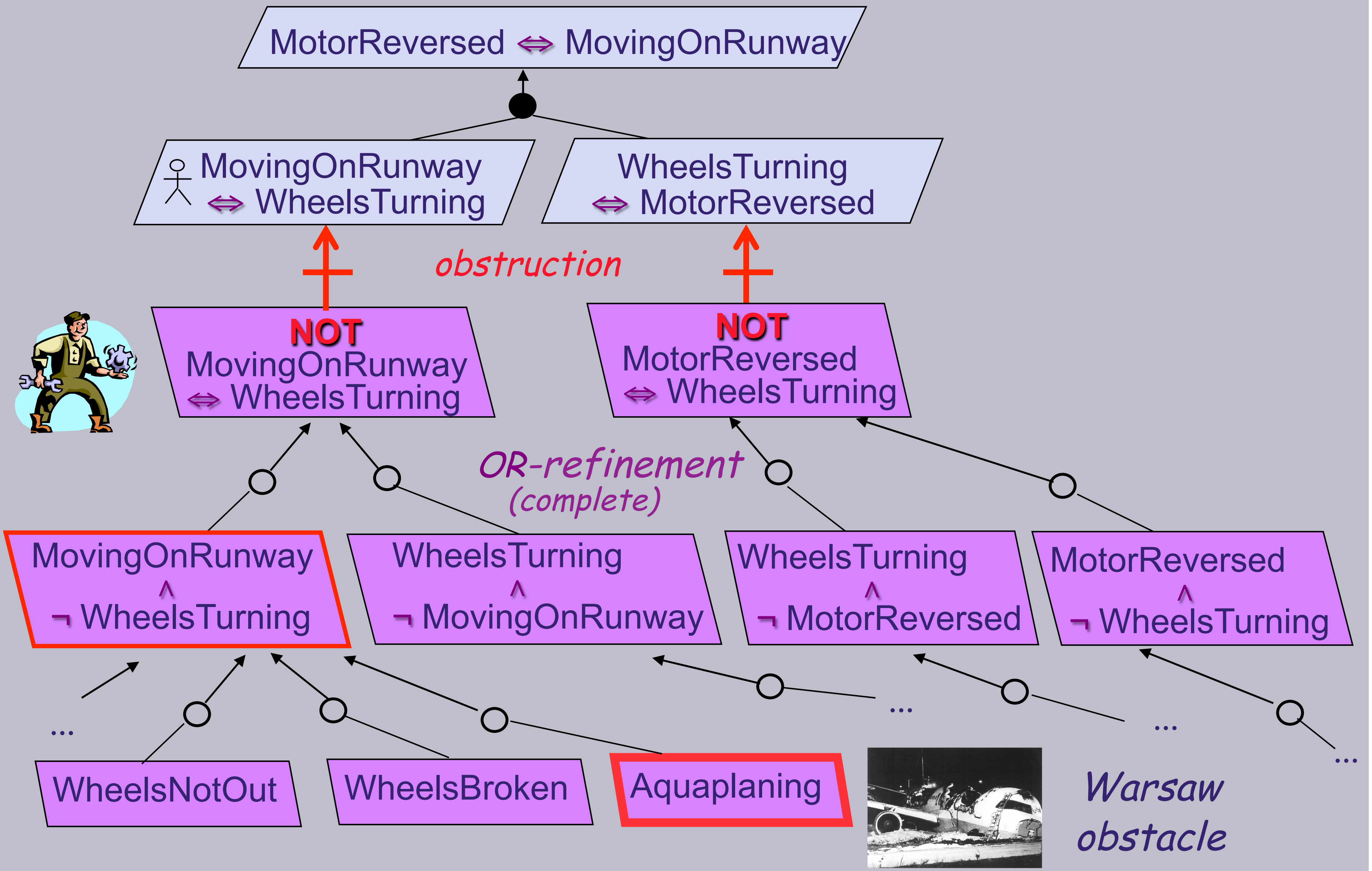
WheelsTurning ⇒ ¬ Aquaplaning

i.e. Aquaplaning ⇒ ¬ WheelsTurning

→ RHS unifiable:

◊ MovingOnRunway ∧ Aquaplaning

Resulting obstacle trees



The regression procedure

- Initial step:
  - take $O := \neg G$
- Inductive step:
  - let

    $A \Rightarrow C$ be the domain property selected

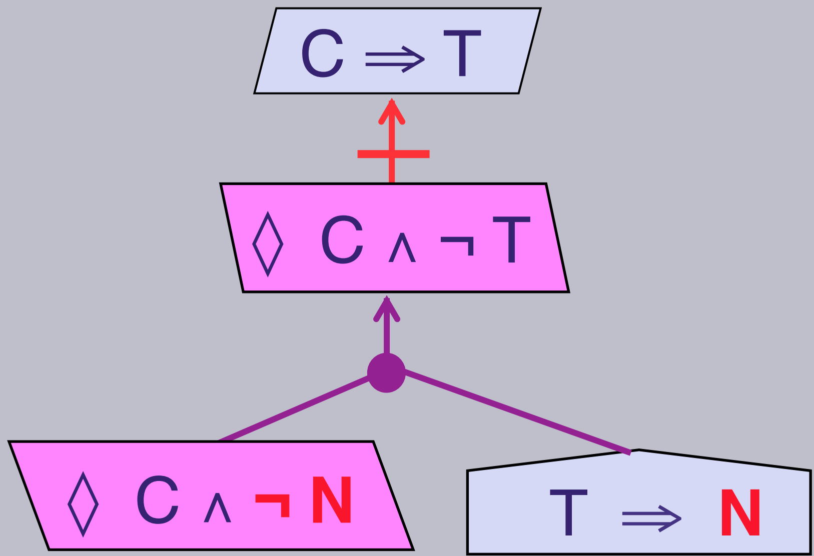
    with C matching some L in O whose occurrences are all positive in O
  - then $\mu :=$ mgu (L, C) (most general unifier)

    $$O := O\,[L\,/\,A.\ \mu]$$

Every iteration produces finer sub-obstacles

Generating obstacles: reusing formal obstruction patterns

- Same idea as goal refinement patterns - *obstructions* here



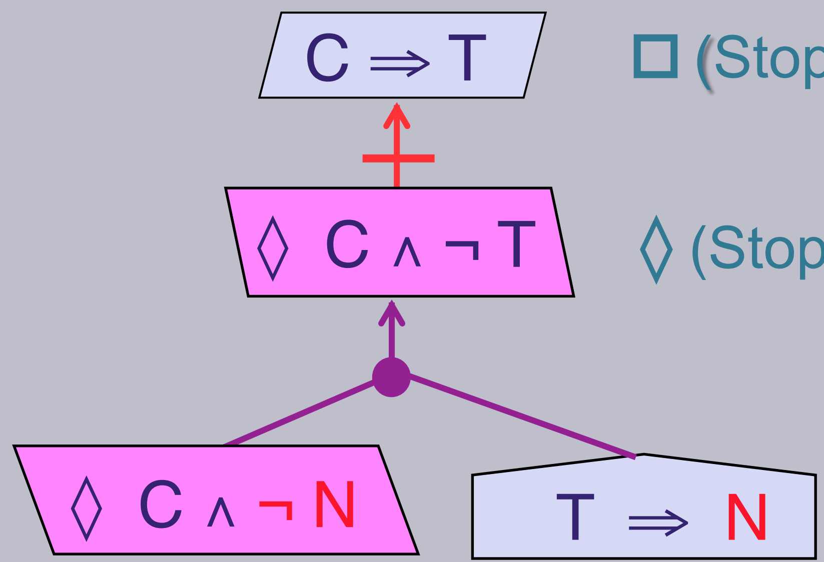
obstacle

domain property:
necessary condition for target condition

- Useful pattern for eliciting relevant domain properties
  - "what are necessary conditions for TargetCondition?"

Generating obstacles: reusing formal obstruction patterns

- Very frequent pattern ...

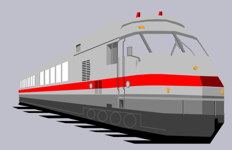


□ (StopSignal → TrainStops)

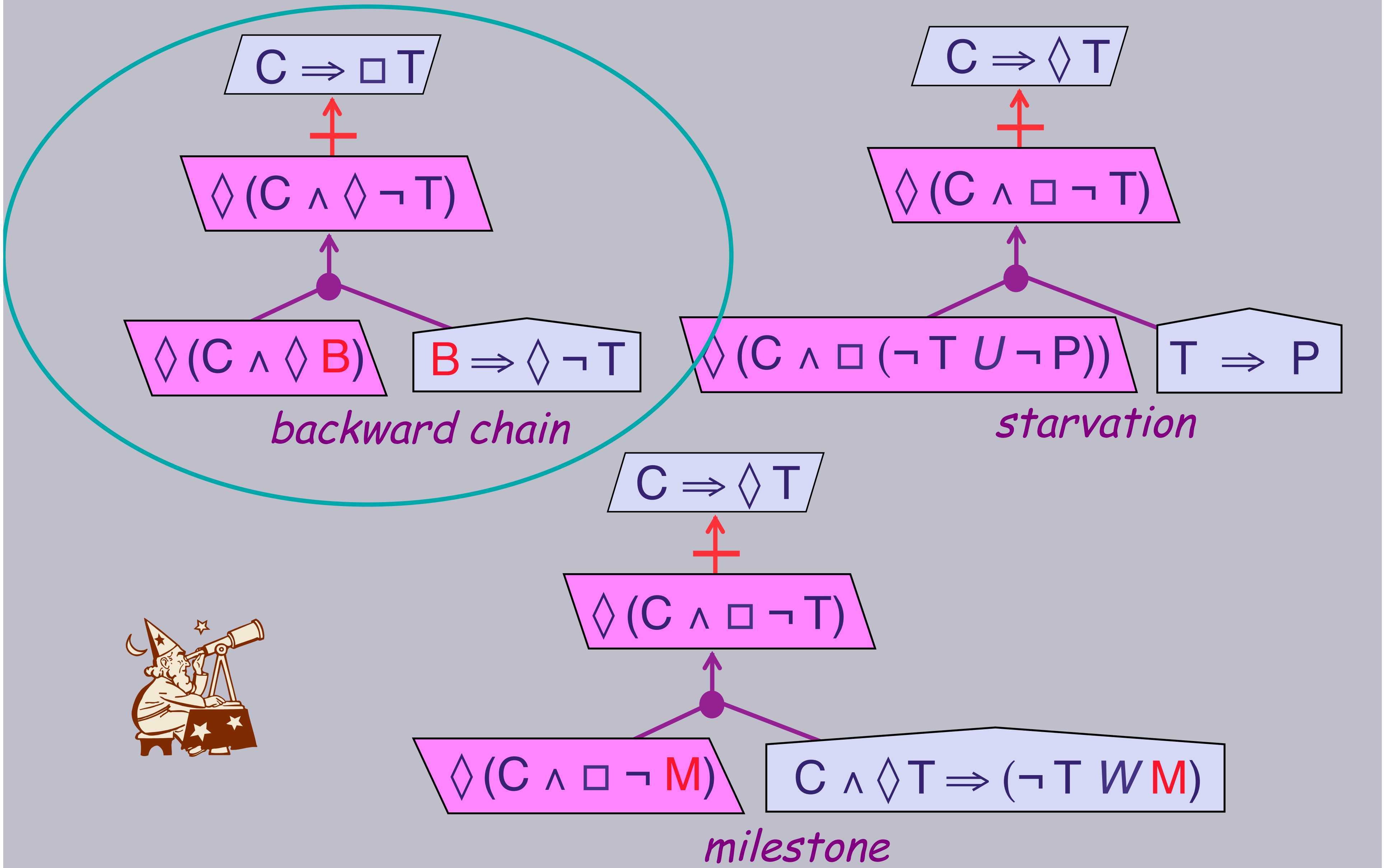
◊ (StopSignal ∧ ¬ TrainStops)

◊ (StopSignal ∧ ¬ DriverResponsive)

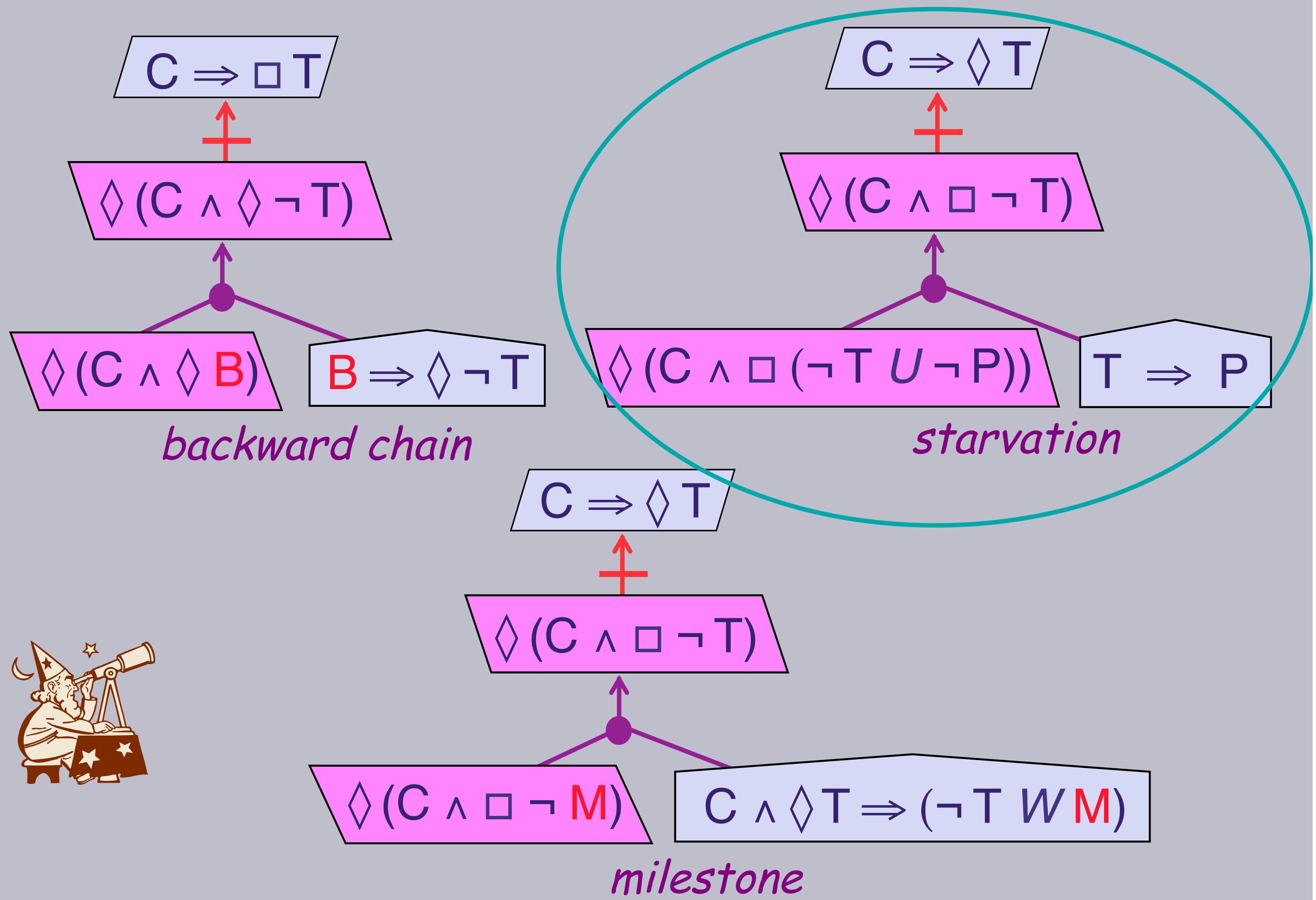
TrainStops ⇒ DriverResponsive



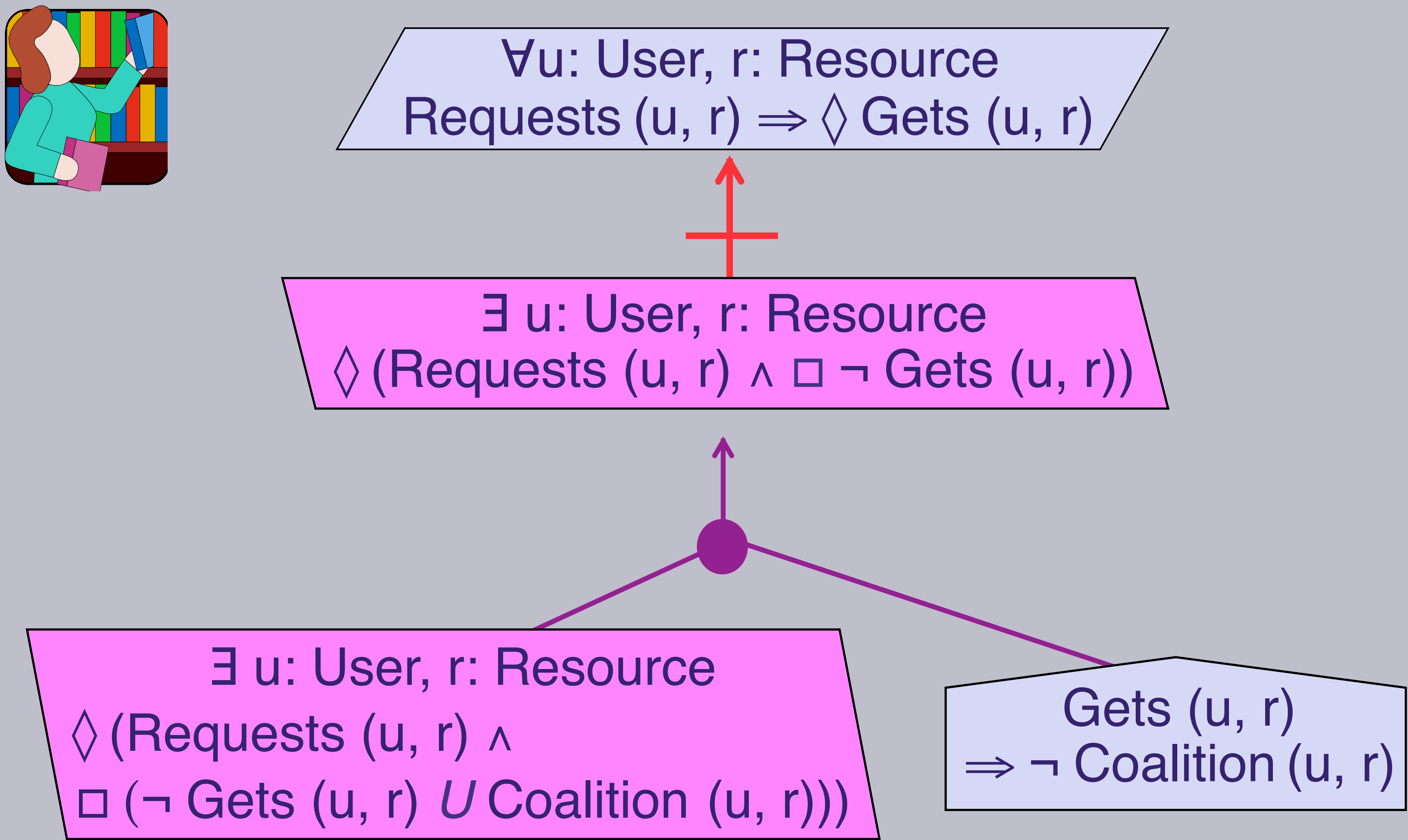
Some other frequent obstruction patterns



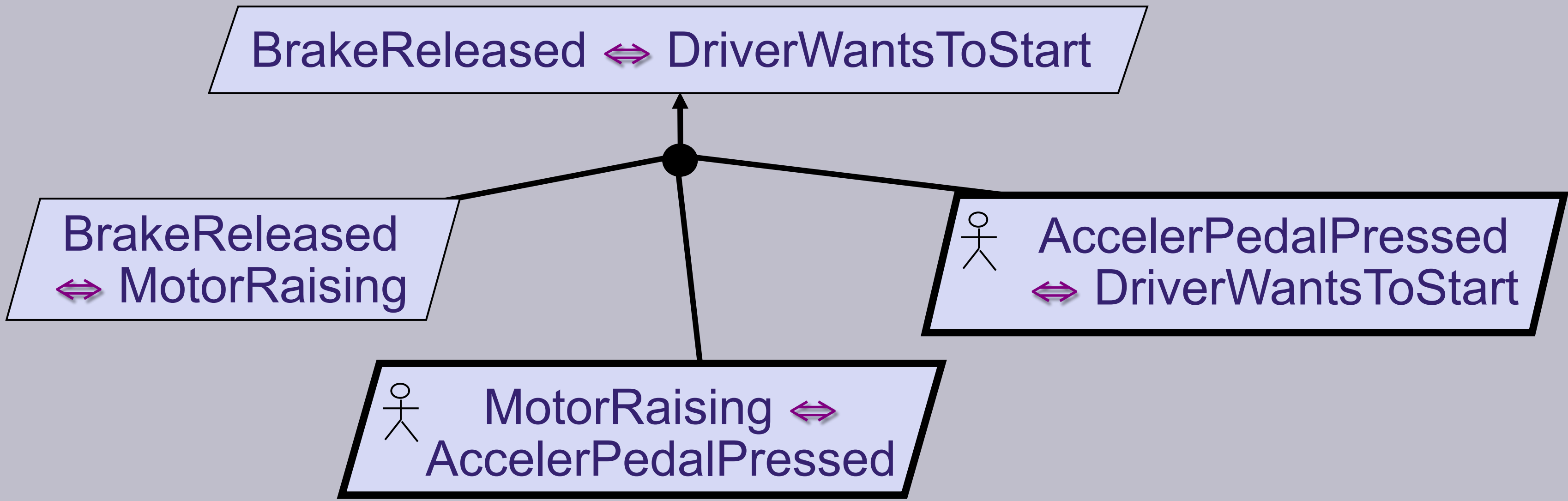
Some other frequent obstruction patterns



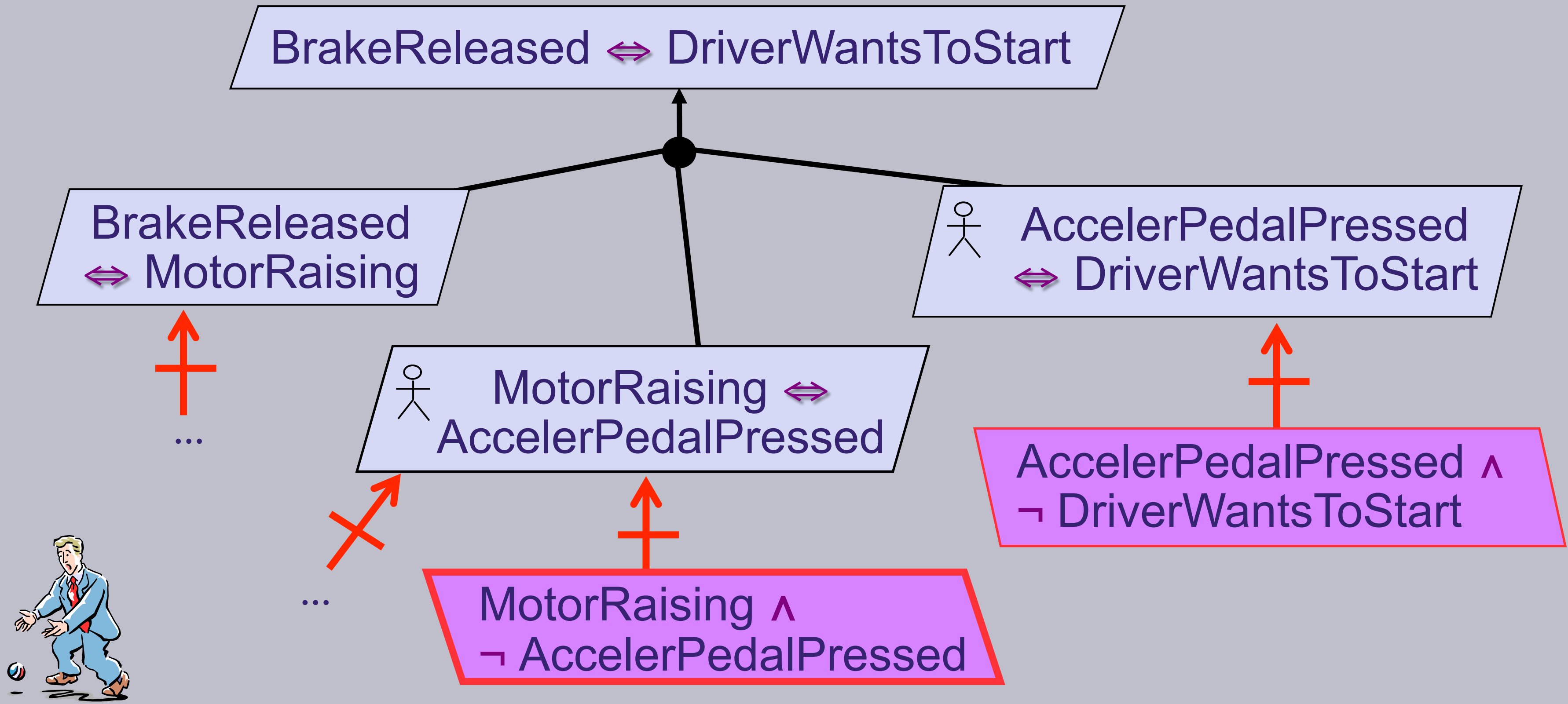
Instantiating the starvation pattern



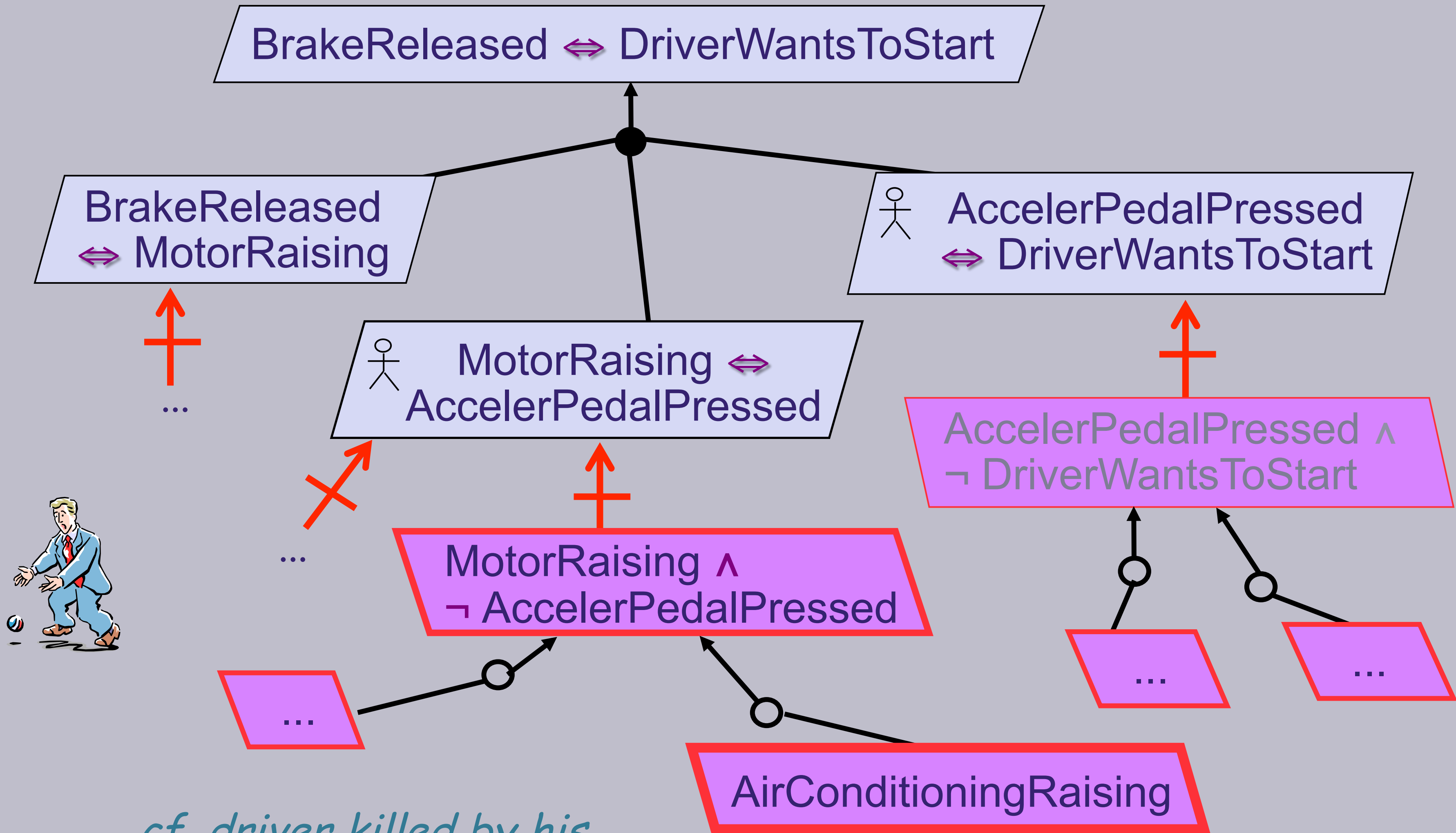
Generating obstacles: another example



Generating obstacles: another example



Generating obstacles: another example

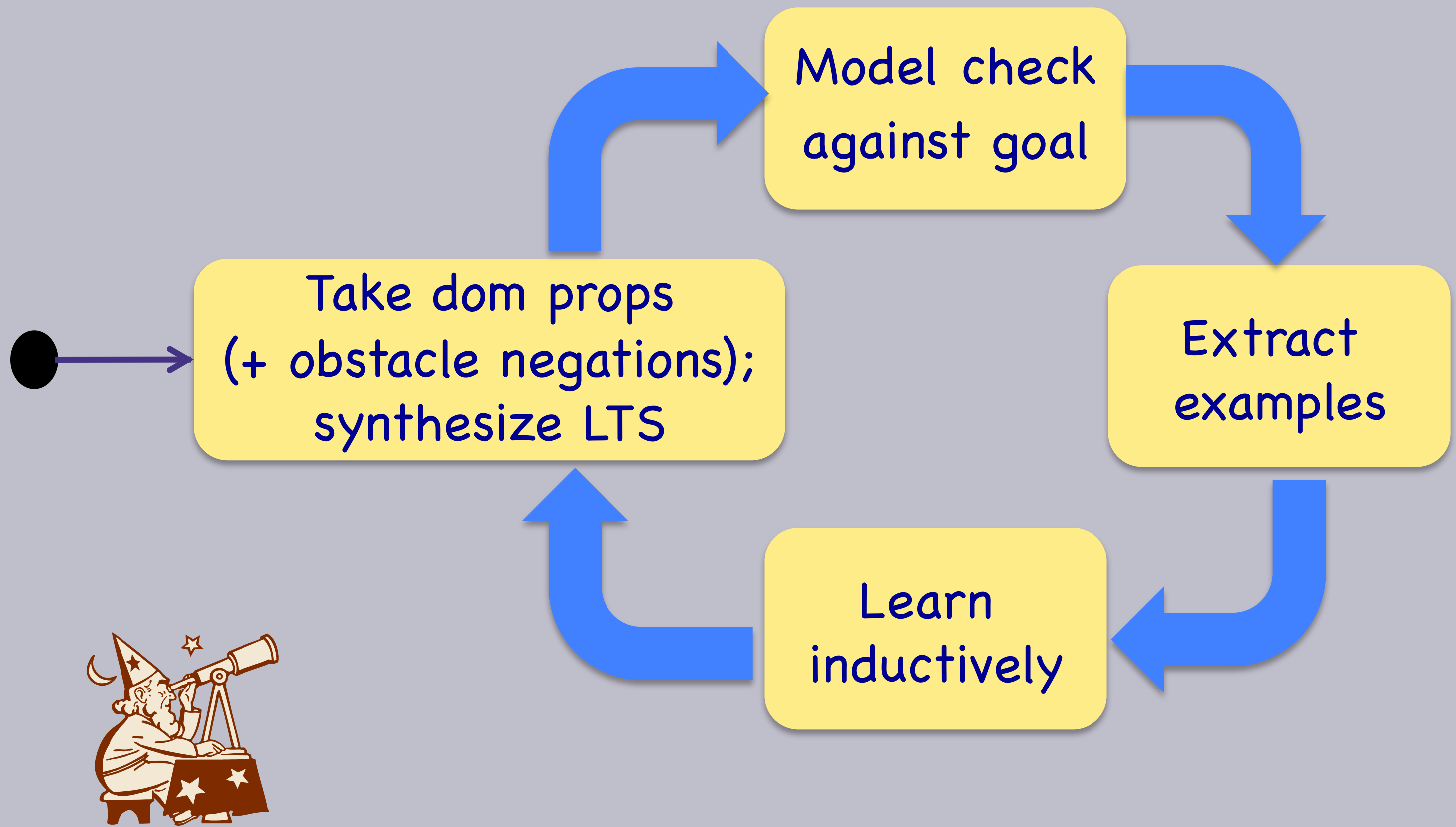


cf. driver killed by his luxurious car on a hot summerday

Outline

- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

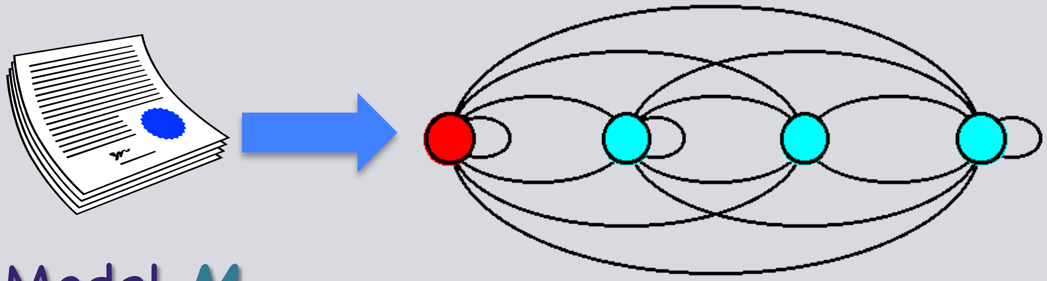
Combining model checking & inductive learning for obstacle generation



[Alrajeh, Kramer, van Lamsweerde, Russo & Uchitel, ICSE'2012]

Using the LTSA model checker

Q:



Model *M*

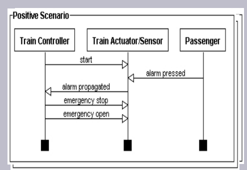
Semantics *L (M)*

⊨ F-LTL property *P*

?

No Yes

A: Counter-example



Model consistent wrt *P*

[Giannakopoulou & Magee, FSE'2003]

Inductive logic programming

Machine learning technique for constructing concept descriptions from examples + logical domain theory [Muggleton 1994]

Given:

K	knowledge base
E$^+$	set of positive examples
E$^-$	set of negative examples
IC	integrity constraints

Find:

H generalisation such that

$\{K, H\} \models E^+$

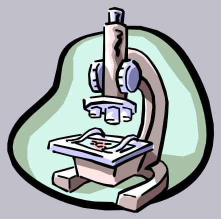
$\{K, H\} \not\models E^-$

$\{K, H, IC\} \not\models false$

Inductive Logic Programming systems available (XHAIL, TAL)

- *scalable for finite domains*
- *sound and complete*
- *fully automated*

[Ray 2009, Corapi et al 2010]



The problem, more precisely

Given

A declarative model: set of LTL goals G + domain properties *D*

$D \not\models G \,, \quad \{D, G\} \not\models \mathit{false}$

Find

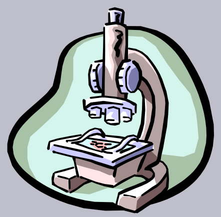
A *domain-complete set* of obstacles $\{O_1, \ldots, O_n\}$ such that

$\{O_i, D\} \models \neg G \,, \quad \{O_i, D\} \not\models \mathit{false}$

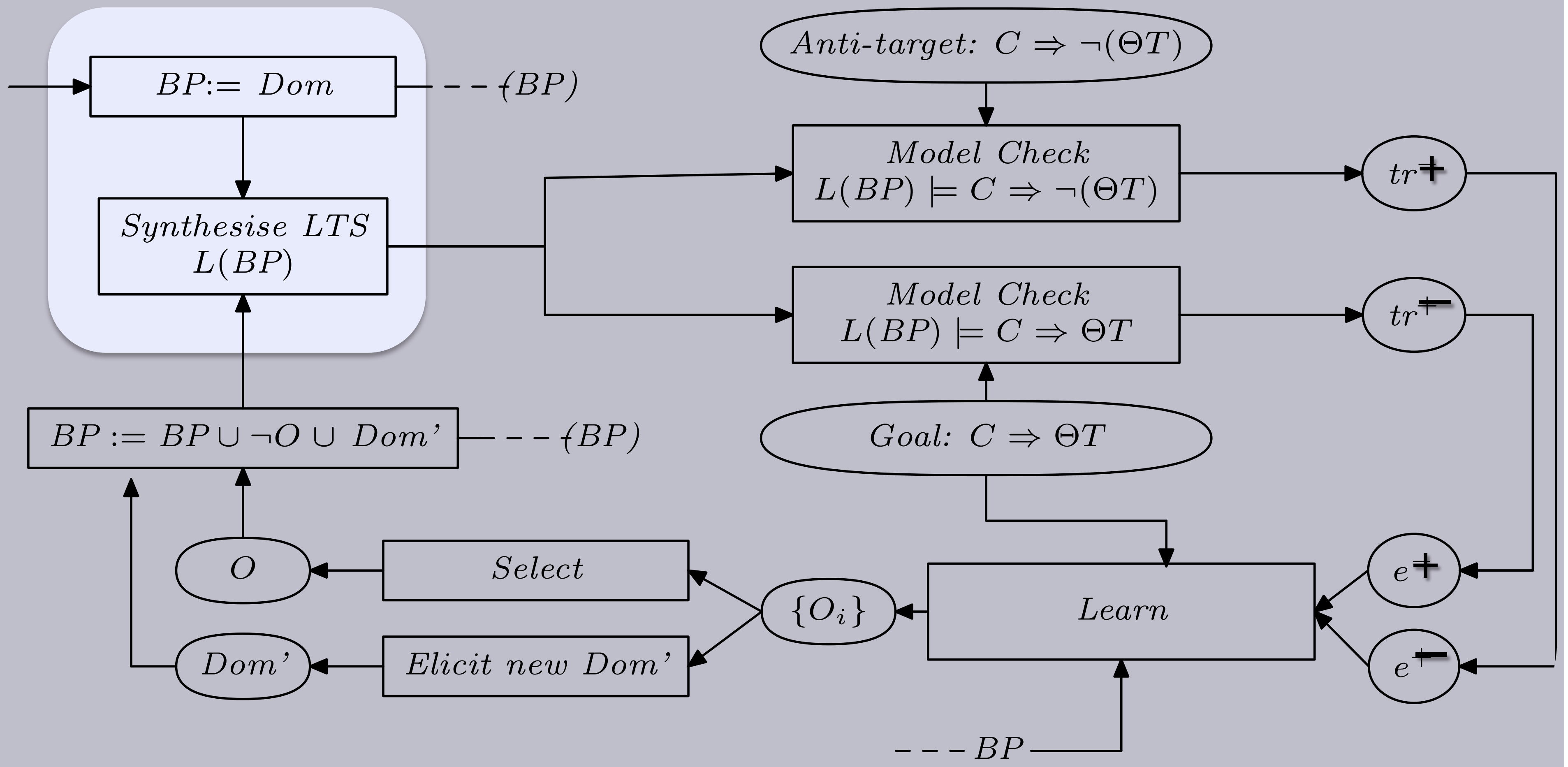
$\{\neg O_1 \,, \ldots \,, \neg O_n \,, D\} \models G$

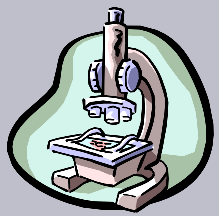
where $\models$ is interpreted as LTL satisfaction relation

wrt all LTS traces

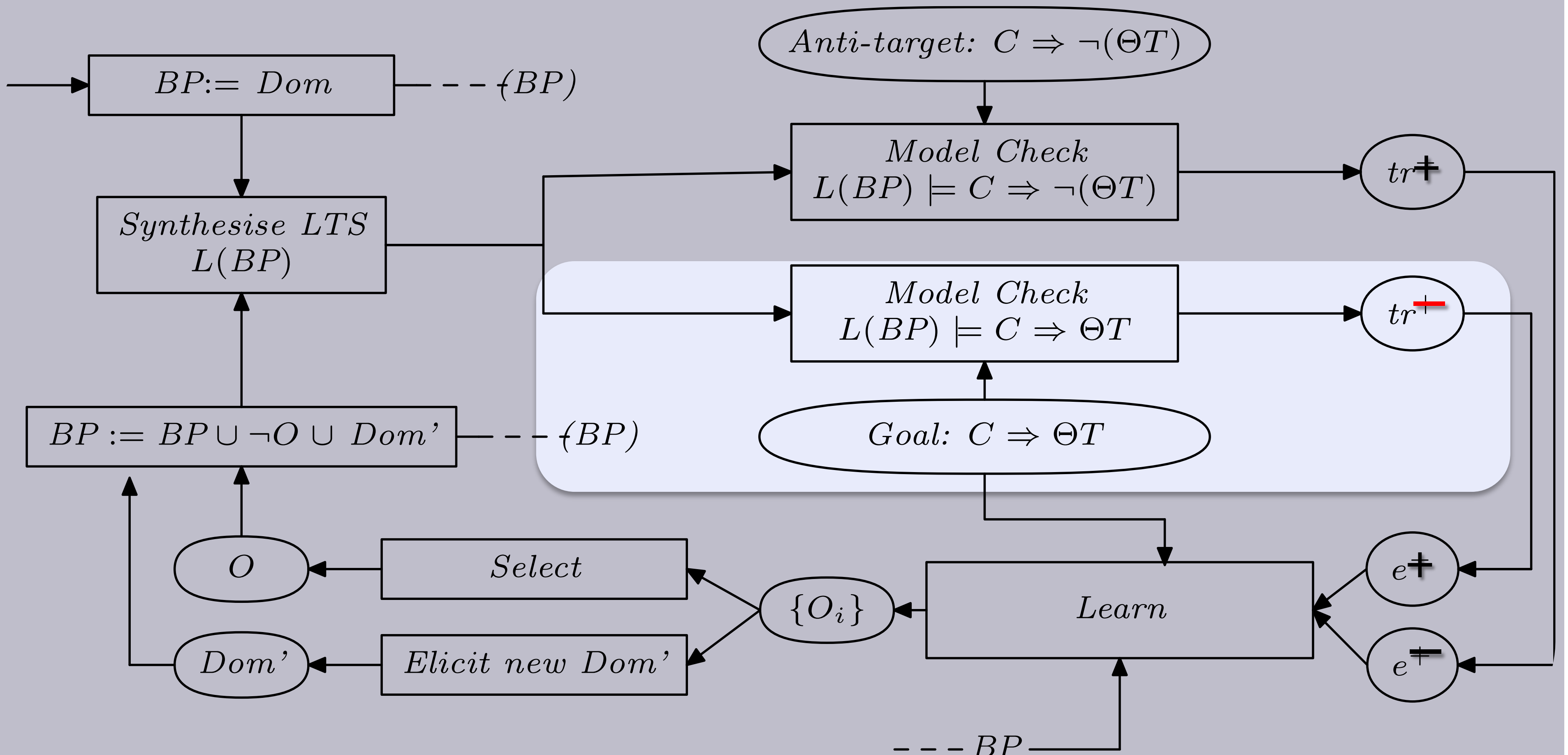


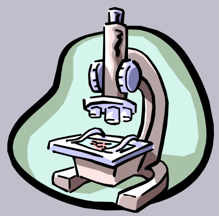
The solution, more precisely



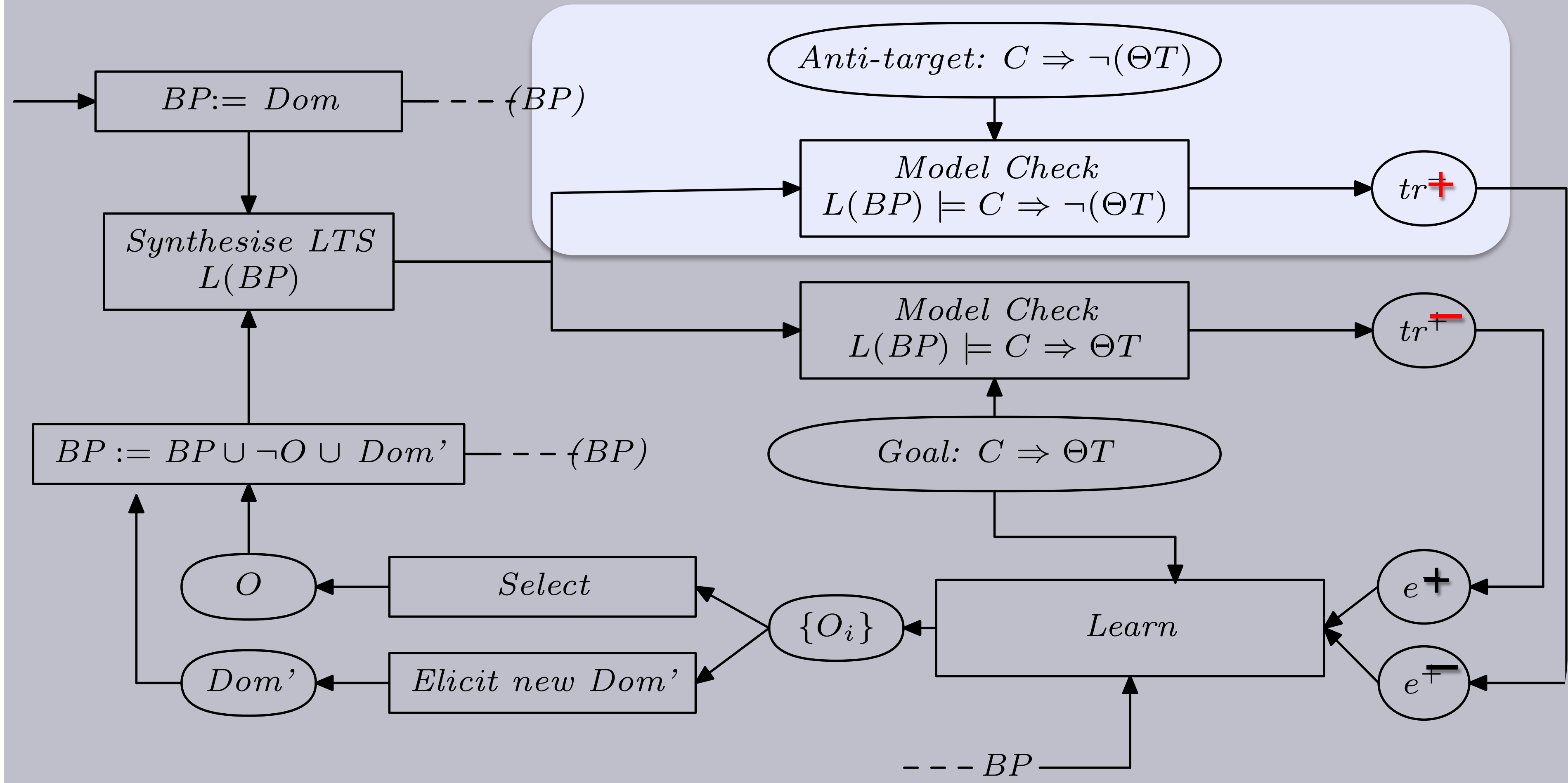


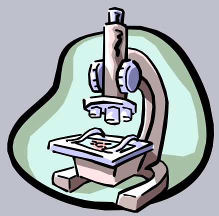
The solution, more precisely



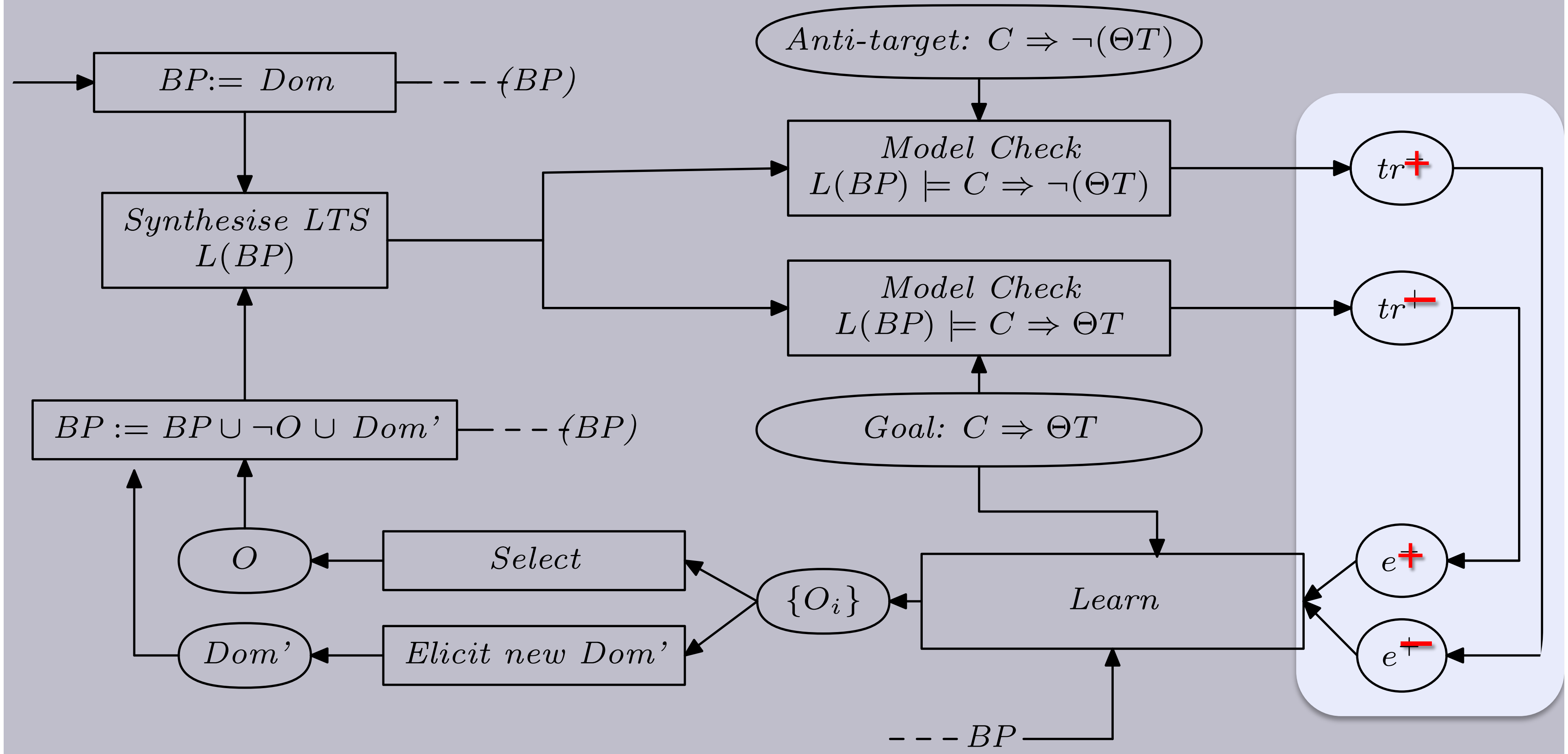


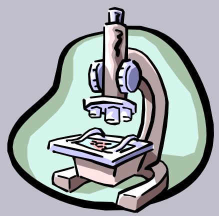
The solution, more precisely



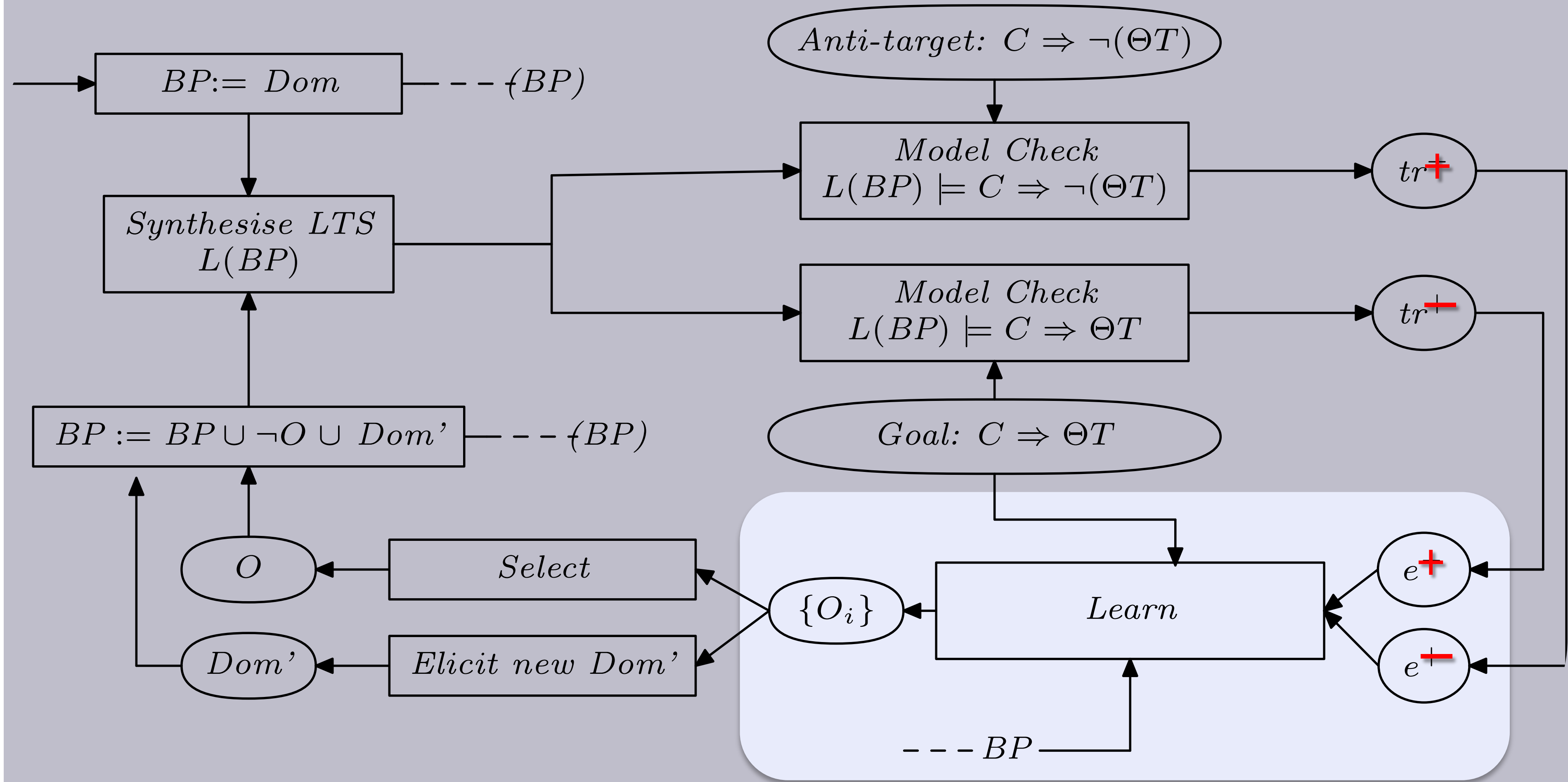


The solution, more precisely

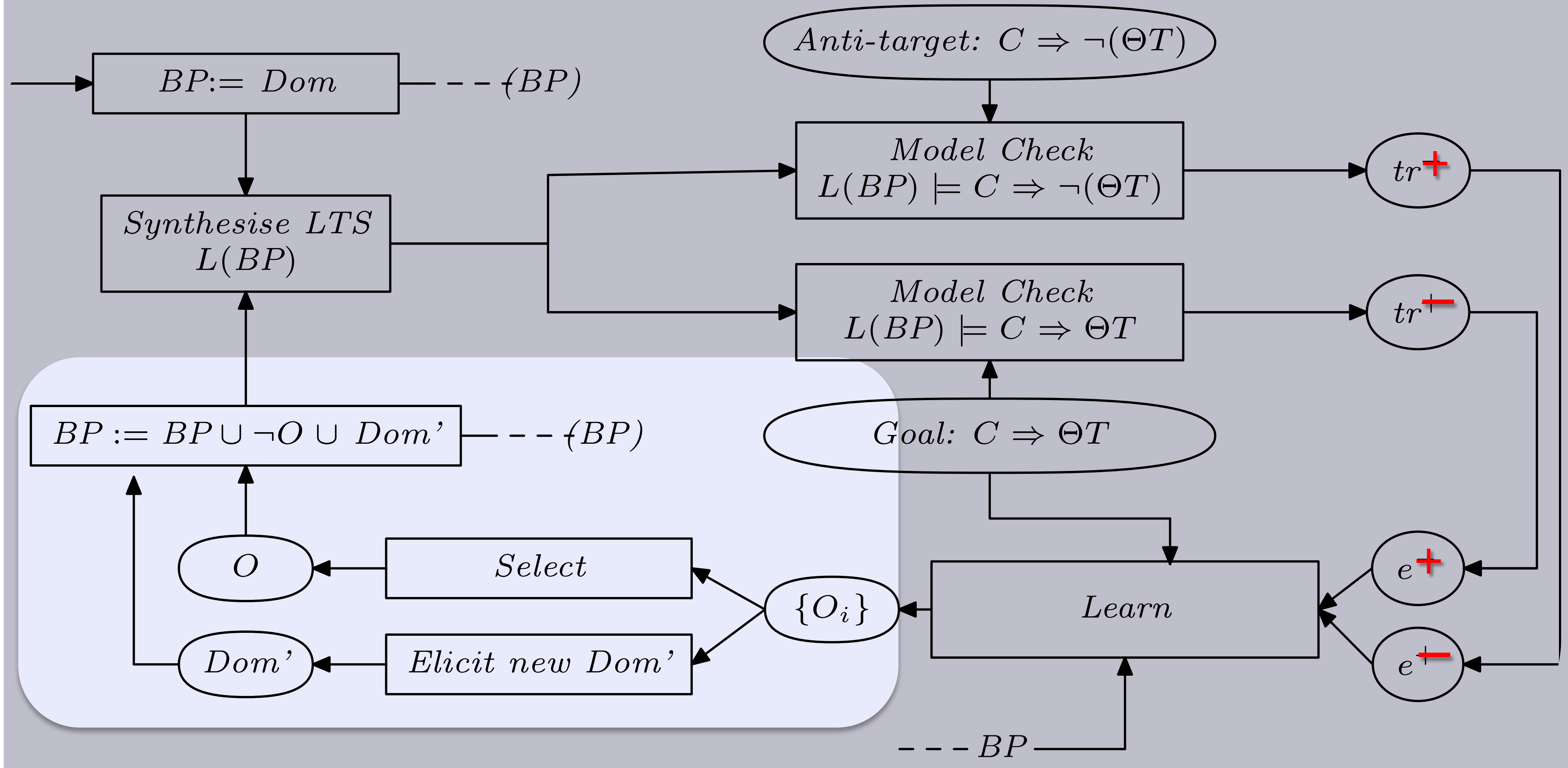




The solution, more precisely



The solution, more precisely



Back to trains and signals …

Input: goals

General form

$C \Rightarrow \Theta T$

Θ : temporal LTL operator $\mathbf{o}$, $\Diamond$, $\Box$, $\Rightarrow$, ...

Goal Achieve [*TrainStoppedAtBlockSignal If StopSignal*]

StopSignal $\Rightarrow$ $\mathbf{o}$ TrainStopped

Input: domain properties

Temporal assertions (necessary conditions for goal target)

+ fluent definitions

Dom props:

TrainStopped ⇒ DriverResponsive

TrainStopped ⇒ SignalVisible

Fluent Definitions:

TrainStopped = < stop_train, start_train, *false* >

StopSignal = < set_to_stop, set_to_go, *false* >

SignalVisible = < clear_signal, obstruct_signal, *true* >

DriverResponsive = < driver_responds, driver_ignores, *true* >

Synthesizing LTL domain props and model checking

- Checking for obstacle feasibility

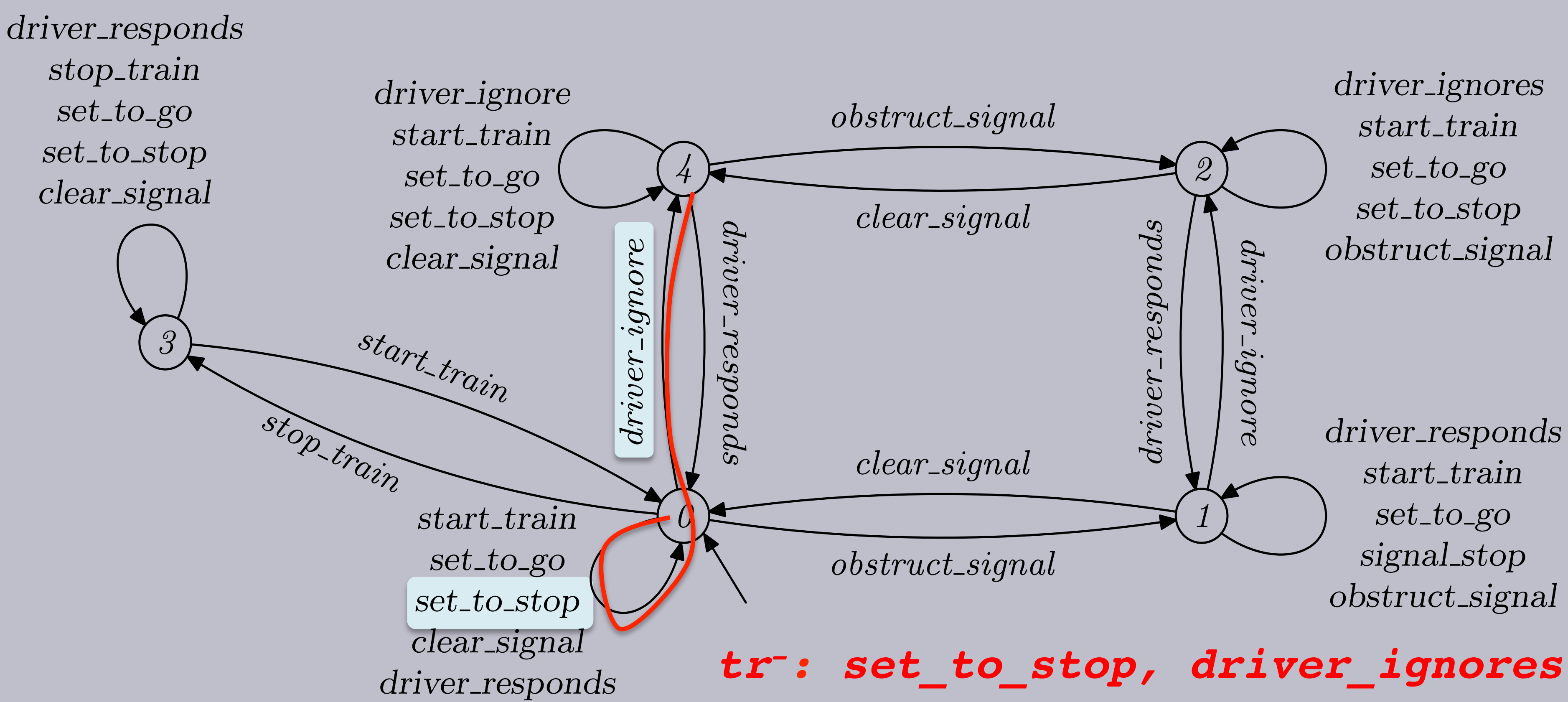
$LTL(D) \models C \Rightarrow \Theta T$ → *counterexample*

- Checking for goal satisfiability

$LTL(D) \models C \Rightarrow \neg \Theta T$ → *witness*

Counterexample generation

$$\begin{array}{c}\text{TrainStopped} \Rightarrow \text{DriverResponsive} \\ \wedge \\ \text{TrainStopped} \Rightarrow \text{SignalVisible}\end{array} \models \text{StopSignal} \Rightarrow \circ\, \text{TrainStopped}$$



Witness generation

TrainStopped ⇒ DriverResponsive

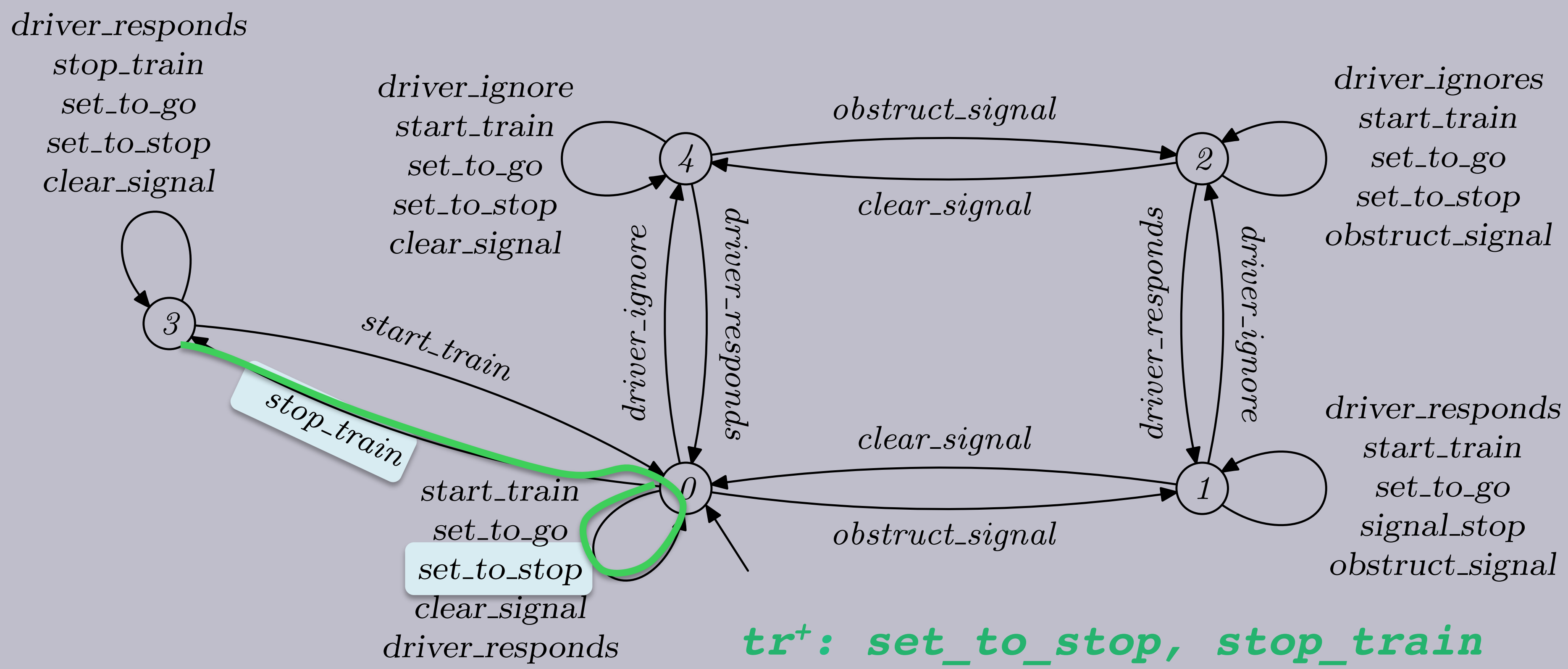
∧

TrainStopped ⇒ SignalVisible

⊨

StopSignal ⇒

¬ o TrainStopped



tr$^+$: set_to_stop, stop_train

Preparation for learning

Domain properties, goals, counterexample and witness(es)
are automatically translated into
the logic programming formalism understood by learning tool

TrainStopped ⇒ DriverResponsive
TrainStopped ⇒ SignalVisible
TrainStopped = <stop_train, start_train, *false*>
StopSignal = <set_to_stop, set_to_go, *false*>
SignalVisible = <clear_signal, obstr_signal, *true*>
DriverResponsive = <responds, ignores, *true*>

StopSignal ⇒ o TrainStopped

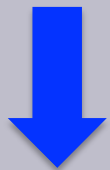
set_to_stop, driver_ignores
set_to_stop, stop_train

```
:- holdsAt(trainStopped,T,S),
   not holdsAt(driverResponsive,T,S).
…
initiates(stop_train,trainStopped).
terminates(start_train,trainStopped).
…
initiates(driver_responds,driverResponsive).
terminates(driver_ignores,driverResponsive).
initially(driverResponsive).
…
holdsAt(trainStopped,T2,S):-
      holdsAt(stopSignal,T1,S), next(T2,T1),
      not obstructed_next(trainStopped,T1,S).
…
```

Translation into a logic program (1)

- Domain properties: fluent definitions …

DriverResponsive = < driver_responds, driver_ignores, *true* >



… add facts to knowledge base ***K***

```
initiates(driver_responds,driverResponsive).
terminates(driver_ignores,driverResponsive).
initially(driverResponsive).
```

Translation into a logic program (2)

- Domain properties: temporal assertions …

TrainStopped ⇒ DriverResponsive

… add to integrity constraints *IC* the rule

```
:- holdsAt(trainStopped,T,S),
   not holdsAt(driverResponsive,T,S).
```

Translation into a logic program (3)

- Goals …

StopSignal ⇒ **o** TrainStopped

… add to the knowledge base *K* the rule

```
holdsAt(trainStopped,T2,S):-
        holdsAt(stopSignal,T1,S),
        next(T2,T1),
        not obstructed_next(trainStopped,T1,S).
```

no obstacle that would prevent the train from stopping

Translation into a logic program (4)

- Counterexamples …

set_to_stop, driver_ignores

… add to the knowledge base the facts

```
happens(set_to_stop,0,cx).
happens(driver_ignores,1,cx).
```

… add to the **positive** examples of obstacle the fact

```
not holdsAt(trainStopped,2,cx).
```

generalization should be inferred to explain why the goal's target is obstructed in this example

Translation into a logic program (5)

- Witnesses ...

set_to_stop, stop_train

... add to the knowledge base the facts

```
happens(set_to_stop,0,wx).
happens(stop_train,1,wx).
```

... add to the **negative** examples of obstacle the fact:

```
holdsAt(trainStopped,2,cx).
```

generalization to be inferred should be consistent with goal's target not being obstructed in this negative example

Learner output: obstacle condition

- Generalised assertion covering counterexample, excluding witness

```
obstructed_next(trainStopped,T,S):-
     holdsAt(stopSignal,T,S),
     not holdsAt(driverResponsive,T,S).
```

$O_1 = \Diamond\ (StopSignal \land \neg\ DriverResponsive)$

Second process iteration

Given

A declarative model: set of LTL goals G + domain properties D + obstacle O_1

$$\{D, \neg O_1\} \not\models G \;, \quad \{D, G\} \not\models false$$

Find

A set of obstacles $\{O_2, \ldots, O_n\}$ such that

$$\{ O_i, D \} \models \neg G \;, \quad \{ O_i, D \} \not\models \mathit{false}$$

$$\{ D \;, \neg O_1 \;, \ldots \;, \neg O_n \} \models G$$

where $\models$ is interpreted as satisfaction relation wrt all LTS traces

Second process iteration (2)

Domain Properties:

TrainStopped ⇒ DriverResponsive

TrainStopped ⇒ SignalVisible

TrainStopped = < stop_train, start_train, *false*>

StopSignal = < set_to_stop, set_to_go, *false* >

SignalVisible = < clear_signal, obstruct_signal, *true* >

DriverResponsive = < driver_responds, driver_ignores, *true* >

Goal:

StopSignal ⇒ o TrainStopped

Negated Obstacle Condition:

$\Box$ (¬StopSignal ∨ DriverResponsive)

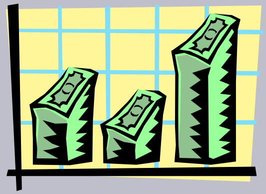
O_2 = ◊ (StopSignal ∧ ¬ SignalVisible)

Getting new domain properties into the loop

- *WHEN?* After obstacles are generated
- *WHY?*
  - expand scope of obstructions
  - refine obstacles
- Focussed, goal-directed ...
  - for *other* goal obstructions: look for properties T ⇒ ***N***

    N: necessary condition for target of goal C ⇒ Θ T
  - for obstacle refinement: look for properties ***S*** ⇒ O

    S: sufficient condition for obstacle to be refined



Benefits of combining model checking & inductive learning

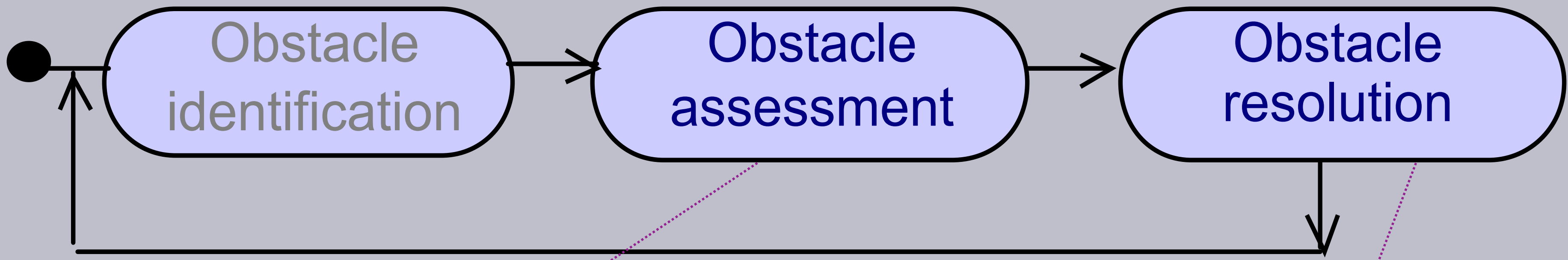
- Tool-supported approach for incremental generation of domain-complete set of obstacles
  - no user intervention required for example provision
- Domain-feasibility of generated obstacles granted for free
  - no need for separate check as in [Lamsweerde&Letier 2000]
- Assists in eliciting relevant domain properties
- Can be integrated with generation of operational reqs [Alrajeh et al 2009]
- Evaluation on LAS case study
  - generation of all formal obstacles that were derived manually in [van Lamsweerde&Letier00], and more

Outline

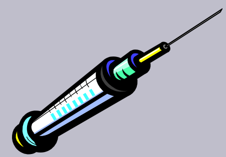
- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning

- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

Brief recall:
risk management at RE time



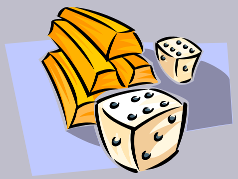
likely?
severe, likely consequences?



resolution =
revised goal model
with countermeasures

- Assessment is aimed at focussing resolution on ***critical*** obstacles

[Cailliau & van Lamsweerde, RE'2012]



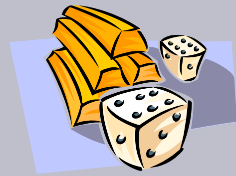
Obstacle assessment calls for a probabilistic framework

- Goals most often will be satisfied **only partially**
  - degree of goal satisfaction depends on probability of obstructing obstacles
- Goals are sometimes stated probabilistically
  - e.g. ORCON standards ...

"ambulances shall be on incident scene within 14 minutes in ***95%*** *of cases"*

- Severity of consequences then depends on difference between
  - ***required*** degree of satisfaction
  - ***estimated*** probability of satisfaction

Probabilistic goals



- **Proba of satisfaction** of *C* ⇒ Θ *T* : proportion between ...
  - # possible behaviors satisfying *C*, Θ *T*
  - # possible behaviors satisfying *C*

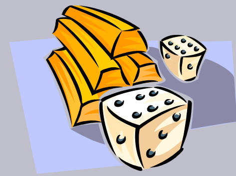
  e.g. P (Achieve [AmbulanceMobilizedWhenAllocated]) =

$$\frac{\text{\# behaviors where allocated ambulance is mobilized}}{\text{\# behaviors where ambulance is allocated}}$$

- Two goals are **dependent** if the set of behaviors non-vacuously satisfying one is also non-vacuously satisfying or denying the other
  - in goal model: if one of them is descendant or conflicting
  - subgoals *are* independent in complete, consistent, minimal refinements:

$$P\ (SG_1 \mid SG_2) = P\ (SG_1 \mid \neg\ SG_2) = P\ (SG_1),$$
$$P\ (SG_2 \mid SG_1) = P\ (SG_2 \mid \neg\ SG_1) = P\ (SG_2)$$

Probabilistic goals (2)



- **Required degree** of satisfaction (*RDS*) of *G* :

  minimal admissible P(*G*)

  (obtained by req elicitation)

  - specifiable in probabilistic TLs

    e.g. $C \Rightarrow Pr_{\geq RDS}\,[\Theta T]$ [Kwiatkowska et al 2002]

  - *G* is probabilistic if $0 < RDS\,(G) < 1$

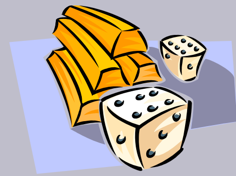
- **Estimated proba** of satisfaction (*EPS*) of *G* :

  P (*G*) computed from the goal/obstacle models from estimates on leaf nodes

- **Severity** of violation of *G* :

  $SV(G) = RDS\,(G) - EPS\,(G)$

Probabilistic goals (3)



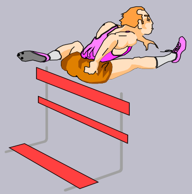
- Desirable conditions extended to probabilistic goals :

$P\,(G \mid Dom) > 0$ *domain-consistency*

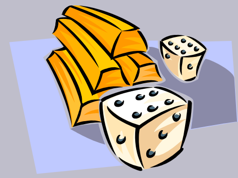
$P\,(G \mid SG_1, \ldots, SG_n, Dom) > 0$ *complete refinement*

$P\,(SG_1, \ldots, SG_n \mid Dom) > 0$ *consistent refinement*

$P\,(G \mid SG_1, \ldots, SG_{i-1}, SG_{i+1}, \ldots, SG_n, Dom)$
$< P\,(G \mid SG_1, \ldots, SG_n, Dom)$ *minimal refinement*



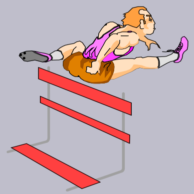
Probabilistic obstacles



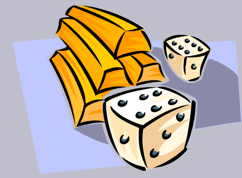
- **Probability of obstacle** : proportion between ...
  - \# possible behaviors satisfying obstacle condition
  - \# possible system behaviors

e.g. G: AmbulanceAllocated $\Rightarrow \Diamond_{\leq 2\ min}$ AmbulanceMobilized

$$P\,(\Diamond\,(\text{AmbulanceAllocated} \wedge \Box_{\geq 2\ min} \neg\, \text{CrewResponsive}) = \frac{\#\ \text{behaviors with ambulance allocated without 2-min response}}{\#\ \text{possible system behaviors}}$$



Probabilistic obstacles (2)



- Conditions extended for probabilistic (sub-)obstacles:

$P(\neg G \mid O, Dom) > 0$ *potential obstruction*

$P(O \mid Dom) > 0$ *domain consistency*

$P(O \mid SO_i) > 0$ for all SO_i *entailment*

$P(O \mid \neg SO_1, \ldots, \neg SO_n, Dom) = 0$ *domain completeness*

e.g.

P (MobilizedAmbulanceNotOnScene |
¬ StuckInTrafficJam, ¬ AmbulanceLost, ¬ AmbulanceBrokenDown)
= 0 ?

$P(SO_i \mid SO_j) = P(SO_i \mid \neg SO_j) = P(SO_i)$,

$P(SO_j \mid SO_i) = P(SO_j \mid \neg SO_i) = P(SO_j)$ *independence*

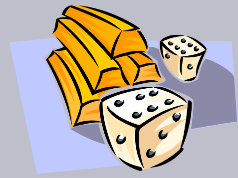
Assessing obstacles

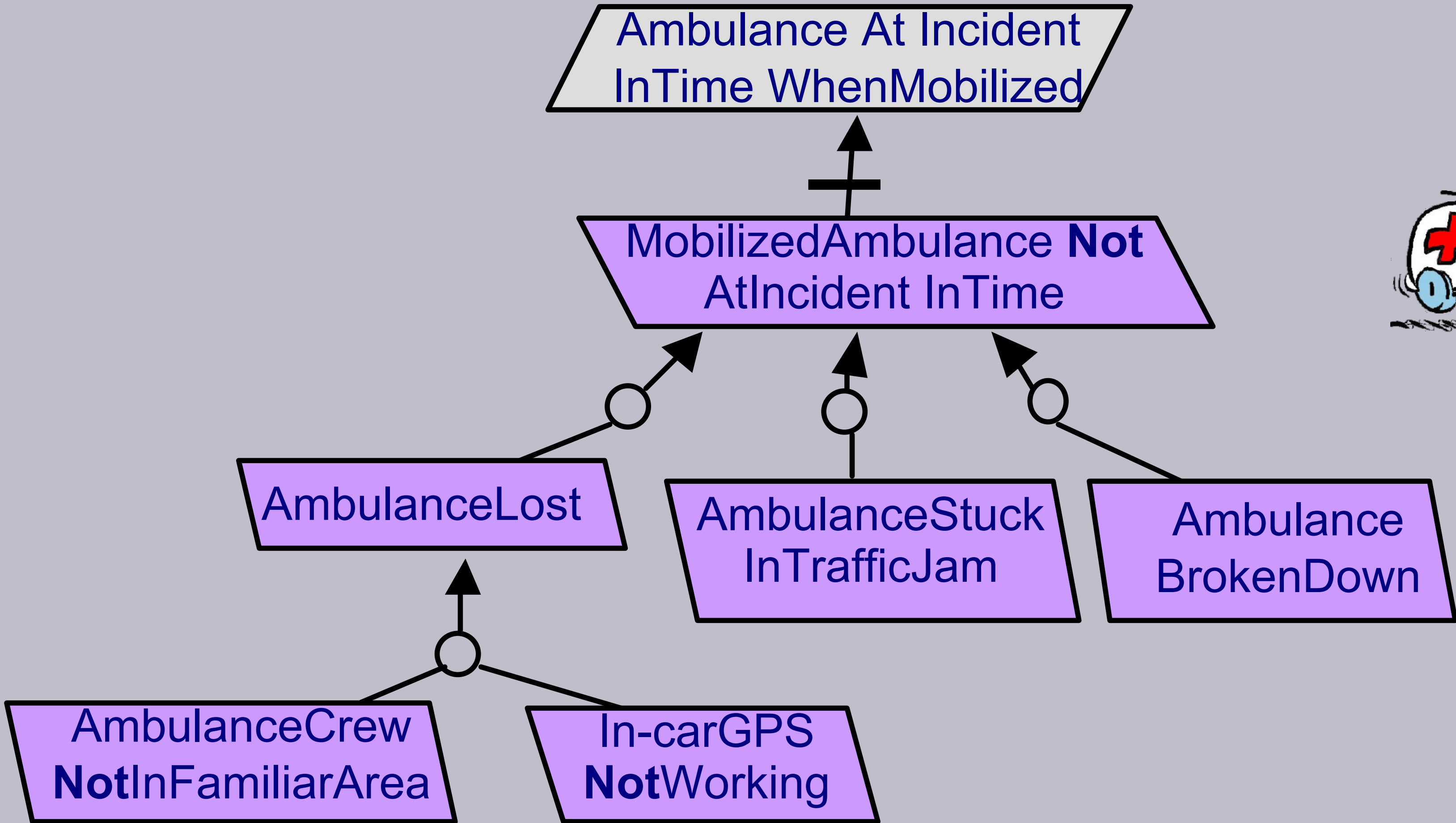
- For **leaf** obstacles: use statistical data, domain expertise
  - e.g. $P(\Diamond(\text{AmbulanceMobilized} \wedge \Box \neg \text{CrewInFamiliarArea})$: occurs in 20% of cases

- For **parent** obstacle: up-propagation through refinement tree
  - AND-refinement: $P(O) = P(SO_1) \times P(SO_2) \times P(O \mid SO_1, SO_2)$
  - OR-refinement: $P(O) = 1 - (1 - P(SO_1) \times P(O \mid SO_1)) \times (1 - P(SO_2) \times P(O \mid SO_2))$

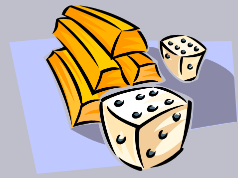
    (for complete refinement in independent obstacles)
- Up-propagation until root $\neg G$ is reached

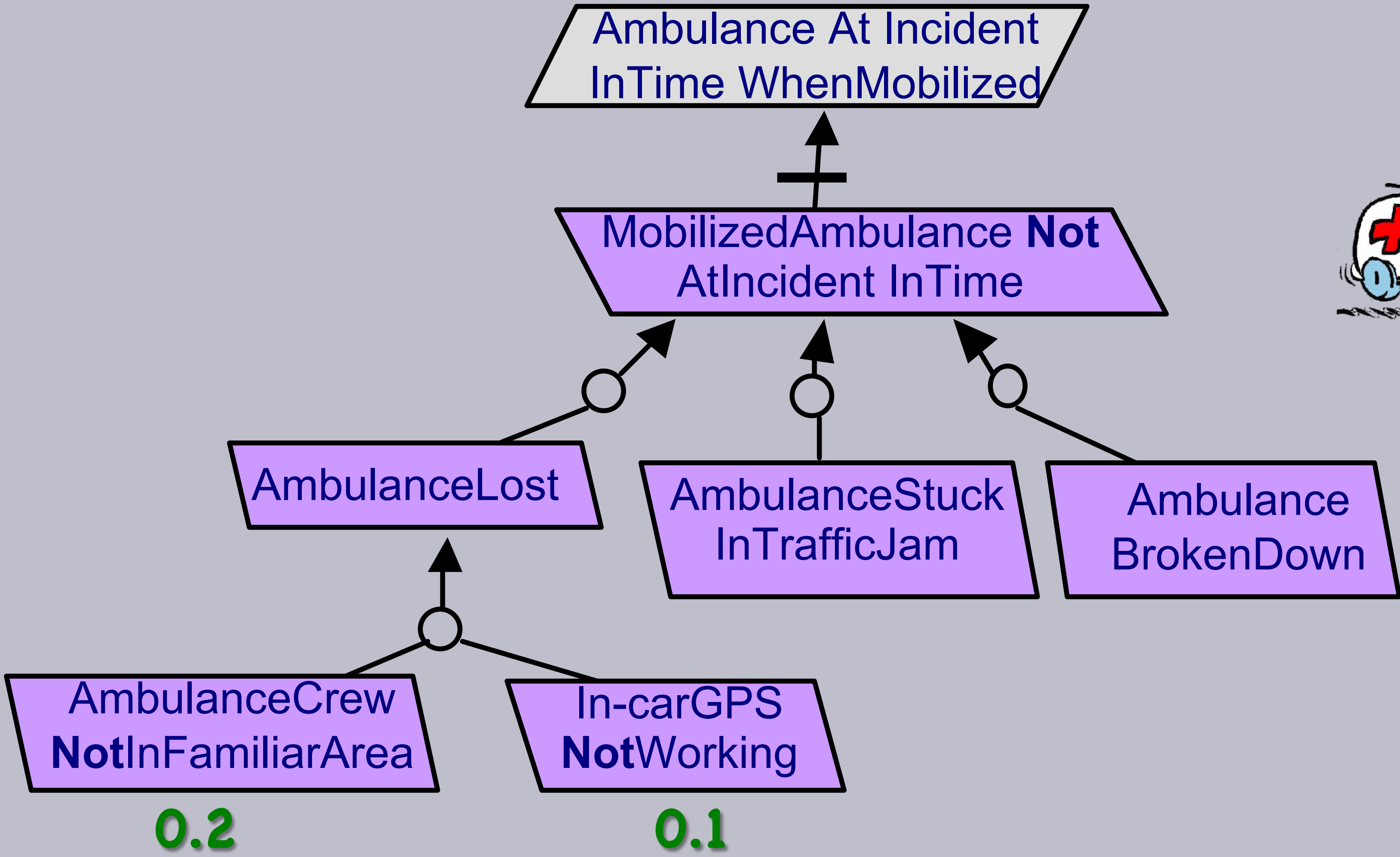
Assessing obstacles: example





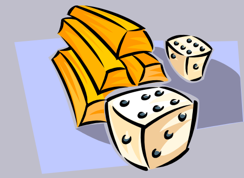
Assessing obstacles: example

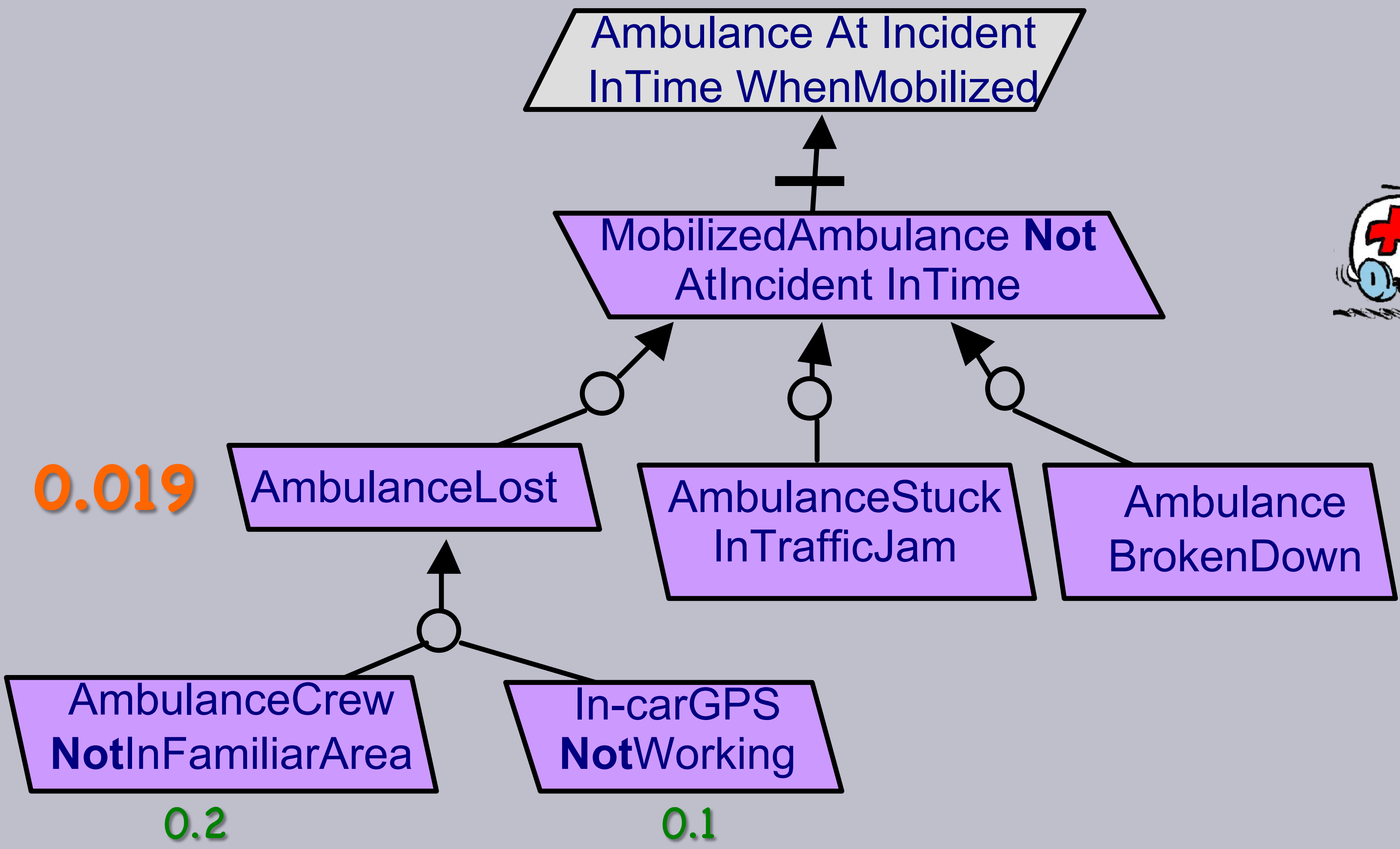




P (AmbulanceLost | NotInFamiliarArea, GPS NotWorking)) = 0.95

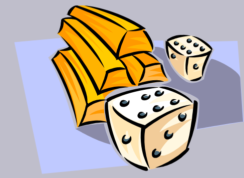
Assessing obstacles: example

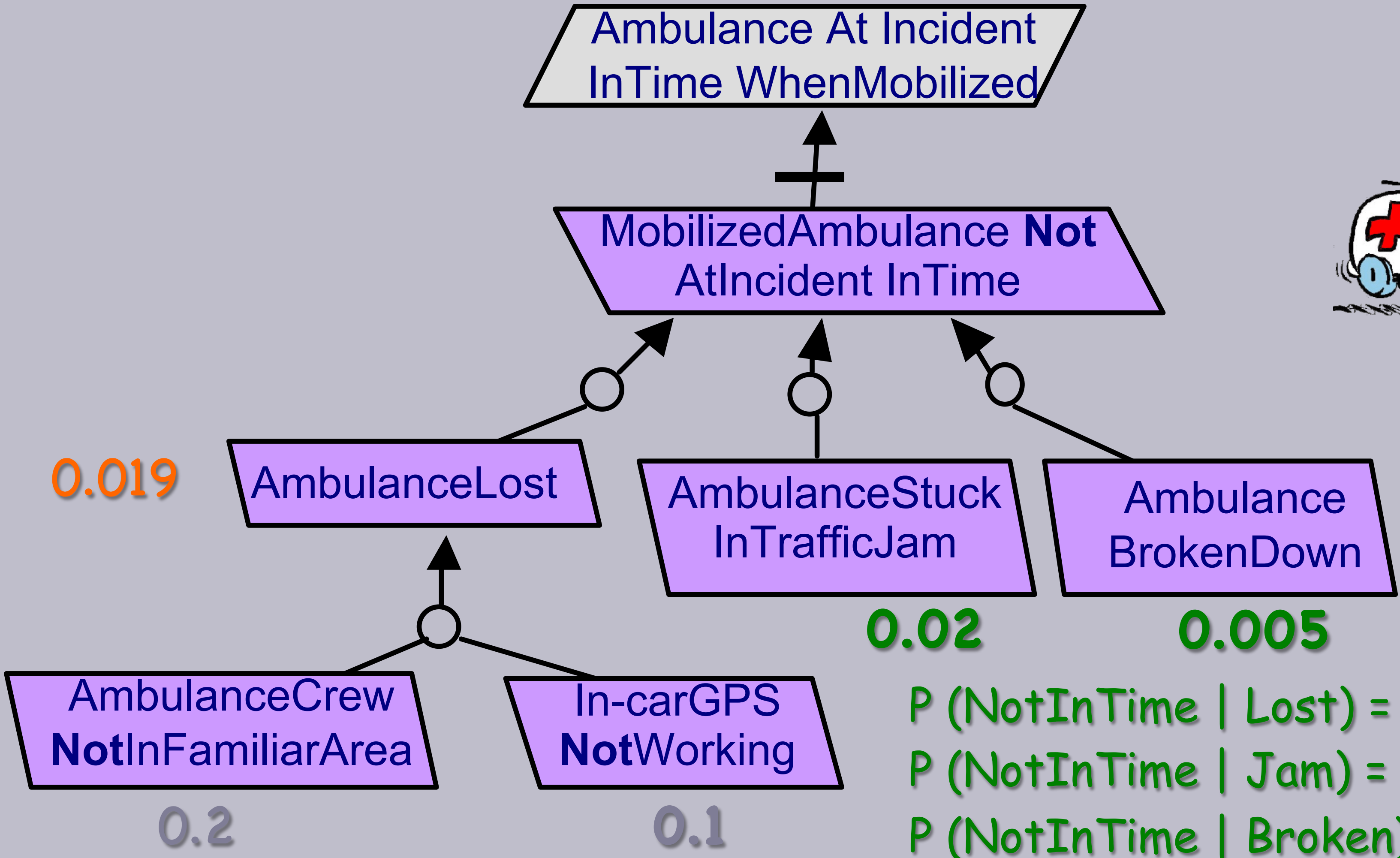




P (AmbulanceLost | NotInFamiliarArea, GPS NotWorking)) = 0.95

Assessing obstacles: example





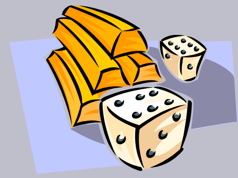
P (NotInTime | Lost) = 0.99

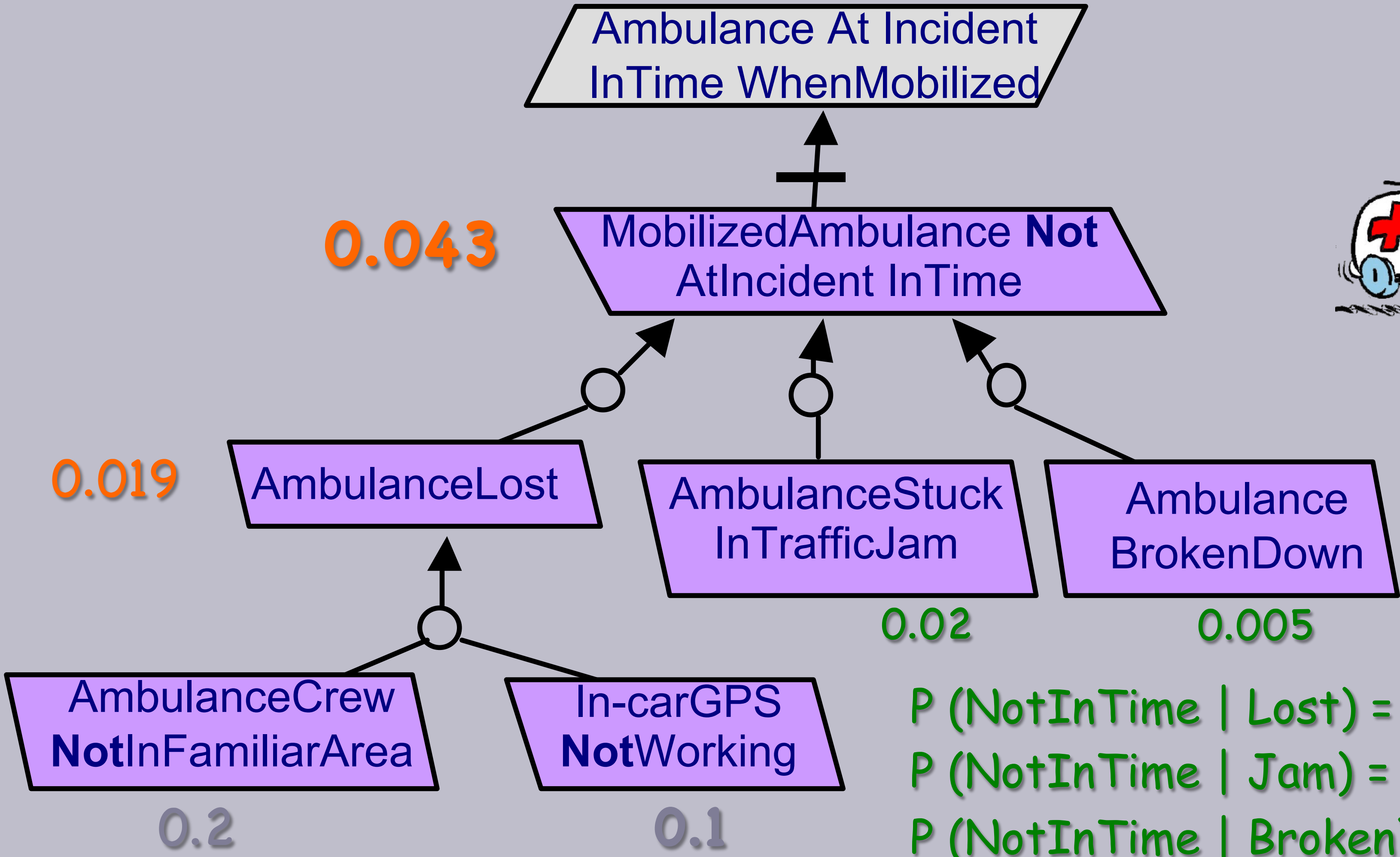
P (NotInTime | Jam) = 0.98

P (NotInTime | Broken) = 1

P (AmbulanceLost | NotInFamiliarArea, GPS NotWorking)) = 0.95

Assessing obstacles: example





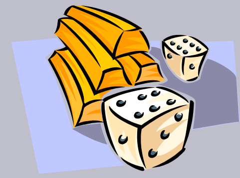
P (NotInTime | Lost) = 0.99

P (NotInTime | Jam) = 0.98

P (NotInTime | Broken) = 1

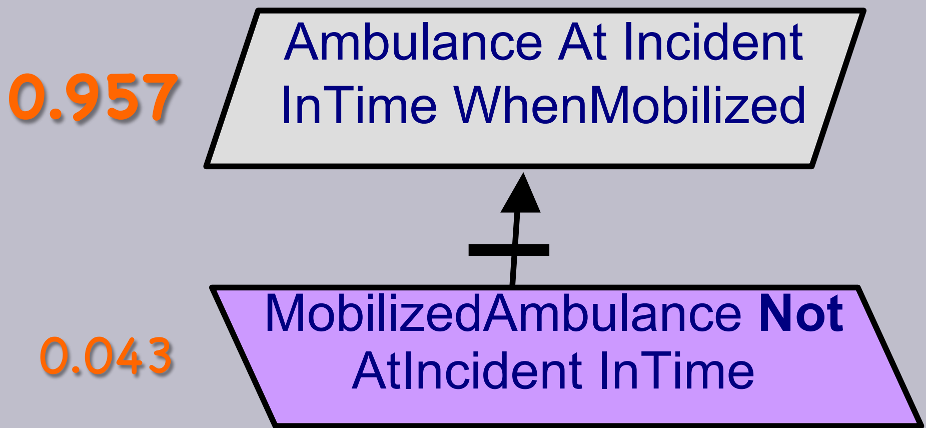
P (AmbulanceLost | NotInFamiliarArea, GPS NotWorking)) = 0.95

Assessing obstacle consequences

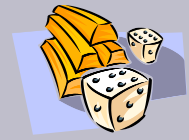


- Obstacle consequence = lower degree of satisfaction of ...
  - obstructed leaf goal,
  - its parent/ancestor goals
- Propagation from root obstacle to obstructed **leaf** goal:

  $1 - P(LG) = P(RO) \times P(\neg LG \mid RO)$



Assessing obstacle consequences: from obstructed leaf goals to higher-level goals



- Up-propagation through goal refinement graph ...
  - for single system with complete AND-refinements:

$$P(G) = P(SG_1, SG_2) + P(SG_1, \neg SG_2) \times P(G \mid SG_1, \neg SG_2) + P(SG_2, \neg SG_1) \times P(G \mid SG_2, \neg SG_1)$$

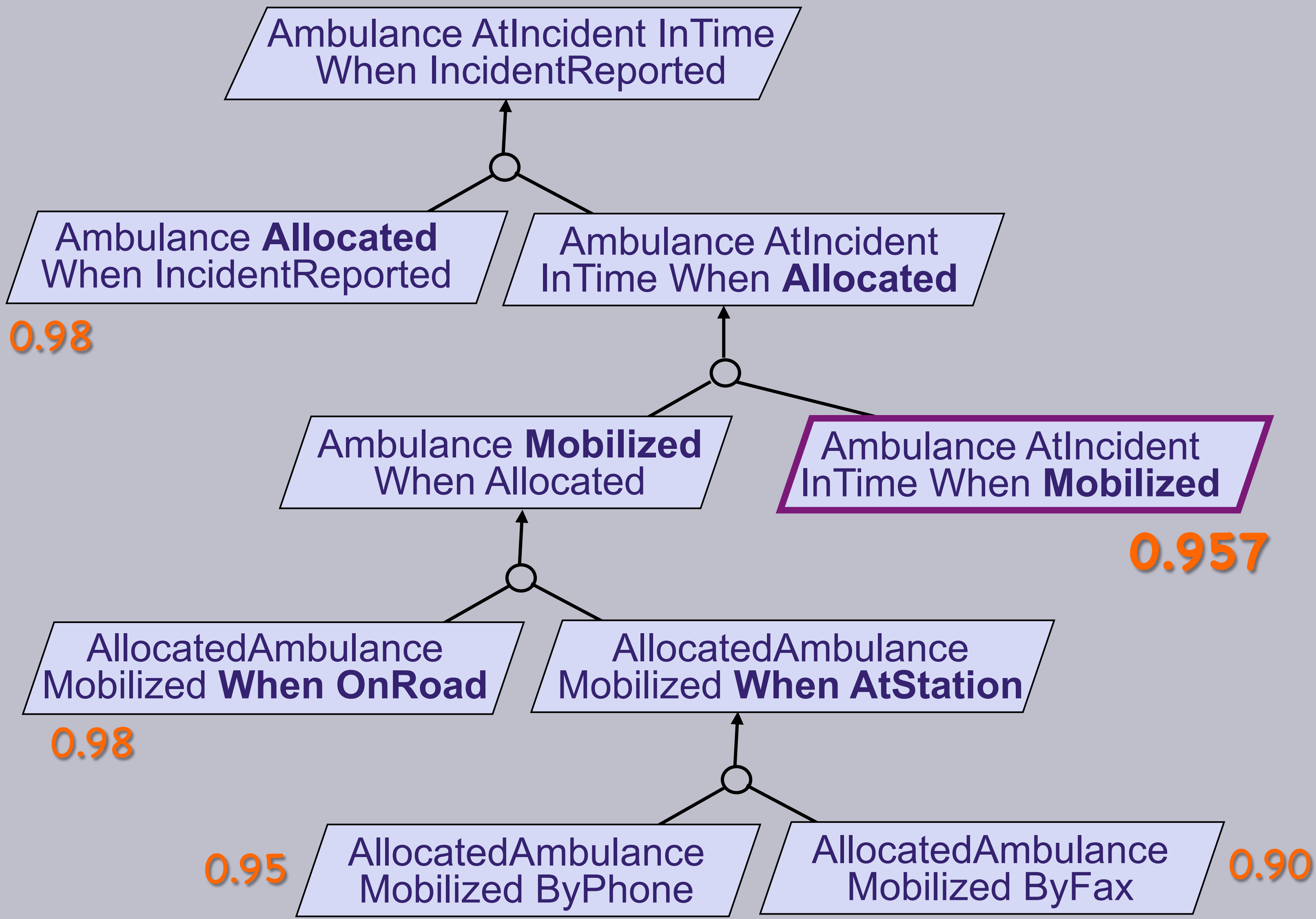
  - further simplification for refinement patterns (complete, minimal, consistent => independent subgoals)

$$P(G) = P(SG_1) \times P(SG_2) \qquad \textit{milestone-driven}$$

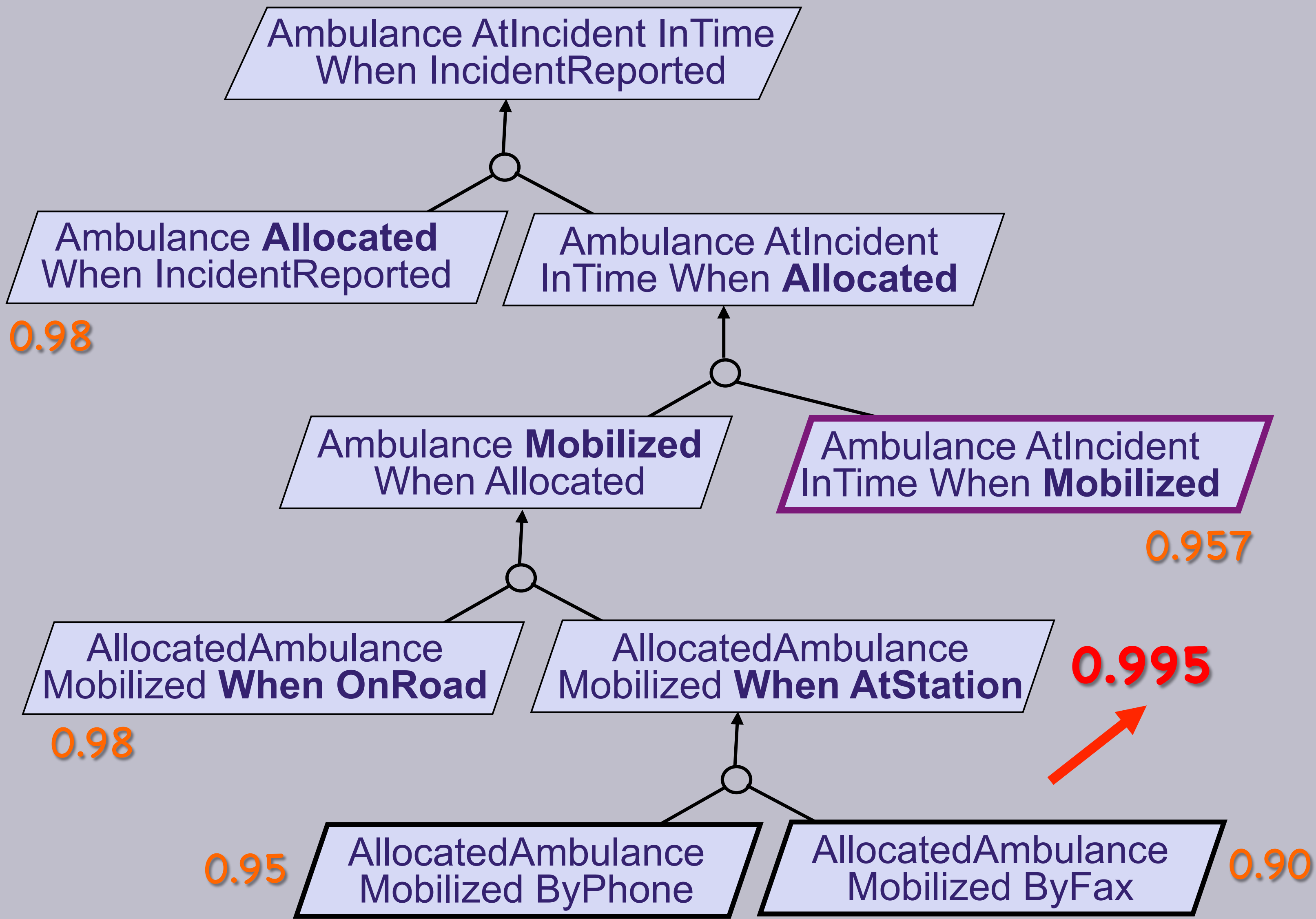
$$P(G) = P(CS) \times P(SG_1) + (1 - P(CS)) \times P(SG_2) \qquad \textit{case-driven}$$

- Two kinds of consequence assessment
  - **global**: severity SV (G) computed from all leaf goal obstructions
  - **local**: single leaf goal obstruction, all other leaf goals with P(LG) = 1

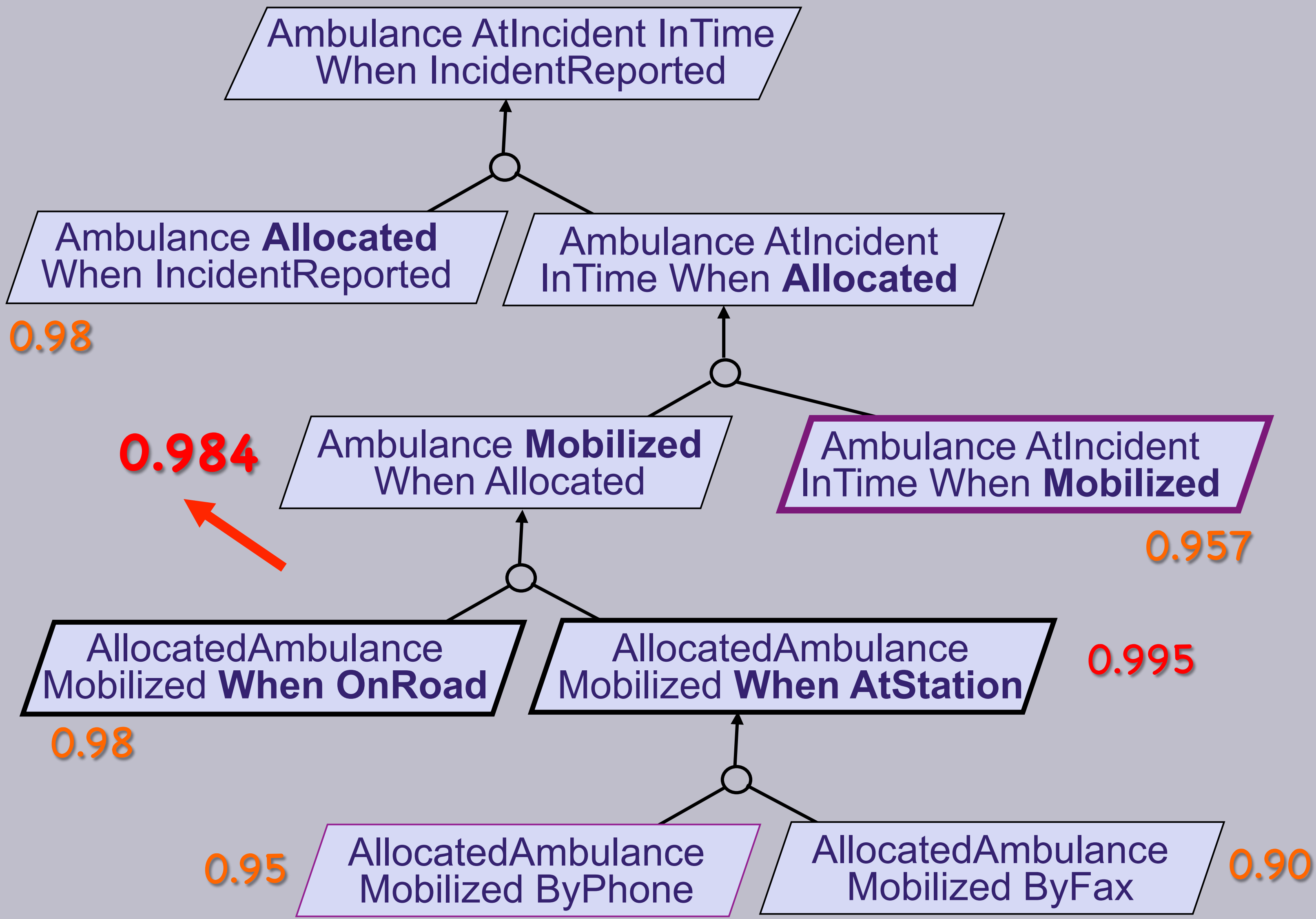
Global impact analysis: example



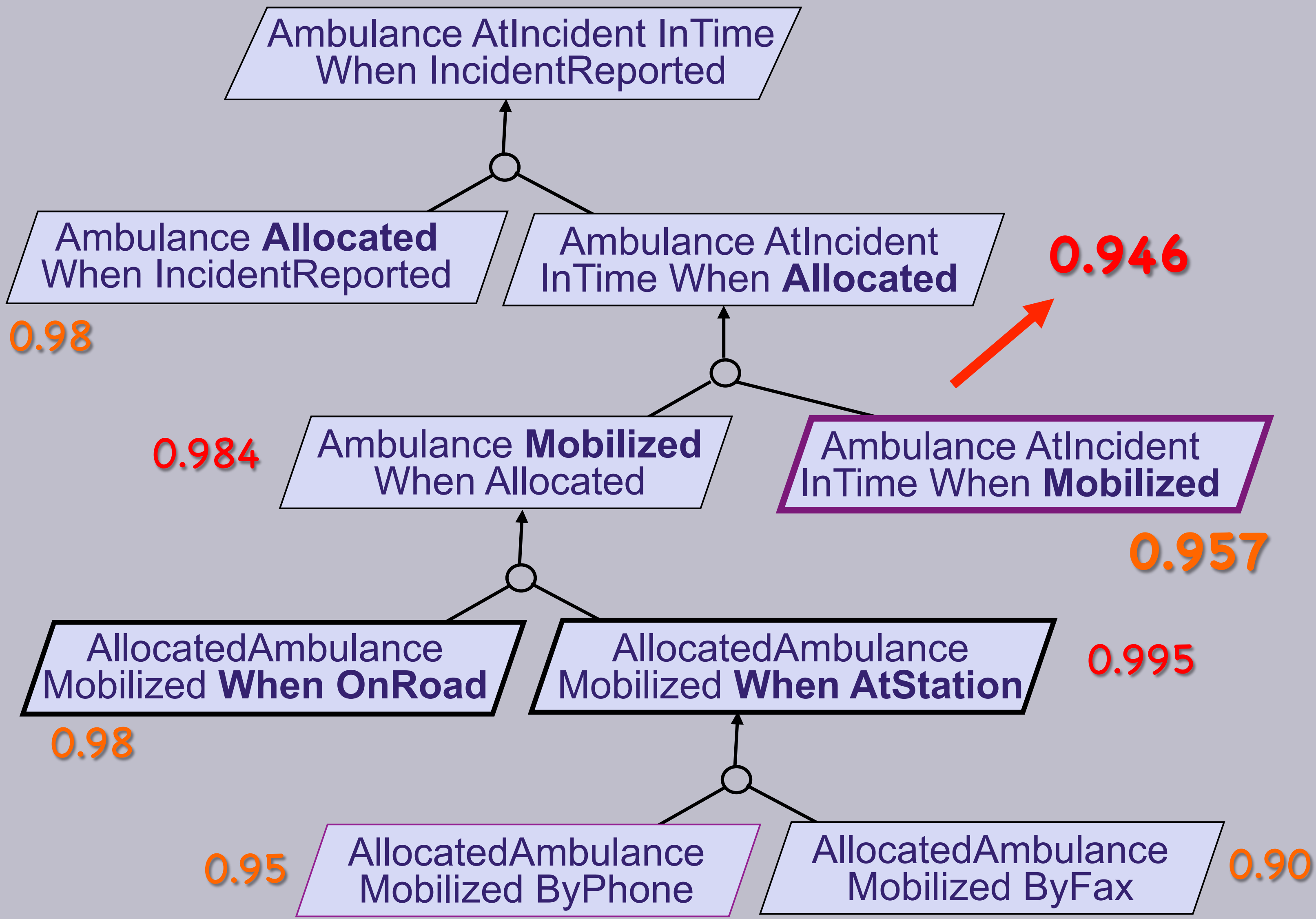
Global impact analysis: example



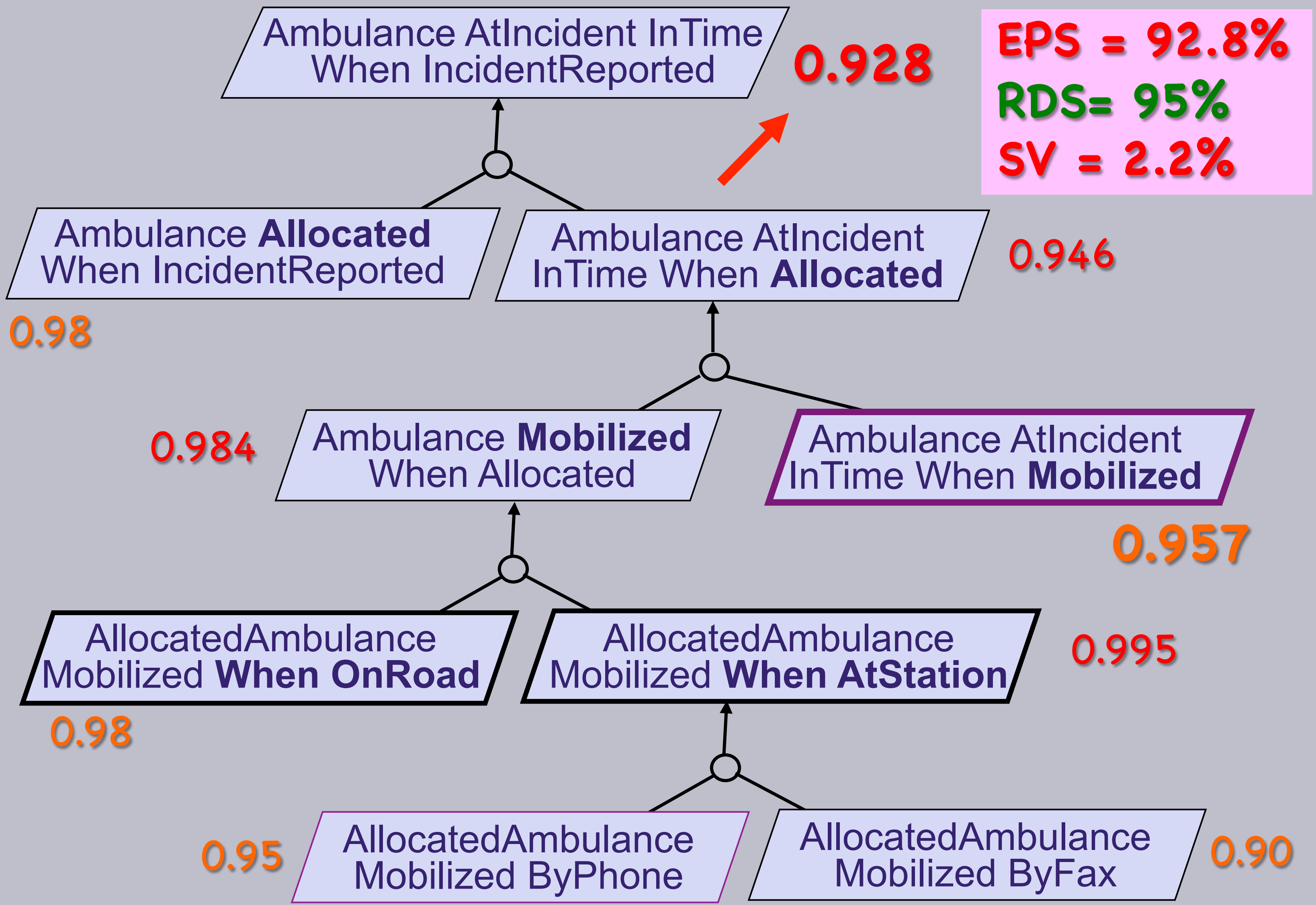
Global impact analysis: example



Global impact analysis: example



Global impact analysis: example



Identifying critical obstacle combinations

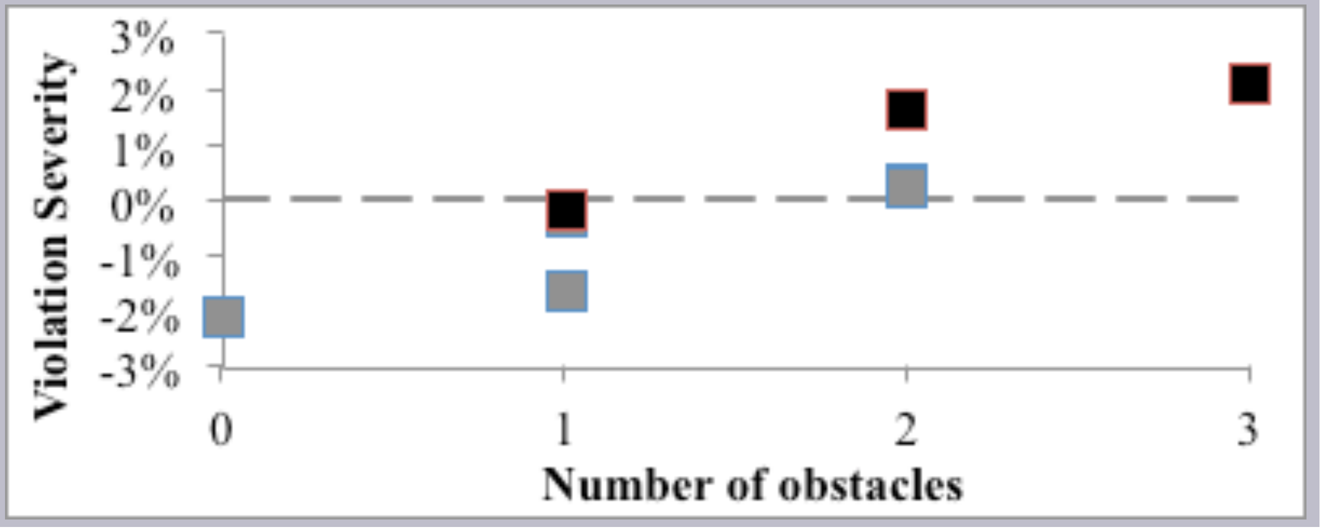
- Aim: focus resolution on most problematic leaf obstacles
- Multi-criteria optimization problem
  - minimal sets of leaf obstacles maximizing severity of goal violations ?
- Brute force solution
  - generate all leaf obstacle combinations
  - compute SV(G) for each obstructed G

    weighted according to goal priority
  - sort leaf obstacle combinations by severity
- Optimized techniques available for generating Pareto fronts [Kung et al, 1975]

Identifying critical obstacle combinations: example

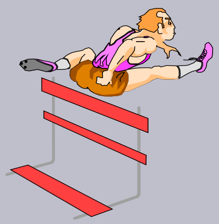
TABLE I. Violation severity for
Achieve [AmbulanceOnSceneInTimeWhenIncidentReported]

Amb. Lost	Amb. Stuck In Traffic	Amb. Broken Down	EPS	RDS	SV
1	1	1	92,77%	95%	**2,23%**
1	1	0	93,20%		**1,80%**
0	1	1	94,54%		**0,46%**
1	0	1	94,61%		**0,39%**
0	1	0	95,02%		-0,02%
1	0	0	95,10%		-0,10%
0	0	1	96,44%		-1,44%
0	0	0	96,92%		-1,92%



Outline

- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis



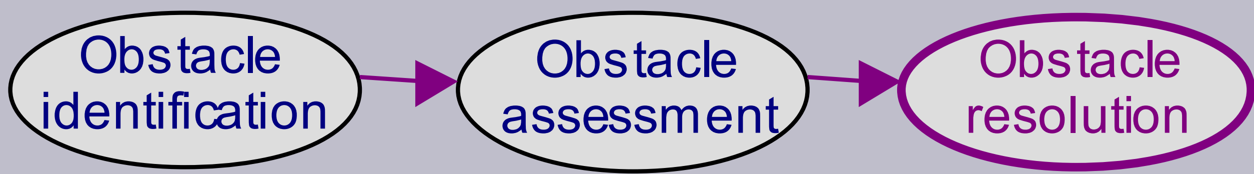
Resolving obstacles

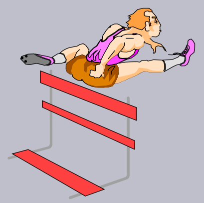
- *At RE time:* integrate countermeasures in the goal model
  - new or modified goals in goal model
  - often to be refined
- For every critical obstacle ...
  - explore alternative resolutions
  - select "best" resolution based on ...

    likelihood/severity of obstacle

    non-functional/quality goals in goal model
- *At system run-time*: obstacle monitoring, run-time resolution

  (non-severe, occasional obstacles) [Feather et al, 1998]





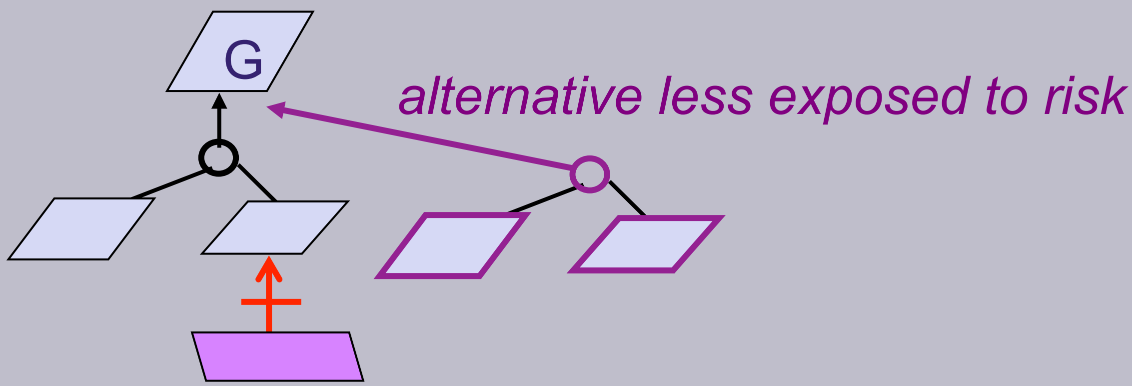
Exploring alternative countermeasures

By use of **model transformation operators**

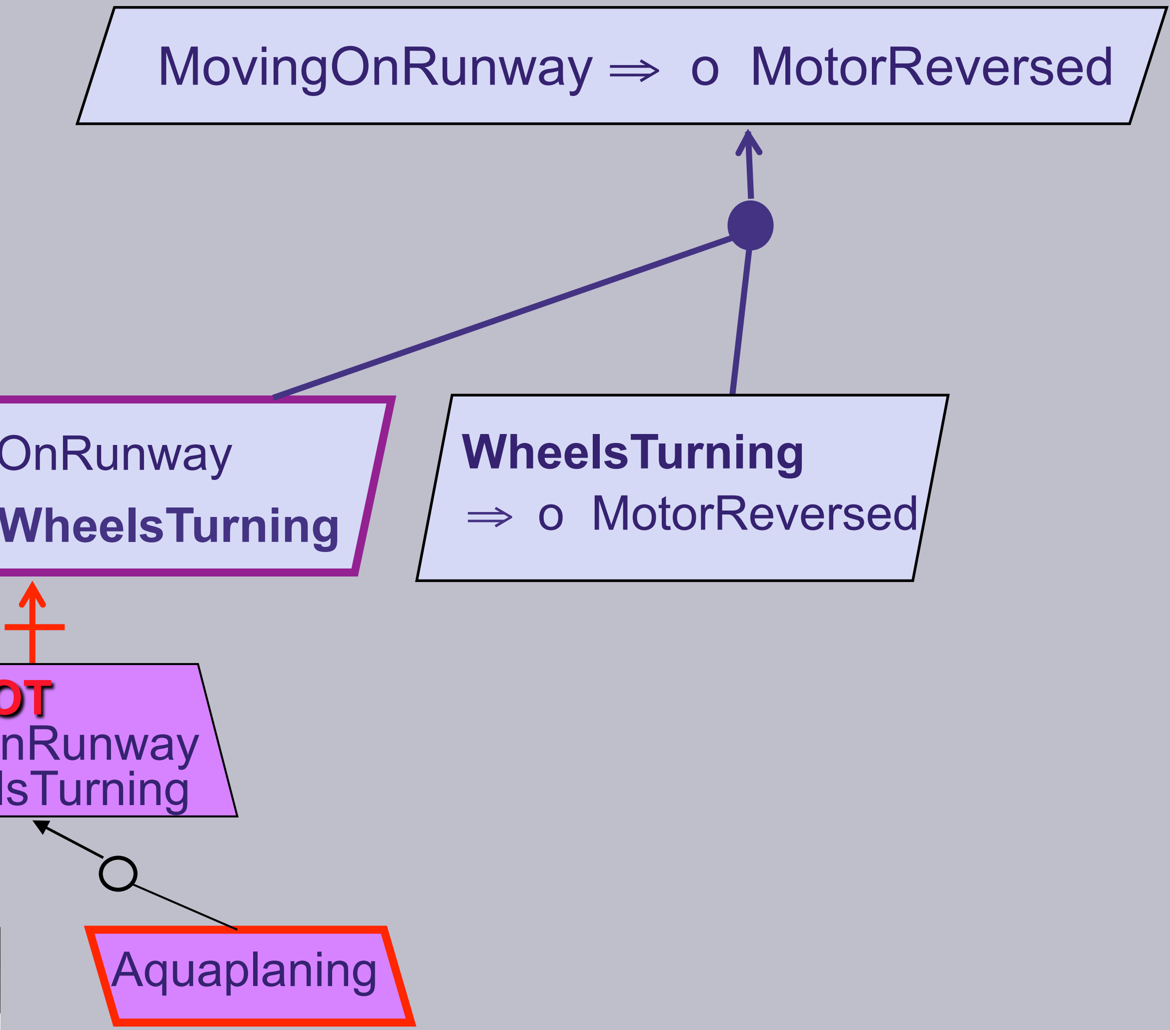
- encode resolution tactics

- **Goal substitution**:

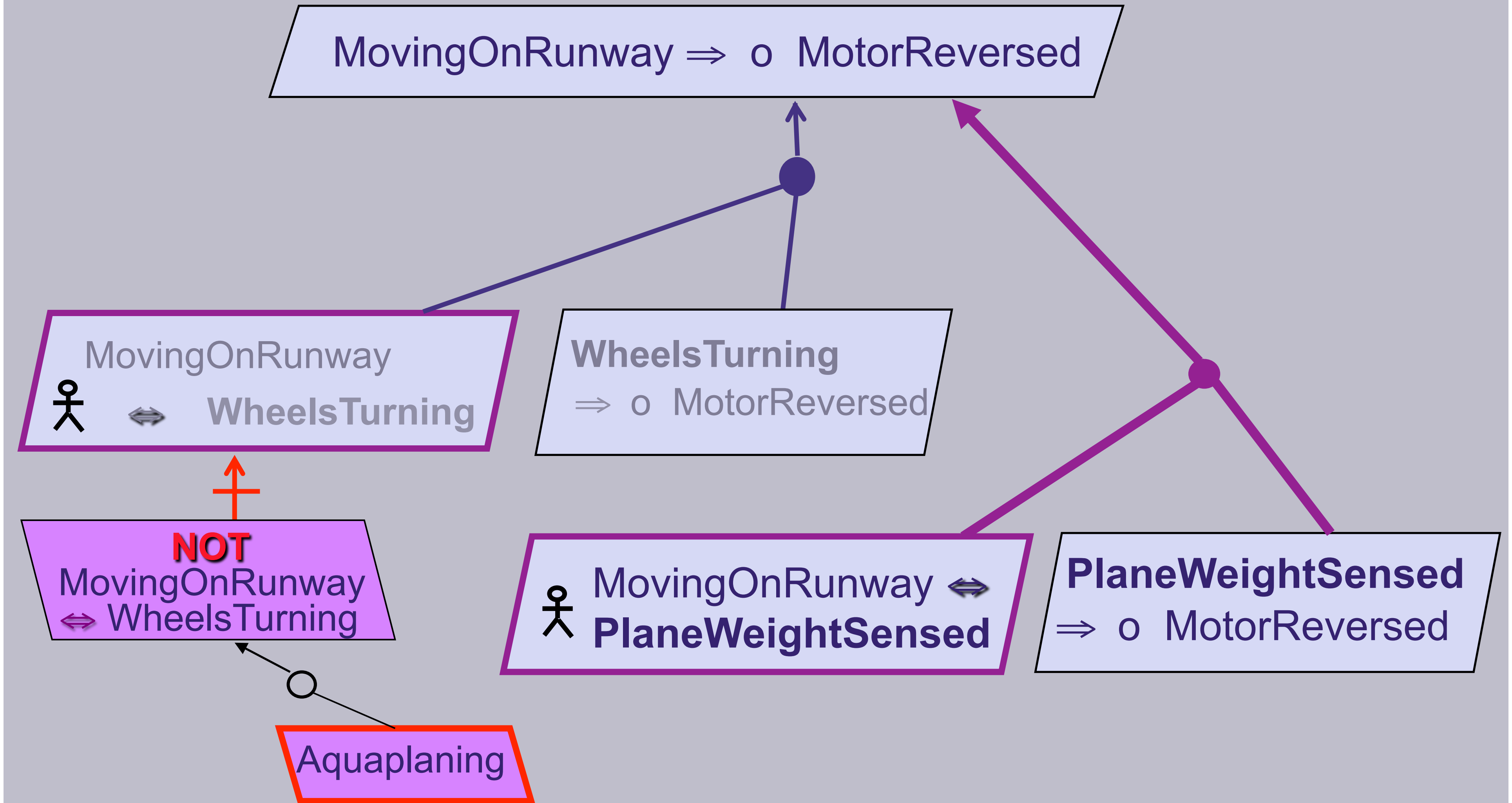
  consider alternative refinement of parent goal
  to avoid obstruction of child goal

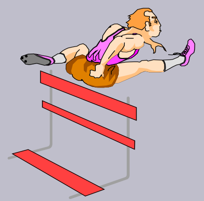


Goal substitution: example



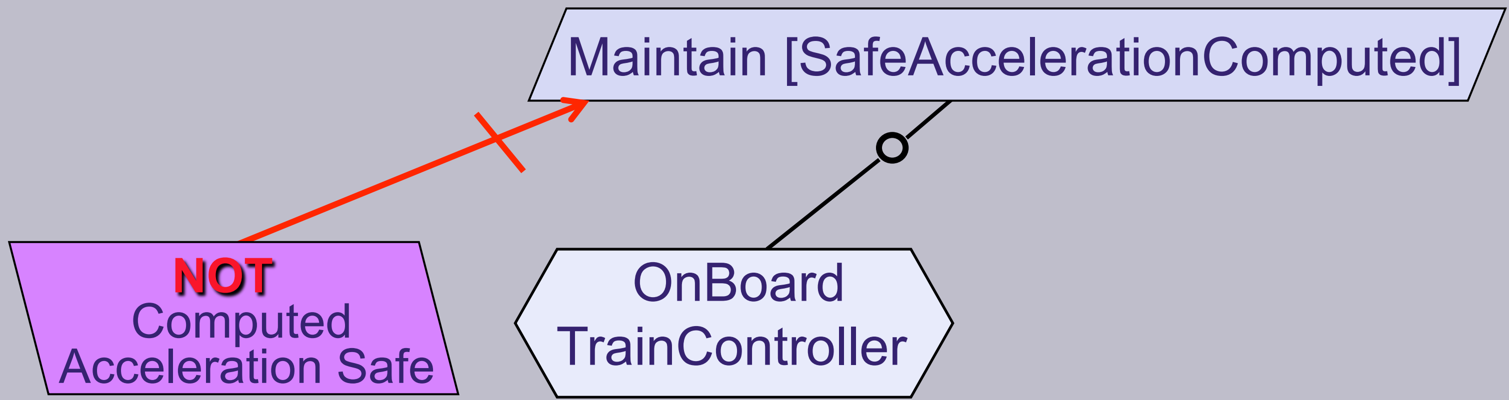
Goal substitution: example

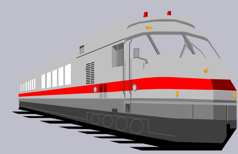


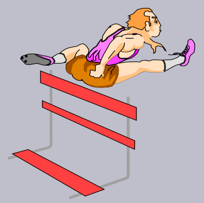


Exploring alternative countermeasures (2)

- **Agent substitution**: consider alternative responsibilities for obstructed goal so as to make obstacle unfeasible

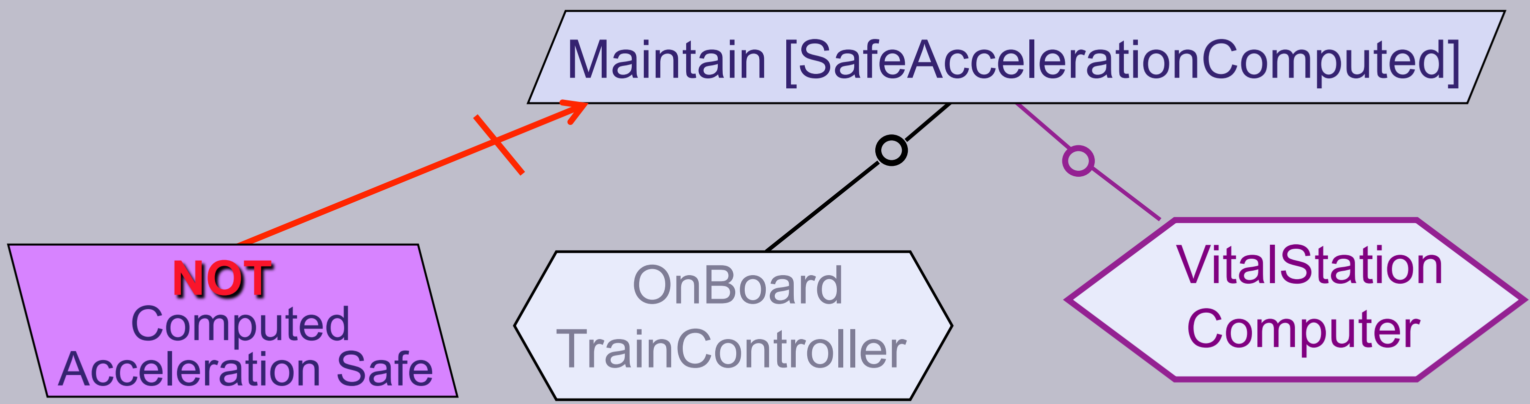


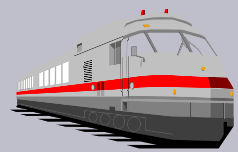


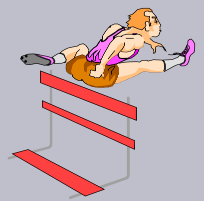


Exploring alternative countermeasures (2)

- **Agent substitution**: consider alternative responsibilities for obstructed goal so as to make obstacle unfeasible

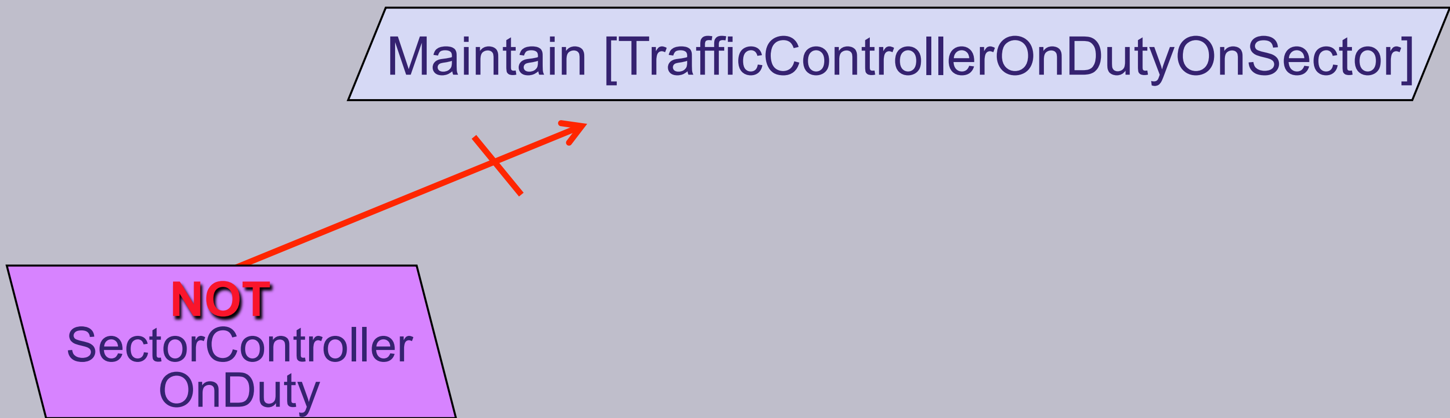




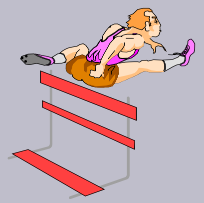


Exploring alternative countermeasures (3)

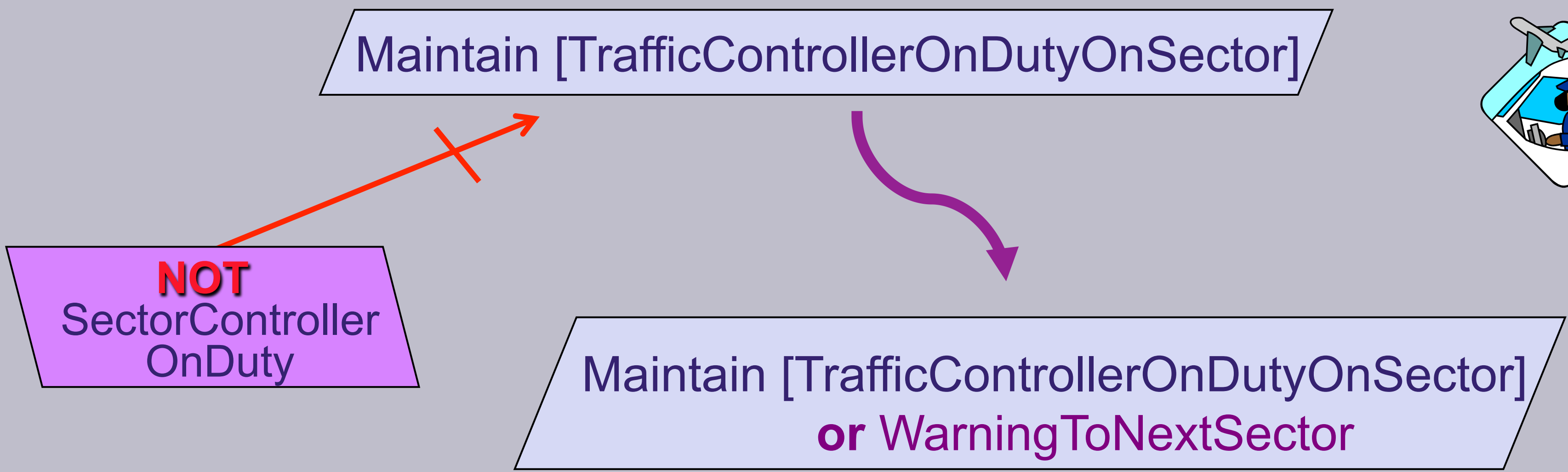
- **Goal weakening**: weaken the obstructed goal so that the weaker version is no longer obstructed
  - for goal specs ***A*** ⇒ ***C***: add conjunct in ***A***
    add disjunct in ***C***

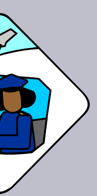


Exploring alternative countermeasures (3)



- **Goal weakening**: weaken the obstructed goal so that the weaker version is no longer obstructed
  - for goal specs $A \Rightarrow C$: add conjunct in A
    add disjunct in C

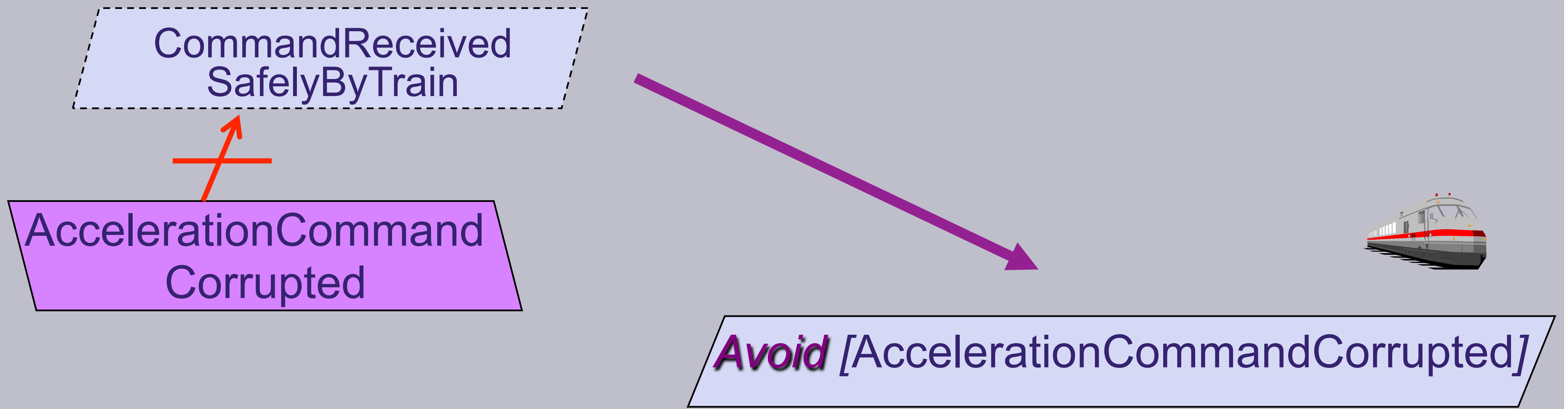


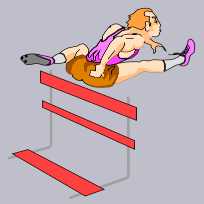


Exploring alternative countermeasures (4)

- **Obstacle prevention**:
  - introduce new goal: ***Avoid* [obstacle]**
  - to be further refined
  - standard resolution tactics for security threats

    Avoid [VulnerabilityCondition]



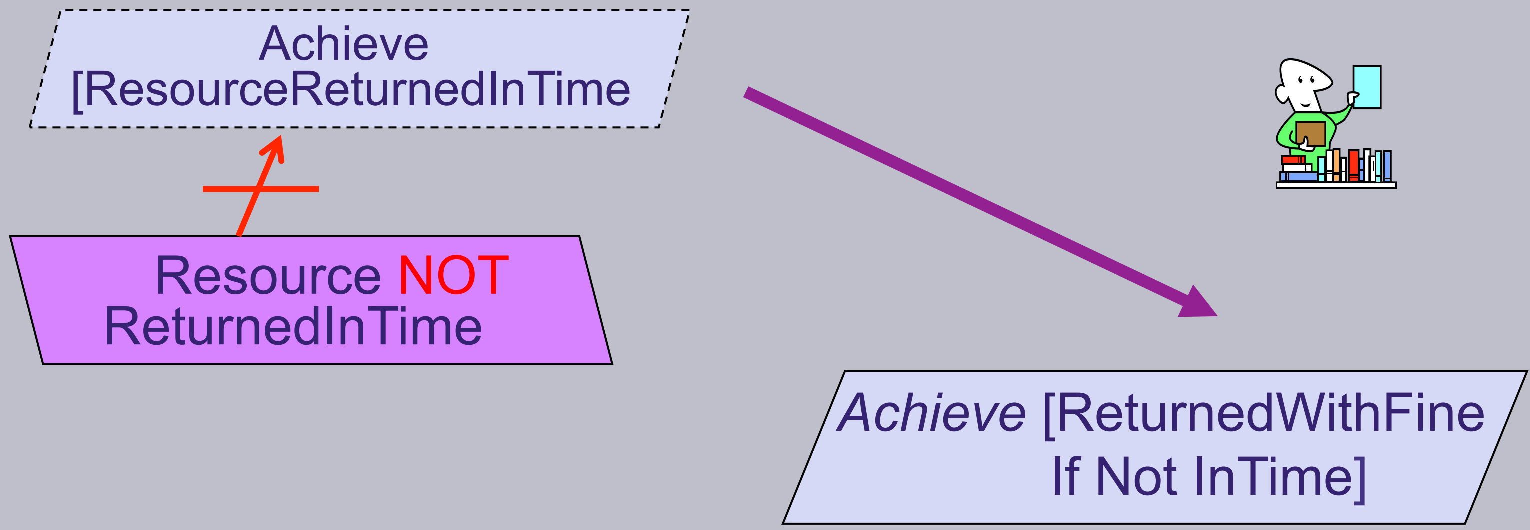


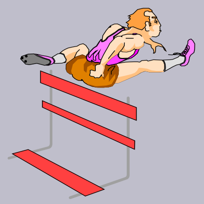
Exploring alternative countermeasures (5)

- **Goal restoration**:

  enforce goal's target condition as obstacle occurs

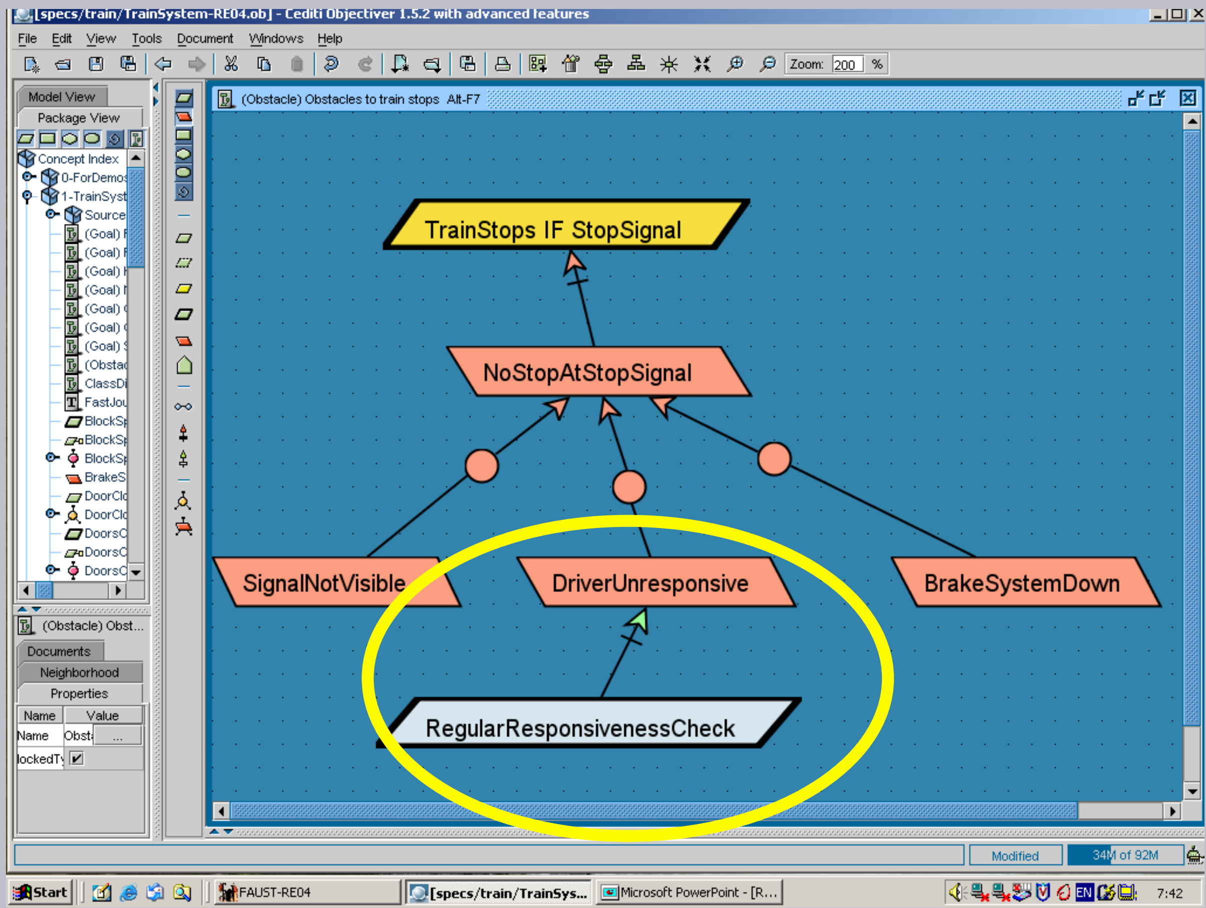
  => new goal: $O \Rightarrow \Diamond\, TargetCondition$

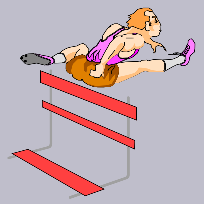




Exploring alternative countermeasures (6)

- **Obstacle reduction**: reduce obstacle likelihood
  by ad-hoc countermeasure





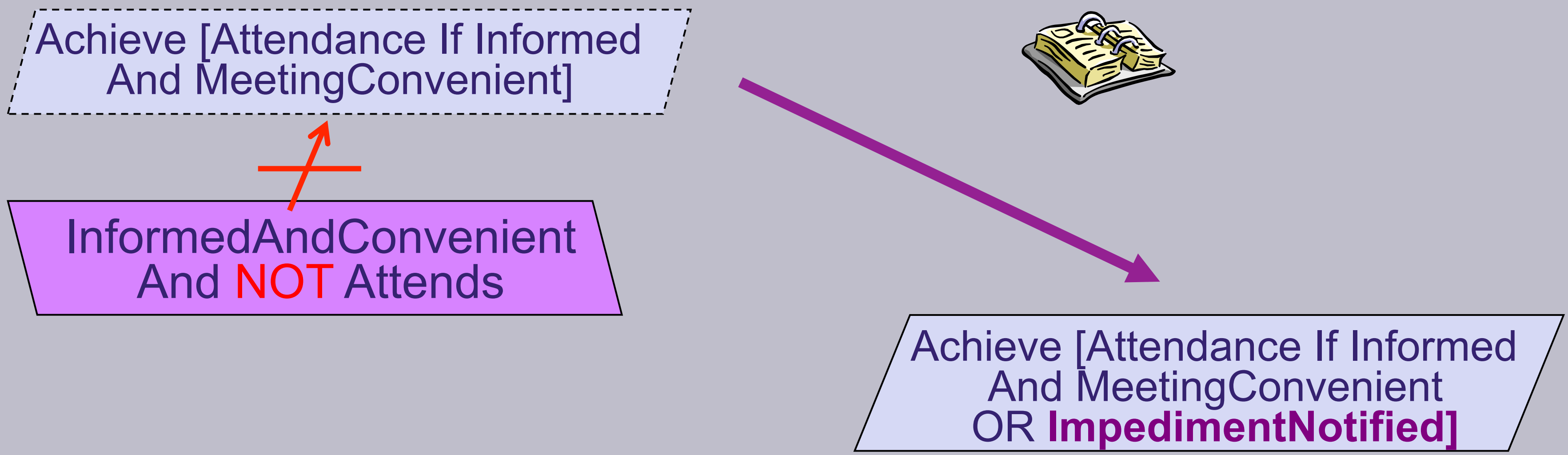
Exploring alternative countermeasures (7)

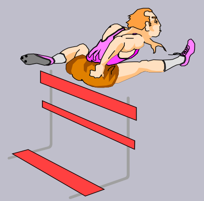
- **Obstacle mitigation**:

  introduce new goal to mitigate **consequences** of obstacle

  - **Weak mitigation**:

    new goal ensures weaker goal version when obstructed





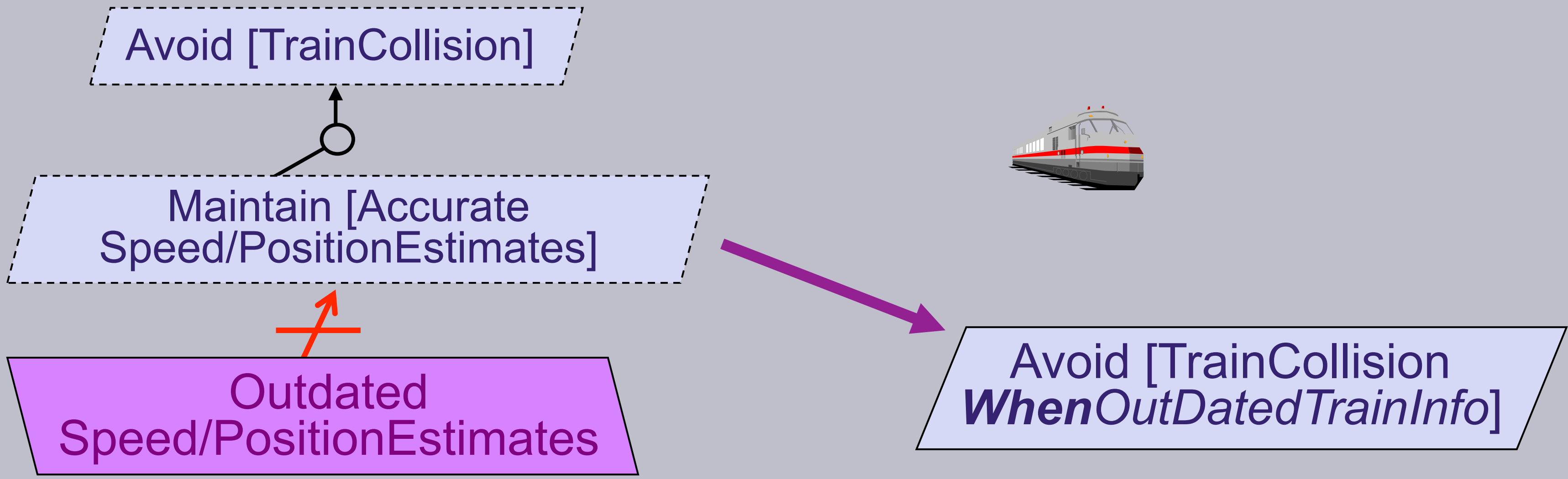
Exploring alternative countermeasures (7)

- Obstacle mitigation:

  introduce new goal to mitigate consequences of obstacle

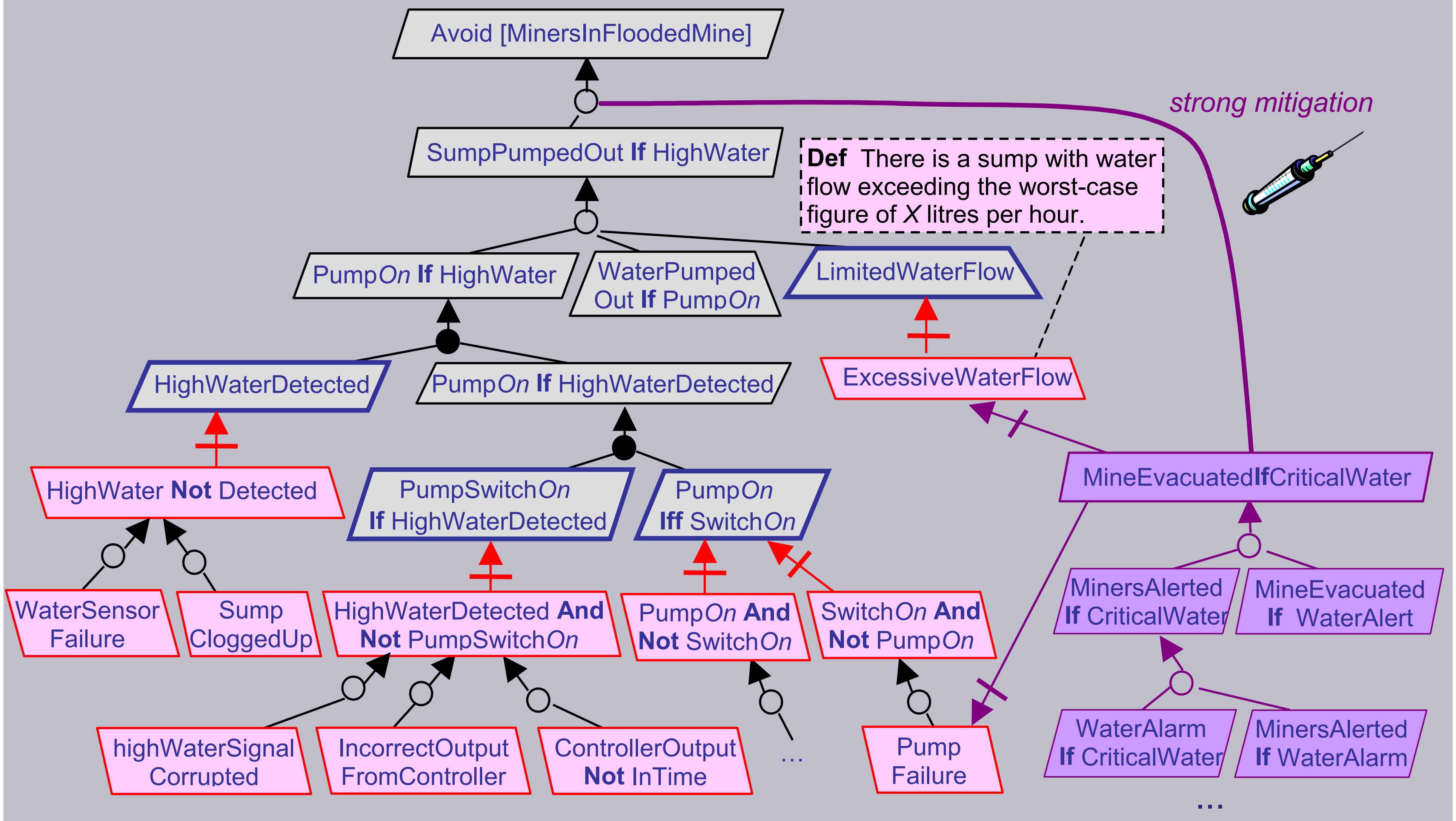
  - **Strong mitigation:**

    new goal ensures ***parent*** of goal when obstructed



Resolution goals must then be further refined in the goal model

Strong mitigation: example



An interesting perspective: obstacle resolution as theory revision

- Given:
  - B: knowledge base (domain properties)
  - E: examples (traces)
  - M: mode declaration (language bias)
  - R_M: a rule space
  - $R \subseteq R_M$: a revisable theory (goal model)
- Find:
  - R′ : a revised theory with distance c (R,R′)
    - obtained by deleting rules, adding/deleting & conditions to/from rules
    - $R' \subseteq R_M$
    - $B \cup R' \models E$
    - c (R, R′) is minimal

Selecting best resolution

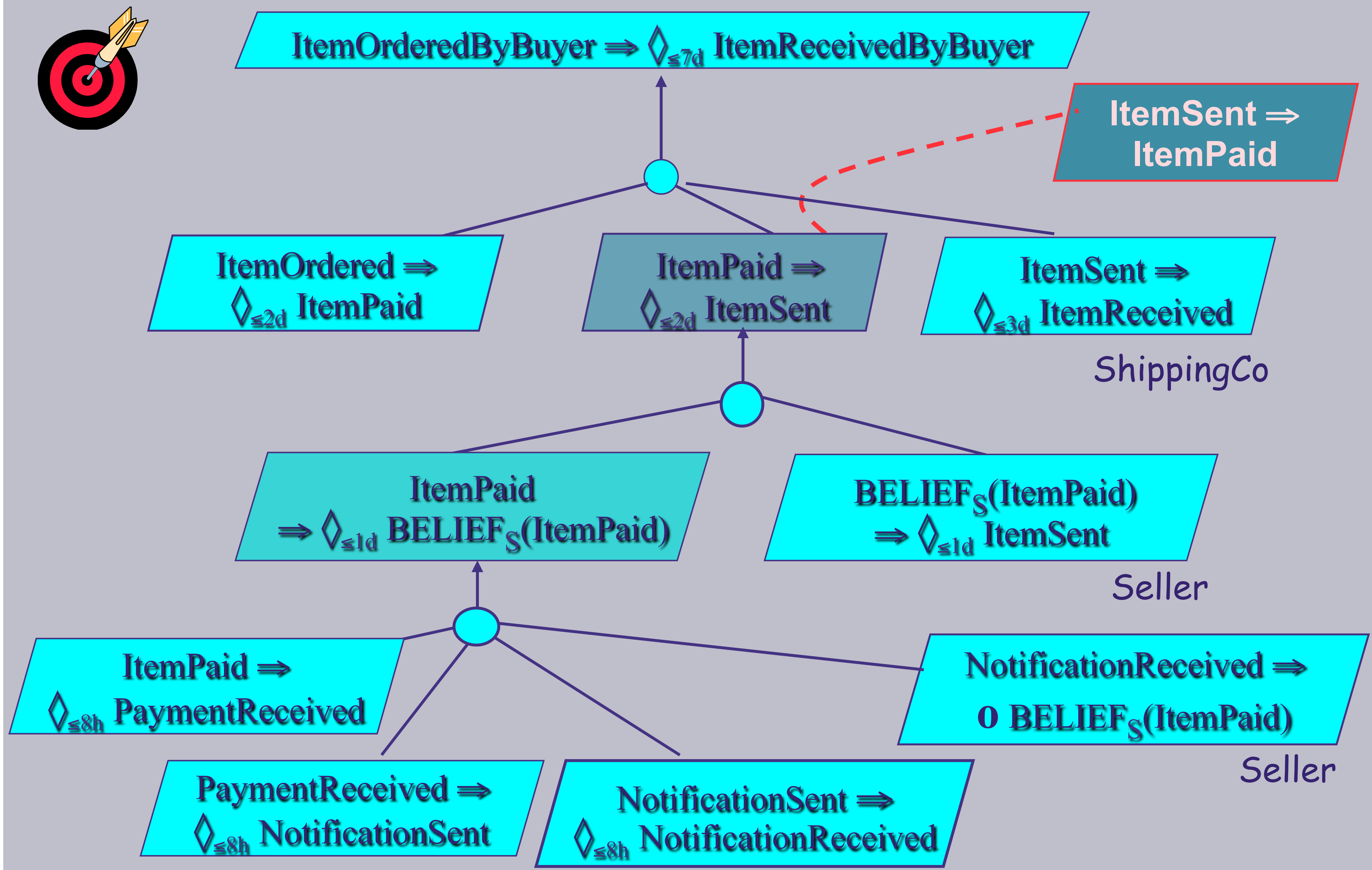
- Evaluation criteria for comparing alternative resolutions ...
  - number of obstacles resolved by the alternative
  - their likelihood & criticality
  - the resolution's contribution to soft goals
  - its cost
- May be based on estimates of ...
  - risk-reduction leverage
  - qualitative/quantitative contribution to soft goals [Mylopoulos et al]
- If obstacle not eliminated, multiple alternatives may be taken

  e.g. FineCharged + ReminderSent (for book copies not returned in time)
- Selected alternative => new/weakened goal in goal model
  - resolution link to obstacle for traceability
  - weakening may need to be propagated in goal model
  - to be refined & checked for conflicts & new obstacles

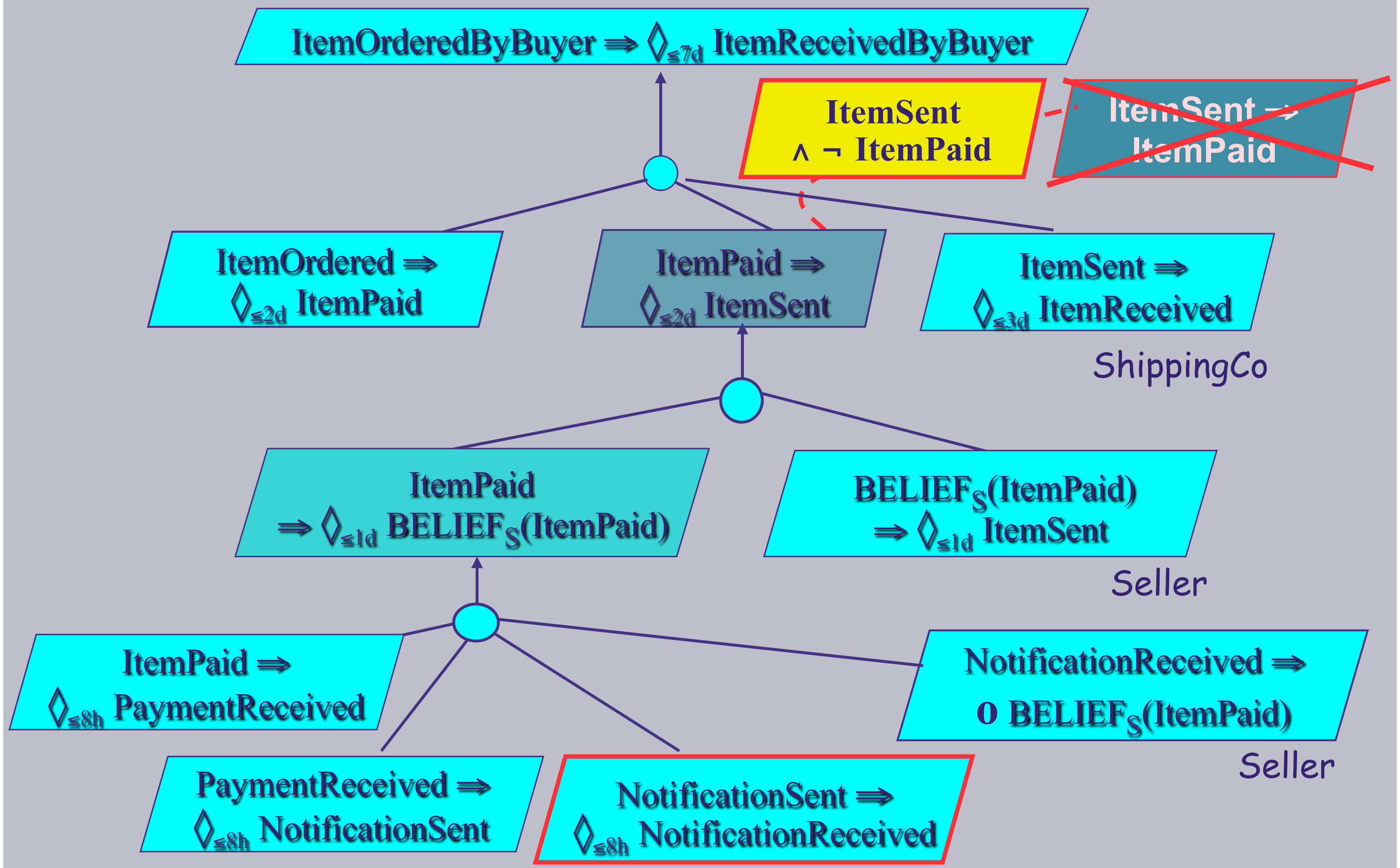
Outline

- Introduction: requirements engineering and risk management
- Background: goal-oriented model building & analysis
  - Basic concepts & modeling technique
  - Specifying model elements
  - Goal refinement and operationalization
- Obstacle analysis for risk-driven RE
- Obstacle identification
  - Regressing goal negations
  - Reusing obstruction patterns
  - Combining model checking & inductive learning
- Obstacle assessment
  - Probabilistic goals & obstacles
  - Assessing the likelihood & severity of obstacles
- Obstacle resolution for a more complete goal model
- Beyond unintentional obstacles: threat analysis

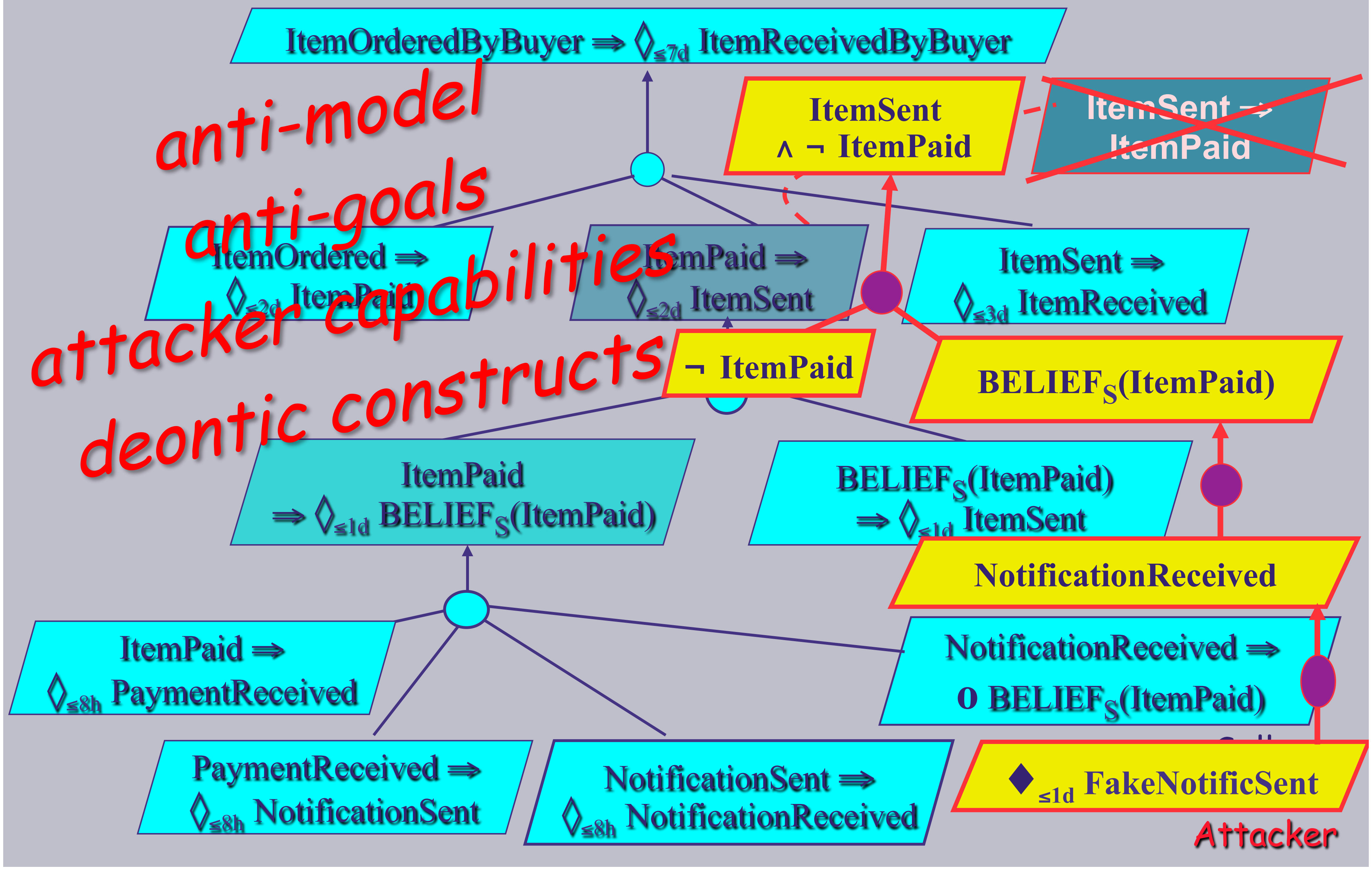
Threat analysis for more secure model



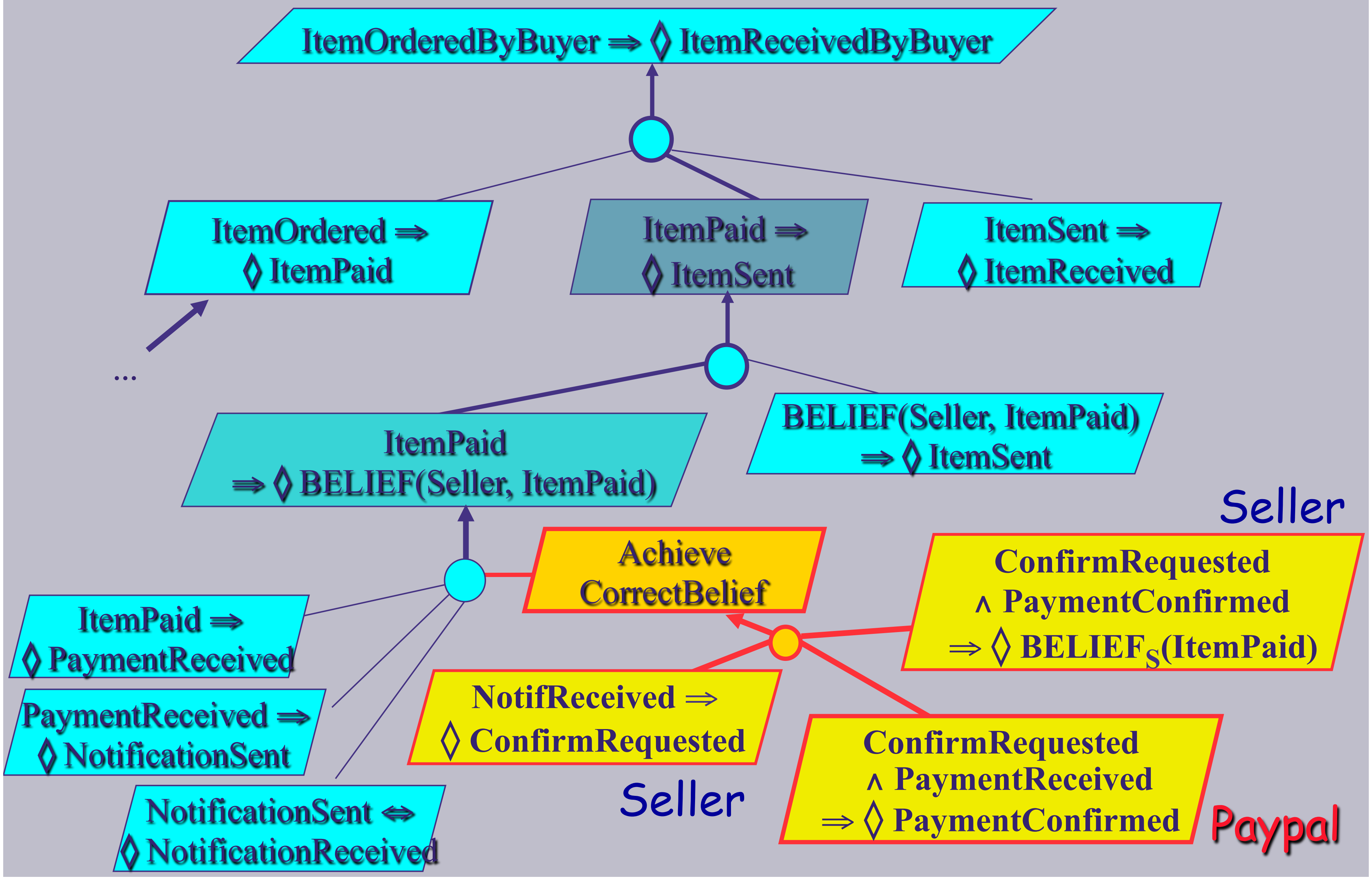
Threat analysis for more secure model



Threat analysis for more secure model



Model completed with countermeasures



Industrial application:

Security of Aircraft in the Future European Environment

Threats against crew & passengers

(External threats)

with Airbus,
British Aerospace,
SAGEM, Marconi, ...

Threats from baggage area

- Modeling terrorist threats (huge anti-goal model)
- For on-board detection & reaction system

Conclusion

- It is important to verify that your software implements its specs correctly... ***BUT*** ...
- ... are those **specs** meeting the software **requirements** (including non-functional ones) ?
- ... are those **requirements** meeting the system's **goals** ? … under realistic **assumptions** ?
- ... are such goals, requirements & assumptions **complete**, consistent, adequate and realistic ?

this is a critical though still largely unexplored area

with many challenging issues for formal methods

Conclusion

- **Problem**-oriented abstractions, declarative specs are needed
  for ... communication with stakeholders
  early, incremental analysis of partial models

- **Systematic** techniques are needed for model construction
  - from high-level goals to detailed operational specs
    from detailed operational specs to high-level goals
  - appropriate mix of **deductive** & **inductive** techniques

- Importance of capturing the right assumptions
  (+ satisfaction args)

Conclusion

- Be pessimistic from beginning about software ***and*** *environment,* anticipate what could go wrong

    hazards, threats, conflicts, ...

- Multi-button approach
    - semi-formal

        for navigation, traceability ... and *accessibility*

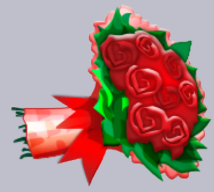
    - formal, when and where needed

        for precise, automated reasoning on model pieces

Rigorous approaches needed

Many opportunities for interesting research!

Thanks ...



- to the KAOS crew *at UCL, CEDITI and CETIC* as *researchers, consultants*, or *tool developers*

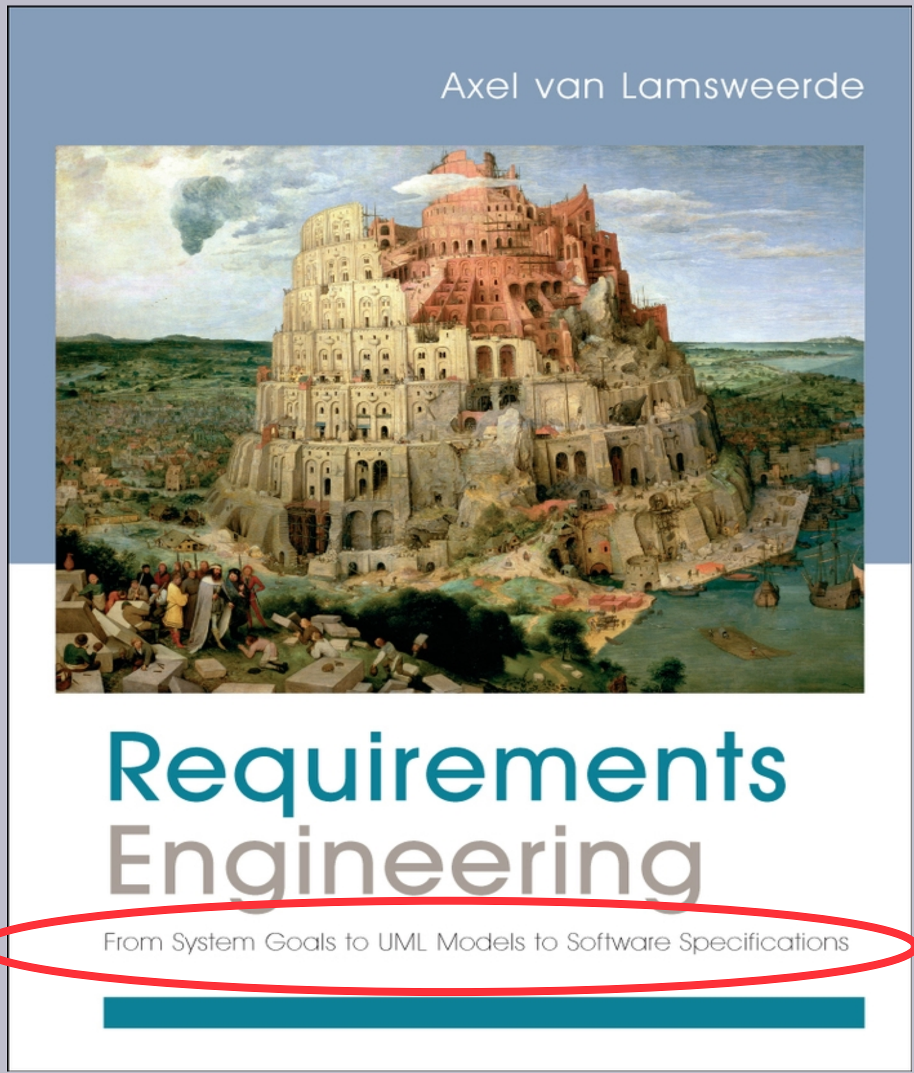
  C. Damas, A. Cailliau, A. Dardenne, R. Darimont,

  R. De Landtsheer, E. Delor, B. Lambeau, E. Letier,

  P. Massonet, C. Ponsard, A. Rifaut, H. Tran Van

- to Steve Fickas and his group at Univ. Oregon
- to Jeff Kramer and his group at Imperial College
- to the EU & Region of Wallonia for significant funding of those efforts

Much, much more info in ...



Wiley, 2009

Fruitful bedtime reading

A. van Lamsweerde & E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering, Special Issue on Exception Handling*, Vol. 26, No. 10, October 2000.

D. Alrajeh, J. Kramer, A. van Lamsweerde, A. Russo, S. Uchitel, Generating Obstacle Conditions for Requirements Completeness, *Proc ICSE' 2012 - 34th Intl Conf on Software Engineering*, Zurich, June 2012, ACM-IEEE.

A. Cailliau & A. van Lamsweerde, A Probabilistic Framework for Goal-Oriented Risk Analysis, Proc. RE'2012: 20th IEEE Intl Conf. on Requirements Engineering, Chicago, Sept. 2012.

A. van Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models", *Proc ICSE' 04 - 26th Intl Conf on Software Engineering*, Edinburgh, May 2004, ACM-IEEE, 148-157.

A. van Lamsweerde, R. Darimont & E. Letier, Managing Conflicts in Goal-Driven Rquirements Engineering, *IEEE Transactions on Software Engineering,* Vol. 24 No. 11, November 1998, pp. 908 - 926.

Fruitful bedtime reading (2)

R. Darimont & A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration". *Proc. FSE-4 - Fourth ACM Conf on Foundations of Software Engineering*, San Francisco, Oct. 1996, 179-190.

E. Letier & A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", *Proc. ICSE'2002 - 24th Intl Conf on Software Engineering*, Orlando, May 2002, IEEE CS Press, 83-93.

E. Letier & A. van Lamsweerde, "Deriving Operational Software Specifications from System Goals", *Proc FSE'2002 - 10th ACM Conf on the Foundations of Software Engineering*, Charleston (South Carolina), November 2002.

A. van Lamsweerde and L. Willemet, Inferring Declarative Rquirements Specifications from Operational Scenarios, *IEEE Transactions on Software Engineering,* Vol. 24 No. 12, December 1998, pp. 1089 - 1114.

E. Letier and A. van Lamsweerde, "Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering", *Proc FSE'04, 12th ACM Intl Symp. Foundations of Software Engineering*, Newport Beach (CA), Nov. 2004, 53-62.