Epistemic Logic and Planning

Shahin Maghsoudi[1], Ian Watson[2]

Computer Science Department - The University of Auckland
[1] mmag005@ec.auckland.ac.nz [2] ian@cs.auckland.ac.nz

Abstract. Artificial Intelligence algorithms can be divided into two groups according to the type of problems they solve. Knowledge-intensive domains contain explicit knowledge, whereas knowledge-poor domains contain implicit knowledge. Logical methods are more suitable for the first type. Neural networks and case-based reasoning (CBR) are more suitable for the second type. This project combines the inferencing power of epistemic logic (type 1) in the adaptation phase of CBR with the performance of case-based planning (type 2). This method is proved to be more efficient then using planning algorithms alone. Planning algorithms are computationally expensive. CBR, using a nearest neighbor algorithm (KNN) is used to make the process faster. A STRIPS planner creates plans for the case-base of a robot that delivers parts in a factory. The manager defines the problem, KNN extracts a plan and a logic sub-system adapts it according to belief revision theorems to resolve the plan inconsistencies.

1. Case-Based Reasoning

CBR is a methodology that solves new problems by remembering solutions to past problems [1]. There are many algorithms used during the retrieval stage of CBR, including: Nearest Neighbor, locally weighted regression and inductive algorithms.

In planning domains the use of CBR is called case-based planning (CBP). In this project problems are plans. A target case is a task assigned by the manager. Plan adaptation is done by a logic sub-system. Traditionally CBR has been conceptualized by the CBR cycle involving the processes: Retrieve, Reuse, Revise, and Retain [2]. Fig 1 shows a enhanced CBR cycle. Revision of Adaptation is between Reuse and Retain. Three types of sub-process can be categorized under the adaptation label:

1. Apply the solution to the problem. Check it is solved or not. Stay in the modification loop until problem is solved.
2. If the original retrieved case did solve the problem add a memo to a field about this new problem and retain it.
3. If the original case was modified then create a new case and retain it.

In Fig. 1, the terms Reuse, Adaptation and Case are used in the following manner:
Reuse = apply the best case to the problem
Adapt = modify the best case, create and modify a new case
Case = information about a past solved problem plus its solution
Target case = information about the present unsolved problem

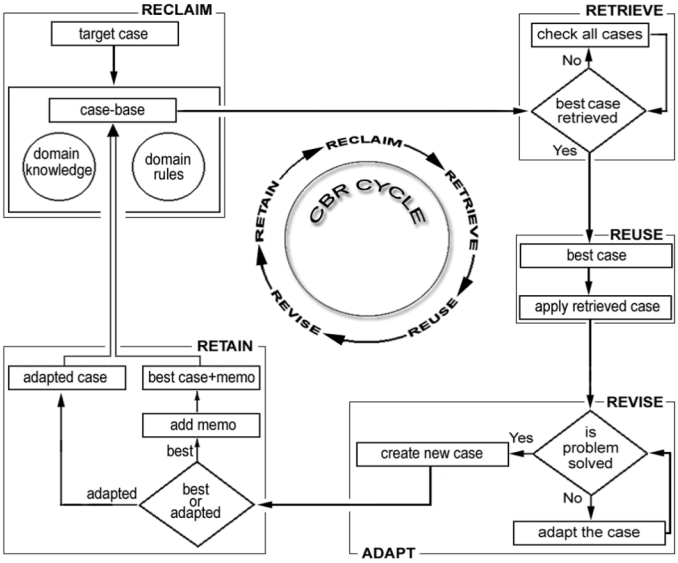

Figure 1 - Enhanced CBR cycle

Figure 1 is inspired by [1] and [2] and substitutes the traditional CBR cycle with a more accurate diagram. In this project, if a retrieved case is inefficient to solve the problem then the adaptation sub-system resolves inconsistencies using epistemic logic to create a new solution (a new case) that will be appended to the case-base.

2. Planning

According to [3] if planning is described in the most abstract level, it is the act of an agent that searches in a plan space instead of situation space to find solutions for problems. Each act is considered by the agent to have some consequences according to its beliefs. If descriptions of [4] are added to the above abstract definition, planning can be seen as finding a sequence of actions to achieve a goal. A goal can be a problem or a conjunction of sub-problems. Fig. 2 is created from the descriptions of paragraph one, page 298, section 8.3 of [4]

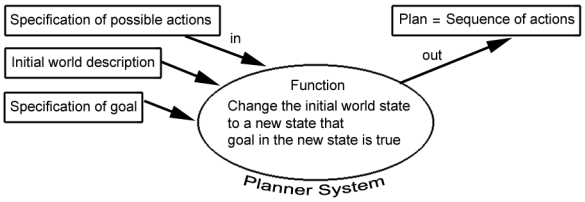

Figure 2 - Automated Planning sub-system

Figure 3 - Autonomous agent

2.1 Elements of a planner

A planner can add actions to the plan whenever it is needed. A planner is different from a general problem solver because it has these elements: representation of actions, representation of states as sets of logical statements, goals and plans. Section 11.2 of [3].
In the planner sub-system of this project, goals are represented by conjunction of several sub-goals. States and actions are represented by sets of logical statements.

2.2 Autonomous planner

An agent can be seen as a self-motivated being that works to achieve its goals.. For designing an autonomous robot different architectures can be used.

Fig. 3 describes the internal elements of the planner sub-system of parts delivery robot of this project according to content of pages 8 to 13 of [4]. **Prior knowledge** is the layout of the factory that the robot already knows, coded in Prolog. **Past experiences** are allowed actions, available parts, effects of actions on position of parts and initial position of parts. They are coded with Prolog in STRIPS format. **Goals** are delivering parts for workstations. The robot is informed of its goals by interaction with the workshop manager as input for planner sub-system. **Observations** with vocabulary of [4] or **Preceptors** with vocabulary of [3] are input from camera, microphone, network, keyboard or file.
In this project, a simulated robot keeps information about its location with internal lists coded in Prolog. **Actions** are **move**, **pick up** and **put down**.

2.3 STRIPS Planner

In this project, STRIPS notation is used for the **planner** and **state definition**. A STRIPS planner has two parts, (domain representation) and (planning algorithm). The domain representation should define three parts, (individual objects in the domain), (relations between objects) and (possible actions).
The world state can be defined with these methods: (situation calculus), (STRIPS) and (event calculus).
In this project the STRIPS model is used in which actions are external to the logic of the program. There are two parts in this planner:

- Representation of actions (STRIPS world representation)
- Search strategy for finding plans (STRIPS planner)

Representation of an action has three parts:

- **Preconditions** - preconditions hold É action performed
- **Add List** - relations that become true after action
- **Delete List** - relations that become false after action

3. Epistemic Logic

3.1 Definition

A set K of logical propositions is a non-absurd belief set **iff**

(1) (K ⊬ ^) or (^ Ï K)

(2) if (K ⊢ B) then (B Î K)

K is a belief set. It does not contain inconsistencies if and only if (it is not possible to make any sentence by logical operations on members of K that contradict other members) and (if B is a logical consequence of the belief set **K** then **B** is a member of K). This definition that insists on non-absurdity is adopted from [5]. Subjects of epistemic logic are "belief" and "knowledge". Episteme "epistɦmɦ" is Greek word for knowledge and Doxa "doxa" for belief. Usually logic of belief is called doxastic and logic of knowledge, epistemic logic. In this paper, epistemic logic is used for logic of *knowledge and belief* as an instance of modal logic [6].

3.2 Revision postulates

(K* 1) K*A is a belief set
(K* 2) A Î K* A
(K* 3) K* A Í K+ A
(K* 4) if (Ø A Ï K) then (K+ A Í K* A)
(K* 5) (K* A = K ^) º (⊢ Ø A)
(K* 6) if [⊢ (A º B)] then (K* A= K* B)
(K* 7) K* (A Ù B) Í (K* A)+ B
(K* 8) if (Ø B Ï K* A) then [(K* A)+ B Í K* (A Ù B)]

Example: if (B Î K) then [(B Î K * A) or (Ø B Î K * A)]
The only reason for excluding a belief in **B**, which is in **K**, from the revision of **K** with respect to **A**, is that it would contradict beliefs in the revision. Where:

- **B** is a belief statement.
- **K** is a set of beliefs.
- **K*A** is belief set K after it is revised by **A**. In other words, Ø **A** is removed from **K** and **A** is added to **K**.

3.3 Rationality criteria

For a human, "actions leading to the achievement of the goal" are considered rational. For a belief set, conjunction of these two rationality criteria defines the belief set:
(1) (^ Ï K) Sentences of the set are consistent or an absurd set cannot be deducted from a belief set.
(2) Any logical consequence of the sentences is member of the set or a belief set is deductively closed = if (K ⊢B) then (B Î K).

3.4 Absurdity

When an agent (human, mechanical or simulated robot) receives new information inconsistent with his present belief set, needs some mechanisms to decide about adding new information to database and remove some of the previous elements. If there is more than one inconsistency what should be done? Reject the new information or change the new (or the old) element to make a new consistent belief set? If a belief set is defined as **K**, then "**K** ^" is an absurd belief set that contains contradictory formulas such as: **q** and Ø **q**.
K ^ º ((**q** Ù Ø **q**) Î K)
In this project non-monotonic extended Epistemic Default Logic (**EDL**) introduced in chapter 4.2 of [6] and system **S5** are used and "^" is excluded from belief set of the robot.

3.5 Commitment function

State of the system is defined as set **S**. Members of **S** are **b1 … bn**. **b** is belief that can be (true), (false), (probable), (necessary) or (possible). These are **epistemic attitudes.**

S = {b1, b2 … bn}

Rules that define how epistemic inputs change the epistemic state are called epistemic commitment functions. In this project one of the commitment functions is:

If robot passes through casting, carrying part number 3, for drilling, then it will not achieve its goal.

Conditional sentences can change the belief set. **Ramsey test** shows the relationship between conditionals and belief change in the following formula.

- **A** is a fact or a logical sentence
- K*A is revision of belief set **K** with respect to **A**

[(A É B) Î K] º (B Î K*A)

(if **A** then **B**) can or can not be consistent with the present content of the system. If it is inconsistent it is called **counter factual** conditional. Page 16 and page 147 of [5]

3.6 Epistemic changes

The system can change from **S1** to **S2** by **Expansion, Revision** or **Contraction.** These are called epistemic changes.

If **K** is a belief set and **A** is a logical sentence, then **A** and **K** can have three types of relations:

- **A Î K** A is accepted
- Ø**A Î K** A is rejected
- **(A Ï K)** and **(Ø A Ï K)** A is undetermined

Definitions of **Expansion, Revision** and **Contraction** are summarized in Table 1.

State of the system / Belief revision	Start	Process	End
Expansion	$(A \notin K) \wedge (\neg A \notin K)$	Add A	$(A \in K)$
	$(A \notin K) \wedge (\neg A \notin K)$	Add $\neg A$	$(\neg A \in K)$
Revision	$(A \in K)$	Add $\neg A$ and remove A	$(\neg A \in K)$
	$(\neg A \in K)$	Add A and remove $\neg A$	$(A \in K)$
Contraction	$(A \in K)$	Remove A	$(A \notin K) \wedge (\neg A \notin K)$
	$(\neg A \in K)$	Remove $\neg A$	$(A \notin K) \wedge (\neg A \notin K)$

Table 1 - Epistemic changes min conditions

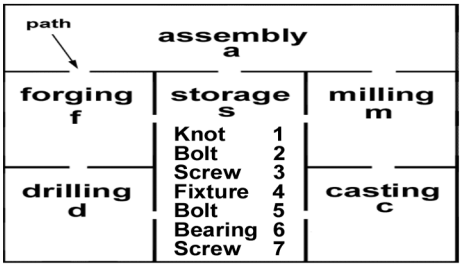

Figure 4 - Workshop layout

4. System

4.1 Definitions

Here keywords are defined according to the project.

Reuse = apply the best plan. Check for presence of inconsistency.

Adapt = If there is no inconsistency then report the plan as a solution. If there is inconsistency then modify the plan string of the case by shuffling the actions using the epistemic revision rule until there is no inconsistency. Make a new case and append it to the case-base.

Case = a past task schedule, plus a plan.(a,f,d,c,m,s from Table 1)

f,3,4,5,6,2,put(3,a),go(f,a),put(4,f),go(a,f),go(m,a),go(c,m),go(s,c),go(d,s),put(5,d),go(f,d),go(a,f),go(m,a),go(c,m),go(s,c),take(3,s),go(a,s),go(m,a),go(c,m),put(6,c),go(m,c),go(a,m),go(f,a),go(d,f),go(s,d),take(4,s),go(a,s),go(m,a),put(2,m),go(a,m),go(f,a),go(d,f),go(s,d),take(2,s),take(6,s),take(5,s),go(a,s),go(f,a).

The first 6 elements of the list are initial location of the robot and required parts for workstations **a,f,d,c,m,s**. The rest of the string is the plan. The plan is read from right to left, for example: go from workstation **f** to **a** then from **a** to **s** then take part number 5 from location **s**…

Target case = present task schedule. **(f, 3, 4 , 5 , 6, 2)** the robot starts from workstation **f** then collects the parts 3, 4, 5, 6 and 2 and deliver them for workstations **a, f, d, c, m, s**.

Domain rules = which workstation requires which parts

Domain knowledge = which workstation is the neighbor of which workstation? Where is the initial location of the parts? They are defined in the Prolog code of the planner sub-system. For example: **truth(isAt(2,s), init)** = Part number 2 is initially stored in the location **s** that is the storage room.

4.2 Problem

A robot supplies parts for workers in a factory and a manager assigns tasks to the robot according to the parts needed by workers. Each worker may require several parts and some parts may be required by more than one worker. If the robot moves near workers needing a part, they might take a part from the robot even though it is carrying it for another worker. In this case an inconsistency is created and the robot cannot fulfill its goal. The robot creates plans before supplying parts. Plans are long lists of actions (e.g., pick up, move, and put down) and

are stored as cases in a case-base. The robot should predict potential inconsistencies in advance and resolve them with epistemic logic formulas, then adapt its plan to achieve its goal. The robot uses a Regression Planner written in Prolog to create plans.

To formalize the problem, q **(i,** b**)**, **T (**W**)**, Y and j are defined.
q **(i,** b**) =** Robot carries part **i** for workstation b**.** This data is collected from the target case.
T (W**)** = Robot travels through workstations W before workstation b. Workstation W requires the part that robot is carrying but robot is not carrying it for W, she is carrying it for b. This data is collected from the retrieved plan.
Y Definitions are conditions that cause the problem. They are constructed from epistemic input (table 2).
Parts 1, 3, 4, 6 and 7 are demanded by more than one workstation therefore, there is a Y assigned for each of them. There are no Y for parts 2 and 5 because these parts are required only by one workstation and robot can freely choose any path and no other workstation takes its parts.

Y1 = **{[**q **(1, a)** Ù **T (m)]** Ú **[**q **(1, m)** Ù **T (a)]}**
Y3 = **{[**q **(3, d)** Ù **T (c)]** Ú **[**q **(3, c)** Ù **T (d)]}**
Y4 = **{[**q **(4, m)** Ù **T (c)]** Ú **[**q **(4, c)** Ù **T (m)]}**
Y6 = **{[**q **(6, a)** Ù **T (d)]** Ú **[**q **(6, a)** Ù **T (f)]** Ú
[q **(6, d)** Ù **T (a)]** Ú **[**q **(6, d)** Ù **T (f)]** Ú
[q **(6, f)** Ù **T (a)]** Ú **[**q **(6, f)** Ù **T (d)] }**
Y7 = **{[**q **(7, a)** Ù **T (m)]** Ú **[**q **(7, a)** Ù **T (f)]** Ú
[q **(7, m)** Ù **T (a)]** Ú **[**q **(7, m)** Ù **T (f)]** Ú
[q **(7, f)** Ù **T (a)]** Ú **[**q **(7, f)** Ù **T (m)] }**

part / workstation	knot - 1	bolt - 2	screw - 3	fixture - 4	bolt - 5	bearing - 6	screw - 7
assembly - a	Y					Y	Y
forging - f		Y				Y	Y
drilling - d			Y			Y	
casting - c			Y	Y			
milling - m	Y			Y	Y		Y

Table 2 - Epistemic input

j defines the problem as Y1 Ú Y3 Ú Y4 Ú Y6 Ú Y7 ⊨ j If any of the Y conditions exist in the retrieved plan then truth of j is satisfied and the plan will be sent to the logic sub-system to resolve its inconsistencies.
If the robot is carrying screw 7 for **f** and passes through **m**, because screw 7 is in the required list of **m** as well, then the worker at **m** will take it and the robot will arrive at **f** empty handed and therefore the plan fails. At the last step the robot believes that it has achieved all of its goals, but does not know that **f** did not receive its part. Resolving such inconsistencies is the role of the epistemic logic sub-system.

Robot believes that:**p** (**f** did receive the part.)
But he should know that:Ø **p** (**f** did not receive the part.)

This epistemic input is given to the robot in the adaptation phase of the CBR sub-system (i.e., the epistemic logic sub-system), in a rule like this: if you want to supply such a part to such a workstation from such a path then the mission will not be successful. The robot now has two choices:
(1) First supply the part for the workstation that is likely to take the parts of others.
(2) Change the sequence of the plan. If robot is going to carry a part for workstation **B** and this part is in the required list of workstation **A**, then robot avoids traveling near workstation **A.**

4.3 Methodology

F. Cuppens in [8] suggests use of epistemic and deontic logics to formalize and solve computer security problems. Approach of this project is somehow similar but not as complex. Here Revision, a sub-set of epistemic logic is used. System has three main sub-systems. A planner sub-system to provide the plans (cases), a CBR and an epistemic logic sub-system that acts in the adaptation phase of the CBR cycle, its output being consistent action plans for the robot.

4.4 Design

This project has three sub-systems:
(1) - CBR sub-system (KNN)
(2) - Planner sub-system (in/out, world descriptions, planning)
(3) - Epistemic logic sub-system (CBR adaptation)
In the planner sub-system STRIPS is used, while a KNN algorithm is used in the CBR sub-system. The Epistemic logic sub-system uses if-then-else rules. The case-base that is collection of plans created by the planner sub-system and acts as input for CBR sub-system.

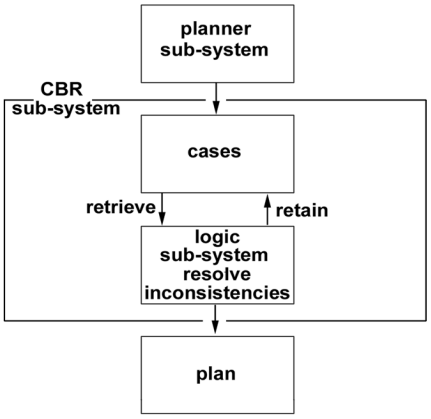

Figure 5 - Methodology diagram

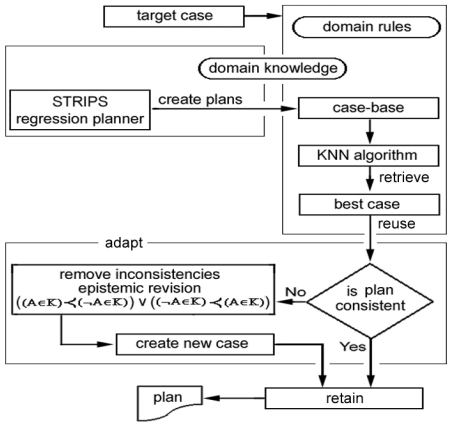

Figure 6 - System diagram

Details of the retain phase of figure 6 is in figure 1.

4.5 CBR sub-system

CBR sub-system uses a KNN algorithm. KNN can be described by three steps.

1. Compare the attributes of the target case with all cases and calculate similarity index of each case.
2. Sort the index array.
3. Choose the case that is most similar to the target case.

$$similarity(T,C) = \sum_{i=1}^{n} F(T_i, C_i) \times w_i$$

- **T** = the target case
- **C** = the source case
- **n** = the number of attributes in each case
- **i** = an individual attribute from 1 to n
- **f** = a similarity function for attribute "i" in cases T and C
- **w** = the importance weighting of attribute i

For calculating $f(T_i, C_i)$ it is possible to use either of the following formulas

$similarityIndex = \sqrt{(T_i)^2 + (C_i)^2}$ or $|T_i - C_i|$.

CBR sub-system requires weight matrices to be used in the KNN formula as part of the domain knowledge. In this project parts are either required or not therefore, is possible to use rules such as

If (required = available) then w=1 else w= 0

4.6 Planner sub-system

A STRIPS planner algorithm used in this project considers a conjunction of goals as one goal. To achieve a set of goals the planner makes a plan to achieve one of them and repeats until it covers all goals. Following flow chart shows the process.

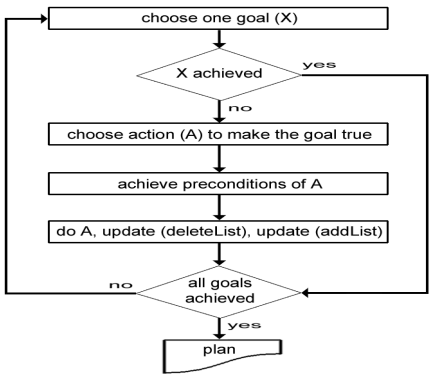

Figure 7 - STRIPS planner

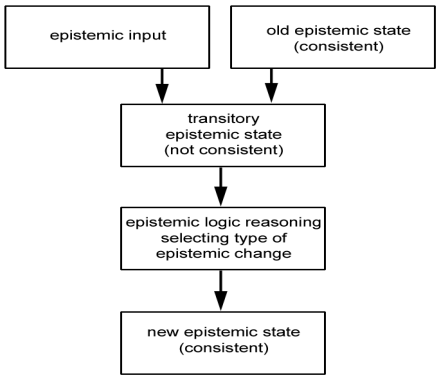

Figure 8 - Epistemic input

4.7 Epistemic logic sub-system

In the adaptation phase of the CBR cycle the retrieved plan is examined for consistency. This is done by examining elements of the belief set of the robot by asking the questions, (is **An,** true? **n** Î {1, 5}) If belief **A** is not true then Ø **A** is added to the robot's belief set and **A** is removed. This is the situation that actions of the plan will be shuffled until **A** is true again. The adapted plan (plus its problem definition part) will be appended to the case-base.

Epistemic revision

Revision as one of the epistemic changes can be formulated as:

$$\left((A \in K) \prec (\neg A \in K)\right) \vee \left((\neg A \in K) \prec (A \in K)\right)$$

By using revision the robot changes its belief in a fact from **A** to Ø **A** or from Ø **A** to **A**. In this formula operator "before" $\prec$ is used to show the state of the system before and after evaluation. The robot believes in **A** before evaluation and in Ø **A** after evaluation or vice versa.

Belief set of the robot = in the above formula **K** is an indicator of the belief set of the robot.
K = [A1, A2 … An] n<6.
An = belief of the robot that required part for workstation **n** is delivered.

Epistemic inputs

The robot collects information that might change its beliefs. New information that can change the **epistemic state** of the system is called epistemic input. The robot receives its data from a file that it reads at initialization. Required parts are **epistemic inputs**. For instance forging workstation requires bolt no. 2 or bearing no. 6.
Old epistemic state and epistemic input create a new inconsistent epistemic state. Rules of the logic sub-system process the new information and decide about an **epistemic change** to bring the system to equilibrium again. Fig. 8 enhances a diagram of section 1.4 of [5].
The epistemic input table (shown in Table 2) is the source of all inconsistencies and is for deciding which part should be delivered to which worker.

5. Conclusion

In an automobile factory for each model of car there are an average of 40,000 parts, hundreds of workstations and many thousands of paths that a supplying robot can choose. If a robot intends to solve its planning problem using algorithms such as regression or partial order planner, it needs to create thousands of plans and each creation takes thousands of hours using present CPUs. In this project plans were created by a regression planner. Then they were used as cases for a CBR sub-system and inconsistencies of the real world and beliefs of the robot were solved by epistemic logic axioms. Therefore by creating a small fraction of the total number of possible plans and combining epistemic logic rules, default reasoning and case-based reasoning, this computationally expensive problem was solved.

Acknowledgements

We would like to thank reviewers. Their comments improved this paper.

References

1. I. Watson - Applying Case-Based Reasoning - Morgan Kaufmann Publishers - 1997
2. Aamodt, Plaza - Foundational issues - AI communications - Vol. 7:1 - 1994 - pages 39 to 59
3. Russell, Norvig - Artificial Intelligence, a Modern Approach - 1995
4. Poole, Macworth, Goebel - Computational Intelligence - 1998 - chapter 8 - Oxford University Press
5. P. Gardenfors - Knowledge in flux, modeling the dynamics of epistemic states - 1988 - The MIT Press
6. J. J. Ch. Meyer, W. van der Hoek - Epistemic logic for AI and computer science - 1995 - Cambridge University Press
7. R. Girle - Possible worlds - 2003 - Acumen publisher
8. F. Cuppens - An Epistemic and Deontic Logic for Reasoning about Computer Security - ESORICS 1990