A COMPARISON OF EVENT CALCULUS AND ACTION LANGUAGES

Onur Aydın
onur.aydin@ceng.metu.edu.tr

Supervisor: Nihan Kesim Çiçekli
nihan@ceng.metu.edu.tr

Co-Supervisor: Ferda Nur Alpaslan
alpaslan@ceng.metu.edu.tr

Department of Computer Engineering
Middle East Technical University
06531 Ankara, Turkey
December, 2003

Abstract

It has been tried to overcome the limitations of classical logic with using causal links between logical facts. Real world is too complex to model or define exactly all the relations so some intelligent inference mechanisms are needed to ignore the unimportant dynamics while watching carefully the essential action and effects. These desires are overcome with the introduction of nonmonotonic formalisms. In this paper two known nonmonotonic formalisms Action Languages and Event Calculus will be reviewed and compared according to their inner representations and expresiveness.

Table of Contents

1. Introduction

Artificial Intelligence and Logic science fields are cooperating tightly to overwhelm some open problems of the literature. Especially the artificial intelligence field of computer science is benefiting from the logic concepts on the solving of some of its sub fields like planning, knowledge representation, deduction for expert systems, and classification of concrete relational information.

Main reason for this interaction is the power of the deduction mechanisms of the Logic. They are found to be useful on solving logical relations under some constraints like completeness of the domain of problem or closed world assumptions. If a problem is clearly expressed as a series of logical expressions obeying the constraints that constitutes the limits of computation then logical inference systems are clearly an option.

Modern logical programming is studied under two sub fields. Elements of the first group shares monotonic approaches. A logical system is monotonic if the truth of a proposition does not change when new information is added to the system. Rules, predicates and default axioms define the operation set. Also with some inference algorithm deduction engine is supplied. Some constrains and assumptions are necessary like uniqueness of the name of axioms and predicates or the closure of the problem domain.

Monotonic logic has been quite successful on various problems. Logicians have contended that reasoning, as performed by humans, is also amenable to analysis using classical logic. However, workers in the field of artificial have shown that classical logic is not sufficiently robust to adequately reason as humans do. Humans do not always reason as would a classical reasoning system. They leap to conclusions based on commonsense reasoning. By commonsense reasoning humans generally refer to such statements:

- "it is my experience that *this* must be the case"
- "there is no good reason not to believe *this*" [1]
- "following the listed assertions *this* should be necessarily the case"
- "it might be possible that the *this* is the case"

The subject of the second sub field of logical programming, called nonmonotonic reasoning is based on a model that shares the commonsense view of world. It is mainly based on classical logic. McCarthy [2] was perhaps the one who first brings this issue into world of computer science and programming. He introduced the idea of necessity of an automated commonsense reasoning.

2. Nonmonotonic Reasoning

Since our primary concern is nonmonotonic reasoning, it will be discussed in a different section here. This papers area of interest that is the Action Languages and Event Calculus are instances of such reasoning. Both of them try to solve the uncompleted problem domains by using their commonsense reasoning axioms summed up with special operators and of course with classical logic definitions. In other words, they share common grounds of classical logic but also they introduced some extension axioms to it in order to deduce.

The motivation and necessity behind these nonmonotonic reasoning systems is well defined in studies of McCarthy and Hayes [3]. In their study, limits of a Turing machine on the edge of implementing a philosophical reasoning system are discussed. According to the authors, an entity is intelligent if it has an adequate model of the world (including the intellectual world of mathematics, understanding of its own goals and other mental processes), if it is clever enough to answer a wide variety of questions on the basis of this model, if it can get additional information from the external world when required, and can perform such tasks in the external world as its goals demand and its physical abilities permit [3].

Building such an autonomous agent with classical logics is just impossible because there is no way of expressing every situation or transition that may occur in real life without making overgeneralizations. Real world is just too complex to express. Despite these facts assume that it is expressed in a theoretical aspect then yet there is another problem: it would be very hard to find answers from such a huge logical database of relations.

Another problem with such a huge amount of rules (also called frame axioms) is to keep it consistent because we know that, classical logics would work only if the set of rules that yields to the decisions are consistent. For that reason some kind of tolerance is necessary when inconsistent observations or even contradictions are met. One way of overwhelming such a situation is using probabilistic or *Action-Value* approach.

In this approach the causalities are not just memorized but kept in an experience pool with certain probability. Such a system can be best visualized with help of nondeterministic automata. For simplicity, assume the agent is put on a closed environment with an initial automaton. This automation contains some of the states of a full set that contains all possible valuations of variables that are fully observable by the agent in the environment including the agent itself. According to the goal given to the agent, it would act as it is desired by using the automaton and taking the transitions in a somewhat nondeterministic way using the

probabilities assigned to the transitions (choosing the action that yields to the goal considering the probably values).

Since the environment is continuously observed by the agent some states and actions can be experienced by the agent that are not initially supplied to it because agent might not be the only interacting element in the closed environment or just because the causes of the subset of all actions given initially to the agent are not completely supplied to the agent. In such cases agent should add the new states and the actions that links the current state and the newly added states with a probability of 1 for instance (probabilities can be updated as the action is met again from the same state). Newly added states would have no transitions out the state so most common state which is the most closest one according to the valuations of variables (or fluents as it will be discussed later) in the automaton might be chosen and can be act as if it is the current state. A possible improvement when a new state is met can be undoing the latest action when possible with a certain probability.

In this example of nonmonotonic system, the transitions could be the logical rules and the states can be a vector of the asserted facts. With addition of intuitive probabilistic action choosing strategy rules, a forward (linear) resolution system would behave in a commonsense deduction mechanism. What makes this system nonmonotonic is the ability to add rules to its world representation.

Best known nonmonotonic logics are circumscription [4, 5, 6], default logic [7], non monotonic modal logics [8, 9, 10], event calculus [11] and action languages [12]. In this paper two of them, event calculus and action languages will be presented and compared according to their similarities and differences.

3. Frame Problem

The original frame problem tends to appear in nonmonotonic systems that adhere incomplete models of a changing world. In such systems there are axioms about changes conditional on prior occurrences. An example is that switching the window button of car opens the window of it and that turning on the lamp of the car changes the illumination of it. Unfortunately, since inferences are to be made only by deduction, axioms are needed for expressing non changes that is switching the button does not change the illumination and that turning on the lamp does not change the state of window. Without such *frame axioms* a system is unable clearly deduce that any states persist. The resulting problem is to come up with an answer without introducing huge numbers of frame axioms potentially relating each changeable variable to each unchangeable one.

A common response is to handle unchangeable variables implicitly by allowing the system to assume by default that a state persists, unless there is an axiom specifying that it is changed by an occurrence, given surrounding conditions. Since such assumptions are not deducible from the axioms of change (even given surrounding conditions), and since the given known conclusions are not cumulative as evidence is added, the frame problem helps motivate the development of nonmonotonic logics intended to minimize the assumptions that must be retracted given further evidence as told before. This is related to discussions of defeasibility and ceteris paribus reasoning in epistemology and philosophy of science [13].

A well known challenge for nonmonotonic reasoning is to determine which assumptions to retract when necessary, as in the *Yale Shooting Problem* [14]. Let a system (or program) assume by default:

1. There is a loaded gun to shoot and loaded guns remains loaded.

2. There is an alive person as target and alive person remains alive.

3. Target is alive.

4. Gun is loaded

5. After a short sneeze, gun is fired to the target.

In the light of these facts there can be two results. If the first fact is in question through the delay of sneezing then second fact is violated by the target. If second fact is in question through the delay then first fact is violated by the target. Intuitively human can conclude that it is more reasonable to enforce the second fact. Some scientists believe that first fact is still holding because there was an indefinite delay before the shooting action [15]. While others say that there is no sign that that to believe the second fact is violated while shooting means something to us as second fact is violated [16, 17, 18]. Here the problem is not being fully depicted the action world; some additional facts can be inserted without sticking to the classical logical rules and facts.

4. Event Calculus

As stated before, Event Calculus is a nonmonotonic formalism to represent action and effect changes to the world of interest. It avoids the representational frame problem, a special kind of frame problem that deals proliferation of the frame axioms (axioms that defines the changes) by using its own axioms and formalism.

Computer programs that take decisions about how they or other agents should act, uses some form of representation of the effects of actions, both their own and those of other agents. In order to maintain a solid theoretical base for the design of these programs there is the need for logical formalisms for representing action, which, although they may or may not be realized directly in computer programs.

4.1 Event Calculus Basics

4.1.1 What the Event Calculus Does

The event calculus is a logical mechanism that derives what's true when given what happens when and what actions do. The story of events constitutes the "what happens when" part, and the description of the effects of actions constitutes the "what actions do" part.

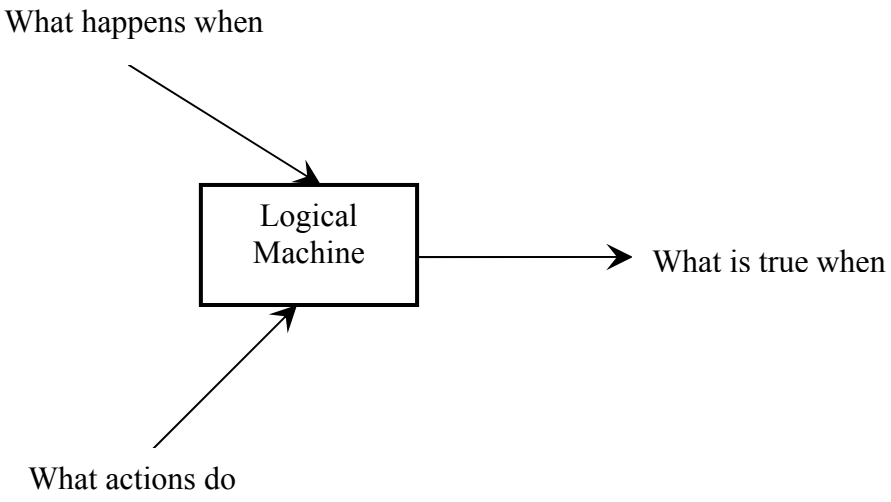

Figure 1: How the Event Calculus Functions

4.1.2 The Semantic of the Event Calculus

The event calculus adopts an approach of introducing suitable predicates and functions for representing the kind of action-related information built on first-order predicate calculus.

The first issue in designing the first-order language is determining the ontology, that is to say the types of things over which quantification is allowed. The underlying ontology of the event calculus consists of actions or events (or rather action or event types), fluents and time points. A fluent is anything whose value is subject to change over time. The temperature in the room, whose numerical value is subject to variation, is an example of type fluent.

Another issue in designing the first-order language is determining the set of basic predicates. Table 1 presents the language elements of the simple event calculus.

Formula	Meaning
Initiates(α, β, τ)	Fluent β starts to hold after action α at time τ
Terminates(α, β, τ)	Fluent β ceases to hold after action α at time τ
$\text{Initially}_P(\beta)$	Fluent β holds from time 0
$\tau 1 < \tau 2$	Time point $\tau 1$ is before $\tau 2$
Happens(α, τ)	Action α occurs at time τ
HoldsAt(β, τ)	Fluent β holds at time τ
Clipped($\tau 1$, β, $\tau 2$)	Fluent β is terminated between times $\tau 1$ and $\tau 2$

Table 1: Some Event Calculus Predicates

4.1.3 The Axioms of the Simple Event Calculus

The following is a suitable collection of axioms relating the various predicates together:

$$HoldsAt(f,t) \leftarrow \neg\, Initially_P(f) \wedge \neg\, Clipped(0,f,t) \qquad \text{(SC1)}$$

A fluent holds at a time t if it held at time 0, and hasn't been terminated between 0 and t:

$$HoldsAt(f,t2) \leftarrow$$

$$Happens(a,t1) \wedge Initiates(a,f,t1) \wedge t1 < t2 \wedge \neg\, Clipped(t1,f,t2) \qquad \text{(SC2)}$$

A fluent holds at time t if it was initiated at some time before t and hasn't been terminated between then and t:

$$Clipped(t1,f,t2) \leftrightarrow$$

$$\exists a,t\ [Happens(a,t) \neg t1 < t < t2 \neg Terminates(a,f,t)] \qquad \text{(SC3)}$$

According to these axioms, a fluent does not hold at the time of the event that initiates it but does hold at the time of the event that terminates it. Put differently, the intervals over which fluents hold are open on the left and closed on the right. Naming the conjuction of these axioms as SC will be usefull for further usage in this text.

4.2 The Frame Problem in the Event Calculus

The frame problem comes up with the question trying to find a way to use logic to represent the effects of actions, without having to explicitly represent all their non-effects. Yale shooting scenario will be considered to see how this problem arises in the context of the event calculus.

4.2.1 The Yale Shooting Scenario

In Yale shooting domain there are three types of action, namely a Load action, a Sneeze action, and a Shoot action, and three fluents, namely Loaded, Alive and Dead. The effect of a Load action is to make Loaded hold. A Shoot action makes Dead hold and Alive not hold so long as Loaded holds at the time. A Sneeze action has no effects. Following is formalized representation of these effects.

$$Initiates(Load,Loaded,t) \qquad \text{(Y1.1)}$$

$$Initiates(Shoot,Dead,t) \leftarrow HoldsAt(Loaded,t) \qquad \text{(Y1.2)}$$

$$Terminates(Shoot,Alive,t) \leftarrow HoldsAt(Loaded,t) \qquad \text{(Y1.3)}$$

The Yale shooting scenario comprises a series of actions of a Load action, a Sneeze action and a Shoot action happening respectively. Following is the formalized representation of this series of actions.

$$Initially_P(Alive) \qquad \text{(Y2.1)}$$

$$Happens(Load,T1) \qquad \text{(Y2.2)}$$

$$Happens(Sneeze,T2) \qquad \text{(Y2.3)}$$

$$Happens(Shoot,T3) \qquad \text{(Y2.4)}$$

$T1 < T2$ (Y2.5)

$T2 < T3$ (Y2.6)

$T3 < T4$ (Y2.7)

Now let S be the conjunction of (Y1.1) to (Y1.3), and let D be the conjunction of (Y2.1) to (Y2.7). Intentionally we should have,

$\sum \wedge \Delta \wedge SC \models HoldsAt(Dead,T4)$.

Unfortunately this resultant is not valid. This is due to the lack of the explicit descriptions of the non-effects of actions. Specifically, there does not exist the statement of the fact that the Sneeze action doesn't unload the gun. So there are, for example, models of SC ^ ∑ ^ Δ in which Terminates(Sneeze,Loaded,T2) is true, Holds(Alive,T4) is true, and HoldsAt(Dead,T4) is false.

In fact, we should consider the possibilities that we must rule out before we have a theory from which the intended conclusions follow. We must also describe the non-occurrence of actions. And, more trivially, we must include formulae that rule out the possibility that, say, the Sneeze action and the Shoot action are identical. This issue is easily dealt with. When describing the effects of actions, we always need to include a set of *Uniqueness of Names* axioms for fluents and actions. In our case, we have the following formulae, which use a notation taken from [25]. These entail that Load ≠ Sneeze, Loaded ≠ Alive, and so on.

UNA[Load, Sneeze, Shoot] (Y3.1)

UNA[Loaded, Alive, Dead] (Y3.2)

4.2.2 Using Predicate Completion

In order to express the non-effects of actions and the non-occurrence of events the completions of the Initiates, Terminates and Happens predicates can be provided. Formulae (Y1.1) and (Y1.2) are replaced by the following.

$Initiates(a, f, t) \leftrightarrow$ (Y4.1)

$[a = Load \wedge f = Loaded] \wedge [a = Shoot \wedge f = Dead \wedge HoldsAt(Loaded, t)]$

$Terminates(a, f, t) \leftrightarrow a = Shoot \wedge f = Dead \wedge HoldsAt(Loaded, t)$ (Y4.2)

We retain formulae (Y2.1) and (Y2.5) to (Y2.7). (Y2.2) to (Y2.4) are replaced by the completion of the Happens predicate.

$$Initially_P(Alive) \quad \text{(Y5.1)}$$

$$Happens(a, t) \leftrightarrow$$

$$[a = Load \wedge t = T1] \wedge [a = Sneeze \wedge t = T2] \wedge [a = Shoot \wedge t = T3] \quad \text{(Y5.2)}$$

$$T1 < T2 \quad \text{(Y5.3)}$$

$$T2 < T3 \quad \text{(Y5.4)}$$

$$T3 < T4 \quad \text{(Y5.5)}$$

Now let W be the conjunction of (Y3.1) and (Y3.2), let S be the conjunction of (Y4.1) and (Y4.2), and let D be the conjunction of (Y5.1) to (Y5.5). Now, as desired, we have,

$$\sum \wedge \Delta \wedge SC \wedge \Omega \models HoldsAt(Dead,T4).$$

Here the core of a satisfactory solution to the frame problem is introduced. Generally, though, especially in non-trivial domains, it's desirable to have some logical mechanism that automatically constructs the completions of the Initiates, Terminates and Happens predicates from individual clauses like those in (Y1.1) to (Y1.3) and (Y2.2) to (Y2.4). As well as being notationally more convenient, this allows the construction of a more modular theory. It also makes our theories more elaboration tolerant [26], that is to say new actions, new fluents, new effects of actions, and new event occurrences can easily be adapted by an extant theory [11].

5. Action Languages:

It is another methodology similar to the Event Calculus which deals with the problem domain with its own causal links. Idea of this language is driven by Texas Action Group, a group of researchers interested in the study of formal and automated reasoning about the effects of actions using action languages, logic programming under the answer set semantics, and related ideas. Ideas are first presented in the studies of V. Lifschiltz and M. Gelfond [21, 22] who lead the Texas Action Group till the moment of the preparation of this paper.

This formalism is an extension to the situation calculus like Event Calculus to overwhelm the deficiencies of it as presented by the owners of the study [23]. The layout of the studies of Action Language and its other implications can be collected under several sub group of task.

5.1 Action Languages, Translations and Query Languages

In the first group a world representation language is defined. In this language physical variation and changes are described by the elements of the language. Its syntax is close to logical programming language. When a program defined in this language, it could be said that a general representation of the world is obtained. Throughout the evolution of the Action Languages several versions are declared by the pioneers of the study and a review of these studies can be obtained from [12].

Initial version was language *A*. It is well equipped to express causal links therefore useful for action domains. In this paper, this language will be provided because it is suitable for comparison with Event Calculus. Later on, language *B* is declared, which is an extension of *A* equipped with additional capability of expressing indirect effects of actions. Final language called *C*, is somewhat more expressive than *B* but not a direct superset of it. It has an ability to express nondeterministic actions and concurrent actions. Also inertia which is a built-in axiom for *B* is not enforced for all fluents and can be overridden if it is desired.

After being defined the Action Language, there has to be a translation scheme from this language to the language of ordinary and extended first order logic (clearly speaking logic programming world). Via this translation scheme, logic programming tools or other formalisms are made available for action languages as inference engines. This translation scheme is going to be depicted in this paper also.

Finally another task active under these studies are querying languages. There are two querying language defined so far [12]. First one is called *P*, and it is used for describing temporal projection of a specific state to obtain information about effects of sequential execution of actions in this state. Other one is called *Q* and it is for the actions that have been already executed. These two languages are not going to be mentioned here also because of not having a direct correspondence from the Event Calculus formalism.

5.2 Action Language A

In fact, this language is created under the light of inspirations from STRIPS formalism. STRIPS is one of the early attempt to solve Frame Problem and it is first defined in [24]. General notion of the strips is the inertia it uses. In this language a static axiom guaranties that a fluent does not change its value unless an action has taken place that could change the value of it.

In language *A*, effects of an operator of language STRIPS, which is a causal link that connects causing fluents to the effected facts, is redefined so that the results of an operator can be conditional [25].

5.2.1 Definitions:

Two kinds of propositions exist in this language. A *Value Proposition* specifies the value of a fluent in a particular situation. It can be in initial situation or after some actions it can have another value from the set of available values. An *Effect Proposition* defines the effects of an executed action on a fluent.

According to this language, every problem domain contains two sets. First set is the fluent names and the second set is the action names set. A *Fluent Expression* is a rule like sentence in the form:

$$F \textbf{ after } A_1; A_2; ...; A_n \quad \text{(A1)}$$

Where F is fluent name and it can be preceded by a ¬ (not symbol in meaning with classical logic). A_i's are actions. If n is 0 then it means:

$$\textbf{initially } F$$

Which means F is initially holding (it is true). Effect proposition is in the form of:

$$A \textbf{ causes } F \textbf{ if } P_1; P_2; ...; P_n \quad \text{(A2)}$$

Where A is action name and F and P_i's are fluent names and if n is 0 then it can be expressed as:

$$A \textbf{ causes } F$$

5.2.2 Yale Shooting Problem:

Let's restate the Yale shooting problem with the language *A*.

initially *Alive*

initially *Loaded*

Load ***causes*** *Loaded*

Shoot ***causes*** ¬*Alive* ***if*** *Loaded*

Shoot ***causes*** ¬*Loaded*

5.2.3 Semantics of Language *A*:

To describe the semantics of *A*, we will define what the *Models* of a domain description are, and when a value proposition is *Entailed* by a domain description. A *State* is a set of fluent names. Given a fluent name *F* and a state *S*, we say that F holds in *S* if $F \in S$; ¬*F* holds in *S* if $F \notin S$. A *Transition Function*, called ϕ, is a mapping from the set of pairs (A; σ) into the set of states, where *A* is an action name and σ is a state,. A structure is a pair (σ_0; ϕ), where σ_0 is a state (the initial state of the structure), and ϕ is a *Transition Function*.

For any structure *M* and any action names A_1; ...; A_n, by $M^{A_1;...;A_n}$ it is denoted that:

$$\phi(A_n; \phi(A_{n-1}; \phi(...; \phi(A_1; \sigma_0) ...));$$

Where ϕ is the transition function of *M*, and S_0 is the initial state of M. We say that a value proposition (A1) is true in a structure *M* if *F* holds in the $M^{A_1;...;A_n}$ and false otherwise.

A structure (σ_0; ϕ) is a model of a domain *D* if every *Value Proposition* from *D* is true in (σ_0; ϕ), and, for every action name *A*, every fluent name *F*, and every state σ, the following conditions are satisfied:

1. if *D* includes an effect proposition describing the effect of *A* on *F* whose preconditions hold in σ, then $F \in \phi(A; \sigma)$;

2. if *D* includes an effect proposition describing the effect of *A* on ¬*F* whose preconditions hold in σ, then $F \notin \phi(A; \sigma)$;

3. if *D* does not include such effect proposition, then $F \in \phi(A; \sigma)$, if $F \in \sigma$.

It is intuitive that there can be at most one such transition function satisfying the conditions that are listed here. Therefore different models of the some problem domain can only differ by the initial states.

A domain description is *Consistent* if it has a model and *Complete* if it has exactly one model. The Yale Shooting domain is complete; its only model is defined by the equations:

$\sigma 0 = \{Alive\};$

$\phi(Load; \sigma) = \sigma \cup \{Loaded\};$

$\phi(Shoot, \sigma) = \sigma \setminus \{Loaded; Alive\};$ ***if*** $Loaded \in \sigma,$

σ ***otherwise****;*

$\phi(Wait; \sigma) = \sigma;$

A value proposition is entailed by a domain description *D* if it is true in every model of *D*. For example, Yale Shooting Problem Domain entails:

$\neg Alive$ ***after*** *Load; Wait; Shoot;*

5.2.4 Translation from *A* to Logic:

Having defined the language, lets define the translation π from *A* into the language of extended logic programs in where the *Negation* and *Not* of a literal are treated differently. *Not* is accepted as follows: when it is present in front of a literal then it states that the literal is evaluated to be false and it is shown by symbol ¬. Negation of a literal on the other hand, defined to hold when there is no evidence against falsification of it and it is shown with *not* in front of literals.

About two different effect propositions we say that they are *Similar* if they differ only by their preconditions. The translation method is defined for any domain description *D* that does not contain similar effect propositions. This condition prevents, for instance, combining in the same domain such propositions as

Let *D* be a domain description without similar effect propositions. The corresponding logic program πD uses variables of three sorts:

1. Situation Variables: $s_1, s_2, ...$
2. Fluent variables: $f_1, f_2, ...$
3. Action variables: $a_1, a_2, ...$

Its only *situation* constant is S_0, its fluent constants and action constants are from, respectively, the fluent names and action names of *D*. There are also some predicate and function symbols; the sorts of their arguments and values will be clear from their use in the rules below.

The program πD will consist of the translations of the individual propositions from *D* and the four standard rules:

$$Holds(f; Result(a; s)) \leftarrow Holds(f; s); not\ Noninertial(f; a; s) \quad (A3)$$

$$\neg Holds(f; Result(a; s)) \leftarrow \neg Holds(f; s); not\ Noninertial(f; a; s) \quad (A4)$$

$$Holds(f; s) \leftarrow Holds(f; Result(a; s)); not\ Noninertial(f; a; s) \quad (A5)$$

$$\neg Holds(f; s) \leftarrow Holds(f; Result(a; s)); not\ Noninertial(f; a; s) \quad (A6)$$

These rules are inspired from *Commonsense Law of Inertia*. According to this, value of a fluent after an action is executed stays unaffected if it does not involved with the action. The rule 3 is used when it is known that fluent is holding before the action and rule 4 is used when it is known that fluent is not holding before the action. Similarly rules (A5) and (A6) are for reverse order inference of the fluents. The auxiliary predicate *Noninertial* is an *Abnormality Predicate* [5].

Lets define how π translates the value and effect propositions. First of all, if *F* is a fluent and *t* is a situation then $Holds(\neg F, t)$ means the same thing with $\neg Holds(F, t)$. Also if $A_1, A_2, \ldots, A_n$ are action names then $[A_1, A_2, \ldots, A_n]$ stands for following:

$$Result(A_n; Result(A_{n-1}; \ldots; Result(A1; S_0)); \ldots))$$

Translation of a value proposition (A1) is as follows:

$$Holds(F; [A_1; \ldots; A_n]) \quad (A7)$$

For describing translation of an effect proposition two definitions are necessary. First, $| F | = F$ and $| \neg F | = F$ for any fluent *F*. This operator functions like an absolute value operator in algebra. Second definition is as follows: the symbol ~ when appear before a literal as $\sim Holds(P_i, s)$, literal complementary of $Holds(P_i, s)$ is meant. Based on these definitions effect proposition *is* as follows:

$$Holds(F; Result(A; s)).\ Holds(P_1; s); \ldots; Holds(P_n; s) \quad (A8)$$

It allows us to prove that *F* will hold after *A*, if the preconditions are satisfied. The second rule is:

Noninertial(| F |; A; s) ← not ~Holds(P1; s); …; not ~Holds(Pn; s) *(A8)*

It disables the inertia rules (A5), (A6) in the cases when *F* can be a effected by *A*. Without this rule, the program would be contradictory: We would prove, using a rule of the form (A7), which an unloaded gun becomes loaded after the action Load, and also, using the second of the rules (A4), that it remains unloaded.

Note the use of *not* in (A8). Here it is wanted to disable the inertia rules not only when the preconditions for the change in the value of F are known to hold, but whenever there is no evidence that they do not hold. If, for instance, it is not known whether Loaded currently holds, then it is not wanted to conclude by inertia that the value of Alive will remain the same after Shoot. It would be wrong draw any conclusions about the new value of Alive. If we replaced the body of (A8) by *Holds(P_1; s); …; Holds(P_n; s),* the translation would become unsound.

Besides (A7) and (A8), the translation of (A2) contains, for each i such that $1 \leq i \leq n$, the rules

$$\text{Holds}(P_i; s) \leftarrow \sim\text{Holds}(F; s); \text{Holds}(F; \text{Result}(A; s)) \quad \text{(A9)}$$

and

$$\sim\text{Holds}(Pi; s) \leftarrow \text{Holds}(F; \text{Result}(A; s)),$$
$$\text{Holds}(P_1; s); \ldots; \text{Holds}(P_{i-1}; s); \quad \text{(A10)}$$
$$\text{Holds}(P_{i+1}; s); \ldots; \text{Holds}(P_n; s):$$

The rule (A9) means the following inference: if the value of *F* has changed after performing *A*, then we can conclude that the preconditions were satisfied when *A* was performed. These rules would be unsound in the presence of similar propositions. The rule (A10) allows concluding that a precondition was false from the fact that performing an action did not lead to the result described by an effect axiom, while all other preconditions were true.

Let's take a look at the final representation of *Yale Shooting Problem* domain *D* after translations are applied and πD is obtained. Here the problem domain includes all the axioms 3-8 and the following rules [21]:

¬Holds(Loaded; S_0) *YS1*

Holds(Alive; S_0) *YS2*

Holds(Loaded; Result(Load; s)) *YS3*

Noninertial(Loaded; Load; s) *YS4*

¬Holds(Alive; Result(Shoot; s)) ← Holds(Loaded; s) *YS5*

Noninertial(Alive; Shoot; s) ← not ¬Holds(Loaded; s) *YS6*

Holds(Loaded; s) ← Holds(Alive; s); ¬Holds(Alive; Result(Shoot; s)) *YS7*

Holds(Loaded; s). Holds(Alive; Result(Shoot; s)). *YS8*

¬Holds(Loaded; Result(Shoot; s)). *YS9*

Noninertial(Loaded; Shoot; s). *YS10*

6. Comparison of Formalisms:

Two formalisms that are reviewed here are two examples of the nonmonotonic methodologies. They both extend the classical logic and the default rules in it. They are using the atoms literals and implication operators as well as negation and not operators in their construct. One obvious difference is that Action Languages are not directly represented in logic programming notation but in a similar more readable context. However there is a translation scheme which converts the Action Language into rule database as in the classical logic.

Their one of the first obvious difference is the concept of action and event. In reality, more or less they define the same notion a trigger that switches states of the environment. In spite of this dissimilar naming, their insight to the states is almost exactly the same. In both formalism there is an perception of fluent and the set of the valuations of these fluents define the states of the current circumstances.

They both aim the same thing solving problems that resemble real world domains. In these domains, there are lots of causal occasions and they affect the fluents and therefore the states. Constantly actions or events occur and fluents change their state. But there is an obvious fact that not all of them related with each other and related with the absolute goal of the agent. In fact most of the time specific actions or events are related with specific set of fluents and states so expressing the non-effect relationships are unnecessary and burdensome. Yet it is necessary for classical logic because it does not care if there are more than one model for the problem. However, in nonmonotonic approaches, an intuitive exactly one model is searched if there exists one. Action Languages and Event Calculus are in fact motivated with this same fallacy of the classical logic.

Action Logic and Event Calculus are both introduce some new axioms to the natural deduction mechanism like axiomatic logic for covering the missing parts

of the classical logic. They define a set of static axioms which should be part of every problem domain. In other words, they are default rules for any representation of any problem space. They relate the elements of the new formalisms. These axioms construct the links between fluents states and actions or events.

Another similarity between these methodologies is the usage of commonsense inertia rules in their additional axioms. Even though it is not so clear to find out the commonsense causal links of Event Calculus at first sight, it also enforces commonsense inertia rules using the initiative, progressive and terminative situations of fluents. Indeed, it is not so surprising to observe that because it is the most common technique to reduce the proliferation of rules for expressing for non-effects of actions or events

Event Calculus additionally introduces timing of the happenings because it not only intends to solve and come up with a solution but also the sequence of actions that yielded to that solution. Therefore also the order of the events are tried to be solved in accordance with the solution.

Finally the Action Language described here used the concept of negation, literal completion and constraint of prevention of similar rules to overcome the awesome frame problem while Event Calculus used predicate completion, unique names of axioms techniques to incorporate the solution of the frame problem.

7. Conclusion:

In this study, two formalisms are represented. Their similar and different properties as discussed. These two formalism aims to solve the same problem but using slightly different approaches. This paper is written to express these approaches and discuss their relation to the grounds of the classical logic and nonmonotonic formalisms.

This study will be a guide line for the task that will precede this work. In the thesis study that initiated this work, these two formalisms will be compared according to their expressiveness and efficiency using a real benchmark problem. After solving the problem with these two formalisms results are going to be analyzed and statistical details are going to be declared.

8. References:

[1] J. Minker, An Overview of Nonmonotonic Reasoning and Logic Programming, Journal of Logic Programming, vol. 17, pp 95-126, 1993.

[2] J. McCarthy, Programs with Commonsense, Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory, vol. 1, pp 77-84, 1958.

[3] J. McCarthy and P. J. Hayes, Some Philosophical Problems From the Standpoint of Artificial Intelligence, Machine Intelligence, vol. 4, pp 463-502, 1969.

[4] J. McCarthy, Circumscription - A Form of Nonmonotonic Reasoning, Artificial Intelligence, vol. 13, pp 27-39, pp 171-172, 1980.

[5] J. McCarthy, Applications of Circumscription to Formalizing Common Sense Knowledge, Artificial Intelligence, vol. 26, pp 89-116, 1986.

[6] V. Lifschitz, Closed World Databases and Circumscription, Artificial Intelligence, vol. 27, pp 229-235, 1985.

[7] R. Reiter, A Logic for Default Reasoning, Artificial Intelligence, vol. 13, pp 81-132, 1980.

[8] D. McDermott and J. Doyle, Nonmonotonic Logic I, Artificial Intelligence, vol. 13, pp 41-72, 1980.

[9] D. McDermott, Nonmonotonic Logic II, Artificial Intelligence, vol. 13, pp 41-72, 1982.

[10] R. Moore, Semantic Considerations on Nonmonotonic Logic, Artificial Intelligence, vol. 25, pp 75-94, 1985.

[11] M. Shanahan, The Event Calculus Explained, Springer Lecture Notes in Artificial Intelligence no. 1600, Springer, pp 409-430, 1999.

[12] M. Gelfond and V. Lifschitz, Action Languages, Electronic Transitions on Artificial Intelligence, vol. 3, no 16, 1998.

[13] G. Harman, Change in View, pp 23-35, MIT Press, 1986.

[14] S. Hanks and D. McDermott, Default reasoning, nonmonotonic logic, and the frame problem, Proceedings of the American Association for Artificial Intelligence, pp 328-333, 1986.

[15] Y. Shoham, Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence. Cambridge, Massachusetts: MIT Press, 1988.

[16] L. Morgenstern, The Problem with Solutions to the Frame Problem, in The Robot's Dilemma Revisited: The Frame Problem in Artificial Intelligence, New Jersey: Ablex Publishing Co, pp 99-133, 1996.

[17] P. Thagard, Computational Philosophies of Science, A Bradford Book, MIT Press, Cambridge, Massachusetts, 1988.

[18] E. Lormand, Frame Problem, MIT Encyclopedia of Cognitive Science, Cambridge, Massachusetts: MIT Press, 1998

[19] A. B. Baker, Nonmonotonic Reasoning in the Framework of the Situation Calculus, Artificial Intelligence, vol. 49, pp 5-23, 1991.

[20] J. McCarthy, Mathematical Logic in Artificial Intelligence, Daedalus, vol. Winter 1988, pp 297-311, 1988.

[21] M. Gelfond and V. Lifschitz, Representing Action and Change by Logic Programs, Journal of Logic Programming 17, pp 301-321, 1993.

[22] V. Lifschitz, Toward a Metatheory of Action, KR'91: Principles of Knowledge Representation and Reasoning, 1991

[23] M. Gelfond and V. Lifschitz and and A. Rabinov, What Are the Limitations of the Situation Calculus, Automated reasoning, Essays in Honor of Woody Bledsoe, Kluwer Academic Publishers, vol. 17, pp 167-181, 1991.

[24] R. E. Fikes and N. J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artificial Intelligence, vol. 2, pp 189-208, 1971.

[25] Pednault and P. D. Edwin, Extending Conventional Planning Techniques to Handle Actions with Context-dependent effects, Proceedings of the Seventh National Conference on Artificial Intelligence, pp. 55-59, Morgan Kaufmann Publishers Inc, 1987.