Reasoning about actions: Non-deterministic effects, Constraints, and Qualification

Chitta Baral
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, U.S.A.
chitta@cs.utep.edu
915-747-6952/5030 (voice/fax)

Abstract

In this paper we propose the language of 'state specifications' to uniformly specify effect of actions, executability condition of actions, and dynamic and static constraints. This language allows us to be able to express effects of action and constraints with same first order representation but different intuitive behavior to be specified differently. We then discuss how we can use state specifications to extend the action description languages $\mathcal{A}$ and $\mathcal{L}_0$.

Introduction and Motivation

In this paper[1] we consider several aspects of reasoning about actions: effects (direct, indirect, nondeterministic) of actions, qualification and executability of actions, constraints and their manifestations as ramifications and/or qualification, and propose a language that facilitates representing and reasoning about all of the above in a uniform manner.

We follow the notation of situation calculus (McCarthy & Hayes 1969) and have three different sets of symbols, called *fluents*, *actions*, and *situations*, respectively. For example, in the statement "Shooting causes the turkey to be dead if the gun was loaded", *shoot* is an action, *dead* and *loaded* are fluents, the initial situation is denoted by s_0, and the situation after the shoot is performed is denoted by $res(shoot, s_0)$. The state corresponding to a particular situation is either represented by the set of fluents true in that situation or by the corresponding interpretation. The particular use will be clear by context.

Consider the action "shoot". It is impossible to perform this action if the agent does not have a gun. We will refer to the condition of *having a gun* as an *executability* condition for the action "shoot". Now let us reconsider the statement "Shooting causes the turkey to be dead if the gun was loaded". Here, the condition of the *gun being loaded* is a *precondition* for the fluent "dead" to hold in the situation obtained after performing the action "shoot". In the literature related to frame problems (Brown 1987) both type of conditions are sometimes considered as part of the *qualification* problem.

We consider two kinds of constraints: *dynamic* and *static*. In this paper we consider dynamic constraints to be general statements that are true about adjacent situations. For example, the statements "the salary of a person does not decrease by performing an action" and "no action can make a dead person alive" are examples of dynamic constraints.

Static constraints are statements about the world that are true in all situations. For example, the statement that "a person can not be at two different places at the same time" is a static constraint.

Now consider the action moveto_B. The effect of this action is to make the fluent at_B true. Now suppose that before the action was executed "at_A" was true, i.e. the agent was at the position A. Not only "at_B" would be true after executing "moveto_B", but also the constraint "a person can not be at two different places at the same time" will dictate that "at_A" be false in the resulting situation. We could explicitly state this as an effect of the action "moveto_B". But there might be several different positions in our world, and also there might be several different actions (fly_to_B, drive_to_B, jump_to_B etc.) which have similar interaction with the constraint. A better approach would be state it as a constraint (i.e. not to state it explicitly for all those actions), and have a mechanism that can infer ramifications of a directly specified effects caused by the constraints. Formalizing this is referred to as *the ramification problem* in the literature.

It was pointed out in (Lin & Reiter 1994) that some constraints instead of causing ramifications, affect the executability of an action. Moreover, sometimes these constraints are indistinguishable (in a first order representation) from the constraints that cause ramifications. For example, consider the action "marry_B", with the effect "married_to_B" and the constraint that

[1]Supported by the grants NSF-IRI-92-11-662 and NSF-CDA 90-15-006.

"a person can not be married to two different persons at the same time". Now suppose "married_to_A" is true in a particular situation S. Intuitively, we would like the action "marry_B" to be inexecutable in situation S, rather than have a ramification of the action "marry_B" that makes "married_to_A" false.

One of the goal of this paper is to propose a language which distinguishes between the above two constraints.

Now let us consider the effect of actions in the absence of any preconditions, executability conditions and constraints. Let S and S' be states[2] and let us represent the effect of an action a by E_a. Let us also represent the set of possible states that may be reached after an action a is performed in a situation with state S by $Res(a, S)$. Using Winsletts' definition (Winslett 1989):

Definition 1 $S' \in Res(a, S)$ if

(a) S' satisfies E_a, and

(b) $\nexists S'' : S''$ satisfies E_a and
$(S'' \setminus S) \cup (S \setminus S'') \subset (S' \setminus S) \cup (S \setminus S')$. □

Now let us consider the action of tossing a coin. Intuitively, the effect of this action is either "head" or "tail" (not both) regardless of which of the fluents were true before the coin was tossed. But if we represent this as

E_{toss} = (head $\wedge \neg$ tail) $\vee$ (tail $\wedge \neg$ head)

and use Winslett's definition or the update operator $\diamond$ defined in terms of symmetric difference, then we have the problem that if the coin was "head" before performing "toss" then it stays "head" after performing "toss" and similarly if it was "tail" before performing "toss" then it stays "tail" after performing "toss". This has also been pointed out in (Crawford 1994; Brewka & Hertzberg 1993; Kartha & Lifschitz 1994; Pinto 1994; Sandewall 1992). Representing, E_{toss} = head $\vee$ tail, does not help either.

Let us now consider a different action "paint" whose effect is to paint a block "red" or "blue". But this time the robot is supposed to be smart and minimizes its job when it realizes that the block is already either "red " or "blue". Here, E_{paint} = (red $\wedge \neg$ blue) $\vee$ (blue $\wedge \neg$ red) is adequate when we use Winslett's definition..

Consider the action of "spinning" a coin on a chess board. Let us consider the fluents "black" and "white" which mean that the coin touches a black square and the coin touches a white square respectively. Intuitively, after the coin is spun the fluent "black", "white" or both could be true in the resultant situation regardless of what was true before. But the only way to represent E_{spin}, the effect of spinning the coin, in first order logic is through the formula
E_{spin} = black $\vee$ white, or its equivalent.

[2] Here states are interpretations.

Here, we again get unintuitive result if we use Winslett's definition. Moreover, since E_{toss} and E_{paint} are equivalent in their propositional representation, any approach that does not specify the differences in their intended meaning will be at least wrong with respect to one of them.

When we examine our formalization we observe that there are two aspects to it. (i) We use first order logic/propositional logic to represent effects of actions. (ii) We use a fixed definition of "closeness" based on symmetric difference. To distinguish between "toss" and "paint" we have two choices: (a) to use a more expressive language to represent E_{toss} and E_{paint}, and/or (b) to use different definition of "closeness" for "toss" and "paint", i.e. to use different update operators for "toss" and "paint".

Kartha and Lifschitz (Kartha & Lifschitz 1994) and Sandewall (Sandewall 1992) follow the second approach by allowing specification of when inertia of a fluent (w.r.t. an action) is not preserved. (This results in different update operators.)

In this paper *we propose to use the first approach of using a more expressive language (than first order logic) to represent effects of actions.*

Now let us modify Definition 1 to take into account constraints. Res(a,S) can then be defined as

Definition 2 $S' \in Res(a, S)$ if

(a') S' satisfies $E_a \cup C$, and

(b') $\nexists S'' : S''$ satisfies $E_a \cup C$ and
$(S'' \setminus S) \cup (S \setminus S'') \subset (S' \setminus S) \cup (S \setminus S')$. □

Now let us reconsider the constraints "a person can not be at two different places at the same time", and "a person can not be married to two different people at the same time". These two constraints when expressed in first order logic are equivalent (modulo renaming). But as discussed before they have different intended meanings, the first causes ramification while the second adds qualification conditions. Here also instead of using different update operators for different constraints *we propose to use a more expressive language to represent constraints. Moreover, the condition (a') suggests that we use the same language to represent effects of actions and constraints.* **This is one of the main thesis of our paper.**

In this paper we propose a language to express (i) effects of actions with their preconditions, (ii) executability conditions of actions, and (iii) constraints, such that the drawbacks of using first order logic (FOL) to express them (as described in this section) is avoided.

Before we go on to introduce the syntax and semantics of the language of "state specifications" for specifying

effects of actions and constraints, we briefly discuss disjunctive logic programs. We later give the semantics of "state specifications" through translations to disjunctive logic programs.

Background: Logic Programming Preliminaries

A disjunctive logic program is a collection of rules of the form

$$L_0 \vee \ldots \vee L_k \leftarrow L_{k+1}, \ldots, L_m, not\ \ L_{m+1}, \ldots, not\ \ L_n \quad (1)$$

where L_i's are literals. When $m = n$ and L_i's are only atoms, we refer to the program as a positive disjunctive logic program.

Definition 3 *(Gelfond 1994)* The *answer set* of a disjunctive logic program Π not containing *not* is the smallest (in a sense of set-theoretic inclusion) subset S of **Lit** such that

(i) for any rule $L_0 \vee \ldots \vee L_k \leftarrow L_{k+1}, \ldots, L_m$ if $L_{k+1}, \ldots, L_m \in S$, then $S \cap \{L_0, \ldots, L_k\} \neq \emptyset$

(ii) if S contains a pair of complementary literals, then $S =$ **Lit**.

The set of answer sets of a program Π that does not contain negation as failure is denoted by $a(\Pi)$. □

Definition 4 *(Gelfond 1994)* Let Π be a disjunctive logic program without variables. For any set S of literals, let Π^S be the logic program obtained from Π by deleting

(i) each rule that has a formula *not* L in its body with $L \in S$, and

(ii) all formulas of the form *not* L in the bodies of the remaining rules.

Clearly, Π^S does not contain *not* , so that then $a(\Pi^S)$ is already defined in Definition 3. If $S \in a(\Pi^S)$, then we say that S is an *answer set* of Π. □

Notational Convenience

In this paper we consider a disjunctive logic program to be of a set of rules of the form (1), where $L_0, \ldots, L_k$ are a conjunction of literals. The answer sets of such programs are defined exactly as in Definition 3 and Definition 4, except that (i) in Definition 3 is replaced by the following:

for any rule $L_0 \vee \ldots \vee L_k \leftarrow L_{k+1} \ldots L_m$ if $L_{k+1}, \ldots, L_m \in S$, then there exists i, $0 \leq i \leq k$, such that all conjuncts in L_i are in S.

State Specifications

For an action a, to specify its effect E_a, we need to make statements about the state reached after 'a' is performed (we refer to this state as the *updated state w.r.t. 'a'*). To integrate preconditions and executability conditions of 'a' we also need to consider the state where, 'a' is to be performed (we refer to this state as *initial state w.r.t. 'a'*). While to represent static constraints we only need to refer to the updated states of actions, to be able to represent dynamic constraints we need to able to refer to both initial states and updated states of actions.

To be able to express the truth and falsity of fluents in the initial states and updated states of actions we use four special operators "in", "out", "was_in" and "was_out". For any fluent f, the intuitive meaning of $in(f)$ is that f is true in the updated state. The intuitive meaning of $was\_in(f)$ is that f is true in the initial state. Similarly the meaning of $out(f)$ is that the fluent f is false in the updated state, and the meaning of $was\_out(f)$ is that the fluent f is false in the initial state.

A state specification is a set of rules of the form

$$\begin{aligned} in(b_0) \vee \ldots \vee in(b_k) \vee out(c_0) \vee \ldots \vee out(c_l) \\ \leftarrow in(d_0), \ldots, in(d_m), out(e_0), \ldots, out(e_n) \\ was\_in(f_0), \ldots, was\_in(f_p), \\ was\_out(g_0), \ldots, was\_out(g_q) \end{aligned} \quad (2)$$

where, k, l, m, n, p and q could be 0.

Intuitively, the rule

$in(p) \leftarrow in(q), out(s), was\_in(t), was\_out(u)$

w.r.t an action a means that if t is true and u is false in the initial state then if q is true and s is false in the updated state then p must also be true in the updated state.

Using state specifications

For every action a, we have a corresponding state specification P_a that specifies (i) the effect of a together with the preconditions, and (ii) the executability conditions of action a. We refer P_a as the "update specification" of the action a.

For example, for the action "shoot", the update specification P_{shoot} is as follows:

$$\left.\begin{array}{l} out(alive) \leftarrow was\_in(loaded) \\ \leftarrow was\_out(has\_gun) \end{array}\right\} P_{shoot}$$

Similarly, the update specifications for the actions 'toss', 'paint' and 'spin' can be given as follows:

$$\left.\begin{array}{l} in(head) \vee in(tail) \leftarrow \\ out(head) \leftarrow in(tail) \\ out(tail) \leftarrow in(head) \end{array}\right\} P_{toss}$$

$$in(red) \vee in(blue) \leftarrow \ \} \ P_{paint}$$

$$\left.\begin{array}{l} in(white) \vee in(black) \leftarrow \\ out(white) \vee in(white) \leftarrow in(black) \\ out(black) \vee in(black) \leftarrow in(white) \end{array}\right\} P_{spin}$$

The set of constraints C is also a state specification. In the absence of constraints to reason about the executability and the effects of an action a we only need to consider its update specification P_a. But in the presence of the constraints C we need to consider the state specification $P_a \cup C$.

The dynamic constraint "No action can make a dead person alive" can be represented by the specification:

$\leftarrow in(alive), was\_in(dead)$

Semantics of State Specifications

We now define $Res(a, S)$, the set of states that may be reached by performing action a in a situation corresponding to state S, when the effects and constraints are given as a state specification P.

Our definition of $Res(a, S)$ when effects and constraints are represented by state specifications will be a fixpoint definition similar to the fixpoint definition[3] of McCain and Turner (McCain & Turner), and is based on translating state specifications to disjunctive logic programs.[4]

Algorithm 1 Translating State Specifications
INPUT - s: state, P: state specification, a: action

OUTPUT- $D_{s,P,a}$: a disjunctive logic program

Step 1. Initial Database
For any fluent f if f is true in the state s then the program contains $holds(f, s) \leftarrow$

else the program contains $\neg holds(f, s) \leftarrow$

Step 2. Inertia Rule

(2.1) $holds(F, res(a, s)) \leftarrow holds(F, s), not\ ab(F, a, s)$

(2.2)
$\neg holds(F, res(a, s)) \leftarrow \neg holds(F, s), not\ ab(\overline{F}, a, s)$

Step 3. Translating the update rules

Each revision rule of the type (2) in the state specification P

is translated to the rule
$(holds(b_0, res(a, s)) \wedge ab(\overline{b_0}, a, s)) \vee \ldots \vee$
$(holds(b_k, res(a, s)) \wedge ab(\overline{b_k}, a, s)) \vee$
$(\neg holds(c_0, res(a, s)) \wedge ab(c_0, a, s)) \vee \ldots \vee$
$(\neg holds(c_l, res(a, s)) \wedge ab(c_l, a, s))$
$\leftarrow holds(d_0, res(a, s)), \ldots, holds(d_m, res(a, s)),$
$\neg holds(e_0, res(a, s)), \ldots, \neg holds(e_n, res(a, s)),$
$holds(f_0, s), \ldots, holds(f_p, s)),$
$\neg holds(g_0, s), \ldots, \neg holds(g_q, s)$

Definition 5 Let S be a state and P be a state specification corresponding to action a. Let $D_{S,P,a}$ be the translation of S and P to a disjunctive logic program obtained by Algorithm 1.

If A is a consistent answer set of $D_{s,P,a}$ then
$S' = \{f : holds(f, res(a, s)) \in A\} \in Res(a, S)$.

If $Res(a, S)$ is an empty set we then say that a is not executable in the situations whose state is S. □

The definition of executability of an action in a state S in Definition 5 allows us to specify executability conditions, effect of actions and constraints in a single framework.

Examples

Proposition 1 Consider the action "toss', its update specification P_{toss} and $S = \{h\}$. (For convenience we use 'h' for head and 't' for 'tail'.)

$Res(toss, S) = \{\{h\}, \{t\}\}$ □

Proof(sketch)
Let Π_1 be the program obtained using Algorithm 1. It is easy to show that Π_1 has the two answer sets $\{holds(h, s), \neg holds(t, s), holds(h, res(toss, s)), ab(\overline{h}, toss, s), \neg holds(t, res(toss, s)), ab(t, toss, s)\}$ and $\{holds(h, s), \neg holds(t, s), holds(t, res(toss, s)), ab(\overline{t}, toss, s), \neg holds(h, res(toss, s)), ab(h, toss, s)\}$.

The above two answer sets corresponds to the states $\{h\}$ and $\{t\}$ Hence, $Res(toss, S) = \{\{h\}, \{t\}\}$. □

Proposition 2 Consider the action "spin', its update specification P_{spin} and $S = \{w\}$. (For convenience we use 'w' for white and 'b' for 'black'.)

$Res(toss, S) = \{\{w\}, \{b\}, \{w, b\}\}$ □

It is easy to see that for the action 'paint' with the effect specified by P_{paint}, $Res(paint, \{blue\}) = \{\{blue\}\}$, $Res(paint, \{red\}) = \{\{red\}\}$, and $Res(paint, \{\}) = \{\{red\}, \{blue\}\}$. (As pointed out by Przymusinski and Turner for this example the semantics proposed by Marek and Truszczyński behaves un-intuitively. For all the other examples in this paper the semantics given in this paper coincides with the semantics obtained by directly extending the semantics of Marek and Truszczyński.)

Example 1 Consider the the action "shoot" in YSP (Hanks & McDermott 1987) and the update specification P_{shoot}.

Consider $S_1 = \{has\_gun, loaded, alive\}$, $S_2 = \{has\_gun, alive\}$, and $S_3 = \{loaded, alive\}$.

[3] McCain and Turner (McCain & Turner) show that the definition of Res(a,S) in Definition 2 is equivalent to $Res(a, S) = \{S' : S' = \{L : L \in Cn((S \cap S') \cup E_a \cup C)\}\}$.

[4] We do not use the direct semantics that can be obtained by directly extending the definition of P-justified revision in (Marek & Truszczyński 1994c; Baral August 1994) due to certain un-intuitive behavior. This was pointed out in a manuscript by Przymusinski and Turner. Moreover, we believe that the semantics through translation to disjunctive logic programs to be more intuitive than the direct semantics (Marek & Truszczyński 1994c; Baral August 1994).

$Res(shoot, S_1) = \{\{has\_gun, loaded\}\}$
$Res(shoot, S_2) = \{\{has\_gun, alive\}\}$
Moreover, $Res(shoot, S_3) = \{\}$, implying that *shoot* is not executable in a situation whose state is S_3. □

Example 2 Let us consider the constraint C_1 which says "a person can not be at two different places at the same time", the action "moveto_B", and let us assume that our world consist of only two places, A and B

The update specification of "moveto_B", denoted by P_{moveto_B} is given by $in(at\_B) \leftarrow$

The constraint C_1 is denoted by the following state specification

$out(at\_A) \leftarrow in(at\_B)$
$out(at\_B) \leftarrow in(at\_A)$

It is easy to see that $Res(moveto\_B, \{at\_A\}) = \{\{at\_B\}\}$.

Now let us consider the constraint C_2 which says that "a person can not be married to two different persons at the same time". the action "marry_B", and let us assume that our world consist of only two marriageable persons, A and B.

The update specification of "marry_B", denoted by P_{marry_B} is given by $in(married\_to\_B) \leftarrow$

The constraint C_2 is denoted by the following state specification

$\leftarrow in(married\_to\_B), in(married\_to\_A)$

It is easy to see that $Res(marry\_B, \{married\_to\_A\}) = \{\}$, implying that the action $marry\_B$ is not executable in a situation whose state is $\{married\_to\_A\}$. □

Extending $\mathcal{A}$ and $\mathcal{L}_0$

The language $\mathcal{A}$ was introduced by Gelfond and Lifschitz in (Gelfond & Lifschitz 1992), to express actions and their effects. It was later extended to $\mathcal{L}_0$ (Baral, Gelfond, & Provetti 1994) to include actual situations and to $\mathcal{AR}$ (Kartha & Lifschitz 1994) to include static constraints.

In $\mathcal{A}$, $\mathcal{L}_0$ and $\mathcal{AR}$, effects of actions were specified through propositions of the form

A **causes** F **if** $P_1, \ldots, P_m, \neg P_{m+1}, \ldots, \neg P_n$, and

A **causes** $\neg F$ **if** $P_1, \ldots, P_m, \neg P_{m+1}, \ldots, \neg P_n$

When using update specifications to express effects and preconditions of actions, propositions of the above form are translated to rules of the form

$in(F) \leftarrow was\_in(P_1), \ldots, was\_in(P_m), was\_out(P_{m+1}), \ldots, was\_out(P_n)$, and

$out(F) \leftarrow was\_in(P_1), \ldots, was\_in(P_m), was\_out(P_{m+1}), \ldots, was\_out(P_n)$

respectively, and included in the update specification of A.

The languages $\mathcal{A}$ and $\mathcal{L}_0$ can be extended by having effects and preconditions of actions being represented by updated specifications. Moreover, the translations of $\mathcal{A}$ (Lifschitz & Turner 1994) and $\mathcal{L}_0$ to disjunctive logic programs can be easily adapted to the proposed extensions by following Algorithm 1 to translate the update specifications of actions.

In $\mathcal{AR}$ static constraints[5] are represented as first order theories. *Our proposal is to represent constraints using state specifications. To translate a domain description description in the resulting language to a disjunctive logic program we use Algorithm 1 w.r.t. every action a and the corresponding state specification $P_a \cup C$, where, P_a is the update specification of a, and C is the constraint.*

Conclusion

Marek and Truszczyński (Marek & Truszczyński 1994c) introduce the language of "revision programs"[6] and use it to represent complex effects of actions, such as the action of "reorganizing" a department whose effect is given as "If John is in the department then Peter must not be there", with a non-classical interpretation of "If ... then ..." implying that "John is preferred over Peter". In (Baral August 1994), Baral shows how to translate "revision programs" to extended logic programs and introduces *was_in* and *was_out*, in the body of rules. Marek and Truszczyński also extend (Marek & Truszczyński 1994c) revision programs to allow disjunctions in the head of rules.

In this paper we build on the efforts in (Marek & Truszczyński 1994c; 1994b; Baral & Gelfond 1994) and introduce the language of state specifications. We show that this language is not only able to represent complex effects, but also, non-deterministic effects of actions, executability conditions of actions, preconditions of different effect of actions, and dynamic and static constraints. Moreover, it allows distinct representations of different non-deterministic effects, and different constraints that have same representation in first order theory (See the discussion on "paint" and "spin", and "marry" and "move" in the introduction.). We then show how to incorporate state specifications to action description languages like $\mathcal{A}$ and $\mathcal{L}_0$, and how to implement state specifications through a translation to disjunctive logic programs[7].

[5] $\mathcal{AR}$ does not allow representation of dynamic constraints.

[6] Revision programs are state specifications, with the following restrictions: (a) No *was_in* and *was_out*, (b) No disjunctions in the head of rules, and (c) No rules with empty heads.

[7] Currently, there exists some systems that can compute the answer sets of disjunctive logic programs.

In the full paper[8] we discuss how to add evaluable predicates and variables to the bodies of the rules of state specifications. We also discuss how to express something using state specifications. More specifically, using the algorithm in (Kosheleva & Kreinovich 1992) we can syntactically translate a first-order theory to a logic program and then using (Marek & Truszczyński 1994a) we can construct the corresponding state specification. Also, it should be noted that 'State Specifications' do not behave like first order theories. For example,
(i) the specification $\{\leftarrow in(a)\}$ has different effect then the specification $\{out(a)\}$, and (ii) the specification $\{in(a) \vee in(b)\}$ has different effect then the specification $\{in(a) \vee in(b) \leftarrow, in(a) \vee out(a) \leftarrow\}$. Specifying something using 'state specifications' is then as easy or hard as using disjunctive logic programs.

Recently, McCain and Turner(McCain & Turner) propose using inference rules for representing constraints so as to distinguish between constraints that cause ramification, and that add to the qualification. Unlike their approach we are able to express both dynamic and static constraints, and non-deterministic effects in a single language. Moreover, one of the fundamental thesis of our approach is that effects, and executability conditions of actions be expressed in the same language as the constraints.

One of our main future goal is to study impact of using state specifications with other action theories. In particular we would like to formalize effects of concurrent actions when the effects are given as state specifications.

Acknowledgement

I would like to thank Hudson Turner for his comments on an earlier draft of the paper.

References

Baral, C., and Gelfond, M. 1994. Reasoning about effects of concurrent actions. In Fronhofer, B., ed., *Theoretical Approaches to Dynamic Worlds.*

Baral, C.; Gelfond, M.; and Provetti, A. 1994. Representing Actions I: Laws, Observations and Hypothesis. Technical report, Dept of Computer Science, University of Texas at El Paso.

Baral, C. 1994. Rule based updates on simple knowledge bases. In *Proc. of AAAI94, Seattle*, 136–141.

Brewka, G., and Hertzberg, J. 1993. How to do things with worlds: on formalizing actions and plans. *Journal of Logic and Computation* 3(5):517–532.

Brown, F., ed. 1987. *Proceedings of the 1987 worskshop on The Frame Problem in AI.* Morgan Kaufmann, CA, USA.

[8] A version of it is accessible through http://cs.utep.edu/chitta/chitta.html

Crawford, J. 1994. Three issues in action. In *Presented in the workshop on Non-monotonic reasoning.*

Gelfond, M., and Lifschitz, V. 1992. Representing actions in extended logic programs. In *Joint International Conference and Symposium on Logic Programming.*, 559–573.

Gelfond, M. 1994. Logic programming and reasoning with incomplete information. *Annals of Mathematics and Artificial Intelligence.* To appear.

Hanks, S., and McDermott, D. 1987. Nonmonotonic logic and temporal projection. *Artificial Intelligence* 33(3):379–412.

Kartha, G., and Lifschitz, V. 1994. Actions with indirect effects (preliminary report). In *KR 94*, 341–350.

Kosheleva, K., and Kreinovich, K. 1992. Any theory expressible in first order logic extended by transitive closure can be represented by a logic program. manuscript.

Lifschitz, V., and Turner, H. 1994. From disjunctive programs to abduction. In preparation.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of Logic and computation, special issue on action and processes (to appear).*

Marek, W., and Truszczyński, M. 1994a. Revision programming. manuscript.

Marek, W., and Truszczyński, M. 1994b. Revision programming, database updates and integrity constraints. In *To appear in 5th International conference in Database theory, Prague.*

Marek, W., and Truszczyński, M. 1994c. Revision specifications by means of programs. manuscript.

McCain, M., and Turner, M. A causal theory of ramifications and qualifications.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 4. Edinburgh: Edinburgh University Press. 463–502.

Pinto, J. 1994. *Temporal Reasoning in the Situation Calculus.* Ph.D. Dissertation, University of Toronto, Department of Computer Science. KRR-TR-94-1.

Sandewall, E. 1992. Features and fluents: A systemetic approach to the representation of knowledge about dynamical systems. Technical report, Institutionen for datavetenskap, Universitetet och Tekniska hogskolan i Linkoping, Sweeden.

Winslett, M. 1989. Reasoning about action using a possible models approach. In *Proc. of 7th national conference on AI*, 89–93.