Using Answer Set Programming to model multi-agent scenarios involving agents' knowledge about other's knowledge

Chitta Baral, Gregory Gelfond
Department of Computer Science
Arizona State University
Tempe, AZ 85281, USA.
chitta|ggelfond@asu.edu

Tran Cao Son, Enrico Pontelli
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
epontell|tson@cs.nmsu.edu

ABSTRACT

One of the most challenging aspects of reasoning, planning, and acting in a multi-agent domain is reasoning about what the agents know about the knowledge of their fellows, and to take it into account when planning and acting. In the past this has been done using modal and dynamic epistemic logics. In this paper we explore the use of answer set programming (ASP), and reasoning about action techniques for this purpose. These approaches present a number of theoretical and practical advantages. From the theoretical perspective, ASP's property of non-monotonicity (and several other features) allow us to express causality in an elegant fashion. From the practical perspective, recent implementations of ASP solvers have become very efficient, outperforming several other systems in recent SAT competitions. Finally, the use of ASP and reasoning about action techniques allows for the adaptation of a large body of research developed for single-agent to multi-agent domains. We begin our discussion by showing how ASP can be used to find Kripke models of a modal theory. We then illustrate how both the muddy children, and the sum-and-product problems can be represented and solved using these concepts. We describe and implement a new kind of action, which we call "ask-and-truthfully-answer," and show how this action brings forth a new dimension to the muddy children problem.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

General Terms

Experimentation, Theory

Keywords

Answer set programming, reasoning about actions

1. INTRODUCTION AND MOTIVATION

Cite as: Using Answer Set Programming to model multi-agent scenarios involving agents' knowledge about other's knowledge, C. Baral et al., *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 259-266

Reasoning about the actions of an agent and the associated frame problem has been an important challenge from the early days of AI. The main issue there was to succinctly represent effects of actions on the world as well as the default that the value of a fluent does not change unless it is affected by an action. This becomes more complicated when certain fluents are related to each other (possibly through causal connections), and when actions are not limited to just world-changing actions but may include sensing actions.

During the last two decades most of these problems have been adequately solved with respect to *domains that assume a single agent interacting with the environment*. The language of answer set programming (ASP), which is a specification language as well as an implementation language, and which has a non-standard connective ← that has causal connotations, played an important role in the solutions, as well as in specifying, and encoding various reasoning about action tasks, such as prediction, planning, explaining observations and diagnosis.

Relatively little influenced by the above[1], there has also been significant progress in formulating and implementing multi-agent systems. In a multi-agent scenario, the most challenging aspects of reasoning, planning, and acting in a multi-agent domain are reasoning about what the agents know about the knowledge of other agents in the domain, and taking it into account when planning and acting. In the past this has been done using general logics such as modal [5] and dynamic epistemic logics [15, 1, 7, 14], and specific logics such as the logic of public announcements [2]. In this paper we use ASP and other reasoning about action techniques for reasoning about agent's knowledge about other agents.

One immediate question is what does this buy us?

What follows are some of the direct, and indirect implications of our approach.

(i) We are able to use ASP explicitly to construct the initial Kripke structure. This is important in planning where one needs to start with an initial "state" and determine a sequence of actions that will take it to a given "state", or lead to a desired trajectory. In most other works, the initial Kripke state is shown [5] but it is not discussed how exactly it was obtained.

(ii) Besides encoding the traditional formulation of the "Muddy Children Problem" where the children's answers are recorded as observations, we are able to encode a new kind of action which we call "ask-and-

[1]There are a few exceptions; for example [13, 8, 11, 4].

truthfully-answer" that encodes the thought process of the children, and as a result can be used in planning.

(iii) Our formulation is a general one, and we illustrate this by encoding another classical multi-agent reasoning scenario, the "Sum-N-Product Problem".

(iv) Our use of ASP and reasoning about action techniques allows us to apply various results from the single agent domain, such as dealing with causal relations between fluents, and with issues related to the frame, qualification and ramification problems, to the multi-agent domain.

2. ANSWER SET PROGRAMMING

A logic program in the language of *AnsProlog* (also known as *A-Prolog*) [6, 3] is a set of rules of the form:

$$a_0 \leftarrow a_1, \ldots, a_m, not\ \ a_{m+1}, \ldots, not\ \ a_n \qquad (1)$$

where $0 \leq m \leq n$, each a_i is an atom of a propositional language[2], and *not* represents *negation-as-failure*. A negation-as-failure literal (or naf-literal) has the form *not* a, where a is an atom. Given a rule of this form, the left and right hand sides are called the *head* and *body*, respectively. A rule may have either an empty head or an empty body, but not both. Rules with an empty head are called *constraints*, while those with an empty body are known as *facts*. A *definite rule* is a rule which does not contain naf-literals, and a *definite program* is composed solely of definite rules.

Let X be a set of ground atoms. The body of a rule of the form (1) is *satisfied* by X if $\{a_{m+1}, \ldots, a_n\} \cap X = \emptyset$ and $\{a_1, \ldots, a_m\} \subseteq X$. A rule with a non-empty head is satisfied by X if either its body is not satisfied by X, or $a_0 \in X$. A constraint is *satisfied* by X if its body is not satisfied by X.

Given an arbitrary program, Π, and a set of ground atoms, X, the *reduct* of Π w.r.t. X, Π^X, is the definite program obtained from the set of all ground instances of Π by:

1. deleting all the rules that have a naf-literal *not* a in the body where $a \in X$, and
2. removing all naf-literals in the bodies of the remaining rules.

A set of ground atoms X is an *answer set* of a program Π if it satisfies the following conditions:

1. If Π is a definite program, then X is a minimal set of atoms that satisfies all the rules in Π.
2. If Π is not a definite program, then X is the answer set of Π^X. (Recall that Π^X is a definite program, and its answer set is defined in the first item.)

A program Π is said to be *consistent* if it has an answer set, and *inconsistent* otherwise. To make answer set programming easier, Niemelä et al. [12] introduced a new type of rule, called *cardinality constraint rule*:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ \ A_{m+1}, \ldots, not\ \ A_n \qquad (2)$$

where each A_i is a *choice atom*. A choice atom has the form $l\{b_1, \ldots, b_k\}u$ where each b_j is an atom, and l and u are integers such that $l \leq u$. Choice atoms can be also written as $l\{p(\bar{X}) : q(\bar{X})\}u$, where $\bar{X}$ is a set of variables—this is a short-hand for the choice atom $l\{p(\bar{s}_1, \ldots, p(\bar{s}_k\}u$, where $\{\bar{s}_1, \ldots, \bar{s}_k\}$ are all the ground instantiations of $\bar{X}$ such that $q(\bar{X})$ is true. As with rules of type (1), rules of this from may have an empty head. A set of atoms X satisfies a choice atom $l\{b_1, \ldots, b_k\}u$ if $l \leq |X \cap \{b_1, \ldots, b_k\}| \leq u$. The notion of satisfaction of a rule with choice atoms can be extended in the usual way. The semantics of logic programs which contain such rules is given in [12].

The possibility of a program having multiple answer sets (or no answer sets) has given rise to an alternative way of solving problems via logic programming, called *Answer Set Programming (ASP)* [10, 12]. This approach revolves around writing programs whose answer sets have a one-to-one correspondence with the solutions of the particular problem. Typically a ASP program consists of:

1. Rules to enumerate the possible solutions of a problem as *candidate* answer sets;
2. Constraints to eliminate answer sets not representing solutions of the problem.

We illustrate ASP with a simple example, capturing the ASP solution to the problem of computing the 3-coloring of a directed graph G. Let us assume we have the three colors red, green, and blue, represented by the constant r, g, and b. Let us also represent the vertices of G as the numbers $0, \ldots, n$. The program $\Pi(G)$ consists of the following:

- A set of atoms $edge(u, v)$ denoting the edges (u, v) of G;
- For each vertex u of G, a set of rules describing the possible colorings of u:

$$1\{color(u, g), color(u, b), color(u, r)\}1$$

- For each edge (u, v) of G, a set of constraints encoding the fact that u and v must be assigned different colors:

$$\leftarrow color(u, r), color(v, r), edge(u, v)$$
$$\leftarrow color(u, b), color(v, b), edge(u, v)$$
$$\leftarrow color(u, g), color(v, g), edge(u, v)$$

It can be shown that for each graph G, (i) $\Pi(G)$ is inconsistent if and only if the 3-coloring problem of G does not have a solution; and (ii) if $\Pi(G)$ is consistent, then the answer sets of $\Pi(G)$ have a one-to-one correspondence with solutions of the 3-coloring problem of G.

3. USING ASP TO FIND KRIPKE MODELS

3.1 Encoding Agents Knowledge

We assume a finite, non-empty set of agents $\{1, \ldots, n\}$, and a set of propositions describing various properties of the world, referred to as *fluents*. The language, $\mathcal{LK}_n$, for representing and reasoning about the knowledge of our agents is comprised of formulae built from our fluents, propositional connectives, and a set of modal operators $\mathbf{K}_i$, with one such operator per agent. Formulae are defined as follows:

- *Fluent formulae*: a fluent formula is a propositional formula built from fluents and Boolean operators.
- *Knowledge formulae*: a knowledge formula is *(i)* a fluent formula, or *(ii)* a formula of the form $\mathbf{K}_i\psi$ where ψ is a knowledge formula, or *(iii)* a formula of the form $\psi \vee \phi$, $\psi \wedge \phi$, or $\neg\psi$, where ψ and ϕ are knowledge formulae.

In addition, for a non-empty set of agents G:

- $\mathbf{E}_G\psi$ denotes the set of formulae $\{\mathbf{K}_i\psi \mid i \in G\}$.
- $\mathbf{C}_G\psi$ denotes the set of formulae of the form $\mathbf{E}_G^k\psi$, where $k \geq 1$ and $\mathbf{E}_G^{k+1}\psi = \mathbf{E}_G^k\mathbf{E}_G\psi$.

Formulae in the language $\mathcal{LK}_n$ allow us to specify:

- The real state of the world;

[2]A rule with variables is viewed as a shorthand for the set of its ground instances.

- The knowledge of an agent about the world—e.g., $\mathbf{K}_i\psi$ where ψ is a fluent formula;
- The knowledge of an agent about the knowledge of other agents—e.g., $\mathbf{K}_i\mathbf{K}_j\psi$, where ψ is a fluent formula;
- The common knowledge among agents—e.g., $\mathbf{C}_G\psi$.

EXAMPLE 1 (MUDDY CHILDREN [5]). *A father says to his three children that at least one of them has mud on his forehead. He then repeatedly asks "do you know whether your forehead is muddy or not?" The first two times all three children answer "no." The third time however, they all answer "yes." It is known that the father and the children can see and hear each other.*

Let us denote the children by 1, 2, and 3. In addition, let m_i be a fluent encoding the fact that child i is muddy. Some formulae in the language $\mathcal{LK}_3$ are: (i) m_i (i is muddy); (ii) $\mathbf{K}_1 m_1$ (child 1 knows that he is muddy); (iii) $\mathbf{K}_1\mathbf{K}_2 m_2$ (child 1 knows that child 2 knows that he is muddy); and (iv) $\mathbf{C}_{1,2,3}(m_1 \vee m_2 \vee m_3)$ (it is common knowledge among the children that at least one of them is muddy).

An $\mathcal{LK}_n$ theory is a set of $\mathcal{LK}_n$ formulas. The semantics of $\mathcal{LK}_n$ theories is given by Kripke structures:

DEFINITION 1. *A Kripke structure M, over a set of fluents Φ, is a tuple $(\mathcal{S}, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$ where $\mathcal{S}$ is a set of state symbols, π is an interpretation which associates with each state symbol a truth assignment to the fluents of Φ, and $\mathcal{K}_i$ is a binary relation on $\mathcal{S}$, for each $1 \leq i \leq n$.*

Given a Kripke structure $M = (\mathcal{S}, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, a state $s \in \mathcal{S}$, and a $\mathcal{LK}_n$-formula φ, the entailment relation $(M, s) \models \varphi$ is defined as follows:

- If φ is a fluent formula, then $(M, s) \models \varphi$ iff $s \models \varphi$;
- $(M, s) \models \mathbf{K}_i\varphi$ iff $\forall t : (s, t) \in \mathcal{K}_i \Rightarrow (M, t) \models \varphi$;
- $(M, s) \models \neg\varphi$ if and only if $(M, s) \not\models \varphi$.

Kripke structures are often viewed as directed labeled graphs, whose set of nodes is $\mathcal{S}$, and whose edges are specified by $\mathcal{K}_i$. Whenever we say that M' is obtained from M by removing an edge between (s, t) labeled by i from M, we remove (s, t) from $\mathcal{K}_i$. Throughout this paper we will make use of the standard notions of a *path* between two nodes in M, and *reachability* without providing formal definitions.

DEFINITION 2. *Given a Kripke structure M, two states s, t in M, and a set of agents G, we say that t is G-reachable from s iff there is a path from s to t involving only edges labeled by agents in G. We say that t is* reachable *from s if t is G-reachable from t and G is the set of all the agents.*

Different logic systems for reasoning about the knowledge represented by a $\mathcal{LK}_n$ theory have been introduced. Each system uses the following two inference rules:

- **R1**: From φ and $\varphi \Rightarrow \psi$ infer ψ
- **R2**: From φ infer $\mathbf{K}_i\varphi$

and satisfies some (or all) of the following axioms:

- **P**: all instances of axioms of propositional logic
- **K**: $(\mathbf{K}_i\varphi \wedge \mathbf{K}_i(\varphi \Rightarrow \phi)) \Rightarrow \mathbf{K}_i\phi$
- **T**: $\mathbf{K}_i\varphi \Rightarrow \varphi$
- **4**: $\mathbf{K}_i\varphi \Rightarrow \mathbf{K}_i\mathbf{K}_i\varphi$
- **5**: $\neg\mathbf{K}_i\varphi \Rightarrow \mathbf{K}_i\neg\mathbf{K}_i\varphi$
- **D**: $\neg\mathbf{K}_i false$

The system $S5$ satisfies all of the above axioms with the exception of (**D**) [9]. Many other logics can be defined by removing some axioms among (**T**), (**R**), (**4**), and (**5**) from $S5$, and/or adding (**D**) to it. Note that in the $S5$ system, the binary relation $\mathcal{K}_i$ is reflexive, transitive, and symmetric.

3.2 Modeling Kripke Structures in ASP

Given a $\mathcal{LK}_n$ theory $\mathcal{T}$, we would like to compute a Kripke structure $M = (\mathcal{S}, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, which is a model of $\mathcal{T}$. In this section, we will develop a program $\Pi_I(\mathcal{T}, m, k)$ whose answer sets represent Kripke models of $\mathcal{T}$ with m states, assuming they exist. Our approach here is analogous to finding plans using ASP or SAT; in that case, the ASP or SAT encodings are with respect to a given plan length m and they find all plans of length m if such plans exist. In our encoding m denotes the number of states, and k denotes the number of reasoning steps.[3] $\Pi_I(\mathcal{T}, m, 0)$ denotes our initial structure.

Let us describe the ASP representation of $\Pi_I(\mathcal{T}, m, k)$. The intuition guiding our encoding is to model the Kripke structure as "existing at each time point." The language of the program $\Pi_I(\mathcal{T}, m, k)$ includes the following components:

- a set of atoms of the form $fluent(F)$;
- a set of constants $s_1, \ldots, s_m$, representing the names of the possible states;
- a set of atoms of the form $state(S, T)$, denoting the fact that S is a state in the Kripke structure at step T;
- a set of atoms of the form $h(\varphi, S, T)$ denoting the fact that the formula φ holds in the state S in the Kripke structure present at step T. These atoms represent the interpretation associated with each state (i.e., π);
- a set of atoms of the form $r(A, S_1, S_2, T)$, which represent the accessibility relations of the Kripke structure present at step T. This states that $(S_1, S_2) \in \mathcal{K}_A$ in the Kripke structure at step T;
- a set of atoms of the form $t(S1, S2, T)$, used to represent the existence of a path from S_1 to S_2 in the Kripke structure present at step T;
- a set of atoms of the form $real(S, T)$, used to denote the real state of the world in the Kripke structure of step T.

We assume formulae to be built from fluents, propositional connectives, and knowledge operators (in keeping with our previous definition). In particular, the fact that a formula of the form $\mathbf{K}_A\varphi$ holds in the current Kripke structure with respect to a state S is encoded by atoms of the form $k(A, \varphi, S, T)$. The formulae φ which comprise our theory $\mathcal{T}$ are described by atoms $init(\varphi)$.

Let us now describe the rules of $\Pi_I(\mathcal{T}, m, 0)$. As our goal is to generate Kripke structures which satisfy $\mathcal{T}$, and the states are given by facts of the form $state(s_i, 0)$, we must have a rule to generate the possible accessibility relations $\mathcal{K}_i$ for our agents:[4]

$$0\{r(A, S_1, S_2, 0) : state(S_1, 0) : state(S_2, 0)\}1 \leftarrow agent(A) \quad (3)$$

This rule states that there might be an edge labeled A from S_1 to S_2. Depending on the application, we have rules to ensure that our accessibility relations are symmetric, transitive, and reflexive (represented by the predicate r which

[3]This last parameter is added for use in the next section.

[4]In all the rules, A, F, L, and S, possibly with indices, denote an agent, a fluent, a fluent literal, and a state, respectively. We simplify the rules by removing the definition of these variables in the right hand side of the rules.

stands for *reachable*):

$$r(A, S_1, S_2, 0) \leftarrow r(A, S_1, S_3, 0), r(A, S_3, S_2, 0) \quad (4)$$
$$r(A, S_1, S_2, 0) \leftarrow r(A, S_2, S_1, 0) \quad (5)$$
$$r(A, S, S, 0) \leftarrow \quad (6)$$

This allows us to concisely define the predicate t as the transitive closure of r:

$$t(S_1, S_2, T) \leftarrow r(A, S_1, S_2, T) \quad (7)$$
$$t(S_1, S_2, T) \leftarrow t(S_1, S_3, T), t(S_3, S_2, T) \quad (8)$$

Using the predicate h as a base, we can express the conditions under which our knowledge formulae are satisfied ($n\_k(A, S, F, T)$ indicates that the formula $\mathbf{K}_A(F)$ does not hold w.r.t. state S in the Kripke structure at time T):

$$n\_k(A, S, F, T) \leftarrow r(A, S, S1, T), h(\neg F, S_1, T) \quad (9)$$
$$n\_k(A, S, \neg F, T) \leftarrow r(A, S, S1, T), h(F, S_1, T) \quad (10)$$
$$k(A, S, L, T) \leftarrow not\ \ n\_k(A, S, L, T) \quad (11)$$

These rules describe the entailment relation $(M_\mathcal{T}, S) \models \mathbf{K}_A L$, where $M_\mathcal{T}$ is the Kripke structure generated by the program and L is a fluent literal. The first rule states that an agent A does not know if a fluent F is true with respect to the state S if there is a state S_1 accessible from S via A where F is false. The second rule is symmetrical, testing the knowledge of $\neg F$ being true. The last rule states that if it cannot be proven that A does not know the value of L, then A knows the value of L.

We define h recursively to deal with the various types of knowledge formulae:[5]

$$h(k(A, G), S, T) \leftarrow literal(G), k(A, S, G, T)$$
$$h(\neg k(A, G), S, T) \leftarrow not\ \ k(A, S, G, T)$$
$$h(or(L_1, L_2), S, T) \leftarrow form(or(L_1, L_2)), h(L_1, S, T)$$
$$h(or(L_1, L_2), S, T) \leftarrow form(or(L_1, L_2)), h(L_2, S, T)$$
$$h(or(L_1, L_2, L_3), S, T) \leftarrow form(or(L_1, L_2, L_3)), h(L_1, S, T)$$
$$h(or(L_1, L_2, L_3), S, T) \leftarrow form(or(L_1, L_2, L_3)), h(L_2, S, T)$$
$$h(or(L_1, L_2, L_3), S, T) \leftarrow form(or(L_1, L_2, L_3)), h(L_3, S, T)$$
$$h(neg\_c(L), S, T) \leftarrow form(L), t(S, S1, T), not\ \ h(L, S1, T)$$
$$h(c(L), S, T) \leftarrow form(L), not\ \ h(neg\_c(L), S, T)$$

This collection of rules extends the definition of the predicate h to knowledge formulae and compound formulae. The possible formulae are described by the predicate $form$; some of the rules of $form$ are:

$$literal(L) \leftarrow fluent(L)$$
$$literal(\neg L) \leftarrow fluent(L)$$
$$form(L) \leftarrow literal(L)$$
$$form(k(A, G)) \leftarrow literal(G)$$
$$form(or(k(A1, G1), k(A2, G2))) \leftarrow literal(G1), literal(G2)$$
$$form(\neg k(A, G)) \leftarrow literal(G)$$
$$form(or(L1, L2, L3)) \leftarrow literal(L1), literal(L2), literal(L3)$$

As we wish to generate Kripke structures which are models of $\mathcal{T}$, we need to enforce the following constraints:

$$real(S, T+1) \leftarrow real(S, T) \quad (12)$$
$$\leftarrow real(S, 0), init(F), not\ \ h(F, S, 0) \quad (13)$$
$$1\{real(S, 0) : state(S, 0)\}1 \quad (14)$$

The second rule states that every formula of the theory $\mathcal{T}$ must be satisfied with respect to the state designated as the real world. The third rule states that one of the possible states of the world is the real state of the world, and the first rule asserts that the real world remains the same as time progresses.

EXAMPLE 2. *Let $\mathcal{T}_0$ be the theory representing the Muddy Children problem before the father's announcement. The knowledge of the three children can be represented by the following formulae:*

$$init(c(or(k(1, m(2)), k(1, \neg m(2))))) \quad (15)$$
$$init(c(or(k(1, m(3)), k(1, \neg m(3))))) \quad (16)$$
$$init(c(or(k(2, m(1)), k(2, \neg m(1))))) \quad (17)$$
$$init(c(or(k(2, m(3)), k(2, \neg m(3))))) \quad (18)$$
$$init(c(or(k(3, m(1)), k(3, \neg m(1))))) \quad (19)$$
$$init(c(or(k(3, m(2)), k(3, \neg m(2))))) \quad (20)$$
$$init(c(\neg k(1, m(1)))) \quad (21)$$
$$init(c(\neg k(1, \neg m(1)))) \quad (22)$$
$$init(c(\neg k(2, m(2)))) \quad (23)$$
$$init(c(\neg k(2, \neg m(2)))) \quad (24)$$
$$init(c(\neg k(3, m(3)))) \quad (25)$$
$$init(c(\neg k(3, \neg m(3)))) \quad (26)$$

The formulae indicate that everyone knows that child 1 knows whether or not child 2 is muddy (formula (15)), and whether or not child 3 is muddy (formula (16))), and that child 1 does not know whether or not he is muddy (formulae (21) and (22)). Similarly for children 2 and 3.

Let $\Pi(0)$ be the program consisting of the rules (3)-(26) and the set of eight states representing the eight possible interpretations of the set of fluents $\{m(1), m(2), m(3)\}$. We can prove the following property about the program $\Pi(0)$:

PROPOSITION 1. $\Pi(0)$ *is consistent and has eight answer sets; each corresponds to Fig. 1 with the only difference being the choice of the real state of the world.*

One of the answer sets of $\Pi(0)$ is shown graphically in Fig. 1, displaying the accessibility relations of the three children within the model (the three digits in each state encode the value of the three fluents, where 1 denotes true and 0 denotes false). This model also coincides with the model represented by most texts discussing this famous problem.

4. THE MUDDY CHILDREN PROBLEM

We are now ready to discuss the Muddy Children problem in greater detail. The action of the father announcing that there is at least one muddy child among the three can be encoded by ensuring that the formula $\varphi \doteq \mathbf{C}_{\{1,2,3\}}(m_1 \vee m_2 \vee m_3)$ is true. This has the effect of enabling the children to refine their knowledge, possibly removing unnecessary states and edges. We can encode this in ASP as follows—where *occ*

[5]The encoding is tailored to meet the grounding requirement of current ASP-solvers. It can be generalized to deal with compound formulae using recursive formula representation.

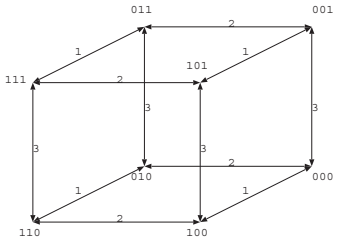

Figure 1: Kripke Model (Step 0)

is a binary predicate that encodes action occurrences at the various steps:

$$occ(announce\_fact(\varphi), 0) \leftarrow$$
$$state(S, 1) \leftarrow occ(announce\_fact(\varphi), 0), state(S, 0), h(\varphi, S, 0)$$
$$r(A, S1, S2, 1) \leftarrow r(A, S1, S2, 0), state(S1, 1), state(S2, 1)$$

The second rule above encodes the fact that the states at step 1 are those states from step 0 where φ holds. The third rule ensures that the edges at step 1 are those edges from step 0 connecting states that are still present at step 1. This is reminiscent of the ASP encoding of effect axioms in reasoning about actions, except that the states here behave like fluents in traditional reasoning about actions. In situation calculus, the second rule above would be expressed as $state(S, s_0) \wedge \varphi(S, s_0) \Rightarrow state(S, do(announce\_fact(\varphi), s_0))$.

Let R_1 be the set of rules above and $\Pi(1) = \Pi(0) \cup R_1$. Computing the answer sets of this program, we obtain the Kripke structure at step 1 shown in Fig. 2.

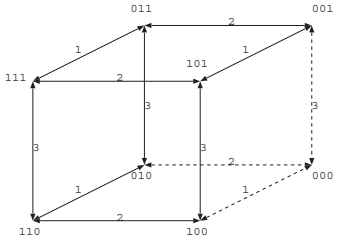

Figure 2: After Father's Announcement (Step 1)

The program has seven answer sets which correspond to the seven possible real-state of the world. However, the resulting Kripke structure is identical across answer sets. This certainly is intuitive since the Kripke structure in step 0 does not depend on the real-state of the world which implies that the Kripke structure in step 1 should not depend on it. The dotted edges are those edges that have been removed from the initial structure (Fig. 1).

The father continues with the action ask, which in effect does not change the knowledge of any child. This is encoded by the set of rules R_2:

$$occ(ask, 1) \leftarrow$$
$$state(S, 2) \leftarrow occ(ask, 1), state(S, 1)$$
$$r(A, S1, S2, 2) \leftarrow r(A, S1, S2, 1), state(S1, 2), state(S2, 2)$$

Let $\Pi(2) = \Pi(1) \cup R_2$. $\Pi(2)$ has as many answer sets as $\Pi(1)$, describing the same Kripke structures.

The children then reply "no" to their father. This is equivalent to the children announcing that they do not know whether or not they are muddy. This is encoded by the action $announce\_k(A, false)$ indicating that agent A does not know $m(A)$. This action has the following effects:

- The states of the Kripke structure are not affected.
- The knowledge formulae $\mathbf{C}_{\{1,2,3\}}(\neg \mathbf{K}_A m_A)$ and $\mathbf{C}_{\{1,2,3\}}(\neg \mathbf{K}_A \neg m_A)$ are true.

In order to compute the subsequent Kripke structure we use the following set of rules R_3:

$$state(S, 3) \leftarrow state(S, 2) \quad (27)$$
$$bad(S_1, S_2, 2) \leftarrow real(S, 2), \quad (28)$$
$$t(S, S_1, 2), S_1 \neq S_2, r(A_1, S_1, S_2, 2),$$
$$occ(announce\_k(A_2, false), 2),$$
$$not\ \ h(\neg k(A_2, m(A_2)), S_2, 2)$$
$$bad(S_1, S_2, 2) \leftarrow real(S, 2), \quad (29)$$
$$t(S, S_1, 2), S_1 \neq S_2, r(A_1, S_1, S_2, 2),$$
$$occ(announce\_k(A_2, false), 2),$$
$$not\ \ h(\neg k(A_2, \neg m(A_2)), S_2, 2)$$
$$bad(S_1, S_2, 2) \leftarrow real(S, 2), \quad (30)$$
$$t(S, S_1, 2), S_1 \neq S_2, r(A_1, S_1, S_2, 2),$$
$$occ(announce\_k(A_2, true), 2),$$
$$h(\neg k(A_2, m(A_2)), S_2, 2), h(\neg k(A_2, \neg m(A_2)), S_2, 2)$$
$$bad(S_1, S_2, 2) \leftarrow bad(S_2, S_1, 2) \quad (31)$$
$$r(A, S_1, S_2, 3) \leftarrow r(A, S_1, S_2, 2), not\ \ bad(S_1, S_2, 2) \quad (32)$$

The rules (28)-(29) identify links labeled A_1 from S_1 to S_2 which, if present, will lead to violating the validity of the formulae $\mathbf{C}_{\{1,2,3\}}(\neg \mathbf{K}_A m_A)$ and $\mathbf{C}_{\{1,2,3\}}(\neg \mathbf{K}_A \neg m_A)$—because they link to states where $\mathbf{K}_A m_A$ or $\mathbf{K}_A \neg m_A$ are satisfied. The rule (30) takes care of the case when A_2 announces that it knows the value of m_{A_2}. Thus, atoms of the form $bad(S_1, S_2, 2)$ identify links that need to be removed from the Kripke structure to enable the satisfaction of the common knowledge derived from the announcement. Rule (31) indicates that $bad(S_1, S_2, 2)$ is symmetric and rule (32) creates the new Kripke structure excluding the bad links.

Let $\Pi(3) = \Pi(2) \cup R_3$. This program also has seven answer sets as $\Pi(2)$. However, the Kripke structure depends on the real state of the world. If the real state corresponds to one of the four possible states with more than one children muddy, the Kripke structure consists of self-loops and three links (Fig. 3, left)—this is the case commonly considered in several publications related to this problem. For the other three cases, the structure consists of only self-loops (Fig. 3, right). The red (dashed) links are removed from the structure after the announcements are made.

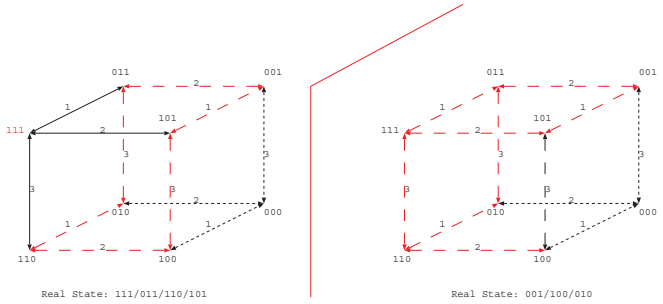

Figure 3: After Children First Answer (Step 3)

Repeating this pair of actions (*ask*, *announce*) a second time yields a Kripke structure whose states have only self-loops in all answer sets. This allows the children to affirmatively answer their father's question.

5. ASK AND TRUTHFULLY ANSWERED

The encoding presented in the previous section relies on determining Kripke structures that satisfy formulae obtained from observations of the behavior of the agents. Intuitively, the encoding acts as an external observer that is recording the actions and responses of the participating agents.

Let us now present an alternative encoding, where the process of asking and answering the question is not passively observed; instead, the behavior of the agents is encoded and the answers are automatically computed from the current Kripke structure. The encoding will allow us to solve the general muddy children problem:

> *Suppose there are n children and the father comes and announce that at least one of them is muddy. He then repeatedly asks "do you know whether you are muddy?" until all children say "yes." Prove that if there are exactly ℓ ($\ell \leq n$) children who are muddy, the father has to ask exactly ℓ questions before* all muddy *children say "yes."*

For simplicity, we will continue to illustrate the program with $n = 3$. We will also continue using the representation developed in the previous section. The program simulates the conversation between the father and the children, automatically computing the children's responses from the current Kripke structure. We refer to this encoding as *Ask and Truthfully Answer*.

5.1 Children Observing the World

The first stage of the encoding corresponds exactly to the collection of rules we denoted with $\Pi(0)$ earlier—these rules capture the fact that each child can *see* the other children.

It is interesting to observe that if we are interested in studying what knowledge each child holds:

$$hreal(F,T) \leftarrow real(S,T), h(F,S,T)$$

we discover that each answer set contains $hreal(k(1,m(2)),0)$, $hreal(k(1,m(3)),0)$, $hreal(k(2,m(1)),0)$, $hreal(k(2,m(3)),0)$, $hreal(k(3,m(1)),0)$, and $hreal(k(3,m(2)),0)$, if the real-state of the world is 111. This confirms that each child is correctly seeing the other children.

5.2 Father's Initial Announcement

The father produces the initial announcement; as before, we capture this using an initial fact of the form:

$$occ(announce\_fact(\varphi),0) \tag{33}$$

(here, φ is $or(m(1),m(2),m(3))$). We are interested in maintaining only states that satisfy the announced facts:

$$state(S,T{+}1) \leftarrow occ(announce\_fact(\varphi),T), h(\varphi,S,T) \tag{34}$$

while we maintain all links that connect states that are maintained in the structure:

$$r(A,S_1,S_2,T{+}1) \leftarrow state(S_1,T{+}1), state(S_2,T{+}1), occ(announce\_fact(\varphi),T), r(A,S_1,S_2,T) \tag{35}$$

5.3 Asking and Answering Questions

The occurrences of *ask* questions, denoted by $occ(ask,T)$, do not have any effect on the structure:

$$state(S,T+1) \leftarrow state(S,T), occ(ask,T) \tag{36}$$

$$r(A,S_1,S_2,T+1) \leftarrow agent(A), state(S_1,T+1), state(S_2,T+1), occ(ask,T), r(A,S_1,S_2,T) \tag{37}$$

On the other hand, each occurrence of an *ask* question triggers the generation of a reply, in the form of instances of *announce_k* actions. The actual formula reported will be determined by each child based on their observations, which are relative to the real state of the world. Note that, even though the children do not know the real world, their answers are based on their observations about the real world. To evaluate any conclusion of the type $\mathbf{K}_i\varphi$ about agent i, one needs to evaluate φ with respect to the set of worlds the agent i considers possible. These worlds are exactly the world accessible from the real world using the accessibility relation of agent i. Thus in our encoding, the child answers *yes* if $(M_T, real) \models \mathbf{K}_i m_i \vee \mathbf{K}_i \neg m_i$, while it responds *no* if $(M_T, real) \models \neg\mathbf{K}_i m_i \wedge \neg\mathbf{K}_i \neg m_i$, where M_T denotes the Kripke structure at step T. The first situation leads to $announce\_k(i,true)$ while the second leads to $announce\_k(i,false)$. The generation of these responses is described by the following rules. In this sense, we can say that the child answers the question *truthfully*.

$$occ(announce\_k(A,true),T) \leftarrow real(S,T), occ(ask,T{-}1), h(k(A,m(A)),S,T) \tag{38}$$

$$occ(announce\_k(A,true),T) \leftarrow real(S,T), occ(ask,T{-}1), h(k(A,\neg m(A)),S,T) \tag{39}$$

$$occ(announce\_k(A,false),T) \leftarrow real(S,T), occ(ask,T{-}1), h(\neg k(A,m(A)),S,T), h(\neg k(A,\neg m(A)),S,T) \tag{40}$$

In turn, the father will keep asking as long as one of the children says that he does not know whether he is muddy or not.

$$occ(ask,T+1) \leftarrow occ(announce\_k(A,false),T) \tag{41}$$

The Kripke structure will be updated as a consequence of the responses produced by the children. The update is used to refine the knowledge of the children—this is modeled by removing links that represent incorrect knowledge of the child. This is captured by a predicate $bad(S_1,S_2,T)$, which is true if the following conditions are satisfied:

- S_1 and S_2 are distinct states reachable from the real state of the world;
- There is a link between S_1 and S_2;
- There is an agent A, who announces φ, and $(M_T, S_2) \not\models \mathbf{K}_A\varphi$.

This is captured by the rules similar to the rules (28)-(31) presented earlier, where the last parameter of the predicate $bad(S_1,S_2,2)$ is replaced by the step parameter T. Let $\Pi(k)$ be the program consisting of

- $\Pi(1)$ (rules (3)-(26)) and R_1 for generating the initial Kripke structure and reasoning about the announcement at the beginning of the story;
- The rules (36)-(37) for reasoning about the effects of the action *ask* where $T < k$;

- The rules (38)-(40) and (41) where $T < k$ for generating action occurrences;
- The rules (27)-(32) where 2 and 3 are replaced by $T-1$ and T with $T < k$, respectively, for reasoning about the effects of the actions of announcing knowledge of the children.

We can prove the following properties of the program $\Pi(k)$.

PROPOSITION 2. *Let us consider the muddy children problem with n children with exactly ℓ muddy children. Every answer set M of the program $\Pi(k)$, with $k \geq 2n+1$:*

- *contains a unique atom of the form $real(s, 0)$;*
- *contains $occ(announce\_k(a, false), i)$ for $i = 2t < 2\ell$ and a such that $h(m(a), s, 0) \in M$; and*
- *contains $occ(announce\_k(a, true), 2\ell)$ for every a such that $h(m(a), s, 0) \in M$.*

The first item corresponds to the fact that there is only one real state of the world in each answer set. The second item indicates that all muddy children do not know whether they are muddy or not before the father finishes with the ℓ^{th} question. The third item corresponds to the fact that, after the father finishes with the ℓ^{th} question, all muddy children know whether they are muddy or not.

The following atoms are extracted from an answer set of $\Pi(7)$ for the case $n = 3$ and the real state of the world is 110 (child 1 and 2 are muddy, 3 is clean):

```
occ(announce_fact(or(m(1),m(2),m(3))),0)
real(2) occ(ask,1) occ(ask,3) occ(ask,5)
occ(announce_k(1,false),2) occ(announce_k(2,false),2)
occ(announce_k(3,false),2) occ(announce_k(1,true),4)
occ(announce_k(2,true),4)  occ(announce_k(3,false),4)
occ(announce_k(1,true),6)  occ(announce_k(2,true),6)
occ(announce_k(3,true),6)
```

6. SUM-AND-PRODUCT

The previous section illustrates an approach to reasoning about knowledge in a multi-agent system where agents announce their knowledge or facts to the whole community. In this approach, the act of exchanging information between agents are recorded as actions (e.g., asking and responding with a knowledge fact). In the muddy children problem, the Kripke structures after the execution of an action depends solely on the Kripke structure before the execution of the action. The approach relies on this fact and provides general rules for dealing with two types of actions, the action of announcing a fact and the action of announcing a knowledge formula. In this section, we show that the proposed approach can be generalized to deal with problems where this dependence does not hold. We use the well-known sum-and-product problem to illustrate this point.

An agent chooses two numbers $1 < x < y$ such that $x + y \leq 100$. The sum $x + y$ is communicated to agent s while the product $x \cdot y$ is communicated to agent p. These communications are private, and the agents are tasked with discerning the values of x and y. The following conversation takes place between the two agents:

a. p states that it does not know the numbers x, y

b. s indicates that it already knew this fact

c. p states that now it knows the two numbers x, y

d. s states that now it knows the two numbers x, y as well.

We wish to determine the values of x, y discovered by the agents.

It is easy to see that this problem only deals with the knowledge of the two agents involved; our objective is to determine Kripke structures that represent a correct model of the knowledge of the agents at the different steps in the story. In this sense, it is similar to the muddy children problem. The key difference between these problems lies in the fact that the announcement (b) refers to the state of knowledge of s with respect to the same initial Kripke structure the announcement (a) refers to.

We will now apply our strategy to solve the sum-and-product problem, which we refer to as Γ. The language of Γ contains

- Rules to declare that each state is of the form $num(X, Y)$, i.e., each state is a pair of numbers;
- Rules to define the fluents $x(X)$, $y(Y)$, $sum(Z)$, and $prod(P)$, respectively denoting the fact that X is the value of x, Y is the value of y, Z is the sum of the two numbers encoding the state, and P is the product of the two numbers encoding the state;
- A set of atoms of the form $state(S, T)$, denoting the fact that S is a state in the Kripke structure at step T.

Additionally, the language will use atoms of the form $h(\varphi, S, T)$, $r(A, S_1, S_2, T)$, $real(S, T)$, and $t(S1, S2, T)$, whose meaning is the same as in the previous section.

The rules describing the states and the choices of x, y are:

$$state(num(X, Y), T) \leftarrow 1 < X, X < Y, X + Y < 101$$
$$h(x(X), num(X, Y), T) \leftarrow$$
$$h(y(Y), num(X, Y), T) \leftarrow$$
$$h(sum(S), num(X, Y), T) \leftarrow S = X + Y$$
$$h(prod(P), num(X, Y), T) \leftarrow P = X * Y$$

We omit the corresponding rules to assert the truth value of the negation of these fluents for brevity.

The initial Kripke structure could be generated in the same way as in the previous section, using the rules (3)-(6). Here, we simplify these rules to the following rules:

$$r(s, num(X, Y), num(X1, Y1), 0) \leftarrow X + Y = X1 + Y1,$$
$$state(num(X, Y), 0), state(num(X1, Y1), 0)$$
$$r(p, num(X, Y), num(X1, Y1), 0) \leftarrow X * Y = X1 * Y1,$$
$$state(num(X, Y), 0), state(num(X1, Y1), 0)$$

As before, we assume that the real state of the world is denoted by:

$$1\{real(S, 0) : state(S, 0)\}1$$

Finally, we need to capture the property that an agent knows the value of the two numbers—i.e., the agent can see only one possible state. We will define this as:

$$s\_knows \equiv \bigvee_{(x,y)\in\mathcal{S}} \mathbf{K}_s(x(X) \wedge y(Y))$$

for the s agent and

$$p\_knows \equiv \bigvee_{(x,y)\in\mathcal{S}} \mathbf{K}_p(x(X) \wedge y(Y))$$

for the p agent.

To complete our representation, we need to represent the sequence of communications between the agents in this prob-

lem. As we have alluded to it before, there is a slight difference between the sequence of exchanges in this problem and those of the muddy children problem. In the case of the muddy children problem, the announcement are encoded as assertions on the current Kripke structure; in the case of the sum and product problem, some of the announcements are referred to a Kripke structure that is independently seen by the agents (i.e., the original Kripke structure). We therefore generalize the action of announcing a fact by attaching to it a time reference. We will use an announcement of the form

$$announce\_fact(\varphi, T)$$

to indicate a public announcement referring to the Kripke structure at the time step T. States not satisfying φ at step T are not acceptable in the Kripke structure at the current time.

The handling of the new type of announcements is encoded as follows:

$$state(S, T+1) \leftarrow state(S, T),$$
$$occ(announce\_fact(F, T_1), T), h(F, S, T_1)$$
$$r(A, S_1, S_2, T+1) \leftarrow state(S_1, T+1), state(S_2, T+1),$$
$$occ(announce\_fact(F, T_1), T), r(A, S_1, S_2, T)$$

These two rules are similar to the rules (34)-(35).

The history of the sum and product problem is then reduced to the following occurrence facts:

$$occ(announce\_fact(0, \neg p\_knows), 0)$$
$$occ(announce\_fact(0, k(s, p\_knows)), 1)$$
$$occ(announce\_fact(2, p\_knows), 2)$$
$$occ(announce\_fact(3, s\_knows), 3)$$

It can be shown that the program Γ has a single answer set, which contains the fact $real(num(4, 13), 0)$. This implies that $(4, 13)$ is a solution of the problem.

7. DISCUSSION AND CONCLUSION

In this paper, we used techniques from reasoning about actions and answer set programming to model multi-agent systems involving agents with knowledge about other agents' knowledge. Since answer set programming (ASP) is a logic as well as a programming paradigm, we were able to formally state and prove (proofs have not been included due to space limitations) the correctness of our executable encodings. We illustrated our approach with respect to two well-known examples: the muddy children problem and the sum-and-product problem. Our approach led to two new aspects: (i) Our ASP specification about the initial Kripke model is able to generate an initial Kripke structure; and (ii) We formulated a new kind of action which we called "ask and truthfully answer" with which we could encode the reasoning done by the children in the muddy children scenario. Both aspects are valuable when planning as one needs to start with the initial "state" and deal with actions such as "ask and truthfully answer" which are similar to sensing actions studied in the past. In addition, our formulation can identify the real state of the world given a history of action occurrences (using a program similar to $\Pi(2)$ or Γ).

In terms of related work, reasoning about agents' knowledge using Kripke structures has been extensively studied in the literature (e.g., [2, 1, 5, 15]). Our logic programming encoding follows the general idea of refining Kripke structure based on public announcements (e.g., as in [2]). In essence, we use ASP to encode the transition between Kripke models defined as follows:

Let $M = (\mathcal{S}, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$ be a Kripke structure, s_0 be the real state of the world, and α be an announcing action. The Kripke structure, $M' = (\mathcal{S}', \pi, \mathcal{K}'_1, \ldots, \mathcal{K}'_n)$, resulting from the execution of an announcement can be computed as follows:

- For $A = announce\_fact(\varphi, \_)$ where φ is a fluent formula:
  - if $s_0 \not\models \varphi$ then M' is undefined; and
  - if $s_0 \models \varphi$ then $\mathcal{S}' = \{s \mid s \in S, s \models \varphi\}$, $\mathcal{K}'_A = \{(s, s') \mid (s, s') \in \mathcal{K}_A, s \in \mathcal{S}', s' \in \mathcal{S}'\}$.
- For $A = announce\_k(\varphi, \_)$ where φ is a knowledge formula, then $\mathcal{S}' = \mathcal{S}$, $\mathcal{K}'_A = \mathcal{K}_A \setminus \{(s_0, s') \mid (M, s') \not\models \varphi\}$.

Overall, the novelty of our work is in terms of the use of ASP and reasoning about action techniques, in writing formulations that computes the initial Kripke model, and in formulating the action "ask and truthfully answer".

8. REFERENCES

[1] A. Baltag, L.S. Moss. Logics for Epistemic Programs. *Synthese*, 139(2):165–224, 2004.

[2] A. Baltag, L.S. Moss, S. Solecki. The logic of public announcements, common knowledge, and private suspicions. *7th TARK*, 1998.

[3] C. Baral. Knowledge representation, reasoning and declarative problem solving. Cambridge U Press. 2003.

[4] C. Baral, E. Pontelli, T. Son. Modeling multi-agent domains in an action language. In *LPNMR*, 2009.

[5] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT press, 1995.

[6] M. Gelfond, V. Lifschitz. The Stable Model Semantics for Logic Programming. ICLP/SLP 1988:1070-1080

[7] J. Gerbrandy, W. Groeneveld. Reasoning about Information Change. Journal of logic, language and information, 1997, 6, 147-169.

[8] H. Ghaderi, H.J. Levesque, Y. Lespèrance. Towards a logical theory of coordination and joint ability. In *AAMAS*, pp. 81, 2007.

[9] J. Y. Halpern. A theory of knowledge and ignorance for many agents. *J. Log. Comput.*, 7(1):79–108, 1997.

[10] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm*, Springer, 1999.

[11] J.-J.Ch. Meyer. Dynamic logic for reasoning about actions and agents. *Logic Based AI*, 281–314. 2000.

[12] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *AMAI*, 25(3,4):241–273, 1999.

[13] S. Shapiro, Y. Lespèrance, H.J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. *AAMAS*, 2002.

[14] H. P. van Ditmarsch, W. van der Hoek and B. P. Kooi. Dynamic epistemic logic with assignment. AAMAS'2005.

[15] H. van Ditmarsch, W. van der Hoek, B. Kooi. *Dynamic Epistemic Logic*. Springer Verlag, 2008.