# Knowledge of Other Agents and Communicative Actions in the Fluent Calculus

**Yves Martin**
Fakultät Informatik
Technische Universität Dresden
01062 Dresden (Germany)
ym1@inf.tu-dresden.de

**Iman Narasamdya**
Department of Computer Science
The University of Manchester
M13 9PL Manchester (UK)
in@cs.man.ac.uk

**Michael Thielscher**
Fakultät Informatik
Technische Universität Dresden
01062 Dresden (Germany)
mit@inf.tu-dresden.de

**Abstract**

The Fluent Calculus has largely been focused on building agents that work individually. However, agents often need to interact with each other to learn more about their environment as well as to achieve their goals. One form of interaction is by means of communication. Effective, goal–oriented communication requires knowledge of other agents. This paper studies the problem of endowing agents with the ability to reason about the knowledge of other agents and with communication skills. Our formalism for the knowledge of other agents generalizes the existing notion of knowledge in the Fluent Calculus. Communication is treated as actions which are called communicative actions. The specification of communicative actions is based on the formalism for the knowledge of other agents. We have also developed an implementation of the theory as an extension to FLUX, which is a programming method that allows to design intelligent agents based on the Fluent Calculus.

## INTRODUCTION

Agents that are able to act autonomously under incomplete information in dynamically changing environments must maintain a representation of their surroundings. Then, using their reasoning capabilities, these agents can draw inferences on the basis of the knowledge that they have. Most of the work so far on the theory and implementation of logic-based agents has been concerned with single agents.

A first approach for agents to treat communication as actions in the context of reasoning about actions was introduced by Cohen and Perrault in (Cohen & Perrault 1979). The formalism chosen in their paper is the STRIPS notation, and they do not consider agents in a multi-agent setting—rather they take only one single system consisting of many agents.

Another approach to communication in multi-agent systems is based on the agent-programming language GOLOG, which is rooted in the logical theory of action of the Situation Calculus (Lespérance *et al.* 1995; Shapiro, Lespérance, & Levesque 1997). However, there are several restrictions to the method described in these papers. In (Lespérance *et al.* 1995), the individual agents have no information about the executed actions of other agents. As a consequence, each agent has to assume a possibly infinite number of exogenous actions which have been performed and could potentially affect any fluent. This could lead to a complete loss of knowledge about the world, and therefore the approach can only be applied to specially designed domains. This deficiency was eliminated by the approach described in (Shapiro, Lespérance, & Levesque 1997). Nevertheless, the implementation of (Shapiro, Lespérance, & Levesque 1997) does not allow for systematic knowledge about other agents. In turn, it is not possible to have separate programs for the individual agents because it is not clear how to deal with queries about the mental states of agents. Moreover, the GOLOG implementation used in (Lespérance *et al.* 1995; Shapiro, Lespérance, & Levesque 1997), employs regression to infer the knowledge state of an agent. For longer communication sequences, this entails that the knowledge has to be regressed over the whole sequence back to the initial situation. The effort for doing this will increase with each new communicative action.

In this paper, we extend the theory for reasoning about action and change of the Fluent Calculus to deal with agents in multi-agent settings. We also show how this theory can be effectively implemented in the high-level agent programming language FLUX. A major motivation for this work is to prepare logically reasoning agents for the Semantic Web. For example, imagine a software agent which goes shopping on the Internet on behalf of its user. The goal of this agent is to buy all required items with a minimal amount of money. In order to achieve this goal, this shopping agent may have to communicate with other agents which manage the virtual stores. Of course, the communication will be much more effective if the shopping agent has knowledge of these selling agents. For example, knowing that some agent sells only electronic devices will prevent our shopping agent from asking for books in this virtual store. With our method, the shopping agent can build up more and more knowledge of others each time he communicates. For subsequent communications, he would then become better in asking the right questions to the right agents.

The approach described in this paper shows how to extend the knowledge of an agent in the Fluent Calculus to contain information of arbitrarily many other agents. Our axiomatization is proved to be sound wrt. basic properties of knowledge. These properties are also shown to be valid

for knowledge updates for both "regular" actions as well as knowledge-producing (i.e., sensing) actions. Based on the knowledge of other agents, we develop a set of communicative actions which allow agents to ask questions, to provide information, or to request the execution of actions from each other. Knowledge of other agents helps eliminate unnecessary communication. For example, if one agent wants to know a property and knows that another agent knows it, then the former should ask the latter about this property. Having this knowledge of the other agent will be defined as a precondition for querying the agent.

In addition to the theoretical framework, we have developed an implementation of knowledge about other agents and the communicative actions in the high-level programming language FLUX. Using the paradigm of constraint logic programming and the notion of incomplete states, we show how knowledge about other agents can be represented in a succinct way within a single knowledge state. This leads to a very nice computational behavior as all the knowledge is immediately available for checking action preconditions and other conditions that are of interest to the agent (see also (Thielscher 2004)).

Our initial treatment of agents in multi-agent setting rests on the following assumptions, which should be relaxed in future extensions of this work: 1. Agents are homogeneous, i.e., share the same ontology, actions and communication protocol. 2. We are dealing with knowledge, thus ignoring the problem of (possibly incorrect) beliefs. We therefore assume that all actions are public. 3. Actions affecting the shared fluents are assumed not to be executed concurrently.

After this introduction, we briefly recapitulate the essentials of the Fluent Calculus and FLUX. Following the recapitulation, we present an extension of the calculus that allows agents to represent and reason about the knowledge of other agents. Afterwards, we give an axiomatization of actions for agent communication based on agent knowledge. The implementation of our theoretical framework in the agent programming language FLUX is described subsequently. We conclude by summarizing the main features of our approach and showing some paths for future research. All programs are available on our web site: `http://www.fluxagent.org`.

## THE FLUENT CALCULUS

The Fluent Calculus shares with the standard Situation Calculus the basic notion of a *situation*. The initial situation is usually denoted by the constant $S_0$. The function $Do(a, s)$ denotes the situation which is reached by performing action $a$ in situation $s$. In order to specify what holds in a situation, the expression $Holds(f, s)$ is used, where $f$ is a *fluent* (i.e., term of sort FLUENT); e.g.,

$$\begin{array}{l} Holds(OnTable(Book), S_0) \wedge \\ (\forall x)\, \neg Holds(Carrying(Agent, x), S_0) \end{array} \quad (1)$$

The Fluent Calculus extends the Situation Calculus by the notion of a *state*. The function $State(s)$ denotes the state (of the environment of an agent) in situation $s$. By definition, every FLUENT term is a state (i.e., term of sort STATE), and if $z_1$ and $z_2$ are states then so is $z_1 \circ z_2$. The *foundational axioms* $\Sigma_{state}$ of the Fluent Calculus stipulate that function "$\circ$" shares essential properties with the union operation for sets (see, e.g., (Thielscher 2001) for details). This allows the definition of the *Holds*-expression as a mere macro thus:

$$Holds(f, s) \stackrel{\text{def}}{=} Holds(f, State(s)) \quad \text{and}$$
$$Holds(f, z) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z'$$

With this, specification (1) entails the following equation for $State(S_0)$:

$$\begin{array}{l} (\exists z)\, (State(S_0) = OnTable(Book) \circ z \wedge \\ \qquad (\forall x)\, \neg Holds(Carrying(Agent, x), z)) \end{array} \quad (2)$$

Based on the notion of a state, the frame problem is solved in the Fluent Calculus by *state update axioms*, which define the effects of an action $a$ as the difference between some $State(s)$ and the successor $State(Do(a, s))$; e.g.,

$$\begin{array}{l} Poss(Pickup(x, y), s) \supset \\ \quad State(Do(Pickup(x, y), s)) = \\ \qquad (State(s) - OnTable(y)) + Carrying(x, y) \end{array} \quad (3)$$

The standard predicate $Poss(a, s)$ means that action $a$ is possible in situation $s$. The functions "$-$" and "$+$" denote, respectively, removal and addition of fluents to states. They have a purely axiomatic characterization in the Fluent Calculus (we again refer to (Thielscher 2001)). For example, tacitly assuming $Poss(Pickup(Agent, Book), S_0)$ and uniqueness-of-names for fluents *OnTable* and *Carrying*, the instance $\{x/Agent, y/Book, s/S_0\}$ of the state update axiom just mentioned applied to equation (2) yields, with the help of the foundational axioms,

$$\begin{array}{l} (\exists z)\, (\ State(Do(Pickup(Agent, Book), S_0)) = \\ \qquad Carrying(Agent, Book) \circ z \wedge \\ \qquad \neg Holds(OnTable(Book), z) \wedge \\ \qquad (\forall x)\, \neg Holds(Carrying(Agent, x), z)\ ) \end{array}$$

### Representing State Knowledge

The knowledge an agent has of its environment can be represented in the Fluent Calculus using the notion of *possible states*. Let the predicate $KState(s, z)$ denote that, according to the knowledge of the agent, $z$ is a possible state in situation $s$. The following axiom, for example, says implicitly that in the initial situation all that is known is that the agent does not hold any object:

$$\begin{array}{l} (\forall z)\, (KState(S_0, z) \equiv \\ \quad (\forall x)\, \neg Holds(Carrying(Agent, x), z)) \end{array}$$

Formally, a property is defined to be known in a situation if it holds in all possible states:

$$Knows(f, s) \stackrel{\text{def}}{=} (\forall z)(KState(s, z) \supset Holds(f, z)) \quad (4)$$

The effects of actions, including knowledge-producing actions, are specified by *knowledge update axioms*, which relate the possible states between successive situations; e.g.,

$$\begin{array}{l} Poss(SenseOnTable(x), s) \supset \\ \quad (\ KState(Do(SenseOnTable(x), s), z) \equiv \\ \qquad KState(s, z) \wedge \\ \qquad [Holds(OnTable(x), z) \equiv Holds(OnTable(x), s)]\ ) \end{array}$$

That is to say, a state $z$ is still possible after $SenseOnTable(x)$ just in case it was possible beforehand and $OnTable(x)$ holds in $z$ iff it holds in the actual $State(s)$.

## FLUX

The Fluent Calculus provides the formal underpinnings of the logic programming method FLUX, whose purpose is to design agents that reason about their actions and sensor information in the presence of incomplete knowledge (Thielscher 2004). FLUX is based on the representation of knowledge states of agents by *open-ended lists* of fluent terms `Z=[F1,...,Fk|_]` accompanied by *constraints* which encode negative and disjunctive information:

| Constraints | Semantics |
|---|---|
| `not_holds(F,Z)` | $\neg \mathit{Holds}(f, z)$ |
| `not_holds_all(F,Z)` | $(\forall \vec{x})\, \neg \mathit{Holds}(f, z)$ <br> $\vec{x}$ variables in $f$ |
| `or_holds([F1,...,Fn],Z)` | $\bigvee_{i=1}^{n} \mathit{Holds}(f_i, z)$ |

As an example, this is a FLUX encoding of the knowledge state which corresponds to the Fluent Calculus axiom (2):

```
Z0 = [on_table(book)|Z],
not_holds_all(carrying(agent,X),Z)
```

The agent infers knowledge of a particular property in such a knowledge state by examining whether the negation of the property is unsatisfiable under the given constraints. To this end, a system of Constraint Handling Rules [1] has been defined in (Thielscher 2004) by which a set of FLUX constraints is solved in accordance with the foundational axioms of the Fluent Calculus.

Agent programs in FLUX are constraint logic programs consisting of three components $P_{kernel} \cup P_{domain} \cup P_{strategy}$. The domain-independent $P_{kernel}$ provides an encoding of the foundational axioms and macros of the Fluent Calculus, including a definition of how incomplete states are updated according to positive and negative effects. The environment of an agent is specified in $P_{domain}$, which in particular contains the precondition and knowledge update axioms for the possible actions of the agent. Finally, $P_{strategy}$ describes the task-oriented behavior of the agent, according to which it reasons, plans, and executes actions. The semantics of a FLUX program is given as a combination of the Fluent Calculus and the standard semantics of logic programming: The computation tree for $P_{strategy}$ and a given query contains nodes representing the execution of actions. The ordered sequence of these nodes determines a particular situation term. With the help of the Fluent Calculus this situation can be formally verified against desired properties of the agent. For more details on syntax and semantics of FLUX we refer to (Thielscher 2004).

# KNOWLEDGE OF OTHER AGENTS

Environments might contain more than one agent. Instead of building an environment along with the inhabiting agents, we are aiming at building agents that are capable of communicating with each other. The agent we are currently developing will subsequently be called "*our agent*". Knowing what other agents know is important to have efficient communication. For example, our agent will only ask another agent about the truth value of some property, if our agent knows that the other agent knows about the property; whether the property holds or does not hold in the environment.

## Representing Knowledge of Other Agents

Our approach to representing the knowledge of other agents is by treating the knowledge as yet another information of the environment. In general, fluents are atomic components of states that represent some particular information of the environment. Thus, the knowledge of other agents shall be represented using a special fluent, which is called *knowledge fluent*:

$$\mathit{KF} : \text{AGENT} \times \text{STATE} \mapsto \text{FLUENT}$$

This fluent function is added to the basic Fluent Calculus signature. The fluent $\mathit{KF}(r, z_r)$ means that the state $z_r$ is a state of which our agent thinks that agent $r$ might think to be actual.

To give a relation between the possible states of our agent and the possible states of other agents, we introduce a new ternary predicate to the basic Fluent Calculus signature, that is,

$$\mathit{KFState} : \text{STATE} \times \text{AGENT} \times \text{STATE}$$

The relation $\mathit{KFState}(z, r, z_r)$ is meant to hold iff the state $z_r$ is a possible state of agent $r$ given that $z$ is a possible state of our agent. A *knowledge fluent state* is a formula

$$\begin{aligned} &\mathit{KFState}(z, r, z_r) \equiv \\ &\quad \mathit{Holds}(\mathit{KF}(r, z_r), z) \ \wedge \ \Phi(z_r) \wedge \\ &\quad (\forall y, z'_y)\, \neg \mathit{Holds}(\mathit{KF}(y, z'_y), z_r) \end{aligned}$$

The above formula says that the predicate $\mathit{KFState}(z, r, z_r)$ relates the possible state $z_r$ of agent $r$ with the possible state $z$ of our agent. This is denoted by the expression $\mathit{Holds}(\mathit{KF}(r, z_r), z)$ saying that the fluent $\mathit{KF}(r, z_r)$ holds in $z$. Moreover, the state $z_r$ is defined using a state formula $\Phi(z_r)$ [2] and no nested knowledge fluent in the state $z_r$.

The relation $K(r, \phi)$ denotes that agent $r$ knows some property $\phi$. Throughout this paper, properties are associated with fluents. Given a possible state $z$ of our agent, $K(r, \phi)$ holds in $z$ iff $\phi$ holds in all possible states belonging to agent $r$ in the state $z$. On this basis, our agent knows that agent $r$ knows property $\phi$ if the property $K(r, \phi)$ holds in all possible states of our agent:

$$\begin{aligned} &\mathit{Knows}(K(r, \phi), s) \stackrel{\text{def}}{=} \\ &\quad (\forall z)\, (\mathit{KState}(s, z) \supset \\ &\qquad (\forall z_r)(\mathit{KFState}(z, r, z_r) \supset \mathit{Holds}(\phi, z_r))) \end{aligned}$$

where the property $\phi$ does not contain any relation of the form $K(r', \phi')$.

**Example 1** Consider a very simple world with only two fluents $F$ and $G$. Suppose that in the initial situation $S_0$, our agent knows that agent $R$ knows that fluent $G$ is true (or

[1] Constraint Handling Rules (CHRs) are a general method of specifying, in a declarative way, rules to process constraints (Frühwrith 1998).

[2] A *state formula* $\Phi(z)$ is a first-order formula with free state variable $z$ and without any occurrences of states other than in expressions of the form $\mathit{Holds}(f, z)$.

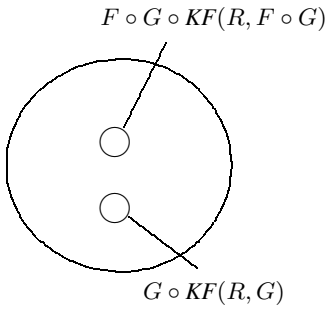


Figure 1: Our agent knows $K(R,G) \wedge (K(R,F) \vee K(R,\neg F))$ in the initial situation $S_0$.

agent $r$ knows $G$). In addition, our agent knows that agent $R$ knows $F$ or agent $R$ knows $\neg F$. The following describes the knowledge state of our agent:

$$\begin{aligned} &\mathit{KState}(S_0, z) \equiv \\ &\quad z = F \circ G \circ \mathit{KF}(R, F \circ G) \vee \\ &\quad z = G \circ \mathit{KF}(R, G) \end{aligned} \tag{5}$$

The above formula is described pictorially in Figure 1. The small circles in the figure denote the possible states of our agent. Our agent knows that agent $R$ knows $G$ because fluent $G$ holds in all knowledge fluents, or formally,

$$\begin{aligned} &(\forall z)(\mathit{KState}(S_0, z) \supset \\ &\quad (\forall z_r)(\mathit{KFState}(z, R, z_r) \supset \mathit{Holds}(G, z_r))) \end{aligned}$$

In one possible state, fluent $F$ holds in the states of all knowledge fluents of agent $R$, but in the other, fluent $F$ does not hold in any states of knowledge fluents of agent $R$. This denotes that our agent has disjunctive knowledge $K(R,F) \vee K(R,\neg F)$. Fluents that hold in the possible states of our agent are determined by Corollary 1, which will be explained later. □

In the presence of many agents, the universal validity of knowledge has to be preserved. For example, it would be inconsistent if our agent knows that agent $r$ knows that property $\phi$ is true, but our agent itself knows that property $\phi$ is false. The validity of knowledge is captured by the following foundational axioms $\Sigma_{knows}$:

$$\mathit{KState}(s, \mathit{State}(s))$$

$$\mathit{KFState}(z, r, z^-)$$

where $z^-$ is defined using the axiom of state existence $(\forall P)(\exists z)(\forall f)(\mathit{Holds}(f, z) \equiv P(f))$ [3] with the predicate variable $P$ substituted by $\{P/\lambda f.\mathit{Holds}(f, z) \wedge (\forall x, z')(f \neq \mathit{KF}(x, z'))\}$. The first axiom, which is inherited from (Thielscher 2000), says that the actual state is always possible in any situation. The second axiom says that a possible state of our agent, without any knowledge fluent, is also a possible state of any other agent. From those axioms, the following corollary can be derived:

[3]This axiom is a foundational axiom in the Fluent Calculus which stipulates a state for any combination of fluents (Thielscher 2001).

**Corollary 1** *Let $f$ be a fluent and $s$ be a situation, then $\Sigma_{state} \cup \Sigma_{knows}$ entails*

$$\begin{aligned} (\exists r_1)\mathit{Knows}(K(r_1, \phi), s) \supset \\ (\forall r_2)(\neg \mathit{Knows}(K(r_2, \neg\phi), s)) \wedge \\ \mathit{Knows}(\phi, s) \wedge \neg \mathit{Knows}(\neg\phi, s) \wedge \\ \mathit{Holds}(\phi, s) \end{aligned}$$

The above corollary says that if there is another agent of which our agent knows that this other agent knows $\phi$, then our agent knows $\phi$ too, but does not know $\neg\phi$. Moreover, there cannot be any agent $r_2$ of which our agent knows that agent $r_2$ knows $\neg\phi$. Finally, property $\phi$ itself must be true in the environment (i.e., the actual state).

**Example 1 (continued)** Equation (5) satisfies the foundational axioms. In accordance with Corollary 1, if our agent knows $K(R,G)$, then our agent knows $G$ too. The knowledge state in equation (5) shows that fluent $G$ holds in all possible states of our agent. Therefore, our agent knows $G$. □

## Knowledge Update Axioms

Knowledge of other agents might have to be updated as well upon updating knowledge states. For example, suppose that our agent knows $K(R,G)$. Our agent performs an action that makes fluent $G$ become false. Without updating the knowledge of agent $R$, our agent's knowledge becomes inconsistent with the foundational axioms $\Sigma_{knows}$, that is, our agent knows $\neg G \wedge K(R,G)$ (recall Corollary 1). To define knowledge update axioms appropriately, the approach taken in this paper is based on the following assumptions:

- Agents are homogenous. It means that these agents have the same set of actions, and those actions are axiomatized in the same way, in terms of their preconditions and update actions. Moreover, these agents share the same ontology and communication protocol.
- The physical effects of actions to the environment are observable by every agent inhabiting the environment.

The first assumption makes preconditions and knowledge update axioms simple to axiomatize, since the precondition and effects of every action do not have to be quantified over agents. The second assumption is needed because our agent inhabits dynamic environments which include other active entities, that is, its fellow agents. Consequently, some state properties are not under the sole control of our agent. Therefore, our agent must take into account actions besides its own when maintaining the state of the environment. One way of addressing this problem is by treating some actions as *exogenous actions* [4].

Updating knowledge of other agents involves removal and addition of infinitely many knowledge fluents to knowledge states. However, the removal operation "$-$", explained in the previous section, was only defined for the subtraction of finite states (Thielscher 2001). In this paper, we introduce function "$\ominus$" generalizing "$-$" to the subtraction of infinite states, that is, $z_1 \ominus \tau = z_2$ where $z_2$ is defined using the

[4]Exogenous actions are actions which are not performed by our agent but do affect some relevant fluents.

axiom of state existence with the predicate variable $P$ substituted by $\{P/\lambda f.\mathit{Holds}(f, z_1) \wedge \neg \mathit{Holds}(f, \tau)\}$.

Knowledge update axioms can now be formally defined. The following definition generalizes the definition in (Thielscher 2004), in the sense that the knowledge of other agents is now taken into consideration.

**Definition 2** A *knowledge update axiom* for action $A(\vec{x})$ is a formula

$$\begin{array}{l}\mathit{Knows}(\mathit{Poss}(A(\vec{x})), s) \supset \\ \quad (\exists \vec{y})(\forall z')(\exists z^*)\,(\,\mathit{KState}(\mathit{Do}(A(\vec{x}), s), z') \equiv \\ \qquad\qquad (\exists z)\,(\mathit{KState}(s, z) \wedge \Psi(z^*, z)) \\ \qquad\qquad \wedge\, \Pi(z', z^*, \mathit{Do}(A(\vec{x}), s))\,)\end{array}$$

where

- the *physical effect* $\Psi(z^*, z)$ is of the form

$$\bigvee_{i=1}^{n} (\exists \vec{y}_i)\,(\Phi_i(z) \wedge z^* = z \ominus \widehat{\vartheta}_i^- + \widehat{\vartheta}_i^+)$$

  where
  - $\Phi_i(z)$ is a state formula;
  - $\widehat{\vartheta}_i^- = \vartheta^- \circ z_{kf}^-$, where $\vartheta^-$ contains the negative physical effect and $z_{kf}^-$ contains all knowledge fluents in $z$; and
  - $\widehat{\vartheta}_i^+ = \vartheta^+ \circ z_{kf}^+$, where $\vartheta^+$ contains the positive physical effect and $z_{kf}^+$ contains all knowledge fluents in $z_{kf}^-$ whose states have been updated according to the physical effects $\vartheta^-$ and $\vartheta^+$.
- the *cognitive effect* $\Pi(z', z^*, \mathit{Do}(A(\vec{x}), s))$ is of the form

$$\begin{array}{l}\bigwedge_{i=1}^{k}[\,\Pi_i(z^*) \equiv \Pi_i(\mathit{State}(\mathit{Do}(A(\vec{x}), s)))\,] \\ \quad\wedge \\ \bigwedge_{i=1}^{l} \mathit{Holds}(F_i(\vec{t_i}), z^*) \wedge \mathit{Holds}(F_i(\vec{t_i}), \mathit{Do}(A(\vec{x}), s)) \\ \quad\wedge \\ (z' = z^* \ominus \vartheta^* \vee z' = z^*)\end{array}$$

  where $\Pi_i(z^*)$ is a state formula and $\vartheta^*$ contains all knowledge fluents of some agent in $z^*$ whose states do not conform to $\Pi_i(z^*)$ or fluent $F_i(\vec{t_i})$ does not hold therein.

□

Note that the physical effect is modelled by updating the possible states of our agent *including* the states of the knowledge fluents. The cognitive effect is modelled by constraining the set of possible states so as to agree with the actual state on the sensed properties and fluent values. Usually, actions with no physical effect do not involve any knowledge fluent, although they might affect the knowledge of other agents. For example, in sensing its own location, our agent gets to know its location regardless of whether his fellow agents get to know his location or not. Sensing actions usually do not have any physical effect and do not involve any knowledge fluent, but they might affect the knowledge of other agents (see the following for an example). In particular, communication does not give any physical effect to the environment, but its cognitive effect might involve removal of some knowledge fluents of some agent from the possible states of our agent. This case is handled by the last conjunct of the cognitive effect part of the above definition.

**Example 1 (continued)** Consider again Formula (5). Suppose our agent performs action *MakeGFalse* whose negative physical effect involves fluent $G$. According to our assumptions, agent $R$ can observe the action, so that our agent is obliged to update the knowledge fluent states of agent $R$. The following is the precondition and knowledge update axiom of action *MakeGFalse*:

$$\mathit{Poss}(\mathit{MakeGFalse}, z) \equiv \top$$

$$\begin{array}{l}\mathit{Knows}(\mathit{Poss}(\mathit{MakeGFalse}), s) \supset \\ \quad (\forall z')\,(\,\mathit{KState}(\mathit{Do}(\mathit{MakeGFalse}, s), z') \equiv \\ \qquad (\exists z)\,(\mathit{KState}(s, z) \wedge z' = z \ominus \widehat{\vartheta}^- + \widehat{\vartheta}^+))\end{array}$$

where $\widehat{\vartheta}^-$ and $\widehat{\vartheta}^+$ correspond to those in Definition 2 with $\vartheta^- = G$ and $\vartheta^+ = \emptyset$. Thus, after having performed action *MakeGFalse*, the knowledge state of our agent is as follows:

$$\mathit{KState}(S_1, z) \equiv z = F \circ \mathit{KF}(R, F) \vee z = \mathit{KF}(R, \emptyset)$$

where $S_1 = \mathit{Do}(\mathit{MakeGFalse}, S_0)$. The above knowledge state says that our agent knows $\neg G \wedge K(R, \neg G)$ in $S_1$. The knowledge $K(R, F) \vee K(R, \neg F)$ remains since the action did not affect fluent $F$.

Afterwards, our agent performs action *SenseF* which is used to sense whether fluent $F$ is true or not in the environment. The action is defined as follows:

$$\mathit{Poss}(\mathit{SenseF}, z) \equiv \top$$

$$\begin{array}{l}\mathit{Knows}(\mathit{Poss}(\mathit{SenseF}), s) \supset \\ \quad (\forall z')\,(\,\mathit{KState}(\mathit{Do}(\mathit{SenseF}, s), z') \equiv \\ \qquad (\exists z)\,(\mathit{KState}(s, z) \wedge z' = z) \wedge \\ \qquad [\Pi_F(z') \equiv \Pi_F(\mathit{State}(\mathit{Do}(\mathit{SenseF}, s)))])\end{array}$$

where $\Pi_F(z) \stackrel{\text{def}}{=} \mathit{Holds}(F, z)$. Suppose, for the sake of argument, that fluent $F$ is actually true, then the knowledge state of our agent becomes as follows:

$$\mathit{KState}(S_2, z) \equiv z = F \circ \mathit{KF}(R, F)$$

where $S_2 = \mathit{Do}(\mathit{SenseF}, S_1)$. The disjunctive knowledge has now vanished, that is, our agent knows $K(R, F)$. This shows that actions having no physical effect might affect the knowledge of other agents. The evolution of the set of possible states is depicted in Figure 2. □

## COMMUNICATIVE ACTIONS

The approach to modelling the communication process is to treat communication itself as constituted by actions. This approach is in the spirit of the formal theory of *speech acts* (Austin 1962; Searle 1969). This theory treats communication as actions that might alter the knowledge of the communication participants. The actions used for the communication are called *communicative actions*. The specification of the actions will benefit greatly from the formalism for the knowledge of other agents.

There are four types of communicative actions developed here. The first type is *ask action*. Communicative actions of this type are used to get some knowledge from other agents. Action function *AskIf* with signature

$$\mathit{AskIf}: \quad \text{AGENT} \times \text{FLUENT} \mapsto \text{ACTION}$$

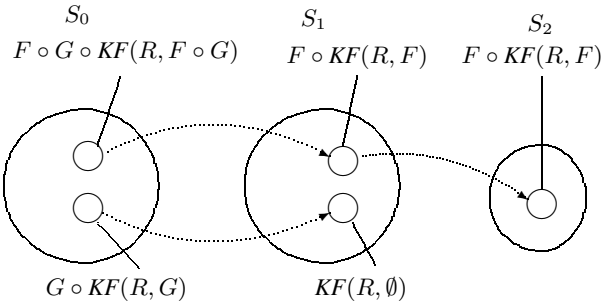


Figure 2: The evolution of the set of possible states while performing action *MakeGFalse* followed by action *SenseF*. In the initial situation $S_0$, our agent knows $K(R, G) \wedge (K(R, F) \vee K(R, \neg F))$.

is a function to construct ask actions that are used to ask another agent about the truth values of fluents. Suppose, for example, the fluent $On(A, B)$ denotes the condition that block $A$ is on block $B$. The action $AskIf(R, On(A, B))$ is meant to ask agent $R$ if block $A$ is on block $B$. Another ask action is the action scheme $AskVal\mathbf{F}(r, \vec{p_1}, \vec{p_2}, \vec{v_2})$ which is used to ask the values of the arguments of fluent $\mathbf{F}$. The arguments in question are denoted by positions $\vec{p_1}$. The arguments whose values $\vec{v_2}$ are known, are denoted by positions $\vec{p_2}$. For example, the action $AskValOn(R, [1], [2], [B])$ is meant to ask agent $R$ which block is on block $B$. For convenience, this action is simply written as $AskValOn(R, [x], On(x, B))$.

The second type of communicative actions is *tell action*. Communicative actions of this type are used to reply to those of type ask action. Action function *TellIf* with signature

$$TellIf: \quad \text{AGENT} \times \text{FLUENT} \times \{-1, 0, 1\} \mapsto \text{ACTION}$$

is a function to construct tell actions that are used to tell another agent about the status of fluents. The value 1 means that the fluent in question holds, value 0 means that the fluent does not hold, otherwise value $-1$ denotes that the teller does not know the status of the fluent. To reply to the action $AskVal\mathbf{F}(r, \vec{p_1}, \vec{p_2}, \vec{v_2})$, there is a scheme of tell actions $TellVal\mathbf{F}(r, \vec{p_1}, \vec{p_2}, \vec{v_2}, \vec{v_1})$, where $\vec{v_1}$ contains all possible values for the arguments $\vec{p_1}$. For example, the action $TellValOn(R, [1, 2], [], [], [[A, B], [B, C]])$ is meant to tell agent $R$ that block $A$ is on block $B$, which in turn is on block $C$. For clarity, this action shall be written as

$$TellValOn(R, [x, y], On(x, y), On(A, B) \circ On(B, C))$$

To achieve its goal, our agent may need some help from other agents, that is, to perform actions which our agent is unable to perform. The action $Request(r, a)$ is a communicative action of type *request* that is used to request agent $r$ to perform action $a$. The last communicative action is the action *Listen* of type *listen*. Since all communicative actions discussed so far have no physical effect, they are unobservable by other agents in the environment. Thus, the action *Listen* is needed by our agent to listen to other communicative actions directed to him.

The following exemplifies how the specification of communicative actions benefit from the formalism for the knowledge of other agent. Our agent shall ask agent $r$ about the status of fluent $f$ only if our agent does not know the status of fluent $f$ and our agent knows that agent $r$ knows that fluent $f$ holds or knows that fluent $f$ does not hold. This is formalized as the precondition of action $AskIf(r, f)$:

$$\begin{aligned} &Poss(AskIf(r, f), s) \equiv \\ &\quad \neg Knows(f, s) \wedge \neg Knows(\neg f, s) \wedge \\ &\quad Knows(K(r, f) \vee K(r, \neg f), s) \end{aligned}$$

The above precondition axiom shows that, to ask agent $r$ about the status of fluent $f$, our agent must infer the knowledge of agent $r$ about fluent $f$. This results in an efficient communication, in the sense that our agent will not ask every agent about the status of fluent $f$, but only those who know the status of the fluent.

Our agent has to follow some rules while it is communicating with other agents. A protocol specifies the rules of encounter governing a dialogue between agents. We develop a simple binary protocol involving the communicative actions that we have already specified. A communication protocol is a set of rules determining the would–be performed actions for each communicative action. Here, we are aiming at developing honest and sincere agents. Thus, the protocol will also reflect the honesty and sincerity of our agent. For example, the action $AskIf(r_2, f)$ performed by agent $r_1$ is replied by agent $r_2$ with the action $TellIf(r_1, f, v)$, where the value of the variable $v$ depends on the knowledge of agent $r_2$ about fluent $f$.

The semantics of communicative actions defined here has a different perspective from the semantics of standard agent communication languages, such as *FIPA Communicative Acts Language* (FIPA:The Foundation for Intelligent Physical Agents 2000). Our approach to the semantics of the language is purely subjective, while FIPA takes an objective approach. For example, the action *TellIf* defined here corresponds to the primitive action *Inform* defined in the FIPA specification. The postcondition of *TellIf* says that, after telling agent $r$ about some property, our agent gets to know that agent $r$ knows the property. Unlike *TellIf*, the postcondition of action *Inform* is specified as follows: after informing agent $r$ about some property, agent $r$ knows the property. Since FIPA Communicative Acts Language is now a standard for agent communication languages, for future work, we have to build an interface bridging our communicative actions and FIPA's language. The interface will allow our agent to communicate with other agents which are not implemented in FLUX.

## KNOWLEDGE OF OTHER AGENTS IN FLUX

This section presents the implementation of our Fluent Calculus approach to communication in FLUX. First, the encoding of knowledge of other agents is described. This description also shows the semantics of FLUX expressions in terms of knowledge of other agents. Furthermore, an extension to the FLUX constraint system is given to handle

knowledge of other agents. Afterwards, this section discusses how knowledge of other agents is inferred in FLUX. The inference method then enables us to encode the communicative actions developed in this paper. For example, the precondition of action *AskIf*$(r, f)$ requires inferring the knowledge of agent $r$. Finally, the encoding of knowledge update axioms, which respects the knowledge of other agents, is explained.

## Encoding Knowledge of Other Agents in FLUX

To begin with, the following explains the notations that we use in this section. Following the Prolog convention, the fluent variable $f$ in the Fluent Calculus corresponds to the term variable `F` in FLUX. Fluent $F$ in the Fluent Calculus is associated with the fluent term `f` in FLUX. Likewise for variables and constants denoting agents.

We have already mentioned that knowledge states of agents are represented as open-ended lists of fluent terms. A FLUX state is a list `Z=[F1,...,Fk|_]` of pairwise different fluents along with finite number of constraints representing the knowledge that our agent has. In other words, a FLUX state represents all possible knowledge states that our agent has. Therefore, if fluent `f` is among `F1,...,Fk`, then the fluent holds in all possible knowledge states of our agent. Thus, it means that our agent knows $F$.

Encoding the knowledge of other agents is not as straightforward. The problem is that there could be infinitely many knowledge fluents, so that it is impossible to enumerate them. The approach taken here is to encode all knowledge fluents belonging to the same agent as one single knowledge fluent, the second argument of which is a FLUX state. To accommodate this, it is required that, for each agent $r$, there exists at most one fluent `kf(R,ZR)` that holds in the list `Z`. Similar to the list `Z`, the list `ZR` is a FLUX state. The fluent `kf(R,ZR)` is the encoding of all knowledge fluents belonging to agent $r$. Thus, if fluent `F` holds in `ZR`, then the fluent holds in all knowledge fluents of agent $r$. Consequently, it means that our agent knows $K(r, f)$. Moreover, applying state constraints to the list `ZR` gives the following semantics, where each item in the second column denotes the knowledge of agent $r$ that our agent has in some situation:

| Constraints | Semantics |
|---|---|
| `not_holds(F,ZR)` | $K(r, \neg f)$ |
| `or_holds([G1,...,Gm],ZR)` | $K(r, \bigvee_{j=1}^{m>0} g_j)$ |
| `or_holds([k(R,G1),`<br>`...,k(R,Gk),`<br>`k(R,neg(G1)),`<br>`...,k(R,neg(Gm))]` | $\bigvee_{j=1}^{k>0} K(r, g_j)$<br>$\vee$<br>$\bigvee_{j=l}^{m>0} K(r, \neg g_j)$ |

It is worth mentioning that knowledge of the form $\neg K(r, f)$ is not yet expressible in FLUX.

According to the above description, the following is the encoding for Example 1:

```
init(Z0) :-
    Z0 = [kf(r,[g|_]) | _],
    or_holds([k(r,f),k(r,neg(f))],Z0),
    duplicate_free(Z0).
```

```
cons_state(Z) <=> nonvar(Z) | cons_kf(Z,Z).     %1a

cons_kf(Z1,_) <=> var(Z1) | cons_state(Z1).     %2a
cons_kf([F|Z1],Z2) <=>                          %3a
    F\=kf(_,_) | cons_kf(Z1,Z2).
cons_kf([kf(R,ZR)|Z1],Z2) <=>                   %4a
    cons_kf1(ZR,Z2),
    cons_kf(Z1,Z2).

cons_kf1(ZR,Z) <=> var(ZR) | true.              %5a
cons_kf1([F|ZR1],Z) <=>                         %6a
    holds(F,Z), cons_kf1(ZR1,Z).
```

Figure 3: FLUX CHRs for knowledge–consistent state.

The specification of the initial conditions is encoded in FLUX by the definition of the predicate `init(Z0)`. There is only one knowledge fluent `kf(r,ZR)` belonging to agent `r`, and fluent `g` holds in the list `ZR`. This encodes the fact that our agent knows $K(R, G)$. The disjunctive knowledge of agent $R$ about fluent $F$ is encoded using the disjunctive constraint `or_holds([k(r,f),k(r,neg(F))],Z0)`. The constraint will be described in detail later. The auxiliary constraint `duplicate_free(Z0)` ensures that there are no multiple occurrences of fluents in the list `Z0` (Thielscher 2004). In the presence of other agents, the definition of this constraint is extended. The extended definition includes the application of the constraint to the list `ZR` of every knowledge fluent `kf(R,ZR)`. Thus, no multiple occurrences of fluents appear in the list `ZR`. The extended definition of `duplicate_free` moreover includes imposing the constraint `not_holds_all(kf(X,ZX),ZR)` to every knowledge fluent `kf(R,ZR)`. This guarantees that there will be no occurrences of nested knowledge fluents in `ZR`.

At this point, nothing prevents us from constructing a FLUX state that does not obey Corollary 1. For example, the following FLUX state is allowed:

```
Z = [kf(r,[f|ZR]) | _], not_holds(f,Z),
duplicate_free(Z)
```

The above state describes that our agent knows $\neg F$, but knows $K(R, F)$. In what follows, a FLUX state is said to be *knowledge–consistent* if it respects Corollary 1.

To keep FLUX states knowledge–consistent, another auxiliary constraint `cons_state(Z)` is introduced. The constraint is applied to our agent's FLUX state. Figure 3 shows part of the definition of the constraint. Essentially, the constraint examines every knowledge fluent `kf(R,ZR)` in the list `Z` using rules $1a - 4a$. For any fluent `F`, if it holds in the list `ZR`, then the constraint `cons_kf1(ZR,Z)` ensures that the fluent also holds in the list `Z`. This means, whenever our agent knows $K(r, f)$, then he knows $f$ too. This makes the FLUX state obey Corollary 1 for positive knowledge of other agents. Other forms of knowledge of other agents can be treated similarly. Having the problem fixed, the previous encoding of Example 1 is revised to

```
init(Z0) :-
    Z0 = [kf(r,[g|ZR]) | Z],
    or_holds([k(r,f),k(r,neg(f))],Z0),
    cons_state(Z0), duplicate_free(Z0).
```

However, it is inefficient to keep FLUX states knowledge–consistent every time a knowledge assertion occurs. The approach taken here is that the constraint `cons_state` is only applied to the initial state, and it is left to the assertion methods to ensure that assertions do not make the state inconsistent. This means, if our agent asserts that he knows that other agent knows about some property, then our agent has to assert that he knows the property as well. The issue of knowledge–consistency suggests the following FLUX program for asserting the knowledge of other agents:

```
holds_kf(R,F,Z) :-
    holds(kf(R,ZR),Z),
    holds(F,ZR), holds(F,Z).
```

The above program says as follows: upon asserting that agent $r$ knows fluent $f$ using the predicate `holds(F,ZR)`, our agent also asserts that he knows fluent $f$ using the predicate `holds(F,Z)`.

It has been shown already that disjunctive knowledge is handled using the constraint `or_holds([T1,...,Tk],Z)`. The knowledge of other agents is encoded such that some of the `Ti`'s are of the form `k(R,F)` (see the encoding of Example 1 above). However, the current constraint handling rules are not sufficient to resolve the term `k(R,F)` such that the whole FLUX state is knowledge–consistent. Therefore, not only do the current rules have to be modified, but some new rules have to be added to the FLUX constraint solver.

Figure 4 depicts the new rules along with the modification of the existing rules in (Thielscher 2004). There are two additional terms used internally in the encoding of the disjunctive knowledge of other agents. The first additional term is of the form `k(R,F,ZR)`, the meaning of which is the same as the term `k(R,F)`. Moreover, the list `ZR` denotes the knowledge fluent states belonging to agent $r$. The second additional term is of the form `h(R,F)`, which is used to designate that the fluent `F` holds in our agent's FLUX state.

To begin with, variables `F` and `R` denote, respectively, a fluent variable and an agent variable. In addition, variables `Z` and `ZR` denote, respectively, our agent's FLUX state and agent r's FLUX state of knowledge fluent `kf(R,ZR)`. CHR $1b$ simplifies a singleton disjunction of a non–knowledge fluent, that is, $\bigvee_{i=1}^{n=1} \mathit{Holds}(f_i, z) \equiv \mathit{Holds}(f_1, z)$. CHR $2b - 4b$ are used to assert the fluent `F` in the list `Z` if the term `k(R,F,ZR)` is the only remaining disjunct. This reduction is justified by our explanation of knowledge assertion, that is, whenever our agent gets to know $K(r, f)$, then he should know $f$ as well.

CHR $5b$ is used to reduce the term `k(R,F,[G|ZR])` to `k(R,F,ZR)` if the fluent `G` is not equal to the fluent `F` or `neg(F)`. The reduction is useful for the following CHRs. Rule $6b$ says that, having reduced by CHR $5b$, if $g$ is equal to $f$, then the whole disjunction is true. This rule is justified by $\Sigma_{\mathit{state}} \cup \Sigma_{\mathit{knows}}$, which entails

$$\mathit{Knows}(K(r, f), s) \wedge [\mathit{Knows}(K(r, f), s) \vee \Psi] \equiv \mathit{Knows}(K(r, f), s)$$

$$\mathit{Knows}(K(r, f), s) \supset \mathit{Knows}(f, s)$$

In contrast, CHR $7b$ will remove term `k(R,neg(F),ZR)` from the disjunction if `F` holds in `ZR`. Correspondingly, CHR

```
or_holds([F],Z) <=> F\=eq(_,_), F\=k(_,_,_)    %1b
                  | holds(F,Z).
or_holds([D],Z) <=> D=k(R,F,ZR), F\=neg(_)     %2b
                  | holds(F,ZR),holds(F,Z).
or_holds([D1,D2],Z) <=> D2=k(R,F,ZR),          %3b
                        D1=h(R,F), F\=neg(_)
                        | holds(F,ZR).
or_holds([D],Z) <=> D=k(R,F1,ZR), F1=neg(F)    %4b
                  | not_holds(F,ZR), not_holds(F,Z).

or_holds(V,Z) <=> member(k(R,F,ZR),V,V1),      %5b
                  nonvar(ZR),  ZR=[F1|ZR1],
                  F\=F1, F\=neg(F1)
                  | or_holds([k(R,F,ZR1)|V1],Z).

or_holds(V,Z) <=> member(k(R,F,ZR),V),         %6b
                  nonvar(ZR), ZR=[F1|_], \+ F\=F1
                  | ( member(h(R,F),V,V1) -> true
                      ; holds(F,Z) ).
or_holds(V,Z) <=>  member(k(R,F,ZR),V,V1),     %7b
                   nonvar(ZR), ZR=[F1|_],\+ F\=neg(F1)
                   | or_holds(V1,Z).

or_holds(V,Z) <=>  member(h(R,F1),V),          %8b
                   member(k(R,F,ZR),V,V1),
                   \+ F\=neg(F1) | or_holds(V1,Z).
or_holds(V,Z), not_holds(F,ZR) <=>             %9b
         member(k(R,F,ZR1),V,V1), ZR\==Z, ZR==ZR1
         | ( member(h(R,F),V1,V2) -> or_holds(V2,Z)
                                     ; or_holds(V1,Z) ).

not_holds(F, Z) \  or_holds(V, Z) <=>          %10b
                   member(k(R,F1,_), V, W), F1==F,
                   \+ member(h(R,F1),V)
                   | or_holds(W, Z).
not_holds_all(F, Z) \ or_holds(V, Z) <=>       %11b
                   member(k(R,F1,_), V, W),
                   inst(F1, F), \+ member(h(R,F1),V)
                   | or_holds(W, Z).

or_holds(V, W, [F1|Z]) <=>                     %12b
              member(D, V, V1), D=k(R,F), F1=kf(R,ZR)
              | or_holds(V1,[k(R,F,ZR)|W],[F1|Z]).
or_holds(V, W, [F1|Z]) <=>                     %13b
              member(D, V, V1),
              ( D=k(R,F1) ; D=k(R,F1,ZR) ),
              \+ member(h(R,F1),V,_)
              | or_holds(V1,[h(R,F1), D|W],[F1|Z]).
or_holds(V, W, [F1|Z]) <=>                     %14b
              member(D, V, V1),
              ( D=k(R,neg(F1)); D=k(R,neg(F1),_) ),
              | or_holds(V1,W,[F1|Z]).
```

Figure 4: An extension and modifications of FLUX CHRs for disjunction.

$9b$ removes the term `k(R,F,ZR)` from the disjunction if `F` does not hold in `ZR`. These rules are also sanctioned by $\Sigma_{state} \cup \Sigma_{knows}$, which entails

$$\mathit{Knows}(K(r,f),s) \wedge [\mathit{Knows}(K(r,\neg f),s) \vee \Psi] \equiv \mathit{Knows}(K(r,f),s) \wedge \Psi$$

$$\mathit{Knows}(K(r,f),s) \supset \mathit{Knows}(f,s)$$

Rule $8b$ removes the term `k(R,neg(F),ZR)` from the disjunction, but the removal is caused by the term `h(R,F)` denoting that `F` holds in `Z`. This rule is entailed by $\Sigma_{state} \cup \Sigma_{knows}$, which implies

$$\mathit{Knows}(f,s) \wedge [\mathit{Knows}(K(r,\neg f),s) \vee \Psi] \equiv \mathit{Knows}(f,s) \wedge \Psi$$

CHRs $10b - 11b$ are used to remove the term `k(R,F,ZR)` from the disjunction if the fluent `F` does not hold in `Z`. This group of rules is justified by the fact that $\Sigma_{state} \cup \Sigma_{knows}$ entails

$$\mathit{Knows}(\neg f,s) \wedge [\mathit{Knows}(K(r,f),s) \vee \Psi] \equiv \mathit{Knows}(\neg f,s) \wedge \Psi$$

$$(\forall \vec{x})\mathit{Knows}(\neg f_1,s) \wedge [\mathit{Knows}(K(r,f_2),s) \vee \Psi] \equiv (\forall \vec{x})\mathit{Knows}(\neg f_1,s) \wedge \Psi$$

where $\vec{x}$ are the variables of $f_1$ and given that $f_1\theta = f_2$ for some $\theta$.

CHR $12b$ is used to evaluate knowledge fluents against all fluents in `Z`. Upon the evaluation, every term `k(R,F)` is replaced by its ternary form `k(R,F,ZR)` if the encountered fluent is `kf(R,ZR)`. CHR $13b$ is used to tag $\mathit{Knows}(f,s)$ by the term `h(R,F)` if there exists a term `k(R,F)` or its ternary form in the disjunction. Finally, CHR $14b$ denotes the universal property of knowledge, namely, if our agent knows $f$, then he does not know that there is another agent knows $\neg f$.

**Example 2** Suppose our agent knows $G$ and $\neg F$. Moreover, he knows that agent $R$ knows $F$ or knows $G$. For simplicity, the following reflects the state without the constraints `duplicate_free` and `cons_state`:

```
?- Z=[g,kf(r,ZR) | Z1], not_holds(f,Z),
   or_holds([k(r,f),k(r,g)],Z).

ZR = [g | ZR1]
Z1 = Z1
Z  = [g, kf(r,[g|ZR1]) | Z1]

Yes

not_holds(f,Z1)
```

The above derivation shows that the disjunctive constraint is simplified to the fact that our agent knows $K(R,G)$. □

## Inferring Knowledge of Other Agents in FLUX

By definition, our agent knows $K(r,f)$ if $K(r,f)$ holds in every possible state $z$ of our agent. This means that fluent $f$ holds in the state $z_r$ of every knowledge fluent $\mathit{KF}(r,z_r)$ occurring in $z$. However, it is impractical to check whether $K(r,f)$ holds in all possible states of our agent. The approach taken here is adapted from (Thielscher 2004), that is, our agent knows $K(r,f)$ if asserting $K(r,\neg f)$ fails. Formally, suppose that a knowledge state $\mathit{KState}(s,z) \equiv \Phi(z)$ is identified with the state specification $\Phi(z)$. Then our agent knows that agent $r$ knows fluent $f$ iff $\{\Phi(z), \mathit{KFState}(z,r,z_r), \neg\mathit{Holds}(f,z_r)\}$ is unsatisfiable. This can be verified using negation–as–failure to prove that the constraint solver derives an inconsistency upon asserting the state constraint `not_holds(F,ZR)`. The following program exemplifies how $K(R,G)$ is inferred from Example 1:

```
?- init(Z0), holds(kf(r,ZR), Z),
   \+ not_holds(g,ZR).

Yes
```

Disjunctive knowledge of other agents is inferred in a slightly more complicated way. Our agent knows $K(r,\phi_1) \vee K(r,\phi_2)$ if it is not true that there is a possible state $z$ of our agent, in which there are knowledge fluents $\mathit{KF}(r,z_{r1})$ and $\mathit{KF}(r,z_{r2})$ (possibly the same knowledge fluent), such that $\phi_1$ does not hold in $z_{r1}$ and $\phi_2$ does not hold in $z_{r2}$. However, since all agent $r$'s knowledge fluents in the Fluent Calculus are encoded in FLUX as one single knowledge fluent, there is no way of accessing particular knowledge fluents in FLUX. Thus, we cannot use the same approach that we have used beforehand. Fortunately, there is also a feasible alternative to infer such knowledge. The alternative has already been hinted by the correctness of the constraint `or_holds([T1,...,Tk],Z)`. Informally, if our agent knows $K(r,\phi_1) \vee K(r,\phi_2)$, then when it is assumed that our agent additionally knows $\neg\phi_1$, he indeed knows $K(r,\phi_2)$. Likewise, when it is assumed that our agent knows $\neg\phi_2$, then he knows $K(r,\phi_1)$. This approach is justified by Corollary 1. The methods for asserting knowledge $\phi$ and inferring $K(r,\phi)$ have already been shown before. Therefore, it is easy to implement the method for inferring disjunctive knowledge of other agents in FLUX.

**Example 3** The precondition of the action $\mathit{AskIf}(r,f)$ requires inferring disjunctive knowledge of agent $r$. The following shows how the precondition is encoded and its application to Example 1:

```
poss(ask_if(R,F),Z) :-
    \+ (knows(R,F); knows_not(R,F)),
    \+ (\+ (holds(kf(R,ZR), Z),
            holds(F,Z),
            \+ not_holds(F,ZR)),
        \+ (holds(kf(R,ZR),Z),
            not_holds(F,Z),
            \+ holds(F,ZR))).

?- init(Z0), poss(ask_if(r,f),Z0)
Yes
```

□

## Knowledge Update Axioms in FLUX

It has been shown in the previous section that knowledge update axioms must consider the knowledge of other agents as well. This demands that the encoding of knowledge update axioms in (Thielscher 2004) be extended. Firstly, the extension is due to the disjunctive knowledge of other agents. For example, suppose it is specified that our agent has some disjunctive knowledge $\bigvee_{i=1}^{k>1} t_i$, where for some $i$ $(1 \leq i \leq k)$,

```
minus(Z, [], Z).
minus(Z, [F|Fs], Zp) :-
   ( \+ not_holds(F, Z) ->
          holds(F, Z, Z1),
          cancel_knows(F,Z1) ;
     \+ holds(F, Z)     -> Z1 = Z ;
     cancel(F, Z, Z1), cancel_knows(F,Z1),
     not_holds(F, Z1) ),
   minus(Z1, Fs, Zp).

plus(Z, [], Z).
plus(Z, [F|Fs], Zp) :-
   ( \+ holds(F, Z)     -> Z1=[F|Z] ;
     \+ not_holds(F, Z) -> Z1=Z ;
     cancel(F, Z, Z2), add_holds(F,Z2),
     not_holds(F, Z2), Z1=[F|Z2] ),
   plus(Z1, Fs, Zp).

update(Z1, ThetaP, ThetaN, Z2) :-
   minus(Z1, ThetaN, Z), plus(Z, ThetaP, Z2).
```

Figure 5: FLUX clauses for updating incomplete states.

$t_i = K(r, f)$. Once our agent performs an action whose negative physical effects include the fluent $f$, then by the assumption of exogenous actions, agent $r$ does no longer know $f$. In the same manner as the approach in (Thielscher 2004), if the status of the fluent is not entailed by the current state specification $\Phi(z)$, such that $\mathit{KState}(s, z) \equiv \Phi(z)$, then the partial knowledge of $f$ in $\Phi(z)$ does not transfer to the resulting state $z \ominus f$.

Figure 5 depicts a set of clauses for updating incomplete states. These clauses are similar to those in (Thielscher 2004), except that there are two new predicates, `cancel_knows(F,Z)` and `add_holds(F,Z)`. The former one is used to cancel disjunctive knowledge if there exists some $K(r, f_1)$ in the disjunctive knowledge, such that the fluent $f$ can be unified with the fluent $f_1$. The cancellation is due to the negative physical effects which involve the fluent $f$. The latter predicate is used to add the term `h(R,F)` to the disjunctive knowledge (recall the use of the term `h(R,F)` in the encoding of disjunctive knowledge of other agents). The addition is due to the positive physical effects of some action which involve the fluent $f$.

Thus far, the encoding of update has only addressed our agent's state and the knowledge of other agents in the form of disjunctive knowledge $\bigvee_{i=1}^{k>0} K(r, f_i)$. In Definition 2, the states of knowledge fluents are also updated. Therefore, for knowledge update axioms, there are two updates, the first one is to update our agent's state, which does not involve any knowledge fluent, and the second one is to update every knowledge fluent in our agent's state.

As all knowledge fluents are encoded as one single knowledge fluent, removing (updating) one single knowledge fluent in FLUX means removing (respectively, updating) all corresponding knowledge fluents in the Fluent Calculus. The above encoding of the predicate `update` can be readily used for updating the states of knowledge fluents. With many agents, there are many single knowledge fluents. This suggests the following scheme of recursive clauses for updating the knowledge of other agents with respect to some action $A$:

$$\begin{aligned}
&\mathit{UpdateKF}(z, A, z) \leftarrow \neg \mathit{Holds}(\mathit{KF}(r, z_r), z).\\
&\mathit{UpdateKF}(z_1, A, [\mathit{KF}(r, z_{r2})|z_2]) \leftarrow\\
&\quad \mathit{Holds}(\mathit{KF}(r, z_{r1}), z_1, z_t),\\
&\quad \Phi_1(z_1) \rightarrow \mathit{Update}(z_{r1}, \vartheta_1^+, \vartheta_1^-, z_{r2}); \ldots;\\
&\quad \Phi_n(z_1) \rightarrow \mathit{Update}(z_{r1}, \vartheta_n^+, \vartheta_n^-, z_{r2});\\
&\quad \mathit{UpdateKF}(z_t, A, z_2).
\end{aligned}$$

where each $\Phi_i(z_i)$ is a state specification, and $\vartheta_i^+$ and $\vartheta_i^-$ are, respectively, positive and negative effects. This scheme is then attached to the encoding of knowledge update axioms.

**Example 4** Consider again Example 1. Actions *MakeGFalse* and *SenseF* can be encoded as follows:

```
state_update(Z1,makegfalse,Z2,[]) :-
    update(Z1,[],[g],ZT),
    update_kf(ZT,makegfalse,Z2).

update_kf(Z1,makegfalse,[kf(R,ZR2)|Z2]):-
    holds(kf(R,ZR1),Z1,ZT), \+ nonground(R),
    update(ZR1,[],[g],ZR2),
    update_kf(ZT,makegfalse,Z1).

update_kf(Z,makegfalse,Z) :-
    \+ holds(kf(_,_),Z).

state_update(Z,sensef,Z,[F]) :-
    F=true -> holds(f,Z) ; true.
```

Provided that fluent $F$ is true in the world, then the following describes the encoding of the evolution of possible states shown in the example:

```
?- init(Z0),
   state_update(Z0,makegfalse,Z1,[]),
   state_update(Z1,sensef,Z2,[true]).

 Z0 = [kf(r,[g|ZR]),g | ZT]
 Z1 = [kf(r,ZR) | ZT]
 Z2 = [kf(r,[f|ZR]),f | ZT]

not_holds(g,ZR), not_holds(f,ZR),
not_holds(g,ZT), not_holds(f,ZT)
   ...
```

The result described above is the same as what Example 1 has shown. In the last situation, our agent knows $K(R, \neg G)$ due to the presence of the constraints `not_holds(g,ZR)` and `not_holds(g,ZT)`. Moreover, since fluent `f` holds in both the list `Z2` and the list `[f|ZR]` of `kf(r,[f|ZR])`, our agent also knows $K(R, F)$. □

## DISCUSSION

We have introduced a formalism for the knowledge of other agents and communication in the Fluent Calculus. The knowledge of other agents is represented as knowledge fluents. These fluents represent the possible states that our agent thinks that other agents think to be in. The representation of the knowledge of other agents and the knowledge updates have been shown to respect the universal validity of knowledge.

Communication is treated as a set of communicative actions. The specification of communicative actions benefits from the formalism for the knowledge of other agents. Moreover, the ability to reason about the knowledge of other agents is important to have efficient communication. This paper has shown one example, that is, to ask another agent about some property, our agent has to infer the knowledge of the agent he asks. In multi-agent settings, where actions of other agents need to be planned as well and these actions have knowledge preconditions, the capability of reasoning about the knowledge of other agents will be crucial.

We have shown how the formalism for knowledge and communicative actions has been implemented in FLUX. This paper has shown that knowledge fluents can be encoded in a succinct way. All knowledge fluents belonging to the same agent are encoded as one single knowledge fluent, whose second argument is a FLUX state. To handle this fluent, an extension to the existing FLUX system (Thielscher 2004) is given. Inferring knowledge of other agents in FLUX is also presented. The inference method is necessary to encode the communicative actions developed in this paper. In particular, due to the encoding of knowledge fluents, disjunctive knowledge of other agents cannot be inferred using the principle of negation–as–failure. This paper has described a feasible alternative using the universal validity of knowledge. Finally, the encoding of knowledge update axioms consists of two stages. The first stage is used to update our agent's state without involving any knowledge fluent. The next stage accounts for updating the states of knowledge fluents.

Related approaches to treating communication as actions are (Acqua, Sadri, & Toni 1999; Cohen & Perrault 1979; Kowalski & Sadri 1999; Lespérance *et al.* 1995; Shapiro, Lespérance, & Levesque 1997). The main difference between these and our approach is as follows. Our approach takes a purely subjective perspective, whereas the other formalisms take an objective perspective, that is, the approaches view the system consisting of many agents as one system. In other words, the other formalisms are mainly used to prove properties of multi–agent systems rather than for building individual agents.

An important limitation of our approach is that agents cannot reason about what other agents know *of other agents*. This limitation, however, was necessary to obtain an effective encoding of the knowledge of other agents in FLUX. Moreover, the negative knowledge of other agents has not been encoded in FLUX. For example, our agent knows $\neg K(r, \phi)$ (agent $r$ does not know $\phi$), has not been encoded in FLUX and is left for future work.

## References


Acqua, D.; Sadri, F.; and Toni, F. 1999. Communicating Agents. In *Proceedings of the Workshop on Multi–Agent Systems in Logic Programming*. In conjunction with ICLP'99.

Austin, J. L. 1962. *How to Do Things with Words*. London: Oxford University Press.

Cohen, P. R., and Perrault, C. R. 1979. Elements of a plan-based theory of speech acts. *Cognitive Science* 3(3):177–212.

FIPA:The Foundation for Intelligent Physical Agents. 2000. FIPA communicative act library specification. URL:`http://www.fipa.org`.

Frühwrith, T. 1998. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming* 37(1–3):95–138.

Kowalski, R. A., and Sadri, F. 1999. From logic programming towards multi–agent systems. *Annals of Mathematics and Artificial Intelligence*.

Lespérance, Y.; Levesque, H. J.; Lin, F.; Marcu, D.; Reiter, R.; and Scherl, R. B. 1995. Foundation of a logical approach to agent programming. In Wooldridge, M.; Müller, J. P.; and Tambe, M., eds., *Proceedings of IJCAI '95 ATAL workshop LNAI*, volume 1037, 331–345. Springer-Verlag.

Searle, J. R. 1969. *Speech Acts: An Essay in The Philosophy of Language*. Cambridge: Cambridge University Press.

Shapiro, S.; Lespérance, Y.; and Levesque, H. J. 1997. Specifying communicative multi–agent systems with congolog. In *Working Notes of the AAAI Fall 1997 Symposium on Communicative Action in Humans and Machines*, volume 1037, 72–82. Cambridge, MA: AAAI Press.

Thielscher, M. 2000. Representing the knowledge of a robot. In Cohn, A.; Giunchiglia, F.; and Selman, B., eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 109–120. Breckenridge, CO: Morgan Kaufmann.

Thielscher, M. 2001. The Qualification Problem: A solution to the problem of anomalous models. *Artificial Intelligence* 131(1–2):1–37.

Thielscher, M. 2004. FLUX: A logic programming method for reasoning agent. *Journal of Theory and Practice of Logic Programming*.