# Combining Uncertainty and Description Logic Rule-Based Reasoning in Situation-Aware Robots*

**Hans-Ulrich Krieger and Geert-Jan M. Kruijff**
German Research Center for Artificial Intelligence (DFKI GmbH)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{krieger,gj}@dfki.de

## Abstract

The paper addresses how a robot can maintain a state representation of all that it knows about the environment over time and space, given its observations and its domain knowledge. The advantage in combining domain knowledge and observations is that the robot can in this way project from the past into the future, and reason from observations to more general statements to help guide how it plans to act and interact. The difficulty lies in the fact that observations are typically uncertain and logical inference for completion against a knowledge base is computationally hard.

## Introduction

A robot continuously builds up beliefs about the world, and about the agents it is working with. It bases these beliefs in its experience. This experience naturally covers the here-and-now. What the robot currently sees, hears, plans to do. At the same time, experience needs to go beyond that. Recording past experience, the robot can reason about what once was – and what might still be, even when the robot isn't looking that way right now. And, looking beyond the moment, the robot can create expectations about what the future might bring. Either based on what has been, or what can be expected to be the case, given general "world" knowledge the robot has.

Every time the robot forms new beliefs, or alters ones that it already entertains, its collection of beliefs needs to be updated. This update takes the model from some state $t$ to a new state $t + 1$. This new state reflects the robot's beliefs about the world. Typically, the robot updates its beliefs on the basis of perceptual input it gets, or deliberative steps it takes, e.g., in dialogue processing or action planning. Experiencing the world and acting on it, are the main drives for maintaining and updating a robot's belief model.

What we would like to achieve is that, within bounds, the belief model represents *all* the robot could possibly know about the aspects of the world it is holding beliefs about. By this we mean both a sense of temporal continuity or persistence, and a sense of completion. By persistence we mean that the robot can infer whether what it believed earlier is still the case, currently. Even when the robot does not have any current experience to confirm or disconfirm that. By semantic completion we mean that when a robot creates a belief based in experience, it can expand that belief by making further inferences about that aspect of reality by using its domain knowledge. There is a certain appeal to a model like that. At each point, it represents *all there is to know about experience* relative to the robot's domain knowledge and inference capabilities. And that means that any process acting on that model only needs to inspect the model to make its decisions, i.e., without needing to request further information from other processes.

Just seeing this logically would be nice, but there is a problem we need to face here. Namely, there is an inherent *uncertainty* to a robot's experience, reflected in the beliefs and inferences it can draw from them. A robot never knows for sure that the object in front of it is a mug – with some likelihood, yes, but it could also be a box. Or the room it believes to be a bedroom is actually a kitchen. A robot is never certain. Any computation for dynamically updating belief models needs to take these uncertainties into account. In this paper we discuss how we can deal with this kind of uncertainty, in combination with safe monotonic logical inference. As a consequence of the uncertainty and the fact that "spelling out" uncertainty is incredibly inefficient, a robot, or more generally, a situation-aware system needs some model of bounded rationality.

## Approach

We describe here an approach that combines uncertainty and description logic rule-based reasoning. Probabilistic information is attached to perceived events (percepts) or interaction with other agents, whereas monotonic reasoning is realized in a highly-optimized rule-based reasoner that covers great parts of OWL (ter Horst 2005) and integrates user-defined rules, going beyond the expressivity of OWL DL.

What is unique to our approach is that uncertainty of perceived information is *not* translated into uncertainty (or fuzziness) built into logical reasoning, but instead is realized through possible worlds, ordered by their likeliness w.r.t. the probability of a recognized external event that is integrated

*The research described here has been financed by the European Integrated projects **CogX** (cogx.eu) and **NIFTi** (nifti.eu) under contract numbers FP7 ICT 215181 and 247870. We would like to thank Bernd Kiefer and the reviewers for their comments.

into the world and the distance of the world to a goal that an agent wants to achieve (see Figure 1).

Thus facts and rules inside a given world are safe and always do hold for this specific possible world. However, it might be the case that such a world is either very unlikely or its ABox is inconsistent, and thus can not be pursued further.

*Bounded rationality* is built into our model in that the approach **(i)** is lazy and the number of worlds in focus might be restricted to the most-probable ones; **(ii)** comes with a monotonicity assumption explained later, so that predecessors of expanded worlds need *not* be recorded; **(iii)** favors a kind of depth-first search that suits online search and only backtracks in case an expanded world is less likely than a recent non-expanded world; **(iv)** incorporates alternative/competing information from the future when backjumping to an older world; **(v)** might involve resource restrictions in order not to fully perform semantic completion (this involves the forward chainer, described later). As we see below, uncertain information reaches us discretized and aggregated, so we are not forced to translate a continuum of possible feature values into a finite set of values.

## Percepts as Multivariate Probability Distributions

Incoming uncertain information (percepts) is represented as a list of independent features over distributions of possible values, for instance:

$$\langle \mathsf{type} = \langle \mathsf{mug} : 0.6; \mathsf{box} : 0.4\rangle,\\ \mathsf{color} = \langle \mathsf{red} : 0.6; \mathsf{blue} : 0.3; \mathsf{green} : 0.1\rangle,\\ \mathsf{shape} = \langle \mathsf{square} : 0.2; \mathsf{round} : 0.8\rangle\rangle$$

Since we regard features as conditionally independent, we turn the conjunction into a disjunction, viz., a *multivariate probability distribution*:

$$\langle\langle \mathsf{type} = \mathsf{mug}, \mathsf{color} = \mathsf{red}, \mathsf{shape} = \mathsf{square}\rangle : 0.072;\\ \langle \mathsf{type} = \mathsf{mug}, \mathsf{color} = \mathsf{red}, \mathsf{shape} = \mathsf{round}\rangle : 0.288;\\ \ldots\ldots\\ \langle \mathsf{type} = \mathsf{box}, \mathsf{color} = \mathsf{green}, \mathsf{shape} = \mathsf{round}\rangle : 0.032\rangle$$

This representation does not change the probability mass which is still 1 for the sum of the individual percepts. Such a distribution over $n$ complex percepts $p_i$ serves two purposes. *Firstly*, the event probabilities are useful to compute the transition probabilities $\Pr(w_{t+1,i} \mid w_t, p_i, a_t)$ of a world $w_t$ at $t$ to $n$ different possible successor worlds $w_{t+1,i}$ at $t+1$ ($1 \leq i \leq n$). The effects of action $a_t$ has led to the construction of world $w_t$ and can be checked against percept $p_i$ to see whether $a_t$ was successfully executed. Thus for the above example, we can produce 12 possible continuations. This is shown in Figure 1.

*Secondly*, the different feature-value combinations encoded in the percepts at $t+1$ are potentially added to copies of the ABox of $w_t$ in order to eventually produce completed ABoxes for $w_{t+1,i}$. For the first percept, we then obtain the following beliefs ($o$ is a new individual): $\{$*mug*$(o, t+1)$, *hasColor*$(o,$ *red*$, t+1)$, *hasShape*$(o,$ *square*$, t+1)$ $\}$.

## Algorithm

We have separated the algorithm into three parts: (i) ***init***( ) which constructs the initial world at time 0, (ii) ***expand***( ) which generates possible continuations, and (iii) ***senseInferAct***( ) which integrates new sensed information from outside, calls ***expand***( ) and checks whether a final world has been reached. It is worth noting that we maintain several global data structures, important to proper backjump (not necessarily, chronological backtracking) to older non-expanded world in case the agent changes its "focus": timeToClock, timeToPosition, timeToAction, and timeToPercepts. Note that time and current below are also global.

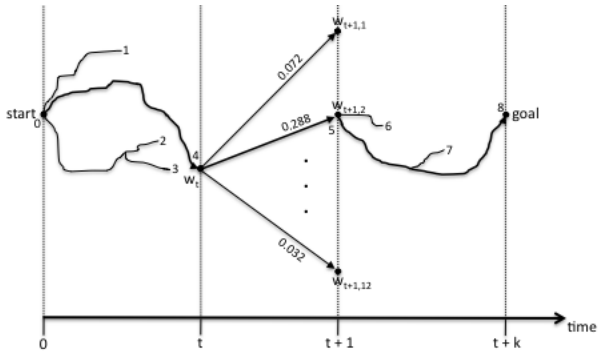


Figure 1: 12 extensions are possible to move from world $w_t$ further into the future (see example above). $w_{t+1,2}$ is the world that comes with the largest transition probability and has the least cost, as indicated by the path connecting $w_{t+1,2}$ and the final goal world. Backtrack points, at which worlds are not further pursued, are marked as 1, 2, 3, 6, and 7.

**Init** We start by constructing an initial world at time 0 (line 06). This world is equipped with some axiomatic knowledge which we make explicit (07+08), but is *not* equipped with information from outside (viewed technically, a blank percept (09) has "led" us to this world). The axioms essentially outline what we could know about *any* world. Since we have not sensed the environment so far, the initial action that is executed here, must be a *wait* action (12). We assign some initial distance estimation to the goal state through ***estimate***( ) (14). The definition of a distance measure, of course, depends on the agent's mission, e.g., Euclidean distance for route finding. Since expansion has not started yet, the agenda of most-probable successor worlds is empty (16).

```
init(initialKnowledge, initialPosition) ≡
01 time := 0
02 timeToClock[0] := getCurrentTime( )
03 timeToPosition[0] := initialPosition
04 timeToAction[0] := wait
05 timeToPercepts[0] := ⟨ blank ⟩
06 current := makeNewWorld( )
07 current.beliefs := initialKnowledge
08 current.beliefs := computeClosure(current.beliefs)
09 current.percept := blank
10 current.here := initialPosition
11 current.now := 0
12 current.action := wait
13 current.travel := 0
14 current.estimate := estimate(null, noop, current)
15 current.total := total(0, current.estimate)
16 activeWorlds := ⟨ ⟩
```

**Expand** When new information from outside is integrated into the system, we construct a new world for each percept from the multivariate distribution (02+03); see the example above. The ABox is (shallow) copied over from current, beliefs generated from the percept are added, and the closure is computed (04+05). This expansion also uncovers inconsistent ABoxes through querying for individuals of type `owl:Nothing`. In such a case, we assign an infinite distance. Only if the distance estimate between the new world and the goal state is finite, we compute the total cost and add next to the agenda (15–17). The lines 08–10 guarantee that an older "passive" world at time $s$ that is regarded to be more likely than the current world at time $t$ ($s < t$) is equipped with the former positions and the performed actions of the agent at times $s+1, \ldots, t$, when brought to the "temporal forefront".

*expand*( ) ≡
01 **local** next
02 **for each** percept ∈ timeToPercepts[current.now + 1]
03 next := *makeNewWorld*( )
04 next.beliefs := *copy*(current.beliefs) ∪ *beliefs*(percept)
05 next.beliefs := *computeClosure*(next.beliefs)
06 next.percept := percept
07 next.now := current.now + 1
08 **if** next.now < time
09 next.action := timeToAction[next.now]
10 next.here := timeToPosition[next.now]
11 **else**
12 next.here := *obtainPosition*( )
13 next.travel := *travel*(current, current.action, percept)
14 next.estimate := *estimate*(current, current.action, next)
15 **if** next.estimate ≠ ∞
16 next.total := *total*(next.travel, next.estimate)
17 *push*(next, activeWorlds)

**Sense, Infer, Act** As long as we have not reached a goal state or have been interrupted (02-06), we keep on adding new beliefs generated from percepts (10) to possible extensions of agenda items (12). The definition of a goal state clearly depends on the problem the agent wants to solve, e.g., a zero distance to a landmark (route finding) or a specific logical entailment (exploration mission). "Safe" information about past events communicated by other agents (e.g., human operator) is added to current and all sleeping agenda items (07). This strategy could lead to the activation of a former, less-likely world that needs to be enriched with percepts and executed actions, as explained in the section on **Expand** above. Thus the inner loop (11–17) might be executed more than once until the time stamp now of the current world is in sync with the global counter time. We also keep a spatial map of the environment that is destructively updated during this loop (17).

Action planning and execution finally started (19–21) after we have opted for a new world (16). We note here that since action execution (21) is supposed to be carried out *asynchronously* in a separate thread, the outer loop (02) keeps on sensing the environment (10). As a result, new percepts can in principle be used to correct an ongoing action, potentially leading to an online replanning step in (19).

*senseInferAct*(initialKnowledge, initialPosition) ≡
01 *init*(initialKnowledge, initialPosition)
02 **while true**
03 **if** *isInterrupted*( )
04 **return** *interrupted*
05 **if** *goalReached*( )
06 **return** *success*
07 *hearsay*( )
08 time := time + 1
09 timeToClock[time] := *getCurrentTime*( )
10 timeToPercepts[time] := *sense*( )
11 **while** current.now < time
12 *expand*( )
13 *recomputeEstimate*(activeWorlds)
14 *sort*(activeWorlds)
15 *restrictLength*(activeWorld)
16 current = *pop*(activeWorlds)
17 *updateMap*(current.now)
18 timeToPosition[time] := current.here
19 current.action := *plan*( )
20 timeToAction[time] := current.action
21 *execute*(current.action)

It is worth noting that this algorithm shares similarities with the $A^\star$ search algorithm in that the agenda is sorted according to a combination (usually the sum) of the accumulated path cost cost (a number) and the distance estimate estimate (a number), establishing a total order. The search space is a fan-out tree "into" the future, but sleeping past worlds on the agenda which gain importance through future knowledge (07) lead to a re-estimation of the total cost (13). As explained above, a past alternative world is then equipped with the things that have happened, since it became inactive. This, however, does *not* necessarily mean that an agent in this world has to travel back in space (even though it has followed a wrong route as a result of false beliefs).

Inside a world, spatial search is carried out in a graph representation of the environment by a variant of $A^\star$. As mentioned above, this map is updated (17) every time we perceive new information and move on to another world. Traveling back in time from $t$ to an inactive world at time $s$ ($s < t$) is achieved by destructively modifying the map, removing the information that has been added between $s$ and $t$, and adding the new alternative information for time stamps $s, s+1, \ldots, t$. Contrary to spatial ABox beliefs which deal with topological relations, the map mostly contains low-level spatial information about objects (e.g., 2D coordinates) and single-step costs between graph nodes. The reason for having only one map and multiple ABoxes is that we can efficiently undo modifications of the map, but would need a truth maintenance-like structure hooked into the ABox to avoid wrong entailments.

## Complexity Considerations

Given $P$ percepts per clock tick and $T$ ticks in a sequence, we can estimate the complexity of the above outlined algorithm and some foreseeable variations in terms of the *number of constructed worlds*:

- full expansion: $\mathcal{O}(P^T)$
- lazy expansion (**this paper**): between $\mathcal{O}(P * T)$ and $\mathcal{O}(P^T)$, depending on the predictive power of *estimate*( )

- lazy expansion + length-restricted agenda: $\mathcal{O}(T)$ (the whole search space can no longer be inspected; only a constant number of worlds is in the focus)
- greedy expansion: $\mathcal{O}(1)$ (only the most-probable world needs to be recorded and is extended without copying)

### Completeness of Algorithm

The algorithm, as displayed above, is able to visit all possible worlds, thus guaranteeing that a final goal can be reached, if it exists. This can be shown by defining the accumulated travel costs to be 0 for each world and by choosing the following distance function that realizes a breadth-first search:

$$\textit{estimate}(\text{from, action, to}) \equiv \begin{cases} \infty & \textbf{if } \text{to.beliefs} \text{ is inconsistent} \\ \text{to.now} & \textbf{otherwise} \end{cases}$$

However, such a behavior is usually not desired, since full expansion is intractable. Finding an optimal estimation is in general impossible, since we do not know how far our final goal is located in the future.

## Forward Chaining and Closure Computation

Logical inference within the individual worlds is performed by *HFC*, a highly-optimized rule-based forward chainer that was originally implemented for reasoning and querying with OWL-encoded ontologies (McGuinness and van Harmelen 2004) over RDF *triples* (Manola and Miller 2004). Usually, bottom-up forward chaining is employed to carry out (all possible) inferences at compile time, so that querying information reduces to an indexing problem at runtime. The process of making implicit information explicit is often called *materialization* or computing the *deductive closure* of a set of ground atoms $A$ w.r.t. a set $R$ of universally-quantified implications $B \rightarrow H$ (if-then rules). The body and the head of a rule consist of a set of clauses, interpreted *conjunctively*. Forward chaining, as we used it here, can be seen as *model building* over the Herbrand interpretation of a function-free definite program (Horn logic as used in Prolog).

It is worth noting that the forward chainer operates in a monotonic and certain conjunctive search space: neither do we delete any information, nor do we attach probabilities to asserted facts. As described in the algorithm above, *HFC* regularly queries new information within a situation-awareness loop. Since the size of relevant *new* information between different closure computations is relatively small, a new fixpoint is usually computed extremely fast, requiring only a few iteration steps. *HFC* efficiently handles ABoxes with millions of facts and provides means to work with thousands of medium-sized ABox in parallel, an important feature that we employ in our approach here.

### Rules in *HFC*

Rules are formulated using an extended RDF triple/tuple notation (additional LHS tests and RHS actions). Rules

1. implement *OWL entailment*, and thus consistency; e.g.,

```
?s owl:sameAs ?o
?s owl:differentFrom ?o
->
?s rdf:type owl:Nothing
?o rdf:type owl:Nothing
```

2. provide *custom functionality*; e.g., by moving from a point-based sensor-oriented representation to an extendable interval-based encoding:

```
?s ?p ?o ?t
->
?s ?p ?o ?t ?t
```

We have opted to go for general tuples instead of using encoding schemes, such as reification, thus making *HFC* able to address two further important areas of functionality:

3. to coalesce information over time, e.g.,

```
?s ?p ?o ?b1 ?e1
?s ?p ?o ?b2 ?e2
->
?s ?p ?o ?b ?e
@test
IntervalNotEmpty ?b1 ?e1 ?b2 ?e2
@action
?b = Min2 ?b1 ?b2
?e = Max2 ?e1 ?e2
```

4. to reformulate LTL safety conditions ($r$ = robot, $a$ = area), such as $\mathbf{G}(\textit{explore}(r) \wedge \textit{risky}(a) \Rightarrow \neg \textit{in}(r, a))$:

```
?r rdf:type Explore ?b1 ?e
?a rdf:type Risky ?b2 ?e
?r near ?a ?b3 ?e
->
DO SOMETHING / MOVE ROBOT AWAY / ...
```

In order to make the entailment rules for RDFS (Hayes 2004) and OWL (ter Horst 2005) also sensitive to time, we have extended them by further temporal arguments (Krieger 2011). For instance, a rule that talks about functional object properties in OWL is implemented as:

```
?p rdf:type owl:FunctionalProperty
?p rdf:type owl:ObjectProperty
?x ?p ?y ?b1 ?e1
?x ?p ?z ?b2 ?e2
->
?y owl:sameAs ?z
@test
IntervalNotEmpty ?b1 ?e1 ?b2 ?e2
```

## References


Hayes, P. 2004. RDF semantics. Technical report, W3C.

Krieger, H.-U. 2011. A temporal extension of the Hayes and ter Horst entailment rules for RDFS and OWL. In *AAAI 2011 Spring Symposium "Logical Formalizations of Commonsense Reasoning"*.

Manola, F., and Miller, E. 2004. RDF primer. Technical report, W3C.

McGuinness, D. L., and van Harmelen, F. 2004. OWL Web Ontology Language Overview. Technical report, W3C.

ter Horst, H. J. 2005. Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *Proceedings of the International Semantic Web Conference*, 668–684.