Annotations in Jason

Table of Contents

Annotations in Jason
- Introduction
- Syntax
- Unification Between Literals
- Unification Between Variables
- Annotated Variables in a Plan Body
- Annotations in Goals and Plans

Introduction

Each belief in an agent's belief base has at least one *annotation*; goals and plans can also have annotations. The whole idea of annotations emerged when we added speech-act based communication, so we needed to annotate the *source* of each belief. In a multi-agent systems, agents' beliefs can come from inter-agent communication, by sensing the envioronment (such beliefs are called percepts), or because the agent added "mental notes" (i.e., beliefs the agent created itself whilst executing plans in order to remind itself of something). In Jason, all beliefs have a pre-defined *source* annotation, which is automatically handled by the interpreter; all other annotations are user-defined an need to be used/controled by the programmer.

Annotations do not change the expressive power of the programming language, but they greatly improve legibility. In fact, they are one of the most interesting additions to the original AgentSpeak language that were introduced in ***Jason***. As agent-oriented programming is heavily influenced by Artificial Intelligence, it often makes sense to represent meta-level information about each individual belief, goal, or plan that an agent has. Even though originally we only needed annotations to denote the sources of information an agent had, they turned out to be an extremely flexible mechanism with many uses and purposes. Much research work extending AgentSpeak has taken advantage of annotations for specific purposes.

However, in order to deal with annotations, we have had to create a more sophisticated unification algorithm. This article presents how it works by means of examples.

Syntax

Annotations are represented with the same syntax of a list in Prolog; however, this needs to be a list of *terms* rather than literals, and it must immediately follow the literal or term they are annotating (plans are annotated in the optional label they have, which is itself a predicate). For example:

```
p(t)[source(ag)]
```

represents a belief literal `p(t)` with a single annotation `source(ag)` which is an annotation handled by Jason to say that this belief originating from agent `ag` telling this agent that `ag` believed `p(t)` to be true. Other possible sources are `source(percept)` and `source(self)`; the former means that belief `p(t)` originated from perceiving the environment and the latter from a mental note. In the following example:

```
p(t)[a1,a2(0)].
```

the literal `p(t)` has two annotations, terms `a1` and `a2(0)`.

One important thing to note is that even though the complete notation for lists in Prolog is used for ***Jason*** annotations, this is semantically treated as as *set* of annotations.

Unification Between Literals

In the case of a unification like `A = B`, the set of annotations of the first argument `A` has to be a *subset* of the annotations of the second argument `B`.

Example:

```
p(t) = p(t)[a1];           // unifies
p(t)[a1] = p(t);           // does not unify

p(t)[a2] = p(t)[a1,a2,a3]; // unifies
```

The "tail" of the list of annotations (in this case working like set difference) can be used:

```
p[a2|T] = p[a1,a2,a3];     // T unifies with [a1,a3]
p[a1,a2,a3] = p[a1,a4|T];  // T unifies with [a2,a3]
```

When the unification is between a triggering event and a plan's trigger, the triggering event is the *first* argument of the unification. So for an event `+!g[a]`, the relevance of the following plans would be as follows:

```
+!g : true <- ...       // relevant for event +!g[a]
+!g[a] : true <- ...    // relevant for event +!g[a]
+!g[a,b] : true <- ...  // not relevant for event +!g[a]
+!g[b] : true <- ...    // not relevant for event +!g[a]
```

However, for an event `+!g` a plan with trigger `+!g[a]` is *not* relevant. In other words, to put an annotation is a plan's trigger means "this plan is relevant only for events with (at least) these annotations, or annotations that unify with these".

Unification Between Variables

Consider the various cases of the a unification `X[As] = Y[Bs]` below; the notation used is `X` and `Y` for variables, and `As`, `Bs`, `Cs`, and `Ds` are sets of annotations.

X and Y are ground

```
X     = p[Cs]  // unify X with p[Cs], where Cs is a set of annotations
Y     = p[Ds]  // unify Y with p[Ds], where Ds is a set of annotations
X[As] = Y[Bs]  // unifies if (Cs union As) is a subset of (Ds union Bs) ...
               // ... after attempting to unify the annotations individually
```

Example:

```
X     = p[a1,a2];
Y     = p[a1,a3];
X[a4] = Y[a2,a4,a5]; // unifies
```

> (note that tail of lists does not work here. **TODO** make it to work as for literals? In the current implementation, the tail in annotations of variables has no meaning)

Only X is ground

```
X     = p[Cs]
X[As] = Y[Bs]  // unifies if (Cs union As) in Bs
               // and Y unifies with p
```

> note: what `Y` should unify with is `p[ (Cs + As) - Bs ]`. **TODO**: the current implementation is as above and not as proposed in this note.

Example:

```
X     = p[a1,a2];
```

```
X[a3] = Y[a1,a2,a3,a4,a5]; // unifies Y with p
X[a3] = Y[a2,a3,a4,a5];    // does not unify
X[a3] = Y[a1,a2,a4,a5];    // does not unify
```

Only Y is ground

```
Y     = p[Ds]
X[As] = Y[Bs]  // unifies if As in (Ds union Bs)
               // and unifies X with p
```

> note: the annotations of `X` is an issue to discuss (what `X` should unify with). It could be `[]`, since `[]` is a subset of anything. It could be: `X = p[(Ds + Bs) - As]`. A minimal subset approach or a maximal subset approach. **TODO**: the current implementation is like above (minimal subset approach).
> The problem is that `X = p[a,b,c]` unifies `X` with `p[a,b,c]`, i.e., the maximal approach. So the current implementation is somewhat inconsistent. Proposal: use always the maximal approach when `Y` is ground and the minimal when `X` is ground.

Example:

```
Y     = p[a1,a3];
X[a1] = Y[a4,a5]; // unifies X with p
X[a6] = Y[a4,a5]; // does not unify
```

Neither X nor Y are ground

```
X[As] = Y[Bs]  // unifies if As is a subset of Bs
               // and X unifies with Y
```

Annotated Variables in a Plan Body

The annotations of the variable and the annotations of its instantiated are combined (using set union) to produce the corresponding event:

```
X=g[a];
...
!X[b]; // produce event +!g[a,b]
```

Annotations in Goals and Plans

(tbd)