Louise A Dennis
January 26, 2021

Action Lifecycles in Gwendolen

Way back in the before times when it was possible to meet in person, Peter, Rafael and I sat down and brainstormed what the Gwendolen semantics might look like if it explicitly recognised that actions might take time to complete and that they could complete with success, failure or timeout (as discussed more generally in [Dennis and Fisher, 2014a, Dennis and Fisher, 2014b]). The result was a set of scrawled diagrams and greek letters that now, 10 months later, even I have trouble interpreting. So this is a second attempt at producing something intelligable.

1 Intentions in Gwendolen

Intentions are crucial to understanding Gwendolen. BDI languages use intentions to store the *intended means* for achieving goals – this is generally represented as some from of *deed stack* (deeds include actions, belief updates, and the commitment to goals). Intention structures also maintain information about the (sub-)goal they are intended to achieve or the event that triggered them. Gwendolen aggregates this information: an intention becomes a stack of tuples of an event, a deed, and a unifier (but I'm going to ignore unifiers in everything that follows as an added complexity which we can deal with later). This tuple is most simply viewed as a matrix structure consisting of two columns in which we record events (new perceptions, goals committed to and so forth), and deeds (a plan of future actions, belief updates, goal commitments, etc.). These columns form an event stack and a deed stack. Rows associate a particular deed with the event that has caused the deed to be placed on the intention. New events are associated with an empty deed, ϵ.

Example The following shows the structure for a single intention to clean a room. We use a standard BDI syntax: $!g$ to indicate the goal g, and $+!g$ to indicate the commitment to achieve that goal (i.e., a new goal that g becomes true is adopted).

event	deed
`+!clean()`	`+!goto(room1)`
`+!clean()`	`+!vacuum(room1)`

This intention has been triggered by a goal to clean — the commitment to the goal *clean*() is the trigger event for both rows in the intention. An intention is processed from top to bottom so we see here that the agent first intends to commit to the goal $goto(room1)$. Once it has committed to that goal it then commits to the goal $vacuum(room1)$. In Gwendolen the process of committing to a goal causes an expansion of the intention stack, first making the goal into an event which has yet to be planned (so is associated with the empty deed ϵ) and then selecting a plan to execute which replaces ϵ and pushes more deeds on the intention to be processed.

If $goto(room1)$ is an achievement goal (which among other things means it it remains on the deed stack until it is "achieved") then our intention is expanded *before* the agent commits to vacuuming the room and the above intention becomes first

event	deed
`+!goto(room1)`	ϵ
`+!clean()`	`+!goto(room1)`
`+!clean()`	`+!vacuum(room1)`

and maybe then (depending upon the plan)

event	deed
`+!goto(room1)`	`+!planRoute(room1)`
`+!goto(room1)`	`+!follow_route`
`+!goto(room1)`	`+!enter(room1)`
`+!clean()`	`+!goto(room1)`
`+!clean()`	`+!vacuum(room1)`

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited.

At any moment, we assume there is a *current intention* which is the one being processed at that time. The stacks that form the intention are further paired with two booleans, *suspended*, and *locked* which indicate the intention's status. A suspended intention is, by default, *not* selected at the intention selection phase of the agent's reasoning.

Because intentions are stacks we are often interested the top row of the intention, *the head*, written $\texttt{hd}_i(i)$, and the rest of the intention, *the tail*, written $\texttt{tl}_i(i)$. Sometimes we are just interested in the top event on the intention, $\texttt{hd}_e$ or the top deed on the intention, $\texttt{hd}_d$.

2 How actions Currently work in GWENDOLEN

Currently actions with durations work via a kludge in GWENDOLEN, in which the action is started using an action command on the top of an intention's deed stack and then the intention is suspended until some success condition for the action is reached using a "wait for" command. In most BDI languages an action starts and the agent stops doing anything else until the action completes, but this doesn't really work for lots of the systems we consider. In particular when an action involves moving from one location to another we generally want to continue monitoring for other things (like approaching collisions, thruster failures, etc.) while the robot is moving and this can't happen if all processing within the agent is basically locked until the action finishes.

In the GWENDOLEN semantics this is represented by Equation (1) (which I've simplified a bit from the rule in the published semantics by removing all reference to unifiers, edge cases like the action command throwing an error, and checks that isn't a *send message* action which has a special rule). The notation I'm using has conditions for a transition above the line, and the actual transition below the line. In this case I'm representing the state of the system as the pair of an environment, ξ, and a big tuple representing all the parts of the agent such as its belief base, goal base, current intention, other intentions, plans, reasoning rules, message inbox etc.. etc., (lots of detail that I'm going to gloss over here).

$$\frac{\texttt{hd}_d(i) = a \quad \xi \xrightarrow{\mathbf{do}(a)} \xi'}{\langle \xi, \langle \ldots i \ldots \rangle \rangle \rightarrow_{\texttt{action}} \langle \xi', \langle \ldots \texttt{tl}_i(i) \ldots \rangle \rangle} \tag{1}$$

So if the top deed on the intention is an action, $\texttt{hd}_d(i) = a$, and the change the action makes to the environment, ξ, is to turn it into ξ', represented by $\xi \xrightarrow{\mathbf{do}(a)} \xi'$ then performing the action transforms the pair of the environment and agent by transforming the environment and removing the top row from the current intention, i, (i becomes $\texttt{tl}_i(i)$). All other parts of the agent state are unchanged (represented by the liberal use of $\ldots$).

Meanwhile the rules for handling wait for are:

$$\frac{\texttt{hd}_d(i) = * \ldots b \quad B, R \models b}{\langle \xi, \langle \ldots i \ldots B, R \ldots \rangle \rangle \rightarrow_{\texttt{wait\_for}} \langle \xi, \langle \ldots \texttt{tl}_i(i) \ldots B, R \ldots \rangle \rangle} \tag{2}$$

$$\frac{\texttt{hd}_d(i) = * \ldots b \quad B, R \not\models b}{\langle \xi, \langle \ldots i, \ldots, B, R \ldots \rangle \rangle \rightarrow_{\texttt{wait\_for}} \langle \xi', \langle \ldots \texttt{suspend}(i), \ldots, B, R \ldots \rangle \rangle} \tag{3}$$

where $B, R \models b$, means that the formula b follows using Prolog-style reasoning from the agent's belief base and Prolog rule-base. $\texttt{suspend}(i)$ suspends an intention.

So, if an intention is waiting for some belief, b, to become true ($* \ldots b$) then if that belief is already/now true the intention continues processing (2). Otherwise the intention is suspended (3). There's a special case for when there are no unsuspended intentions, but I'm ignoring that for now.

Lastly, intentions get unsuspended when new beliefs are added:

$$\frac{\texttt{hd}_d(i) = +b}{\begin{array}{c}\langle \xi, \langle \ldots i, I, B \ldots \rangle \rangle \rightarrow_{\texttt{add\_belief}} \\ \langle \xi, \langle \ldots \texttt{tl}_i(i), \mathbf{unsuspend}(I, b) \cup \texttt{new}(+b, \epsilon), B \cup \{b\}, \ldots \rangle \rangle\end{array}} \tag{4}$$

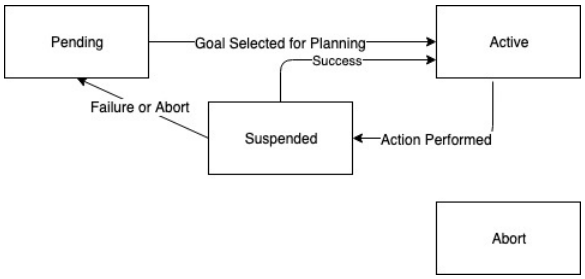
where $\mathbf{unsuspend}(I, b)$ unsuspends all suspended intentions in I that are waiting for b to become true (this is actually concealing a multitude of sins since sometimes reasoning rules are needed to determine if an intention should be unsuspended). $\texttt{new}(e, d)$ creates a new intention from an event and a deed.

This rule adds new belief to the belief base and a new intention noting the appearance of the new belief. At the same time it unsuspends all intentions which are waiting for b to be achieved as part of their suspend condition.

3 New Proposal

The new proposal is that actions should be paired with a success, failure and abort condition. I'm going to write this as $a : (\phi_s, \phi_f, \phi_a)$ where ϕ_s is the success condition, ϕ_f is the failure condition and ϕ_a is the abort condition. And we obviously want some semantics that looks a bit like a mash up of our action and wait for transition rules – except in the case of failure/abort we don't want to just carry on processing the intention, we want to do something else!!

This is where the diagram about goal life cycles came in which we adapted (I think, looking at our scrawl) to be something like:



A goal that is *Pending* is a goal on the top of an intention paired with a "no plan yet" deed (ϵ) which we discussed in Section 1. When the intention is selected and a plan is applied to the goal, expanding the intention, it becomes *Active*. When the top deed is an action then the whole intention is suspended, and this is what we mean by the Goal being *Suspended* in the lifecycle. If the action completes with sucess then the intention becomes active again and continues processing (like in the existing Wait For case). *However* if the goal fails or aborts then we drop everything to do with the current plan and go back to the pending state. We don't really distinguish between failures and aborts – the distinction might be useful for diagnosis, but doesn't make much difference to goal processing.

There is also an *Abort* state where we might want to abandon the goal entirely, not just attempt to replan it – there is a drop goal method in the AIL which should do this. Judging by our scribbles we weren't certain when a goal should be aborted altogether but it might be after repeated planning failures.

So if I were to make a first pass at adapted rules, I would change the action rule to something like:

$$\frac{\mathtt{hd}_d(i) = a : (\phi_s, \phi_f, \phi_a) \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad B, R \models \phi_s}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{tl}_i(i), B, R \ldots \rangle \rangle} \tag{5}$$

$$\frac{\mathtt{hd}_d(i) = a : (\phi_s, \phi_f, \phi_a) \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad B, R \models \phi_f \vee B, R \models \phi_a}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \text{whatever happens when we go back to pending} \rangle \rangle} \tag{6}$$

$$\frac{\mathtt{hd}_d(i) = a : (\phi_s, \phi_f, \phi_a) \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad B, R \not\models \phi_s \quad B, R \not\models \phi_f \quad B, R \not\models \phi_a}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{suspend}(\mathbf{done}(i)), B, R \ldots \rangle \rangle} \tag{7}$$

So if the action succeeds immediately (or trivially) (5) then we carry on processing the intention in an *Active* state. If the action fails immediately (6) we do "whatever" (I could probably write this out, but it would require some thought and probably more notation and isn't needed just at this point) and if the action neither succeeds nor fails immediately then we suspend the intention with some marker, **done** to show we've actually performed the action.

When we unsuspend intentions when we add beliefs we can extend **unsuspend** further to check these "done"[1] actions.

And then we have:

$$\frac{\mathtt{hd}_d(i) = \mathbf{done}(a : (\phi_s, \phi_f, \phi_a)) \quad B, R \models \phi_s}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{tl}_i(i), B, R \ldots \rangle \rangle} \tag{8}$$

$$\frac{\mathtt{hd}_d(i) = \mathbf{done}(a : (\phi_s, \phi_f, \phi_a)) \quad B, R \models \phi_f \vee B, R \models \phi_a}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \text{whatever happens when we go back to pending} \rangle \rangle} \tag{9}$$

$$\frac{\mathtt{hd}_d(i) = \mathbf{done}(a : (\phi_s, \phi_f, \phi_a)) \quad B, R \not\models \phi_s \quad B, R \not\models \phi_f \quad B, R \not\models \phi_a}{\langle \xi, \langle \ldots i, B, R, \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{suspend}(\mathbf{done}(i)), B, R \ldots \rangle \rangle} \tag{10}$$

[1] To be honest I'm not sure a **done** annotation is the best way to do this, but I think it suffices for now and we can see how the implementation works out first.

Which are the same as the new action rules except they don't actually try to do the action, they just check if the action has finished yet. To be honest we shouldn't need (10) since the intention should only become unsuspended if one of the three conditions is true, but I'm including it just in case there is some weird scenario I've not thought of where it is needed.

References

[Dennis and Fisher, 2014a] Dennis, L. A. and Fisher, M. (2014a). Actions with durations and failures in bdi languages. Technical Report ULCS-14-003, University of Liverpool, Department of Computer Science.

[Dennis and Fisher, 2014b] Dennis, L. A. and Fisher, M. (2014b). Actions with durations and failures in bdi languages. In Schaub, T., editor, *21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 995–996. IOS Press.