Gwendolen Semantics: 2017

Louise A. Dennis

February 11, 2017

1 GWENDOLEN Semantics

An operational semantics for GWENDOLEN was presented in [1]. However the language has been amended several times since then. This technical report updates that semantics to present the current operational semantics of the language.

It should be noted that the GWENDOLEN distribution comes with extensive tutorials detailing the language syntax, the motivation behind its constructs and providing many examples of programs in the language. This technical report therefore focuses exclusively on the technical details of the semantics of the underlying implementation.

2 Intentions

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. This tuple is most simply viewed as a matrix structure consisting of three columns in which we record events (new perceptions, goals committed to and so forth), deeds (a plan of future actions, belief updates, goal commitments, etc.), and unifiers. These columns form an event stack, a deed stack, and a unifier stack. Rows associate a particular deed with the event that has caused the deed to be placed on the intention, and a unifier. New events are associated with an empty deed, ϵ.

Example The following shows the full 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). Constants are shown starting with lower case letters, and variables with upper case letters.

event	deed	unifier
`+!clean()`	`+!goto(Room)`	`Room = room1`
`+!clean()`	`+!vacuum(Room)`	`Room = 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 $goto(Room)$, where $Room$ is to be unified with $room1$. Once it has committed to that goal it then commits to the goal $vacuum(Room)$. In GWENDOLEN the process of committing to a goal causes an expansion of the intention stack, pushing more deeds on it to be processed. So $goto(Room)$ is expanded *before* the agent commits to vacuuming the room and the above intention becomes

event	deed	unifier
`+!goto(Room)`	`+!planRoute(Room, Route)`	`Room = room1`
`+!goto(Room)`	`+!follow(Route)`	`Room = room1`
`+!goto(Room)`	`+!enter(Room)`	`Room = room1`
`+!clean()`	`+!vacuum(Room)`	`Room = room1`

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. Typically an intention will remain suspended until some belief condition occurs, normally that a belief is acquired via perception or from the receipt of a message. If an intention is locked, conversely, then it must be selected at the intention selection phase.

3 Plans, Applicable Plans and Intentions

A GWENDOLEN agent also has a *plan library* which is an ordered list of plans. Plans are matched against intentions and manipulate them. There are three main components to a plan,

1. A *trigger event* which may match the top event of an intention.

2. A *guard*: the guard is checked against the agent's state for plan applicability.

3. A *body* which is the new deed stack that the plan proposes for execution.

We use the syntax $trigger : \{guard\} \leftarrow body$ to represent plans.

Plans only match intentions which contain unplanned goals (i.e., those associated with the "no plan yet" deed, ϵ). For instance after a commitment to `goto(Room)` the above intention might appear as:

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

which would match the plan

$+!goto(Room) : \{upstairs(Room)\} \leftarrow +!goto(stairs); +!goto(Room)$

This plan says that in order to achieve the goal $goto(Room)$ in the case where the room is upstairs, ($upstairs(Room)$), first the goal $goto(stairs)$ must be achieved and then the goal $goto(Room)$ achieved.

This would transform the intention to:

event	deed	unifier
`+!goto(Room)`	`+!goto(stairs)`	`Room = room1`
`+!goto(Room)`	`+!goto(Room)`	`Room = room1`
`+!clean()`	`+!goto(Room)`	`Room = room1`
`+!clean()`	`+!vacuum(Room)`	`Room = room1`

3.1 Applicable Plans

Applicable plans are an interim data structure that describe how a plan from an agent's plan library changes the current intention. An applicable plan describes the new rows that will replace the top row of the intention. The new rows are generated from an event, a unifier and a stack of deeds. The new intention rows are generated by creating a row for each deed and associating the event and unifier with each of those rows (so the event and unifier are duplicated several times).

Therefore, an applicable plan is a tuple, (p_e, p_{ds}, p_θ), of an event p_e, a deed stack p_{ds}, and a unifier p_θ. The applicable plan in the example above would be

$$(+!goto(Room), [+!goto(stairs); +!goto(Room)], \{Room = room1\}) \quad (1)$$

Applicable plans are used because GWENDOLEN first determines a list of applicable plans and then picks one plan to be applied. The function $\mathcal{S}_{\text{plan}}$ is used to select *one* applicable plan from a set. By default, this treats the set as a list and picks the first plan, but it may be overridden by specific applications.

Applicable Plan Generation Method The function $\mathbf{appPlans}$, generates a set of applicable plans from the current intention, i, and an agent's internal state.

There are two cases. In the first case the top deed on the intention is not ϵ (i.e., no planning is needed). In this case the set of applicable plans is for continuing to process intention i without any changes (i.e., it represents the top row of the intention). So the set of applicable plans is the singleton:

$$\{(\mathtt{hd}_e(i), \mathtt{hd}_d(i), \theta^{\mathtt{hd}(i)}) \quad | \quad \mathtt{hd}_d(i) \neq \epsilon\} \tag{2}$$

where $\mathtt{hd}_e(i)$ is the top event in i, $\mathtt{hd}_d(i)$ is the top deed, and $\theta^{\mathtt{hd}(i)}$ is the top unifier.

In the case where the top deed on the intention is ϵ, $\mathbf{appPlans}$ generates the set

$$\begin{array}{l}\{(p_e, p_d, \theta^{\mathtt{hd}(i)} \cup \theta) \quad | \\ \quad p_e : \{p_{gu}\} \leftarrow p_d \in P \ \wedge \ \mathtt{hd}_e(i)\theta^{\mathtt{hd}(i)} \models p_e, \theta' \ \wedge \ ag \models p_{gu}\theta', \theta\}\end{array} \tag{3}$$

where P is the agent's library of plans. $\mathtt{hd}_e(i) \models p_e, \theta'$ means that the plan's trigger event follows from the top event on the current intention returning a unifier, θ'. This allows for Prolog-style reasoning on plan triggers.

The notation $ag \models g, \theta$ means that the guard, g, is satisfied by agent ag given unifier θ. Again this allows Prolog-style reasoning. Plan guards may refer to the agent's belief base, goal base or outbox. For instance $\mathcal{B}b$ means some belief, b should follow by logical inference from the agent's belief base and $\mathcal{G}g$ means that some goal g should follows by logical inference from the goal base.

The notation $t\theta$ indicates the application of unifier θ to term t. So, for instance, $\mathtt{hd}_e(i)\theta^{\mathtt{hd}(i)}$ is the result of applying the unifier $\theta^{\mathtt{hd}(i)}$ to the top event on the intention.

4 The Environment

A feature of BDI agent programming languages is that BDI programs do not, in general, stand alone but exist within a computational environment. GWENDOLEN

Notation	Description
$\xi.\mathbf{do}(a)$	Executes an action. Returns a unifier.
$\xi.\mathbf{getMessages}(ag)$	Returns a set of new messages for agent ag.
$\xi.\mathbf{Percepts}(ag)$	Returns a set of new perceptions (logical formulae) for agent, ag.
$\xi.done$	True if the environment is incapable of further independent action.

Table 1: Methods implemented by GWENDOLEN Environments

programs expect to interact with environments programmed in Java which implement a specific interface. This means the semantics of some rules will depend upon the environment used. Environments offer various functions – executing agent actions, supplying sets of perceptions etc. The execution of these functions may also induce a change in the environment itself according to its own semantics.

We represent the environment as ξ. Table 1 summarises the functions that all environments are required to offer by the GWENDOLEN semantics. Some environments only change when one of these functions is called but others may be independently dynamic (e.g., because other agents, not programmed in GWENDOLEN are acting in them). We therefore also allow a transition relation on environments ξ: $\xi \rightarrow_\xi \xi'$ and represent the transitions caused by the functions in table 1 as $\xi \xrightarrow{\mathbf{do}(a)}_\xi \xi'$, $\xi \xrightarrow{\mathbf{getMessages}}_\xi \xi'$ and $\xi \xrightarrow{\mathbf{Percepts}(ag)}_\xi \xi'$. $done$ does not change the environment.

5 Multi-Agent System Semantics, Scheduling, Reasoning Cycle

A GWENDOLEN agent is executed as part of a multi-agent system which includes an environment and a *scheduler*. The scheduler is specific to the application and so its policy for the order in which agents (and where relevant the environment) are executed varies.

We represent the operational semantics of the multi-agent system a set of transition rules. The first rules, $\rightarrow_s$ operate on tuples of the environment, a set of agents and the scheduler and represents how the scheduler chooses the next agent for execution. The agent then transitions through stages in a *reasoning cycle* (represented with $\rightarrow_a$). At each stage in the reasoning cycle specific rules are selected

which cause transitions on the agent (and sometimes also on the environment).

We assume the existence of the following functions: $next\_job(s)$ returns a tuple of an agent (or the environment) and an updated version of the scheduler depending on the scheduler policy; *sleeping(a, s)* returns true if the scheduler lists a as asleep; $sleep(a)$ returns true if the agent's status is that it has no further reasoning at the moment and $sleep(a, s)$ returns an updated scheduler that lists a as sleeping. $\rightarrow_a^*$ represents the transitive closure of the semantics on an agent's reasoning cycle so $\langle \xi, a, \mathbf{A} \rangle \rightarrow_a^* \langle \xi', a', \mathbf{F} \rangle$ represents the effect of a run of the agent's reasoning cycle (from stage **A** to **F** – see below) on both the agent and the environment. $\xi \rightarrow_\xi \xi'$ represents an update of the environment according to its own semantics (not considered here).

The following rules represent the operation of the scheduler.

$$\frac{\neg \xi.done \quad next\_job(a) = \langle \xi, s' \rangle \quad \xi \rightarrow_\xi \xi'}{\langle \xi, A, s \rangle \rightarrow_s \langle \xi', A, s' \rangle} \tag{4}$$

$$\frac{\begin{array}{c} \exists a \in A. \neg sleeping(a, s) \quad next\_job(s) = \langle a, s' \rangle \\ \langle \xi, a, \mathbf{A} \rangle \rightarrow_a^* \langle \xi', a', \mathbf{F} \rangle \quad \neg sleep(a') \end{array}}{\langle \xi, A, s \rangle \rightarrow_s \langle \xi', A[a \backslash a'], s' \rangle} \tag{5}$$

$$\frac{\begin{array}{c} \exists a \in A. \neg sleeping(a, A) \quad next\_job(s) = \langle a, s' \rangle \\ \langle \xi, a, \mathbf{A} \rangle \rightarrow_a^* \langle \xi', a', \mathbf{F} \rangle \quad sleep(a') \end{array}}{\langle \xi, A, s \rangle \rightarrow_s \langle \xi', A[a \backslash a'], sleep(a', s') \rangle} \tag{6}$$

It should be noted that, among other things, $next\_job(s)$ can change the internal state of the scheduler, for instance altering the set of agents marked as sleeping if, for instance, new perceptions are available in the environment that might mean the agent now has something to do.

The GWENDOLEN reasoning cycle is a set consisting of size stages (**A, B, C, D, E,** and **F**). Each stage is a list of rules which are discussed in section 6. The agent reasoning cycle transitions, $\rightarrow_a$, by picking the first applicable rule, r, from the list in the current reasoning stage, RS, transitioning the agent (and in some cases environment) according to the rule $\rightarrow_r$ and then moving the reasoning cycle on according to the function $next$ (see (9)).

$$\frac{\exists r \in rules(RS).\ \langle \xi, a \rangle \rightarrow_r \langle \xi', a' \rangle}{\langle \xi, a, RS \rangle \rightarrow_a \langle \xi', a', next(a', RS) \rangle} \tag{7}$$

$$\frac{\neg \exists r \in rules(RS).\ \langle \xi, a \rangle \rightarrow_r \langle \xi', a' \rangle}{\langle \xi, a, RS \rangle \rightarrow_a \langle \xi, a, next(a, RS) \rangle} \tag{8}$$

A GWENDOLEN agent is a tuple $\langle ag, i, I, P, Pl, B, R, In, Out, S\rangle$ of an identifier, current intention, intention set, plan library, applicable plan set, belief base, rule base, inbox, outbox and sleep flag (more in this in section 6). The definition of $next$ in (9) sometimes uses the current intention, i, and intention set, I, to compute the next reasoning stage. In these cases we represent the agent a as $\langle \ldots i \ldots\rangle$ or $\langle \ldots i, I \ldots\rangle$ as appropriate.

$$
\begin{aligned}
next(\langle \ldots i, I \ldots\rangle, \mathbf{A}) &= \begin{cases} \mathbf{E} & i = [] \wedge \forall i' \in I.\, \texttt{is\_suspended}(i') \\ \mathbf{B} & i \neq [] \vee \exists i' \in I.\, \neg\texttt{is\_suspended}(i') \end{cases} \\
next(a, \mathbf{B}) &= \mathbf{C} \\
next(\langle \ldots i \ldots\rangle, \mathbf{C}) &= \begin{cases} \mathbf{E} & i = [] \\ \mathbf{D} & i \neq [] \end{cases} \\
next(a, \mathbf{D}) &= \mathbf{E} \\
next(a, \mathbf{E}) &= \mathbf{F} \\
next(a, \mathbf{F}) &= \mathbf{A}
\end{aligned}
\tag{9}
$$

where $\texttt{is\_suspended}(i)$ is true if the intention, i, is suspended.

6 Stage Rules: The Agent Reasoning Cycle

We represent an agent as a tuple $\langle ag, i, I, P, Pl, B, R, In, Out, S\rangle$ where:

- ag is a unique identifier for the agent (it's name);
- i is the current intention (see section 2); Note that there can be no current intention which we will indicate with the expression $i = null$.
- I is a stack of intentions $\{i, i', ..\}$;
- P is an ordered list of the agent's plans (see section 3);
- Pl is a set of currently applicable plans (see section 3);
- B is a set of the agent's beliefs which are pairs of ground first-order formulae and a string indicating the *source* of the belief. In GWENDOLEN all beliefs are automatically assigned the source `self` unless they are acquired by perception in which case they are assigned the source `percept`;
- R is a set of Prolog-style rules used in reasoning;

- In is the agent inbox. Elements of inbox have the form $\downarrow^{id,ilf}\ m$ where id is the identifier of the sender, ilf is the illocutionary force of the message and can be *tell*, *perform*, or *achieve*, and m is the message content, a ground first-order formula.
- Out is the agent outbox. Messages in this set have the format $\uparrow^{id,ilf}\ m$ where id is the identifier of the recipient, ilf is the illocutionary force and m is the message content, a ground first-order formula.
- S is a boolean indicating whether the agent should be *slept* by the scheduler or not.

In its initial state the current intention is $null$, the intention set consists of one intention for each of the initial goals provided by the programmer. These intentions are of the form $(\texttt{start}, +!_{\tau_g} g, \emptyset)$ where `start` is a special event used for intentions with no specific trigger. Its plan library is a set of plans provided by a programmer. The applicable plans are empty. The belief base and rule base are as defined by the programmer. The inbox and outbox are empty and the sleep flag is false.

Many of the transition rules make a check on a deed to see what type it is (e.g. the addition of a belief, the deletion of a goal). We represent these checks implicitly using the notation shown in table 2. Many of the rules also check intentions for various properties and manipulate them. Table 3 summarises various operations on intentions that are used in the rules.

It is generally unwieldy to present the full agent tuple in the description of a transition rule. As a result we restrict ourselves to presenting only those parts of the intention that are changed by the rule as we did in (9).

We now discuss each stage of the reasoning cycle in turn.

6.1 Stage A

Stage A of the GWENDOLEN reasoning cycle consists of a list of three rules which are focused around managing intention selection:

[`select_intention`, `sleep`, `drop_intention`]

Select Intention (`select_intention`)

$$\frac{\begin{array}{c}\neg\texttt{empty}(i) \quad \neg\texttt{locked}(i) \quad \exists i'' \in I \cup \{i\}.\ \neg\texttt{is\_suspended}(i'') \\ \mathcal{S}_{\text{int}}(I \cup \{i\}) = (i', I') \quad \texttt{hd}_e(i') \neq -!_{\tau_g} g \vee \texttt{hd}_d(i') \neq \epsilon\end{array}}{\langle \xi, \langle \ldots i, I \ldots \rangle\rangle \rightarrow_{\texttt{select\_intention}} \langle \xi, \langle \ldots i', I' \ldots \rangle\rangle} \tag{10}$$

a	An action.
b	A belief.
$+b$	A belief addition.
$-b$	A belief removal.
$b\{source\}$	A belief, from source $source$.
$!_\tau g$	A goal of type τ.
$+!_\tau g$	A goal addition.
$-!_\tau g$	A goal drop.
$\times!_\tau g$	A goal which can't be planned.
`lock`	An lock.
`unlock`	An unlock.
$\uparrow^{ag,ilf}\ m$	A message m sent to ag.
$\downarrow^{ag,ilf}\ m$	A message m received from ag.
$\top$	An structure who's logical content is trivially true.
ϵ	A special marker indicating that some event has no plan yet.

Table 2: Notations for deed type checks

Notation	Description
$\mathtt{U}_\theta$	Compose a unifier with the top unifier on the intention.
$\mathtt{empty}(i)$	The deed stack of the intention is empty.
$\mathtt{events}(i)$	The stack of events associated with intention i.
$\mathtt{hd}_e(i)$	The top event on the intention.
$\mathtt{hd}_d(i)$	The top deed on the intention.
$\theta^{\mathtt{hd}(i)}$	The top unifier on the intention.
`@`	Add a new event, deed stack, and unifier to the top of the intention.
$;_p$	Add a new event, deed, and unifier as the top row of the intention.
$\mathtt{tl}_i(i)$	Drop the top row of the intention.
$\mathtt{drop}_E(e, i)$	Drop all rows in the intention above and including the first appearance of e as a trigger.
$\mathtt{lock}(i)$	Mark the intention as locked.
$\mathtt{locked}(i)$	The intention is locked.
$\mathtt{suspend}(i)$	Mark the intention as suspended.
$\mathtt{is\_suspended}(i)$	The intention is suspended.
$\mathtt{unlock}(i)$	Mark the intention as unlocked.

Table 3: Operations on Intentions

$$\frac{\neg\texttt{empty}(i) \quad \texttt{locked}(i) \quad \texttt{hd}_e(i) \neq -!_{\tau_g} g \vee \texttt{hd}_d(i) \neq \epsilon \quad \exists i'' \in I \cup \{i\}.\, \neg\texttt{is\_suspended}(i'')}{\langle \xi, \langle \ldots i, I \ldots \rangle \rangle \rightarrow_{\texttt{select\_intention}} \langle \xi, \langle \ldots i, I \ldots \rangle \rangle} \tag{11}$$

where $\texttt{empty}(i)$ is true if intention i has an empty deed stack, $\texttt{locked}(i)$ is true if intention i is locked, and $\texttt{is\_suspended}(i)$ is true if intention i is suspended. Table 3 summarises all the operations on intentions.

This rule has two cases, one for when the current intention isn't locked and one for when it is. When the intention isn't locked the system uses the application specific selection function $\mathcal{S}_{\text{int}}$ to pick a new current intention (by default this treats the intention set I as a LIFO queue and selects the first unsuspended intention from the queue). The rule is inapplicable if the current intention is empty or the selected intention's trigger is a drop goal event.

Sleep (`sleep`)

$$\frac{(i = null \vee \texttt{empty}(i) \vee \texttt{is\_suspended}(i)) \quad \forall i' \in I.\, \texttt{is\_suspended}(i')}{\langle \xi, \langle \ldots i, I, \ldots S \rangle \rangle \rightarrow_{\texttt{sleep}} \langle \xi, \langle \ldots, i, I, \ldots \top \rangle \rangle} \tag{12}$$

Table 3 summarises all the operations on intentions such as `empty` etc,.

This rule sets an agent's sleep flag if all its intentions are empty or suspended. The agent will then be marked as sleeping by the scheduler once the reasoning cycle is concluded.

Drop Intention (`drop_intention`)

$$\frac{i \neq null \quad \texttt{empty}(i) \quad I \neq \emptyset \quad \mathcal{S}_{\text{int}} I = (i', I')}{\langle \xi, \langle \ldots i, I \ldots \rangle \rangle \rightarrow_{\texttt{drop\_intention}} \langle \xi, \langle \ldots i', I' \ldots \rangle \rangle} \tag{13}$$

Table 3 summarises all the operations on intentions such as `empty` etc,.

This rule drops the intention i if it is empty and selects a new current intention from the intention set. The additional $i \neq null$ is necessary since a few rules can leave the agent state with no current intention.

6.2 Stage B

Stage B of the GWENDOLEN reasoning cycle consists of a list of two rules based on generating a set of applicable plans: [`generate_plan`, `no_plan`]

Generate Plan (`generate_plan`**)**

$$\frac{\mathbf{appPlans}(i) \neq \emptyset}{\langle \xi, \langle \ldots i, Pl \ldots \rangle \rangle \rightarrow_{\texttt{generate\_plan}} \langle \xi, \langle \ldots i, \mathbf{appPlans}(i) \ldots \rangle \rangle} \tag{14}$$

appPlans is as described in section 3.

No Applicable Plans (`no_plan`**)**

$$\frac{\mathbf{appPlans}(i) = \emptyset \quad \texttt{hd}_e(i) = +!_\tau g}{\langle \xi, \langle \ldots i, Pl \ldots \rangle \rangle \rightarrow_{\texttt{no\_plan}} \langle \xi, \langle \ldots i, [(\texttt{x}!_\tau g, [\epsilon], \theta^{\texttt{hd}(i)})] \ldots \rangle \rangle} \tag{15}$$

$$\frac{\mathbf{appPlans}(i) = \emptyset \quad \neg\texttt{hd}_e(i) = +!_\tau g}{\langle \xi, \langle \ldots i, Pl \ldots \rangle \rangle \rightarrow_{\texttt{no\_plan}} \langle \xi, \langle \ldots i, [(\texttt{hd}_e(i), [], \emptyset)] \ldots \rangle \rangle} \tag{16}$$

$\mathbf{appPlans}(i)$ is empty if there is no plan applicable to the current intention. This rule differentiates between whether the intention trigger is a goal commitment (in which case the rule creates an applicable plans consisting of an unplanned "problem goal" event ($\texttt{x}!_\tau g$) (which might, for instance, be responded to by suspending the intention until the agent's beliefs have changed and some plan does become applicable). Otherwise it generates an applicable plan with an empty deed stack. This will have the effect of removing the top row of the intention and replacing it by nothing – i.e., it ignores the event that had no applicable plan for handling it. The reasoning behind this is that such events (notifications of beliefs acquired or dropped generally only require a planning response in special cases and can normally be ignored).

6.3 Stage C

Stage C of the GWENDOLEN reasoning cycle consists of a list of a single rule for modifying the current intention according to the applicable plan: [`apply_plan`]

Apply Plan (`apply_plan`**)**

$$\frac{(e, Ds, \theta) = \mathcal{S}_{\text{plan}}(Pl)}{\langle \xi, \langle \ldots i \ldots Pl \ldots \rangle \rangle \rightarrow \langle \xi, \langle \ldots (e, Ds, \theta) \texttt{ @ } \texttt{tl}_i(i) \ldots \emptyset \ldots \rangle \rangle} \tag{17}$$

where $\texttt{tl}_i(i)$ represents intention, i, with its top row removed and (e, Ds, θ) `@` $\texttt{tl}_i(i)$ represents the applicable plan (e, Ds, θ) expanded and added to the top of the intention, i in place of its top row as described in section 3. Table 3 summarises all the operations on intentions such as `empty` etc,.

This rule selects a plan from the agent's applicable plans as determined by the application specific $\mathcal{S}_{\text{plan}}$ (by default this is the first applicable plan found in the plan library and, where a unifier is required, this is the first returned by checking against the agents internal state (this lists beliefs and goals, etc., in alphabetical order)). The plan is represented as a tuple of the trigger event, the plan's deed stack and unifier. The top row of the current intention is dropped and the applicable plan is "glued" in its place.

6.4 Stage D

Stage D of the GWENDOLEN reasoning cycle consists of a list of rules for processing the top deed on the current intention:

[`empty`, `add_achieve_goal`, `add_perform_goal`, `drop_goal`, `add_belief`, `drop_belief`, `lock_unlock`, `wait_for`, `problem_goal`, `action`, `send`, `null`]

Handle Empty Deed Stack (`empty`)

$$\frac{\texttt{empty}(i)}{\langle \xi, \langle \ldots i \ldots \rangle \rightarrow_{\texttt{empty}} \langle \xi, \langle \ldots i \ldots \rangle \rangle} \tag{18}$$

Table 3 summarises all the operations on intentions such as `empty` etc,.

This rule does nothing if the current intention's deed stack is empty (which can occur if there is no plan for handling the intention's trigger event). This leaves the intention unchanged and it will be removed during the select intention phase (Stage A).

Handle Add Achieve Goal (`add_achieve_goal`)

$$\frac{\texttt{hd}_d(i)\theta^{\texttt{hd}(i)} = +!_a g \quad B, R \models g, \theta_g}{\begin{array}{c}\langle \xi, \langle \ldots i \ldots B, R \ldots \rangle \rangle \\ \rightarrow_{\texttt{add\_achieve\_goal}} \\ \langle \xi, \langle \ldots \texttt{tl}_i(i)\ \texttt{U}_\theta\ \theta_g \ldots B, R \ldots \rangle \rangle\end{array}} \tag{19}$$

$$\frac{\texttt{hd}_d(i)\theta^{\texttt{hd}(i)} = +!_a g \quad B, R \not\models g}{\begin{array}{c}\langle \xi, \langle \ldots i \ldots B, R \ldots \rangle \rangle \\ \rightarrow_{\texttt{add\_achieve\_goal}} \\ \langle \xi, \langle \ldots (+!_a g, \epsilon, \theta^{\texttt{hd}(i)});_p i \ldots B, R \ldots \rangle \rangle\end{array}} \tag{20}$$

where $B, R \models g, \theta_g$ means that the formula g (which is the goal with the top unifier from the intention applied to it) follows using Prolog-style reasoning from the agent's belief base when the additional unifier θ_g is applied. $\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta_g$ indicates the union of unifier θ_g with the unifier on the top of the intention $\mathtt{tl}_i(i)$. $(e, d, \theta);_p i$ represents the addition of a row (e, d, θ) to the top of an intention i. Table 3 summarises all the operations on intentions such as `empty` etc,.

GWENDOLEN recognises two types of goal, *achieve* goals and *perform* goals (goal types a and p respectively). This rule handles the commitment to an achieve goal. An achieve goal is one that triggers a plan if it not already believed but does no more than set a unifier if it is. If it is to trigger a plan, then we register the commitment to planning the goal as an event on the top of the intention stack. In this case the top row of the intention is not dropped so the deed intending a commitment to the goal remains. This means that if, after execution of the plan, the goal is not achieved then it will be replanned.

Handle Add Perform Goal (`add_perform_goal`)

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = +!_p g}{\begin{array}{c}\langle \xi, \langle \ldots i \ldots \rangle\rangle \\ \rightarrow_{\mathtt{add\_perform\_goal}} \\ \langle \xi, \langle \ldots (+!_p g, \epsilon, \theta^{\mathtt{hd}(i)});_p (\mathtt{hd}_e(i), null, \theta^{\mathtt{hd}(i)});_p \mathtt{tl}_i(i) \ldots \rangle\rangle\end{array}} \tag{21}$$

where $(e, d, \theta);_p i$ represents the addition of a row (e, d, θ) to the top of an intention i. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

Perform goals always trigger planning but are not replanned if they fail to achieve some state of the world. This being the case we replace the top deed (the request to commit to the goal) on the intention with $null$ so that this is automatically processed once the system reaches that row of the intention. We then add a new top row with the trigger event of the new goal and a no plan yet deed.

Handle Drop Goal (`drop_goal`)

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = -!_{\tau_g} g \quad \exists e \in \mathtt{events}(i).\mathit{unify}(e, +!_{\tau_g} g)}{\langle \xi, \langle \ldots i \ldots \rangle\rangle \rightarrow_{\mathtt{drop\_goal}} \langle \xi, \langle \ldots \mathtt{tl}_i(\mathtt{drop}_E(e, i)) \ldots \rangle\rangle} \tag{22}$$

where $\mathit{unify}(e_1, e_2)$ indicates that two events can be unified. $\mathtt{drop}_E(e, i)$ is a function that recurses through an intention dropping every row after the first occurrence of e – i.e. it prunes the intention back to the point where the event first occurred. Table 3 summarises all the operations on intentions such as `events` etc,.

This rule searches the current intention for the most earliest add goal event that unifies with the goal to be dropped and then deletes all rows on the intention above that. It then deletes the new top row which will be he one that contains the instruction to commit to the goal (if an achieve goal) or $null$ (if a perform goal).

Handle Add Belief (`add_belief`**)**

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = +b}{\begin{array}{c}\langle \xi, \langle \ldots i, I, B \ldots \rangle\rangle \rightarrow_{\mathtt{add\_belief}} \\ \langle \xi, \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)}, \mathit{unsuspend}(I, b) \cup \mathtt{new}(+b, \epsilon, \emptyset), \\ B \cup \{b\}, \ldots \rangle\rangle\end{array}} \tag{23}$$

where $\mathit{unsuspend}(I, b)$ unsuspends all suspended intentions in I that are waiting for b to become true. $\mathtt{new}(e, d, \theta)$ creates a new intention from an event, deed and unifier. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

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.

Handle Drop Belief (`drop_belief`**)**

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = -b \quad B^1 = \{b' | b' \in B \wedge \mathit{unify}(b', b)\}}{\begin{array}{c}\langle \xi, \langle \ldots i, I, B \ldots \rangle\rangle \rightarrow_{\mathtt{drop\_belief}} \\ \langle \xi, \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)}, I \cup \mathtt{new}(-b, \epsilon, \emptyset), B \backslash B^1, \ldots \rangle\rangle\end{array}} \tag{24}$$

where $\mathit{unify}(b', b)$ means that b' and b unify with each other. $\mathtt{new}(e, d, \theta)$ creates a new intention from an event, deed and unifier. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

This rule drops a belief from the belief base. At the same time it generates a new intention containing the event that the belief has been dropped. Appropriate handling of this event can allow the agent to form plans in reaction to it.

Handle Lock and Unlock (`lock_unlock`**)**

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = \mathtt{lock}}{\langle \xi, \langle \ldots i \ldots \rangle\rangle \rightarrow_{\mathtt{lock\_unlock}} \langle \xi, \langle \ldots, \mathtt{lock}(\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)}) \ldots \rangle\rangle} \tag{25}$$

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = \mathtt{unlock}}{\langle \xi, \langle \ldots i \ldots \rangle\rangle \rightarrow_{\mathtt{lock\_unlock}} \langle \xi, \langle \ldots \mathtt{unlock}(\mathtt{tl}_i(i)\mathtt{U}_\theta\theta^{\mathtt{hd}(i)}) \ldots \rangle\rangle} \tag{26}$$

Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

This allows an intention to be "locked" as the current intention, for instance to allow a complete sequence of belief changes be processed before any other reasoning takes place. Once finished the intention has to be unlocked.

Handle Wait For

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = *b \quad B, R \models b, \theta_b}{\begin{array}{c}\langle \xi, \langle \ldots i \ldots B, R \ldots \rangle\rangle \rightarrow_{\mathtt{wait\_for}} \\ \langle \xi, \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ (\theta^{\mathtt{hd}(i)} \cup \theta_b) \ldots B, R \ldots \rangle\rangle\end{array}} \tag{27}$$

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = *b \quad B, R \not\models b, \theta_b \quad \exists i' \in I.\, \neg\mathtt{is\_suspended}(i')}{\begin{array}{c}\langle \xi, \langle \ldots i, I, B, R \ldots \rangle\rangle \rightarrow_{\mathtt{wait\_for}} \\ \langle \xi', \langle \ldots \mathtt{suspend}(i\theta^{\mathtt{hd}(i)}), I, B, R \ldots \rangle\rangle\end{array}} \tag{28}$$

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = *b \quad B, R \not\models b, \theta_b \quad \forall i' \in I.\mathtt{is\_suspended}(i')}{\begin{array}{c}\langle \xi, \langle \ldots i, I, B \ldots In \ldots S \rangle\rangle \rightarrow_{\mathtt{wait\_for}} \\ \langle \xi', \langle \ldots null, I \cup \{\mathtt{suspend}(i\theta^{\mathtt{hd}(i)})\}, B, R \ldots \top \rangle\rangle\end{array}} \tag{29}$$

where $B, R \models b, \theta_b$ means that the formula b follows using Prolog-style reasoning from the agent's belief base and Prolog rule-base when the additional unifier θ_b is applied. $\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ (\theta^{\mathtt{hd}(i)} \cup \theta_b)$ indicates the union of unifier $(\theta^{\mathtt{hd}(i)} \cup \theta_b)$ with the unifier on the top of the intention $\mathtt{tl}_i(i)$. $\mathtt{suspend}(i)$ suspends an intention. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

If an intention is waiting for some belief, b, to become true then if that belief is now true the intention continues processing. Otherwise the intention is suspended. If all intentions are suspended then the agent is told to sleep at the next opportunity.

Ignore Unplanned Problem Goal (`problem_goal`**)**

$$\frac{\mathtt{hd}_e(i) = \mathtt{x!}_{\tau_g} g \quad \mathtt{hd}_d(i) = \epsilon}{\langle \xi, \langle \ldots i \ldots \rangle\rangle \rightarrow_{\mathtt{problem\_goal}} \langle \xi, \langle \ldots i \ldots \rangle\rangle} \tag{30}$$

Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

This rule ignores an unplanned problem goal. It simply does nothing but allows the reasoning cycle of the agent to continue processing on the assumption that planning of the goal may become possible later.

Handle General Action (`action`)

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = a \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad \xi.\mathbf{do}(a) = \theta_a \quad a \neq \uparrow^{ag,ilf}\ m}{\langle \xi, \langle \ldots i \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ (\theta^{\mathtt{hd}(i)} \cup \theta_a) \ldots \rangle \rangle} \tag{31}$$

$$\frac{\begin{array}{c}\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = a \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad \neg\xi.\mathbf{do}(a) \quad \mathtt{hd}_e(i) = +!_{\tau_g} g \\ a \neq \uparrow^{ag,ilf}\ m\end{array}}{\langle \xi, \langle \ldots i \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots (\mathtt{x}!_{\tau_g} g, \epsilon, \theta^{\mathtt{hd}(i)} \cup \theta_a);_p i \ldots \rangle \rangle} \tag{32}$$

$$\frac{\begin{array}{c}\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = a \quad \xi \xrightarrow{\mathbf{do}(a)} \xi' \quad \neg\xi.\mathbf{do}(a) \quad \mathtt{hd}_e(i) \neq +!_{\tau_g} g \\ a \neq \uparrow^{ag,ilf}\ m\end{array}}{\langle \xi, \langle \ldots i \ldots \rangle \rangle \rightarrow_{\mathtt{action}} \langle \xi', \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)} \ldots \rangle \rangle} \tag{33}$$

where $\xi.\mathbf{do}(a\theta^{\mathtt{hd}(i)}) = \theta_a$ means that the environment successfully executes a returning unifier θ_a. $\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta_g$ indicates the union of unifier θ_g with the unifier on the top of the intention $\mathtt{tl}_i(i)$. $(e, d, \theta);_p i$ represents the addition of a row (e, d, θ) to the top of an intention i. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

In this rule, the agent attempts the action (unless it is a send action – $\uparrow^{ag,ilf}\ m$). If the action succeeds it returns a unifier and the environment updates. Otherwise, if the trigger event at the top of the intention is a goal then this is generates a problem goal event for handling by some plan.

Handle Send Action (`send`)

$$\frac{\xi \xrightarrow{\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag})} \xi' \quad \mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = \uparrow^{ag',ilf}\ m \quad \xi.\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag}) = \theta_a}{\begin{array}{c}\langle \xi, \langle ag, i, I \ldots Out \ldots \rangle \rangle \rightarrow_{\mathtt{send}} \\ \langle \xi', \langle ag, \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ (\theta^{\mathtt{hd}(i)} \cup \theta_a), I \cup \{\mathtt{new}(+\uparrow^{ag',ilf}\ m, \epsilon, \emptyset)\}, \\ \ldots Out \cup \{\uparrow^{ag',ilf}\ m\} \ldots \rangle \rangle\end{array}} \tag{34}$$

$$\frac{\begin{array}{c}\xi \xrightarrow{\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag})} \xi' \\ \mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = \uparrow^{ag',ilf}\ m \quad \neg\xi.\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag}) \quad \mathtt{hd}_e(i) = +!_{\tau_g} g\end{array}}{\begin{array}{c}\langle \xi, \langle ag, i, I \ldots Out \ldots \rangle \rangle \rightarrow_{\mathtt{send}} \\ \langle \xi', \langle ag, (\mathtt{x}!_{\tau_g} g, \epsilon, \theta^{\mathtt{hd}(i)} \cup \theta^{\mathtt{hd}(i)});_p i, I \ldots Out \ldots \rangle \rangle\end{array}} \tag{35}$$

$$\frac{\xi \xrightarrow{\mathbf{do}(\uparrow^{ag',ilf} m_{ag})} \xi' \quad \mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = \uparrow^{ag',ilf}\ m \quad \neg\xi.\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag}) \quad \neg\mathtt{hd}_e(i) = +!_{\tau_g} g}{\begin{array}{c}\langle \xi, \langle ag, i, I \ldots Out \ldots \rangle\rangle \rightarrow_{\mathtt{send}} \\ \langle \xi', \langle ag, \mathtt{tl}_i i\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)}, I \ldots Out \ldots \rangle\rangle\end{array}} \tag{36}$$

where $\xi.\mathbf{do}(\uparrow^{ag',ilf}\ m_{ag})$ is the environment executing the sending of a message, m, from ag to ag' with illocutionary force ilf. $\mathtt{new}(e, d, \theta)$ creates a new intention from an event, deed and unifier (in this case the event is the sending of a message to ag'). $\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta_g$ indicates the union of unifier θ_g with the unifier on the top of the intention $\mathtt{tl}_i(i)$. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

This rule behaves much as the rule for handling general actions with the exception that when a send action succeeds a new intention is generated registering the event that a message was sent and the message itself is added to the agent's outbox.

Handle Null (`null`)

$$\frac{\mathtt{hd}_d(i)\theta^{\mathtt{hd}(i)} = null}{\langle \xi, \langle \ldots i \ldots \rangle\rangle \rightarrow_{\mathtt{null}} \langle \xi, \langle \ldots \mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta^{\mathtt{hd}(i)} \ldots \rangle\rangle} \tag{37}$$

where $\mathtt{tl}_i(i)\ \mathtt{U}_\theta\ \theta_g$ indicates the union of unifier θ_g with the unifier on the top of the intention $\mathtt{tl}_i(i)$. Table 3 summarises all the operations on intentions such as $\theta^{\mathtt{hd}(i)}$ etc,.

The null action is used as a place holder to note, when a perform goal has been committed to, a record of the relevant trigger event in an intention stack. This rule simply ignores the null action when it is encountered and deletes that row from the intention.

6.5 Stage E

Stage E of the GWENDOLEN reasoning cycle consists of a list of a single rule for handling perception: [`perceive`]

Perceive

$$\frac{\begin{array}{c}\xi \xrightarrow{\mathbf{Percepts}(ag)}_{\xi} \xi_1 \quad \xi_1 \xrightarrow{\mathbf{getMessages}(ag)}_{\xi} \xi' \\ P = \xi.\mathbf{Percepts}(ag) \\ OP = \{b \mid b \in B \backslash P \wedge \mathit{source\_of}(b) = \mathtt{percept}\} \\ P \backslash B \cup OP \cup \xi.\mathbf{getMessages}(ag) \neq \emptyset\end{array}}{\begin{array}{c}\langle \xi, \langle \ldots I, B \ldots In \ldots S \rangle\rangle \rightarrow_{\mathtt{perceive}} \\ \langle \xi', \langle \ldots \\ I \cup \{\mathtt{new}(\mathtt{start}, +b, \emptyset) \mid b \in P \backslash B\} \cup \{\mathtt{new}(\mathtt{start}, -b, \emptyset) \mid b \in OP\}, \\ B \ldots In \cup \xi.\mathbf{getMessages}(ag) \ldots \top \rangle\rangle\end{array}} \tag{38}$$

$$\frac{\begin{array}{c}\xi \xrightarrow{\mathbf{Percepts}(ag)}_{\xi} \xi_1 \quad \xi_1 \xrightarrow{\mathbf{getMessages}(ag)}_{\xi} \xi' \\ P = \xi.\mathbf{Percepts}(ag) \\ OP = \{b \mid b \in B \backslash P \wedge \mathit{source\_of}(b) = \mathtt{percept}\} \\ P \backslash B \cup OP \cup \xi.\mathbf{getMessages}(ag) = \emptyset\end{array}}{\langle \xi, \langle \ldots I, B \ldots In \ldots \rangle\rangle \rightarrow_{\mathtt{perceive}} \langle \xi', \langle \ldots I, B \ldots In \ldots \rangle\rangle} \tag{39}$$

where $\xi.\mathbf{Percepts}(ag)$ returns a set of new beliefs to the agent which are all annotated as coming from the source `percept`. $\mathit{source\_of}(b)$ returns the source of a belief b. $\xi.\mathbf{getMessages}(ag)$ returns a set of new messages to the agent. $\mathtt{new}(e, d, \theta)$ creates a new intention from an event, deed and unifier. In this case the event is a special distinguished event `start` which is used to indicate an intention with no trigger.

This rule adds all messages to the inbox. It also creates new intentions, each triggered by the event of acquiring or losing a percept. A key part of the working of the rule depends on AIL's annotation of all beliefs in the belief base with a source and its use of a special annotation for beliefs whose source is perception. If some change is bought about either to the agent's inbox or to its intentions then the agent's sleep flag is set to true (i.e., the agent will not sleep at the end of this reasoning cycle).

Note that in the EASS variant of GWENDOLEN the perception rule also updates the belief base directly, unlike this rule which creates intentions to update the belief base and leaves these to later reasoning cycles for execution.

6.6 Stage F

Stage F of the GWENDOLEN reasoning cycle consists of a list of a single rule for processing messages in the inbox: [messages]

Handle Messages (messages)

$$\frac{}{\langle \xi, \langle \ldots I \ldots In \ldots \rangle \rangle \rightarrow_{\texttt{messages}} \langle \xi, \langle \ldots I \cup \{\texttt{new}(+received(ag, ilf, m), \epsilon, \emptyset) \mid \downarrow^{ag,ilf} \ m \in In\} \ldots [] \ldots \rangle \rangle} \quad (40)$$

where $\texttt{new}(e, d, \theta)$ creates a new intention from an event, deed and unifier. In this case the event is a belief that the agent has received message m from agent, ag with illocutionary force, ilf. It is up to the programmer to decide how messages should be handled, there is no default mechanism for handling messages of any particular illocutionary force (unlike many BDI languages which give a specific semantics to such constructs)

This rule does not poll the environment for messages. It takes all messages currently in an agent's inbox and converts them to intentions (triggered by a perception that the message has been received), emptying the inbox in the process. It should be noted that it does not store the message anywhere once the inbox is emptied. It assumes that some plan will act appropriately to the message received event. If this does not happen then the message content may be lost.

Acknowledgments. This research was funded by EPSRC via the research project "Verifiable Autonomy" at Liverpool (EP/L024845). Further details of the project are available at http://wordpress.csc.liv.ac.uk/va

References

[1] L. A. Dennis and B. Farwer. Gwendolen: A BDI Language for Verifiable Agents. In *AISB'08 Workshop on Logic and the Simulation of Interaction and Reasoning*. AISB, 2008.