A Formal Semantics for Brahms Technical Report*

Richard Stocker[1], Maarten Sierhuis[2], Louise Dennis[1], Clare Dixon[1], Michael Fisher[1]

[1] Department of Computer Science, University of Liverpool, UK
[2] PARC, Palo Alto, USA
Contact: Richard Stocker (R.S.Stocker@liverpool.ac.uk)

1 Introduction

In this paper we present the reader with 41 rules to describe the formal semantics of Brahms. With so many rules it is difficult to get a clear picture of how Brahms operates, so we present the reader with Figures 1 and 2 to help explain the semantics.

To aid the reader we will follow the flow through the Brahms semantics in Figures 1 and 2. It should be noted that this is only the core aspects of the Brahms semantics, functions such as suspension, detectables, variables, etc. have been removed for simplicity. The Figures 1 and 2 have been drawn with a relation to traditional data flow diagrams where rectangles with rounded edges start the data flow, rectangular boxes with sharp edges represent a process, diamond boxes represent a yes/no choice, and an elongated oval represents the termination state. Arrows are used in the diagrams to show how the flow moves from process to process, arrows emanating from a diamond are labelled either yes or no to signify which choice they represent. To help describe the flow through these diagrams the states have been labelled A1-A22 and S1-S8. The agent's states are identified using A and the scheduler's by S. It should be realised that the agent's and the scheduler diagrams are not mutually exclusive, i.e., some agent states require the scheduler to be in a certain state before the agent is moved onto another state. Some process states in the diagrams are shaded, these are to identify that non-determinism can occur in these states, e.g., state A14 is shaded and refers to updating a belief or a fact, the non-determinism here occurs because belief and fact updates are assigned a 'certainty', i.e., a percentage chance that the update will occur.

The scheduler, in Figure 2, starts off by initialising everything from agents, to objects, etc. in state $S1$. During this initialisation the scheduler informs the agents to start executing, the scheduler then moves into $S2$ where it waits for a response from all the agents. The agents, in Figure 1, start off by moving into $A1$ where they initialise themselves, then move on to $A2$ where they then wait for the scheduler. Once the agents have received the command from the scheduler to start executing they move into state $A4$ where they generate a set containing all active thoughtframes. The agent then cycles through states $A4$, $A5$ and $A6$ where it executes all thoughtframes in the set and checks for more thoughtframes to become active until no more thoughtframes are active. The box $A6$ is shaded because the thoughtframes are chosen non-deterministically. The agent then moves into state $A7$ where a set of all active workframes is selected, if this set is empty then the agent moves to $A9$ to set itself as idle. If there are active

* Work partially funded in the UK through EPSRC grants EP/F033567 and EP/F037201.

workframes then the agent moves to state $A11$ where it randomly selects one of these workframes, again $A11$ is shaded due to this random choice. The semantics then check whether the workframe is empty or not in $A10$, if the workframe deed stack is empty then the agent is directed back to state $A4$ to process its thoughtframes. If the workframe deed stack is not empty then it pops the top element off the stack in $A12$. $A13$ then checks if the event is an activity or a conclude. If the event is a conclude then the agent moves to state $A14$ where it processes this conclude, this state is shaded because the belief and fact attributed to this conclude may or may not be updated based upon the belief and fact certainty of the conclude. If the event is an activity then the agent moves to state $A15$ where it selects a duration for this activity between the minimum and maximum value, the box is shaded to represent this non-determinism. The agent then sends this value to the scheduler in state $A16$ and waits for a response from the scheduler. Once the scheduler receives all durations from all the agents it moves to state $S4$ and calculates which is the shortest. If all the agents had found no active workframes in state $A8$ and moved to state $A9$ then they would all have sent the scheduler a duration of -1, if this is the case then the scheduler will be directed to states $S7$ and $S8$ from state $S5$ to terminate the simulation. If the scheduler did find a duration greater than -1 in $S5$ then it moves its clock forward by this duration in state $S6$ and moves back to waiting for a duration from all the agents in state $S2$. The agents will now have received a duration from the scheduler and will move from $A16$ into $A18$, if the scheduler had sent a -1 for the duration then they will move to $A19$ and terminate. When the scheduler sends a duration greater than -1 the agents move into state $A22$ where they check to see whether they have an activity to deduct time from, if they had set themselves idle then they would not have a current activity to do this with. They then process states $A21$ and $A20$ to update their clocks and deduct time from their activities, they are then directed back to state $A10$ to continue popping events off the deed stack. Once the deed stack becomes empty they will be directed from state $A10$ to $A4$ to start processing thoughtframes and eventually move onto the next workframe.

2 Semantics: Notation

The following conventions refer to components of the system, and agent and object states.

Agents: ag is used to express the identity of an *agent*, e.g., ag_{Alex} would represent an agent named $Alex$, while Ag represents the *set* of all agents. When referring to arbitrary agents we use names such as i and j, and when we are referring to the number of agents we use n. For example, when we use the term $\forall ag_i \in Ags$ we are referring to all arbitrary agents in the set of all agents, and when we are using something that requires two arbitrary agents, such as communication, we will say that arbitrary agent ag_i communicates to arbitrary agent ag_j.

Beliefs: b represents the atomic formula of a *belief*, while B represents a *set* of beliefs. In Brahms the overall system may have beliefs which are represented by B_ξ.

Facts: f represents the atomic formula of a *fact*, while F represents a *set* of facts.

Workframes: Workframes are represented as the tuple

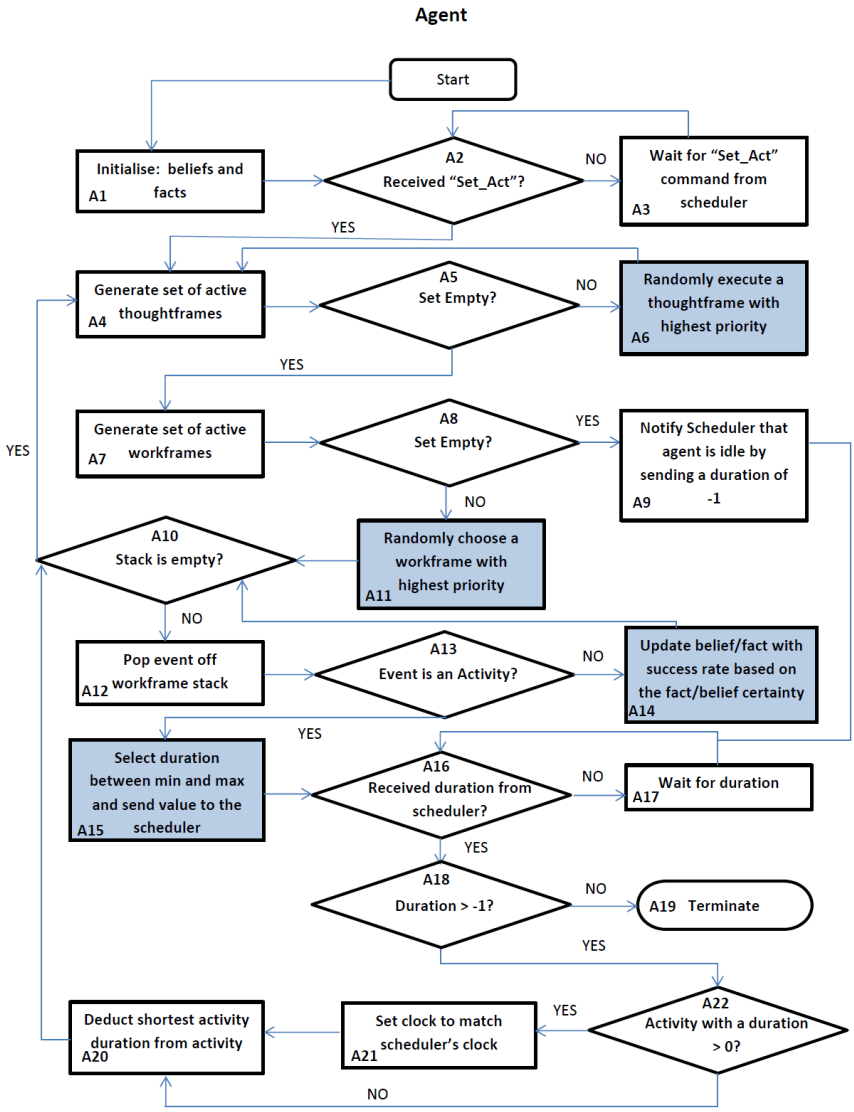

Fig. 1. Overview of a Brahms Agent's Semantics

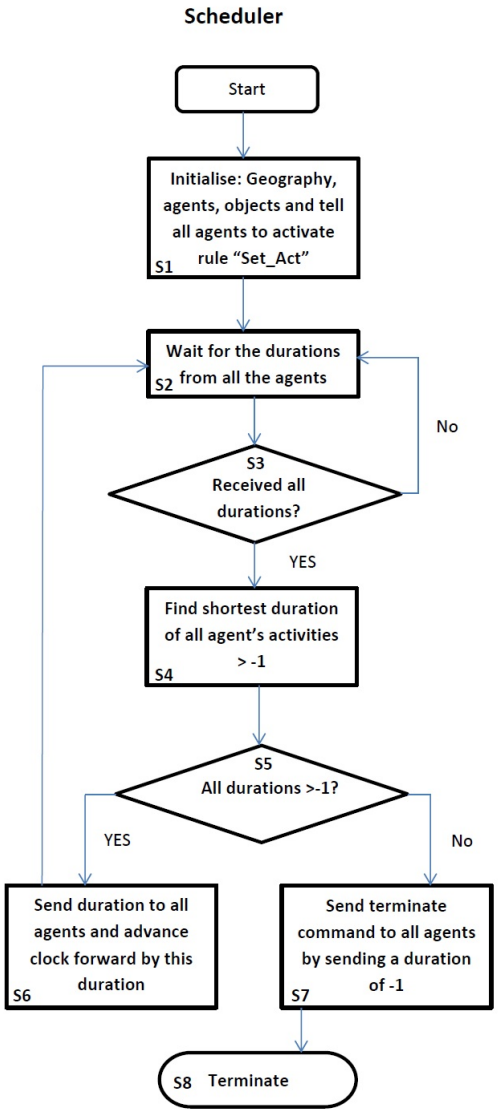

Fig. 2. Overview of the Scheduler's Semantics

$$\langle W^g, W^{pri}, W^r, W^D, W^V, W_{ins} \rangle$$

Where

- W^g is the workframe's guard.
- W^{pri} is the workframe's priority. Priorities are represented in Brahms as a natural number, $\mathbb{N}$, however in this semantics we add decimal values to these numbers to account for priorities of suspended, impassed and current workframes over generic workframes yet to be instantiated.
- W^r is the workframe's repeat variable. The repeat variable can take the values $true$, $false$, and $once$.
- W^D is the workframe's detectables. This is a tuple $\langle d^g, d^{type} \rangle$, where d^g represents the detectables guard condition and d^{type} represents the detectables type; $impasse$, $continue$, $complete$, or $abort$.
- W^V is the workframe's variables, β^V for the current workframe. A single variable is identified using v, each variable has a type which is identifed by v^{type} which can take the values $forone$, $foreach$, and $collectall$.
- $\langle W_0 ... W_n \rangle$ is a set representing instansiations of the workframe. These instantiations are necessary when a workframe contains variables, an instantiation is created for every possible combination of assignments that the variables can have; variables can be assigned to agents, objects and locations.
- $W^{Concludes}$ is used to represent all the conclude statements inside the workframes stack of instructions. This is used when a workframe has been instructed to process only concludes and ignore actvities.

When referring to workframes W refers to any arbitrary workframe, β represents the current workframe, e.g., β^{pri} would refer to the current workframe's priority, and $\mathcal{WF}$ represents a set of workframes. Occasionally to save space in the tuple we represent the first 6 elements of the workframe tuple as W_d, i.e., the workframe's header data. This shortened form of the tuple looks as follows $\langle W_d, W_{ins} \rangle$ where W_{ins} represents the stack of instructions the workframe is to perform, such as concludes and activities.

Thoughtframes: Thoughtframes are represented in a similar fashion except α represents the *current thoughtframe*, TF represents a *set* of thoughtframes, while $\mathcal{T}$ represents any arbitrary *thoughtframe*.

Activities and Concludes: Activities and concludes are broken down into the following types

- `Prim_Act`^{t} is a primitive activity of duration t.
- `Comms(`ag_j, b`)`^{t} is a communication activity to agent j, sending belief b with a duration t.
- `Move(Loc = new)`^{t} is a move activity from the current location `Loc` to the new location `new` t.
- $conclude(b)$ is a conclude asserting the belief b.
- $conclude(f)$ is a conclude asserting the fact f.

Environment: In this semantics additional details outside of the agent's and object's own perceptions are referred to as belonging to the environment. To represent this environment we use the identification ξ.

Time: T represents the time in general, while a specific duration for an activity is represented by t. The time T is always associated with either an agent, object or the environment, e.g., T_i refers to current time of agent ag_i and T_ξ refers to the time of the global clock, or the system clock, in the environment. Time is represented as a natural number, $\mathbb{N}$, with the exception of the termination condition which takes the value of -1.

Stage: The semantics are organised into "stages". Stages refer to the names of the operational semantic rules that may be applicable at that time, wild cards ($*$) are used to refer to multiple rules with identical prefixes. There is also a "*fin*" stage which indicates an agent is ready for the next cycle, and an "$idle$" stage which means it currently has no applicable thoughtframes or workframes. To describe the stage of an agent i we use the notation ag_i^{stage}

Methods: To keep the semantic rules as simple as possible we shorten some actions into Java like method calls. The methods used are as follows:

- $MinTime(\forall ag_i | T_i \in B_\xi)$. This method is used when all the agents have informed the scheduler of when their next activity is due to finish. This method examines all the durations of all the agent's activities and identifies which is the smallest, $\forall ag_i | T_i \in B_\xi$ expresses that the method examines the durations, in the environment's belief base, for all the agents.
- $Max\_Pri()$. This method is used to find the thoughtframe or workframe of the highest priority, e.g., $Max\_Pri(\forall \mathcal{T} \in TF_i | B_i \models \mathcal{T}^g)$ finds the thoughtframe in the set of all thoughtframes such that the thoughtframe's guard condition is met in the agent's belief base.
- $selectVar()$. This method matches all the agents/objects/locations that meet the requirements set out in the workframe or thoughtframe's guard condition and assigns each set of agents etc. to a workframe or thoughtframe instance.
- $Random()$. This method is used to show a that random selection is being made, e.g., $Random(W_0 ... W_n)$ randomly selects a workframe out of the set of workframes $W_0 ... W_n$.
- $concludes(W_1 ... W_n)$. This method is used in the rule Var_all, it takes all the workframes instances $W_1 ... W_n$ and extracts all the conclude statements from it. This rule is needed because a $collectAll$ variable takes all the conclude statements from every instance and processes them at the same time.

3 Semantics: Structure

The operational semantics are broken up into two parts; the scheduler semantic rules and the agents semantic rules. We use a 5-tuple description (shown in Definition 1) to represent the state of the scheduler, a 9-tuple description (shown in Definition 2) to represent the state of the agent, and a transition rule (shown in Definition 3) to show how the states transform. We use first-order logic with set theoretic operations, but restricted to the sets available within the semantic structures, to express when the rule is active and to state how the tuple changes when the rule fires.

Definition 1. *The system configuration is a 5-tuple description* $\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$ *where*

Ags	*- is the first element of the tuple in the set of all agents;*
ag_i	*- is the second is the current agent under consideration;*
B_ξ	*- is the third is the belief base of the system;*
F	*- is the fourth is the set of facts in the environment;*
T_ξ	*- is and the fifth is the current time of the system;*

Definition 2. *The agents and objects within a system have a 9-tuple representation* $\langle ag_i, \mathcal{T}, W, stage, B, F, T_i, TF, WF \rangle$ *where*

ag_i	*- is the first element is the identification of the agent;*
$\mathcal{T}$	*- the second is the current thoughtframe;*
W	*- the third is the current workframe;*
$stage$	*- the fourth is the stage the agent is at;*
B	*- the fifth is the set of beliefs the agent has;*
F	*- the sixth is the set of facts;*
T_i	*- the seventh is the time of the agent;*
TF	*- eighth is the set of thoughtframes the agent has;*
WF	*- and the ninth is the agent's set of workframes.*

The fourth element of the tuple, the stage, explains which set of rules the agent is currently considering or if the agent is in a finish (fin*) or idle (*$idle$*) stage.*

$$\langle Starting\,Tuple \rangle \xrightarrow[Conditions\,Required\,For\,Actions]{Actions\,Performed} \langle Resulting\,Tuple \rangle$$

Fig. 3. Simplified Template for the Operational Semantics Transition Rules

Definition 3. *A transition rule is denoted by Figure 3 where*

$\langle Starting\,Tuple \rangle$	*- represents the system's or the agent's tuple before the rule is applied;*
$Conditions\,Required\,For\,Actions$	*- states the conditions required for the rule to fire;*
$Actions\,Performed$	*- represents the actions performed by the*

	rule;
Resulting Tuple	*- represents the tuple after the rule has fired;*

3.1 Timing

The timing in Brahms works by the use of a global system clock coupled with agents having their own internal clocks. The system scheduler asks each agent how long each of their activities are, finds the time of the shortest activity and then tells each agent to move their clock forward by this time. However it should be noted that during a simulation agents are not aware of their internal clocks, the clocks are used behind the scenes to keep all agents synchronised. Traditionally Brahms simulations are modelled with a 'Clock' agent to broadcast a simulation time to all the agents to give them an awareness of time. Workframes that the agents are currently working on can be interrupted if a new higher priority thought/workframe becomes active, or if a fact change in the system causes an impasse via a detectable. The following structure shows how agents are moved forward in time by the scheduler. It shows every agent from Ag_0 to Ag_n being moved forward in time, once an agent moves forward in time it reaches an intermediary point X where it will then make a $Choice$ on its next set of actions. ξ represents the scheduler, showing that all the agents and the scheduler move as one from time point to time point.

$$\frac{\begin{array}{c} Ag_0 \xrightarrow{LocalClock+t} X, X \xrightarrow{Choice} Ag'_0 \\ \vdots \\ Ag_n \xrightarrow{LocalClock+t} X, X \xrightarrow{Choice} Ag'_n \end{array}}{\xi \xrightarrow{LocalClock+t} \xi'}$$

4 Semantic Rules

4.1 Scheduler Semantics

The scheduler is the central system of Brahms, it decides when and what value the global clock will take and it starts and terminates the execution of the system. For the scheduler to start/continue execution all agents must be in a '*fin*' (*finished*) or 'idle' (*idle*) state and the global clock must not be less than zero. For Brahms to terminate all the agents need to be in an idle state where they have no workframes/thoughtframes which have their guard condition met.

Sch_run. Start agents running for the new clock tick. This rule states that if all agents in the system are either in a finished or idle state and the global clock is not minus one then all agents are directed to the 'Set_Act' semantic rule.

RULE: Sch_run

$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

$$\xrightarrow[\forall ag_i \in Ags | ag_i^{stage} \in \{fin, idle\} \wedge (T_\xi \neq -1)]{ag_{i'} = ag_i[ag_i^{stage} \in \{fin, idle\} / ag_i^{stage} \in \{Set\_Act\}]}$$

$$\langle Ags, ag_{i'}, B_\xi, F, T_\xi \rangle$$

Sch_rcvd. Receives the activity durations from all agents. This rule identifies when the Scheduler has received all the durations from all agents. It states that if all agents are in a waiting or idle state then the Scheduler will check all the agents end activity times, calculate the smallest value and set its time to this. For this rule to activate all the agents need to be considering the rules Pop_PA*, Pop_MA* or Pop_CA* where * represents a wild card for any suffix of the word.

RULE: Sch_rcvd

$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

$$\xrightarrow[\forall ag_i \in Ags | stage \in \{Pop\_PA*, Pop\_MA*, Pop\_CA*)\} \vee idle, (T_\xi \neq -1)]{T_{\xi'} = T_\xi[T_\xi / T_\xi + MinTime(\forall ag_i | T_i \in B_\xi)]}$$

$$\langle Ags, ag_i, B_\xi, F, T_{\xi'} \rangle$$

the notation $ag_{i'} = ag_i[ag_i^{stage} \in \{fin, idle\} / ag_i^{stage} \in \{Set\_Act\}]$ indicates that the stage value of ag_i has been replaced by Set_Act.

Sch_term. This termination condition happens when all agents are in an idle state, to signal the termination it sets the global clock to minus one.

RULE: Sch_Term

$$\langle Ags, ag_i, B_\xi, F, T_\xi \rangle$$

$$\xrightarrow[\forall ag_i \in Ags | stage \in \{idle\}]{T_{\xi'} = T_\xi[T_\xi / T_\xi = -1]}$$

$$\langle Ags, ag_i, B_\xi, F, T_{\xi'} \rangle$$

4.2 Agent Semantics

The Brahms system operates on a simple cycle of handling:

$$Thoughtframes \rightarrow Detectables \rightarrow Workframes$$

4.3 Set_* rules

Rules with the prefix of 'Set_*' are used at the start of every cycle. These are used to determine whether or not the agent/object will be *idle* (no active workframe or thought-frame) for the duration of this cycle. Those that are *idle* will do nothing until this rule is next invoked by the system, those that are not *idle* are directed to checking thought-frames.

Set_Act. If the agent is currently checking 'Set_*' rules, has no current thoughtframe and the agent has a workframe or a thoughtframe with its guard condition met then this rule directs the agent to the 'Tf_*' rules. Whether or not the agent has an active workframe or not is not an issue.

RULE: Set_Act

$$\frac{\langle ag_i, \alpha, \beta, Set\_*, B_i, F, T_i, TF_i, WF_i\rangle}{\alpha\in\{\emptyset\}\wedge(\exists\mathcal{T}\in TF_i|B_i\models\mathcal{T}^g\vee\exists W\in WF_i|B_i\models W^g)}\ ag_i[ag_i^{stage}\in\{Set\_*\}/ag_i^{stage}\in\{Tf\_*\}]$$

$$\langle ag_i, \alpha, \beta, Tf\_*, B_i, F, T_i, TF_i, WF_i\rangle$$

Set_Idle. If the agent has no current thoughtframes or workframes with their preconditions met then place the agent in an idle state. Additionally the agent can not have an active thoughtframe but can possibly have an active workframe.

RULE: Set_Idle

$$\frac{\langle ag_i, \alpha, \beta, Set\_*, B_i, F, T_i, TF_i, WF_i\rangle}{\alpha\in\{\emptyset\}\wedge\beta\in\{\emptyset\}\wedge\neg\exists\mathcal{T}\in TF_i|B_i\models\mathcal{T}^g\wedge\neg\exists W\in WF_i|B_i\models W^g}\ ag_i[ag_i^{stage}\in\{Set\_*\}/ag_i^{stage}\in\{idle\}]$$

$$\langle ag_i, \alpha, \beta, idle, B_i, F, T_i, TF_i, WF_i\rangle$$

4.4 Tf_* rules (Thoughtframes)

The agent is now in a state where it is selecting a thoughtframe to run. The agent will not have any thoughtframes currently active. When selecting the thoughtframe to run it will choose the thoughtframe with the highest priority, but if there is more than one then a random selection will be made.

Tf_Select. If there is a thoughtframe(s) with preconditions met then perform a selection based on the thoughtframe's priority. The agent can not have a current thoughtframe but can possibly have an active workframe. The thoughtframe is selected using the Max_pri method which choses the thoughtframe based on the priority. The agent is then passed onto rules to execute the thoughtframe, the chosen rule depends on the repeat variable of the thoughtframe(true, false or once).

RULE: Tf_Select

$$\frac{\langle ag_i, \alpha, \beta, Tf\_*, B_i, F, T_i, TF_i, WF_i\rangle}{\alpha\in\{\emptyset\}\wedge\exists\mathcal{T}\in TF_i|B_i\models\mathcal{T}^g}\ \alpha'=\alpha[\alpha/Max\_Pri(\forall\mathcal{T}\in TF_i|B_i\models\mathcal{T}^g)]\wedge ag_i[ag_i^{stage}\in\{Set\_*\}/ag_i^{stage}\in\{Tf\_true, Tf\_false, Tf\_once\}]$$

$$\langle ag_i, \alpha', \beta, \{Tf\_true, Tf\_false, Tf\_once\}, B_i, F, T_i, TF_i, WF_i \rangle$$

Tf_true (Repeat = true). If the repeat variable on the thoughtframe is true then the agent is just directed to 'Pop_Tf*' rules.

RULE: Tf_true

$$\langle ag_i, \alpha, \beta, Tf\_true, B_i, F, T_i, WF_i, TF_i \rangle$$

$$\xrightarrow[\alpha^{r=true} \wedge \beta \in \{\emptyset\}]{ag_i[ag_i^{stage} \in \{Tf\_true\}/ag_i^{stage} \in \{Pop\_Tf*\}]}$$

$$\langle ag_i, \alpha, \beta, Pop\_Tf*, B_i, F, T_i, TF_i, WF_i \rangle$$

Tf_once (Repeat = once). If repeat variable is set to once, change to false then move to 'Pop_Tf*' rules.

RULE: Tf_once

$$\langle ag_i, \alpha, \beta, Tf\_once, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[\alpha^{r=once} \wedge \beta \in \{\emptyset\}]{\alpha' = \alpha[\alpha^{r=once}/\alpha^{r=false}] \wedge TF'_i = TF_i[\alpha/\alpha'] \wedge ag_i[ag_i^{stage} \in \{Tf\_once\}/ag_i^{stage} \in \{Pop\_Tf*\}]}$$

$$\langle ag_i, \alpha, \beta, Pop\_Tf*, B_i, F, T_i, TF_i, WF_i \rangle$$

Tf_false(Repeat = false). If repeat variable is set to false, then delete thoughtframe from the set of thoughtframes.

RULE: Tf_false

$$\langle ag_i, \alpha, \beta, Tf\_false, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[\alpha^{r=false} \wedge \beta \in \{\emptyset\}]{TF'_i = TF_i[TF_i - \alpha] \wedge ag_i[ag_i^{stage} \in \{Tf\_false\}/ag_i^{stage} \in \{Pop\_Tf*\}]}$$

$$\langle ag_i, \alpha, \beta, Pop\_Tf*, B_i, F, T_i, TF'_i, WF_i \rangle$$

Tf_exit. If there are no thoughtframes to be executed then the agent is directed towards checking all the detectables.

RULE: Tf_exit

$$\langle ag_i, \alpha, \beta, Tf\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[\neg \exists \mathcal{T} \in TF_i | B \models \mathcal{T}^g \wedge \alpha \in \{\emptyset\}]{ag_i[ag_i^{stage} \in \{Tf\_*\}/ag_i^{stage} \in \{Det\_*\}]}$$

$$\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i TF_i, WF_i \rangle$$

4.5 Wf_* rules (Workframes)

The agent is now in a state where it is selecting a workframe to run. When selecting the workframe to run it will choose the workframe with the highest priority, if there is more than one workframe with the highest priority then a random selection is made between these workframes.

Wf_select. If there is no current workframe then a simple selection process occurs taking the workframe with the highest priority. The agent must have no workframes or thoughtframes assigned to it.

RULE: Wf_Select

$$\frac{\langle ag_i, \alpha, \beta, Wf\_*, B_i, F, T_i, TF_i, WF_i\rangle}{\substack{\beta'=\beta[\beta/Max\_Pri(\forall W\in WF_i \mid B_i \models W^g)]\wedge ag_i[ag_i^{stage}\in\{Set\_*\}/ag_i^{stage}\in\{Wf\_true, Wf\_false, Wf\_once\}] \\ \alpha\in\{\emptyset\}\wedge\beta\in\{\emptyset\}\wedge\exists W\in WF_i \mid B_i\models W^g}}$$
$$\langle ag_i, \alpha, \beta', \{Wf\_true, Wf\_false, Wf\_once\}, B_i, F, T_i, TF_i, WF_i\rangle$$

Wf_suspend. If an agent is currently working on a workframe, but there exists a workframe with its guard condition met that has higher priority then the current workframe is suspended and the progress the agent has made through this workframe is recorded. The priority of the suspended workframe is increased by 0.2, priorities are usually integers but this gives suspended workframes higher priority over those which normally would have the same priority. Note. β_d represents the workframe's deed stack and β_{ins} refers to the workframe's instructions, such as the workframe's repeat values, etc.

RULE: Wf_Suspend

$$\frac{\langle ag_i, \alpha, \beta, Wf\_*, B_i, F, T_i, TF_i, WF_i\rangle}{\substack{\beta'=\beta[\beta^{pri}/(\beta^{pri}+0.2)]\wedge WF'_i=WF'_i[WF_i\cup\beta']\wedge\beta''\in\{\emptyset\} \\ \alpha\in\{\emptyset\}\wedge\beta\notin\{\emptyset\}\wedge\exists W\in WF_i \mid B_i\models W^g\wedge W^{pri}>(\beta^{pri}+0.3)}}$$
$$\langle ag_i, \alpha, \beta'', Wf\_*, B_i, F, T_i, TF_i, WF_{i'}\rangle$$

Wf_true (Repeat = true). If there does not exist such a workframe with a greater priority then execute the currently selected workframe. 0.3 is added to the current workframes priority when checking whether to suspend, so that the current workframe is not suspended for another suspended workframe of priority only 0.2 higher. The agent is then passed onto rules for processing variables, rules with prefix 'Var_*'

RULE: Wf_true

$$\frac{\langle ag_i, \alpha, \beta, Tf\_true, B_i, F, T_i, WF_i, TF_i\rangle}{\substack{ag_i[ag_i^{stage}\in\{Wf\_true\}/ag_i^{stage}\in\{Pop\_Wf*\}] \\ \beta^{r=true}\wedge\alpha\in\{\emptyset\}\wedge\neg\exists W\in WF_i \mid B_i\models W^g\wedge W^{pri}>\beta^{pri}+0.3)}}$$
$$\langle ag_i, \alpha, \beta, Pop\_Wf*, B_i, F, T_i, TF_i, WF_i\rangle$$

Wf_once (Repeat = once). If the current workframe has the repeat value once then the repeat value of this workframe is changed to false and the agent is passed onto rules for processing variables.

RULE: Wf_once

$$\frac{\langle ag_i, \alpha, \beta, Wf\_once, B_i, F, T_i, TF_i, WF_i\rangle}{\beta^{r=once} \wedge \alpha \in \{\emptyset\} \wedge \beta' = \beta[\beta^{r=once}/(\beta^{r=false}] \wedge {WF'}_i = WF_i[\beta/\beta'] \wedge ag_i[ag_i^{stage} \in \{Wf\_once\}/ag_i^{stage} \in \{Var\_*\}]}{\neg \exists W \in WF_i | B_i \models W^g \wedge W^{pri} > \beta^{pri+0.3}} \langle ag_i, \alpha, \beta, Var\_*, B_i, F, T_i, TF_i, {WF'}_i\rangle$$

Wf_false(Repeat = false). If the current workframe has the repeat value false then it is deleted from the set of workframes and the agent is passed onto processing variables.

RULE: Wf_false

$$\frac{\langle ag_i, \alpha, \beta, Wf\_(false), B_i, F, T_i, TF_i, WF_i\rangle}{{WF'}_i = WF_i[WF_i - \beta] \wedge ag_i[ag_i^{stage} \in \{Wf\_false\}/ag_i^{stage} \in \{Var\_*\}]}{\beta^{r=false} \wedge \neg \exists W \in WF_i | B_i \models W^g \& W^{pri} > \beta^{pri+0.3}} \langle ag_i, \alpha, \beta, Var\_*, B_i, F, T_i, TF_i, {WF'}_i\rangle$$

4.6 Det_* rules (Detectables)

Detectables are additional guards contained within a workframe which when activated (though facts not beliefs) will trigger a belief update from the facts and will then decide how the rest of the workframe will be executed. The possible executions are Continue, Complete, Impasse and Abort.

Det_cont. When a detectable's guard condition is met and the detectable is of type Continue then the workframe updates its beliefs from the facts detected and carries on unchanged.

RULE: Det_cont

$$\frac{\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i, TF_i, WF_i\rangle}{B_i' = B_i \cup d^g \wedge ag_i[ag_i^{stage} \in \{Det\_*\}/ag_i^{stage} \in \{Wf\_*\}]}{\exists d \in \beta^D | d^g \models F \wedge d^{type = continue} \wedge (\neg \exists d' in \beta^D | d'^g \models F \wedge (d'^{type = impasse} \vee d'^{type = abort} \vee d'^{type = complete}))} \langle ag_i, \alpha, \beta, Wf\_*, B_i', F, T_i, TF_i, WF_i\rangle$$

Here d is used to represent a detectable, β^D is the workframe β's set of detectables. Notation to express parts of the detectables: d^g represents the detectables guard condition and d^{type} refers to the detectables type whether it is continue, complete or abort.

Det_comp. When a detectable's guard condition is met and the detectable is of type *complete* then the workframe updates its beliefs from the facts detected and deletes all activities from the workframe leaving only concludes.

RULE: Det_comp

$$\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i, TF_i, WF_i\rangle$$

$$\xrightarrow[\exists d \in \beta^D \,|\, d^g \models F \wedge d^{type=complete} \wedge (\neg \exists d' \, in \, \beta^D \,|\, d'^g \models F \wedge (d'^{type=impasse} \vee d'^{type=abort}))]{\beta' = \beta[\beta_{ins} / \beta^{Concludes}] \wedge B_i' = B_i \cup d^g \wedge ag_i[ag_i^{stage} \in \{Det\_*\} / ag_i^{stage} \in \{Wf\_*\}]}$$

$$\langle ag_i, \alpha, \beta', \, Wf\_*, B_i', F, T_i, \, TF_i, \, WF_i \rangle$$

$\beta^{Concludes}$ is used to refer to conclude events within the workframe β.

Det_impasse. When the detectable is of type *impasse* the beliefs are updated from the facts detected but the workframe is suspended. To suspend the workframe a new workframe is created out of this workframe instance and added to the set of workframes with repeat set to false. The priority of this new workframe is fractionally larger than the previous (but smaller than a suspended).

RULE: Det_impasse

$$\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

$$\xrightarrow[\exists d \in \beta^D \,|\, d^g \models F \wedge d^{type=impasse} \wedge (\neg \exists d' \, in \, \beta^D \,|\, d'^g \models F \wedge d'^{type=abort})]{\beta' = \beta[\beta^{pri} / (\beta^{pri} + 0.1) \wedge \beta^g \cup \neg d^g)] \wedge B_i' = B_i \cup d^g \wedge WF'_i = WF_i \cup \beta' \wedge ag_i[ag_i^{stage} \in \{Det\_*\} / ag_i^{stage} \in \{Wf\_*\}]}$$

$$\langle ag_i, \alpha, \beta', \, Wf\_*, B_i', F, T_i, \, TF_i, \, WF_i \rangle$$

Det_abort. If the detectable is of type *abort* then the belief base is updated and the agent's assignment to the workframe is removed.

RULE: Det_abort

$$\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

$$\xrightarrow[\exists d \in \beta^D \,|\, d^g \models F \wedge d^{type=abort}]{\beta' \in \{\emptyset\} \wedge B_i' = B_i \cup d^g \wedge ag_i[ag_i^{stage} \in \{Det\_*\} / ag_i^{stage} \in \{Wf\_*\}]}$$

$$\langle ag_i, \alpha, \beta', \, Wf\_*, B_i', F, T_i, \, TF_i, \, WF_i \rangle$$

Det_empty. If there are no active detectables found then the agent is moved to the 'workframes' rule set denoted '**Wf_***'.

RULE: Det_empty

$$\langle ag_i, \alpha, \beta, Det\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

$$\xrightarrow[\neg \exists d \in \beta^D \,|\, d^g \models F]{ag_i[ag_i^{stage} \in \{Det\_*\} / ag_i^{stage} \in \{Wf\_*\}]}$$

$$\langle ag_i, \alpha, \beta, \, Wf\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

4.7 Var_* rules (Variables)

Variables are used to represent quantification in Brahms. Variables operate on both workframes and thoughtframes, however for simplicity only workframes have been modelled to handle variables. Thoughtframes would operate variables in exactly the same way.

RULE: Var_empty

$$\frac{\langle ag_i, \alpha, \beta, \mathit{Var\_*}, B_i, F, T_i, TF_i, WF_i\rangle \xrightarrow[\beta \notin \{\emptyset\} \wedge \beta^V \in \{\emptyset\}]{ag_i[ag_i^{stage} \in \{\mathit{Var\_*}\}/ag_i^{stage} \in \{\mathit{Pop\_*}\}]} \langle ag_i, \alpha, \beta, \mathit{Pop\_*}, B_i, F, T_i, TF_i, WF_i\rangle}{}$$

Note. Where β^V represents the variables contained within workframe β.

Var_set. Workframes with variables have an additional stack. This additional stack stores instances of the workframe with the differing instantiations that can be created with the variables. If the set of options is empty then a selection process called 'selectVar()' is called. 'selectVar()' will match all agents/objects which match the name and conditions, assign each to an instance of the workframe then places the instances onto the stack. Note. $\langle \beta_d, [\emptyset], [\beta_{ins}]\rangle$ represents a workframe β with a deed stack d, a set of empty workframe instances and the workframe's set of instructions β_{ins}

RULE: Var_set

$$\frac{\langle ag_i, \alpha, \beta, \mathit{Var\_*}, B_i, F, T_i, TF_i, WF_i\rangle \xrightarrow[\beta = \langle \beta_d, \emptyset, \beta_{ins}\rangle]{\beta' = \langle \beta_d, [\emptyset \cup selectVar()], \beta_{ins}\rangle} \langle ag_i, \alpha, \beta', \mathit{Var\_*}, B_i, F, T_i, TF_i, WF_i\rangle}{}$$

Var_one. When the variable is of type 'forone' and a set of workframe instances has been generated then the first workframe instance is selected and set as the current workframe. The subset of variables in the workframe are then deleted. This is how Brahms performs unification.

RULE: Var_one

$$\frac{\langle ag_i, \alpha, \beta, \mathit{Var\_*}, B_i, F, T_i, TF_i, WF_i\rangle \xrightarrow[\beta = \langle \beta_d, W_0 \ldots W_n, \beta_{ins}\rangle \wedge \exists v \in \beta^V \mid v^{type} = forone]{\beta' = \langle \beta_d, Random(W_0 \ldots W_n), \beta_{ins}\rangle \wedge ag_i[ag_i^{stage} \in \{\mathit{Var\_*}\}/ag_i^{stage} \in \{\mathit{Pop\_*}\}]} \langle ag_i, \alpha, \beta', \mathit{Pop\_*}, B_i, F, T_i, TF_i, WF_i\rangle}{}$$

'Random' refers to a random selection of one of the instances and 'v^{type}' represents the variables type (forone, foreach or collectall).

Var_each. When the variable is of type 'foreach' and the subset of the workframe is not empty then the instances of the workframes are added to the set of workframes and the first instance is set as the current workframe. The instances are given a slightly

increased priority and a repeat value of false so they will never be repeated. This represents Brahms operating on a multitude of tasks sequentially.

RULE: Var_each

$$\frac{\langle ag_i, \alpha, \beta, \, Var\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle}{\substack{WF'_i = WF_i \cup (W_0[W_0^{pri}/(\beta^{pri}+0.1), W_0^r/W_0^r = false] \ldots W_n[W_n^{pri}/(\beta^{pri}+0.1), W_n^r/W_n^r = false]) \\ \beta = \langle \beta_d, W_0 \ldots W_n, \beta_{ins} \rangle \wedge \exists v \in \beta^V \mid v^{type} = foreach}}$$
$$\langle ag_i, \alpha, W_0, Pop\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

Var_all. The 'collectall' variable operates in a similar fashion to the previous variables, however when it selects the first workframe from the subset it merges all the concludes from the other work frames into this workframe. This effectively is how Brahms handles a job which has multiple consequences, e.g., By completing task A, I also complete task B.

RULE: Var_all

$$\frac{\langle ag_i, \alpha, \beta, \, Var\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle}{\substack{\beta' = concludes(W_0 \ldots W_n) \wedge ag_i[ag_i^{stage} \in \{Var\_*\}/ag_i^{stage} \in \{Pop\_*\}] \\ \beta = \langle \beta_d, W_0 \ldots W_n, \beta_{ins} \rangle \wedge \exists v \in \beta^V \mid v^{type} = forall}}$$
$$\langle ag_i, \alpha, \beta', Pop\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle$$

$concludes(W_1 \ldots W_n)$ is a method which takes all the workframe instances $W_1 \ldots W_n$ and extracts the concludes statements.

4.8 Pop_* rules (Popstack)

Thoughtframes and workframes all have their own stack of instructions. These rules presented demonstrate how the events are "popped" off these instruction stacks. The events can be *activites* or *concludes*, so these rules show how Brahms treats these different instructions.

Pop_Wfconc*. When a conclude action is found it is removed from the top of the instruction stack. Concludes can update the *beliefs*, the *facts* or both. Three different rules are used for concludes: one for updating *beliefs*; one for *facts*; and one for both. Brahms additionally has probabilities that beliefs will be updated, these probabilities have not been taken into account in these semantics. **Pop_Wfconc*** is neccessary only for workframes, there is no rule **Pop_Tfconc*** thoughtframes since there are no activities to interupt execution.

RULE: Pop_WfconcB

$$\frac{\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, \, TF_i, \, WF_i \rangle}{\substack{B'_i = (B_i/b) \cup b' \\ b \in B_i \wedge \beta = \langle \beta_d, conclude(b')^{belief}; \beta_{ins} \rangle}}$$

$$\langle ag_i, \alpha, \beta, Pop\_*, B'_i, F, T_i, TF_i, WF_i \rangle$$

The "belief" superscript on the conclude is to show the conclude is for updating beliefs only. The statement $conclude(b')$ represents a conclude statement a belief update b' and $B'_i = (B_i/b) \cup b'$ represents removing the old belief where b and replacing it with the new belief b'.

RULE: Pop_WfconcF

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\xrightarrow[f \in F \wedge \beta = \langle \beta_d, conclude(f')^{fact}; \beta_{ins} \rangle]{F = (F/f) \cup f'}$$
$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F', T_i, TF_i, WF_i \rangle$$

RULE: Pop_WfconcBF

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\xrightarrow[b \in B_i \wedge b \in F \wedge \beta = \langle \beta_d, conclude(b')^{belief \wedge fact}; \beta_{ins} \rangle]{F' = (F/b) \cup b' \wedge B'_i = (B_i/b) \cup b'}$$
$$\langle ag_i, \alpha, \beta, Pop\_*, B'_i, F', T_i, TF_i, WF_i \rangle$$

Pop_concWf*. When agents have finished performing an activity they need to finalise belief updates before they can flag themselves as finished for the cycle. This rule here is for doing exactly this, if a conclude is the next event it will carry out the belief/fact update. Here only 'Pop_concWfB' is described, this shows how it is done with just belief updates. Fact and belief/fact updates will be as previously shown.

RULE: Pop_concWfB

$$\langle ag_i, \emptyset, \beta, Pop\_concWf*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\xrightarrow[b \in B_i \wedge \beta = \langle \beta_d, conclude(b')^{belief}; \beta_{ins} \rangle]{B'_i = (B_i/b) \cup b'}$$
$$\langle ag_i, \alpha, \beta, Pop\_concWf*, B'_i, F, T_i, TF_i, WF_i \rangle$$

Pop_notConc. This rule is for when the agent is finalising beliefs after an activity but has not found a *conclude* event, the event could be an *activity* or simply empty.

RULE: Pop_notConc

$$\langle ag_i, \alpha, \beta, Pop\_concWf*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\xrightarrow[\beta = \langle \beta_d, (Prim\_Act \vee Move \vee Comms); \beta_{ins} \rangle]{ag_i[ag_i^{stage} \in \{Pop\_concWf*\}/ag_i^{stage} \in \{fin\}]}$$
$$\langle ag_i, \alpha, \beta, fin, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_PA*. When a primitive activity is started the agents send the duration of their current activity to the scheduler. The scheduler receives all the activity times then determines which activity time is the smallest and updates its own clock based on this duration. When an agent's time is different to the system clock's it then changes accordingly and subtracts the time increment from the duration of its activity.

Pop_PASend. This is the rule the agent's use to send the duration of their next event to the scheduler.

RULE: Pop_PASend

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[T_\xi = T_i \wedge \beta = \langle \beta_d, Prim\_Act^t; \beta_{ins} \rangle]{B_\xi = B_\xi \cup (T_i = T_i + t)}$$

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_PA(t>0). This rule is invoked when the agent's time is no longer the same as the schedulers time. Additionally this rule checks whether the current activity's duration will be greater than zero after updating the times and durations.

RULE: Pop_PA(t>0)

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[T_\xi != Ti \wedge (T_i + t - T_\xi) > 0 \wedge \beta = \langle \beta_d, Prim\_Act^t; \beta_{ins} \rangle]{t' = (T_\xi - T_i) \wedge T_i = T_\xi \wedge Prim\_Act^t = Prim\_Act^t[t/t'] \wedge ag_i[ag_i^{stage} \in \{Pop\_concWf*\}/ag_i^{stage} \in \{fin\}]}$$

$$\langle ag_i, \alpha, \beta, fin, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_PA(t=0). This rule is for when the agent's activity is due to finish at the end of the next clock tick. This rule directs the agent to only executing conclude statements before finishing for the cycle.

RULE: Pop_PA(t=0)

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\xrightarrow[T_\xi != Ti \wedge (T_i + t - T_\xi) = 0 \wedge \beta = \langle \beta_d, Prim\_Act^t; \beta_{ins} \rangle]{T_i = T_\xi \wedge \beta = \langle \beta_d, \beta_{ins} - Prim\_Act^t \rangle}$$

$$\langle ag_i, \alpha, \beta, Pop\_concWF*, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_move*. Move activities are very similar to primitive activities, except when the activity terminates a belief update is performed to change the agents and the environments beliefs of the agent's current location. This belief update occurs when the agent notices that the duration of the move has reached zero after the clock update. **Pop_moveSend**. This is the rule the agent's use to send the duration of their next event to the scheduler.

RULE: Pop_moveSend

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\frac{B_\xi = B_\xi \cup (T_i = T_i + t)}{T_\xi = T_i \wedge \beta = \langle \beta_d, move(Loc = new)^t; \beta_{ins} \rangle}$$
$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

Note. 'Loc = new' refers to the allocation of the *location* to the *new location*.

Pop_move(t>0). Like for primitive activities, the move activity needs a rule for when the activity still has time remaining after the clock tick.

RULE: Pop_move(t>0)

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{t' = (T_\xi - T_i) \wedge T_i = T_\xi \wedge move(Loc = new)^t = move(Loc = new)^t[t/t'] \wedge ag_i[ag_i^{stage} \in \{Pop\_concWf*\}/ag_i^{stage} \in \{fin\}]}{T_\xi != Ti \wedge (T_i + t - T_\xi) > 0 \wedge \beta = \langle \beta_d, move(Loc = new)^t; \beta_{ins} \rangle}$$
$$\langle ag_i, \alpha, \beta, fin, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_move(t=0). Likewise, the move activity needs a rule for when the activity duration ends.

RULE: Pop_move(t=0)

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{T_i = T_\xi \wedge \beta = \langle \beta_d, \beta_{ins} - move(Loc = new)^t \rangle \wedge B_i' = B_i[Loc = old/Loc = new] \wedge F' = F[Loc = old/Loc = new]}{T_\xi != Ti \wedge (T_i + t - T_\xi) = 0 \wedge \beta = \langle \beta_d, move(Loc = new)^t; \beta_{ins} \rangle}$$
$$\langle ag_i, \alpha, \beta, Pop\_concWF*, B_i', F', T_i, TF_i, WF_i \rangle$$

Note. 'old' refers to the previous *location* of the agent.

Pop_comm*. Communication is very similar to a move activity, except the agent doesn't update its own beliefs or the environments beliefs but it updates another agents beliefs.
Pop_commSend. Sends the scheduler the time of next event when processing a communication.

RULE: Pop_commSend

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{B_\xi = B_\xi \cup (T_i = T_i + t)}{T_\xi = T_i \wedge \beta = \langle \beta_d, Comms(ag_j, b')^t; \beta_{ins} \rangle}$$
$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

Note. $Comms(ag_j, b')$ represents a communication to agent j, sending the belief b'.

Pop_comm(t>0). For when the communication has time remaining after the system clock tick.

RULE: Pop_comm(t>0)

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

$$\frac{t'=(T_\xi - T_i) \wedge T_i = T_\xi \wedge Comms(ag_j, b')^t = Comms(ag_j, b')^t[t/t'] \wedge ag_i[ag_i^{stage} \in \{Pop\_concWf*\}/ag_i^{stage} \in \{fin\}]}{T_\xi != Ti \wedge (T_i + t - T_\xi) > 0 \wedge \beta = \langle \beta_d, Comms(ag_j, b')^t; \beta_{ins} \rangle} \rightarrow$$
$$\langle ag_i, \alpha, \beta, fin, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_comm(t=0). Rule for when the communication activity duration ends.

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{T_i = T_\xi \wedge \beta = \langle \beta_d, \beta_{ins} - Comms(ag_j, b')^t \rangle \wedge B'_j = B_j[b/b']}{T_\xi != Ti \wedge (T_i + t - T_\xi) = 0 \wedge \beta = \langle \beta_d, Comms(ag_j, b')^t; \beta_{ins} \rangle \wedge b \in B_j} \rightarrow$$
$$\langle ag_i, \alpha, \beta, Pop\_concWF*, B_i, F, T_i, TF_i, WF_i \rangle$$

Note. Belief exchange via communication is handed directly in Brahms, i.e. when an agent communicates with another, it directly changes the other agent's beliefs.

Pop_emptyTf. Concludes do not use up any simulation time during execution, since thoughtframes only contain concludes then an agent will keep executing thoughtframes until it no longer has any to execute. This rule is for selecting a new thoughtframe when the current one becomes empty.

RULE: Pop_emptyTf

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{\alpha \in \{\emptyset\} \wedge ag_i[ag_i^{stage} \in \{Pop\_*\}/ag_i^{stage} \in \{Tf\_*\}]}{\alpha = \langle \alpha_d, \emptyset \rangle} \rightarrow$$
$$\langle ag_i, \alpha, \beta, Tf\_*, B_i, F, T_i, TF_i, WF_i \rangle$$

Pop_emptyWf. A workframe which only contains concludes will act like a thoughtframe. This rule is for such workframes so the agent can keep select another workframes when the current one becomes empty.

RULE: Pop_emptyWf

$$\langle ag_i, \alpha, \beta, Pop\_*, B_i, F, T_i, TF_i, WF_i \rangle$$
$$\frac{\beta \in \{\emptyset\} \wedge ag_i[ag_i^{stage} \in \{Pop\_*\}/ag_i^{stage} \in \{Wf\_*\}]}{\beta = \langle \beta, \emptyset \rangle} \rightarrow$$
$$\langle ag_i, \alpha, \beta, Wf\_*, B_i, F, T_i, TF_i, WF_i \rangle$$