The Semantics of MALLET–An Agent Teamwork Encoding Language

Xiaocong Fan[1], John Yen[1], Michael S. Miller[2], and Richard A. Volz[2]

[1] School of Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16802
[2] Department of Computer Science
Texas A&M University, College Station, TX 77843
{zfan,jyen}@ist.psu.edu, {mmiller,volz}@cs.tamu.edu

Abstract. MALLET is a team-oriented agent programming language for specifying teamwork knowledge and behaviors; one interpreter of MALLET has already been implemented in the CAST (Collaborative Agents for Simulating Teamwork) system. This paper defines an operational semantics for MALLET in terms of a transition system. This is important not only in guiding the implementation of other interpreters for MALLET, but also in formally studying the properties of team-based agents specified in MALLET.

1 Introduction

Agent teamwork has been the focus of a great deal of research in both theories [1–4] and practices [5–8]. A team is a set of agents having a shared objective and a shared mental state [2]. While the notion of joint goal (joint intention) provides the glue that binds team members together, it is not sufficient to guarantee that cooperative problem solving will ensue [3]. The agreement of a common recipe among team members is essential for them to achieve their shared objective in an effective and collaborative way [4]. Languages for specifying common recipes (plans) and other teamwork related knowledge are thus highly needed both for agent designers to specify and implement cohesive teamwork behaviors, and for agents themselves to easily interpret and manipulate the mutually committed course of actions so that they could collaborate smoothly both when everything is progressing as planned and when something goes wrong unexpectedly.

The term "team-oriented programming" has been used to refer to both the idea of using a meta-language to describe team behaviors (based on mutual beliefs, joint plans and social structures) [9] and the effort of using a reusable team wrapper for supporting rapid development of agent teams from existing heterogeneous distributed agents [10, 11]. This paper adopts the former meaning and focuses on the semantics of an agent teamwork encoding language called MALLET (Multi-Agent Logic Language for Encoding Teamwork), which has been developed and used in the CAST (Collaborative Agents for Simulating Teamwork) system [8] to specify agents' individual and teamwork behaviors.

There have been several efforts in defining languages for describing team activity [12, 13, 3]. What distinguishes MALLET from the existing efforts has two-fold. First, MALLET is a richer generic language for encoding teamwork knowledge. Teamwork knowledge may include both declarative knowledge and procedural knowledge. Declarative knowledge (knowing "that") describes objects, events, and their relationships. Procedural knowledge (knowing "how") focuses on the way needed to obtain a result, where the control information necessary to use the knowledge is embedded in the knowledge itself. MALLET supports the specification of both declarative and procedural teamwork knowledge. For instance, MALLET has reserved keywords for specifying team structure-related knowledge such as agents in a team, roles an agent can play, etc., as well as inference knowledge (horn-clauses). MALLET also has constructs for specifying control flows (e.g., sequential, conditional, iterative, etc.) in a team process. Tidhar also adopted such an synthesized approach [9], where the notions of social structure and plan structure respectively correspond to the team structure and team process in our term. While MALLET does not describe team structure in the command and control dimension as Tidhar did, it is more expressive than the simple OR-AND plan graphs in describing complex team process.

Second, MALLET is a richer language for encoding teamwork process. In MALLET, the constraints for task assignments, preconditions of actions, dynamic agent selection, decision points within a process and termination conditions of a process can be explicitly specified. The recipe language used in [3] lacks the support for specifying decision points in a process, which is often desirable in dealing with uncertainty. While the OR nodes of a plan graph [9] can be used for such a purpose, the language cannot specify complex execution orders. Team/agent selection (i.e., the process of selecting a group of agents that have complimentary skills to achieve a given goal) is an important aspect of collaborative activity [14]. No existing languages except MALLET allow the task of agent-selection to be explicitly specified in a team process. Using MALLET, a team of agents can collaboratively recruit doers for the subsequent activities based on the constraints associated with the agent-selection statement.

The structure of this paper is as follows. Section 2 gives the syntax of MALLET. We prepare our work in Section 3 and give the transition semantics in Section 4. Section 6 concludes the paper.

2 Syntax

The syntax of MALLET is given in Table 1. A MALLET specification is composed of definitions for agents, teams, membership of a team, team goals, initial team activities, agent capabilities, roles, roles each agent can play, agents playing a certain role, individual operators, team operators, plans (recipes), and inference rules.

Operators are atomic domain actions, each of which is associated with preconditions and effects. Individual operators are supposed to be carried out by only one agent independently, while team operators can only be invoked by more

Table 1. The Syntax of MALLET

```
   CompilationUnit ::= ( AgentDef | TeamDef | MemberOf | GoalDef | Start |
                       CapabilityDef | RoleDef | PlaysRole | FulfilledBy | IOperDef |
                       TOperDef | PlanDef | RuleDecl)*
          AgentDef ::= ’(’ <AGENT> AgentName ’)’
           TeamDef ::= ’(’ <TEAM> TeamName ( ’(’ ( AgentName )+ ’)’ )? ’)’
          MemberOf ::= ’(’ <MEMBEROF> AgentName
                       ( TeamName | ’(’ ( TeamName )+ ’)’ ) ’)’
           GoalDef ::= ’(’ <GOAL> AgentOrTeamName ( Cond )+ ’)’
             Start ::= ’(’ <START> AgentOrTeamName Invocation ’)’
     CapabilityDef ::= ’(’ <CAPABILITY> ( AgentName | ’(’ (AgentName)+’)’)
                       ( Invocation | ’(’ ( Invocation )+ ’)’ ) ’)’
           RoleDef ::= ’(’ <ROLE> RoleName (Invocation | ’(’(Invocation)+’)’ )’)’
         PlaysRole ::= ’(’ <PLAYSROLE> AgentName ’(’ ( RoleName )+ ’)’ ’)’
       FulfilledBy ::= ’(’ <FULFILLEDBY> RoleName ’(’ ( AgentName )+ ’)’ ’)’
          IOperDef ::= ’(’ <IOPER> OperName ’(’ (<Variable>)* ’)’
                       ( PreConditionList )* (EffectsList )? ’)’
          TOperDef ::= ’(’ <TOPER> OperName ’(’ (<Variable>)* ’)’
                       ( PreConditionList )* ( EffectsList )? ( NumSpec )? ’)’
           PlanDef ::= ’(’ <PLAN> PlanName ’(’ (<Variable>)* ’)’
                       ( PreConditionList | EffectsList | TermConditionList )*
                       ’(’ <PROCESS> MalletProcess ’)’ ’)’
          RuleDecl ::= ’(’ ( Pred )+ ’)’
              Cond ::= Pred | ’(’ <NOT> Cond ’)’
              Pred ::= ’(’ <IDENTIFIER> ( <IDENTIFIER> | <VARIABLE>)* ’)’
        Invocation ::= ’(’PlanOrOperName (<IDENTIFIER> | <VARIABLE>)* ’)’
  PreConditionList ::= ’(’ <PRECOND> ( Cond )+ ( ’:IF-FALSE’ ( <FAIL> |
                       <WAIT> ( ( <DIGIT> )+ )? | <ACHIEVE> ) )? ’)’
       EffectsList ::= ’(’ <EFFECTS> ( Cond )+ ’)’
 TermConditionList ::= ’(’<TERMCOND> (<SUCCESS> | <FAILURE>)? (Cond)+’)’
           NumSpec ::= ’(’ <NUM> ( ‘ =′ |‘ <′ |‘ >′ |‘ ≤′ |‘ ≥′ ) ( <DIGIT > )+ ’)’
      PrefCondList ::= ’(’ <PREFCOND> ( Cond )+ ( ’:IF-FALSE’ ( <FAIL> |
                       <WAIT> ( ( <DIGIT> )+ )? | <ACHIEVE> ) )? ’)’
          Priority ::= ’(’<PRIORITY> ( <DIGIT> )+ ’) ’
            ByWhom ::= AgentOrTeamName | <VARIABLE> | MixedList
         MixedList ::= ’(’ ( <IDENTIFIER> | <VARIABLE> )+ ’)’
            Branch ::= ’(’(PrefCondList)?(Priority)? ’(’<DO>ByWhom Invocation’)”)’
     MalletProcess ::= Invocation
                       |’(’<DO> ByWhom MalletProcess ’)’
                       |’(’<AGENTBIND> VariableList ’(’ ( Cond )+ ’)’ ’)’
                       |’(’<JOINTDO> ( <AND> | <OR> | <XOR> )?
                            ( ’(’ ByWhom MalletProcess ’)’ )+ ’)’
                       |’(’<SEQ> ( MalletProcess )+ ’)’
                       |’(’<PAR> ( MalletProcess )+ ’)’
                       |’(’<IF>’(’<COND>(Cond)+’)’MalletProcess(MalletProcess)?’)’
                       |’(’<WHILE> ’(’ <COND> ( Cond )+ ’)’ MalletProcess ’)’
                       |’(’<FOREACH> ’(’ <COND> (Cond)+’)’MalletProcess’)’
                       |’(’<FORALL> ’(’ <COND> (Cond)+ ‘)’MalletProcess’)’
                       |’(’<CHOICE> ( Branch)+ ’)’
```

than one agent who play specific roles as required by the operators. Before doing a team action, all the involving agents should synchronize their activities and satisfy the corresponding preconditions.

Plans are decomposable higher-level actions, which are built upon lower-level atomic operators hierarchically. Plans play the same role as recipes in the SharedPlan theory. A plan in MALLET specifies which agents (variables), under what pre-conditions, can achieve what effects by following what a process, and optionally under what conditions the execution of the plan can be terminated.

The process component of a plan plays essential role in supporting coordinations among team members. A process can be specified using constructs such as sequential (SEQ), parallel (PAR), iterative (WHILE, FOREACH, FORALL), conditional (IF) and choice (CHOICE). An invocation statement is used to directly execute an action or invoke a plan; since there is no associated doer specification, each agent coming to such a statement will do it individually. A DO process is composed of a doer specification and an embedded process. An agent coming to a DO statement has to check if itself belongs to the doer specification. If so, the agent simply does the action and moves on; otherwise the agent waits to be informed of the outcome of the action. A joint-do process (JOINTDO) specifies a share type (i.e., *AND, OR, XOR*) and a list of ($ByWhom\ process$) pairs. For the share type "*AND*", each of the pairs must be executed before the complementation of the joint-activity, which requires all the involved agents acting simultaneously. For an "*XOR*", exactly one must be executed to avoid potential conflicts, and for an "*OR*", at least one must be executed (with no potential conflicts). An agent-bind statement is used to dynamically select agents to satisfy various constraints such as finding an agent that is capable of some role or action. An agent-bind statement becomes eligible for execution at the point when progress of the embedding plan has reached it, as opposed to being executed when the plan is entered. The scope for the binding to a variable extends to either the end of the embedding plan, or the beginning of the next agent-bind statement that also binds this variable, whichever comes first.

3 Preparation

The following notational conventions are adopted. We use i, j, k, m, n as indexes; a's [3] to denote individual agents; A's to denote sets of agents; b's to denote beliefs; g's to denote goals; h's to denote intentions; ρ's to denote plan templates; p's to denote plan preconditions; q's to denote plan effects; e's to denote plan termination-conditions; β and α's to denote individual operators; Γ's to denote team operators; s and l's to denote statements within a Mallet-process; ψ and ϕ's to denote first-order formulas; t's to denote terms; bold $\boldsymbol{t}$ and $\boldsymbol{v}$ to denote vector of terms and variables. A substitution (binding) is a set of variable-term pairs $\{[x_i/t_i]\}$, where variable x_i is associated with term t_i (x_i does not occur free in t_i). We use $\theta, \delta, \eta, \mu, \tau$ to denote substitutions.

[3] We use a's to refer to a and a with a subscript or superscript. The same applies to the description of other notations.

Given a team specification in MALLET, let *Agent* be the set of agent names, *Ioper* be the set of individual operators, *TOper* be the set of team operators, *Plan* be the set of plans, B be the initial set of beliefs (belief base), and G be the initial set of goals (goal base).

Let $P = Plan \cup Toper \cup Ioper$. We call P the plan (template) base, which consists of all the specified operators and plans. Every invocation of a template in P is associated with a substitution: each formal parameter of the template is bound to the corresponding actual parameter. For instance, given a template (**plan** ρ $(v_1 \cdots v_j)$

(**pre-cond** $p_1 \cdots p_k$) (**effects** $q_1 \cdots q_m$) (**term-cond** $e_1 \cdots e_n$) (**process** s)).
A plan call $(\rho\ t_1 \cdots t_j)$ will instantiate the template by binding $\theta = \{\boldsymbol{v}/\boldsymbol{t}\}$, where the evaluation of t_i may further depend on some other (environment) binding μ. Note that such instantiation process will substitute t_i for all the occurrence of v_i in the precondition, effects, term-condition, and plan body s (for all $1 \leq i \leq j$). The instantiation of ρ wrt. binding η is denoted by $\rho \cdot \eta$, or $\rho\eta$ for simplicity.

We define some auxiliary functions. For any operator α, let $pre(\alpha)$ and $post(\alpha)$ return the conjunction of the preconditions and effects specified for α respectively, let $\lambda(\alpha)$ returns the binding if α is an instantiated operator. For team operator Γ, $|\Gamma|$ returns the minimal number of agents required for executing Γ. For any plan ρ, in addition to $pre(\rho)$, $post(\rho)$ and $\lambda(\rho)$ as defined above, $tc(\rho)$, $\chi(\rho)$, and $body(\rho)$ return the conjunction of termination-conditions, the termination type ($\in$ {**success**, **failure**, ϵ}), and the plan body of ρ, respectively. The precondition, effects and termination-condition components of a plan are optional. When they are not specified, $pre(\rho)$ and $post(\rho)$ return **true** and $\chi(\rho) = \epsilon$. For any statement s, $isPlan(s)$ returns **true** if s is of form $(\rho\ \boldsymbol{t})$ or (**Do** A $(\rho\ \boldsymbol{t})$) for some A, where ρ is a plan defined in P; otherwise, it returns **false**. (**SEQ** $s_1\ \cdots\ s_i$) is abbreviated as $(s_1; \cdots; s_i)$. ε is used to denote the empty Mallet process statement. For any statement s, $\varepsilon; s = s; \varepsilon = s$. (**wait until** ϕ) is an abbreviation of (**while** (**cond** $\neg\phi$) (**do** self skip)) [4], where *skip* is a built-in individual operator with $pre(skip) = true$ and $post(skip) = true$ (i.e., the execution of *skip* changes nothing).

Messages Control messages are needed in defining the operational semantics of MALLET. A control message is a tuple $\langle type, aid, gid, pid, \cdots \rangle$, where $aid \in Agent$, $gid \in wffs$, $pid \in P \cup \{nil\}$, and $type \in \{$ *sync, ctell, cask, unachievable* $\}$. A message of type *sync* is used by agent *aid* to synchronize with the recipient with respect to the committed goal *gid* and current activity *pid*; a message of type *ctell* is used by agent *aid* to tell the recipient about the status of *pid*; a message of type *cask* is used by agent *aid* to request the recipient to perform *pid*; a message of type *unachievable* is used by agent *aid* to inform the recipient of the inachievability of *pid*.

MALLET has a built-in domain-independent operator **send***(receivers, msg)*, which is used for inter-agent communications. $pre(send) = true$. We assume the

[4] The keyword "*self*" can be used in specifying doers of a process. An agent always evaluate *self* as itself.

execution of **send** always succeeds. If $\langle typ, a, \cdots \rangle$ is a control message, the effect of $send(a, \langle typ, a, \cdots \rangle)$ will assert $(typ\ a\ \cdots)$ as a fact into the agent a's belief base. For instance, when agent a_1 receives message $\langle sync, a_2, g, p \rangle$, predicate $(sync\ a_2\ g\ p)$ will be appended as a fact into the belief base of a_1.

Goals and Intentions A goal g is a pair $\langle \phi, A \rangle$, where $A \subseteq Agent$ is a set of agents responsible for achieving a state satisfying ϕ. When A is a singleton, g is an individual goal; otherwise, it is a team goal.

An *intention slice* is of form $(\psi, A) \leftarrow s$, where the execution of statement s by agents in A is to achieve a state satisfying ψ. An *intention* is a stack of intention slices, denoted by $[\omega_0 \backslash \cdots \backslash \omega_k]$ $(0 \leq k)$, where ω_i $(0 \leq i \leq k)$ are of form $(\psi_i, A_i) \leftarrow s_i$. ω_0 and ω_k are the bottom and top slice of the intention, respectively. The ultimate goal state of intention $h = [(\psi_0, A_0) \leftarrow s_0 \backslash \cdots \backslash \omega_k]$ is ψ_0, referred to by $o(h)$. The empty intention is denoted by $\top$. For $h = [\omega_0 \backslash \cdots \backslash \omega_k]$, $[h \backslash \omega'] \triangleq [\omega_0 \backslash \cdots \backslash \omega_k \backslash \omega']$. If $\omega_i = true \leftarrow \varepsilon$ $(0 \leq i \leq k)$, then $h = [\omega_0 \backslash \cdots \backslash \omega_{i-1} \backslash \omega_{i+1} \backslash \cdots \backslash \omega_k]$. Let H be the intention set.

Definition 1 (configuration). *A Mallet configuration is a tuple $\langle B, G, H, \theta \rangle$, where B, G, H, θ are the belief base, the goal base, the intention set, and the current substitution, respectively. And, (1) $B \not\models \perp$, (2) for any goal $g \in G$, $B \not\models g$, and $g \not\models \perp$ hold.*

B, G, H, θ are used in defining Mallet configurations, because beliefs, goals, and intentions of an agent are dynamically changing, and a substitution is required to store the current environment bindings for free variables. Plan base P is omitted since we assume P will not be changed at run time.

Similar to [15] we give an auxiliary function to facilitate the definition of semantics of intentions.

Definition 2. *Function agls is defined recursively as: $agls(\top) = \{\}$,
and for any intention $h = [\omega_0 \backslash \cdots \backslash \omega_{k-1} \backslash (\psi_k, A_k) \leftarrow s_k]$ $(k \geq 0)$,
$agls(h) = \{\psi_k\} \cup agls([\omega_0 \backslash \cdots \backslash \omega_{k-1}])$.*

Note that goals in G are top-level goals specified initially, while function *agls* returns a set of achievement goals generated at run time in pursuing some (top-level) goal in G.

4 Operational Semantics

Usually there are two options to defining semantics for an agent-oriented programming language: operational semantics and temporal semantics. For instance, temporal semantics is given to MABLE [16]; while 3APL [17] and AgentSpeak(L) [18] have operational semantics, and transition semantics is defined for ConGolog based on Situation calculus [19]. Temporal semantics is better for property verification using existing tools, such as SPIN (a model checking tool which can check whether temporal formulas hold for the implemented systems), while operational semantics is better for implementing interpreters for the language.

We define an operational semantics for MALLET in terms of a transition system in the hope that it can guide the implementation of interpreters. Each transition corresponds to a single computation step which transforms the system from one configuration to another. A computation run for an agent is a finite or infinite sequence of configurations connected by transition relation $\rightarrow$. The meaning of an agent is a set of computation runs starting from the initial configuration. We assume a belief update function $BU(B, p)$, which revises the belief base B with a new fact p. The details of BU is out the scope of this paper. For convenience in defining semantics, we assume two domain-independent operators working on B: $\mathbf{unsync}(\psi, \rho)$ and $\mathbf{untell}(\psi, s)$. Their effects are to remove all the predicates that can be unified with $sync(?a, \psi, \rho)$ and $ctell(?a, \psi, s, ?id)$, respectively, from B.

4.1 Semantics of beliefs, goals and intentions in MALLET

We allow *explicit negation* in B, and for each $b(\boldsymbol{t}) \in B$, its explicit negation is denoted by $\tilde{b}(\boldsymbol{t})$. Such treatment enables the representation of unknown.

Definition 3. *Given a Mallet configuration $M = \langle B, G, H, \theta \rangle$, for any wff ϕ, any belief or goal formula ψ, ψ', any agent a,*

1. $M \models Bel(\phi)$ *iff* $B \models \phi$,
2. $M \models \neg Bel(\phi)$ *iff* $B \models \tilde{\phi}$,
3. $M \models Unknown(\phi)$ *iff* $B \not\models \phi$ *and* $B \not\models \tilde{\phi}$,
4. $M \models Goal(\phi)$ *iff* $\exists \langle \phi', A \rangle \in G$ *such that* $\phi' \models \phi$ *and* $B \not\models \phi$,
5. $M \models \neg Goal(\phi)$ *iff* $M \not\models Goal(\phi)$,
6. $M \models Goal_a(\phi)$ *iff* $\exists \langle \phi', A \rangle \in G$ *such that* $a \in A$, $\phi' \models \phi$ *and* $B \not\models \phi$,
7. $M \models \neg Goal(\phi)$ *iff* $M \not\models Goal(\phi)$, $M \models \neg Goal_a(\phi)$ *iff* $M \not\models Goal_a(\phi)$,
8. $M \models \psi \wedge \psi'$ *iff* $M \models \psi$ *and* $M \models \psi'$,
9. $M \models Intend(\phi)$ *iff* $\phi \in \bigcup_{h \in H} agls(h)$.

4.2 Transition system

We start with the semantics of termination. As shown in the syntax, termination-conditions can be specified for a plan (we assume the execution of operators always succeed). Given a configuration $\langle B, G, H, \theta \rangle$, a plan template $(\rho\ \boldsymbol{v})$ and an invocation $(\rho\ \boldsymbol{t})$, let $\eta = \{\boldsymbol{v}/\boldsymbol{t}\}$. $\langle B, G, H, \theta \rangle \models isTermed(\rho)$, iff either (1)on entering, $\nexists \tau \cdot B \models pre(\rho)\theta\eta\tau$, and it is specified that plan invocation $(\rho\ \boldsymbol{t})$ fails when $pre(\rho)$ is **false**; or (2)in execution, $\exists \tau \cdot B \models tc(\rho)\theta\eta\tau$ [5]; or (3)on exiting, $\nexists \tau \cdot B \models post(\rho)\theta\eta\tau$. If $\langle B, G, H, \theta \rangle \models isTermed(\rho)$ holds, a predicate of form $(termed\ \rho\ \boldsymbol{t})$ will be asserted into B, so that in later transitions ($isTermed(\rho)$ may be inderivable then) the termination can be propagated upwards to a higher plan level.

[5] It is a successful termination if $\chi(\rho) = \mathbf{succeed}$, and a failure termination if $\chi(\rho) = \mathbf{failure}$. For simplicity, failure termination is assumed in the follows.

Definition 4 (semantics of termination). *Let s be any Mallet statement.* $B \models termed(s)$ *iff*

$$\begin{aligned}
(termed\ \rho\ \boldsymbol{t}) \in B,&\ \textit{if } s = (\rho\ \boldsymbol{t}),\ \textit{where } (\rho\ \boldsymbol{v}) \in Plan\\
(termed\ \rho\ \boldsymbol{t}) \in B,&\ \textit{if } s = (\mathbf{Do}\ A\ (\rho\ \boldsymbol{t})),\ \textit{where } (\rho\ \boldsymbol{v}) \in Plan\\
B \models termed(l_1) \vee termed(l_2),&\ \textit{if } s = (\mathbf{if}\ (\mathbf{cond}\ \psi)\ l_1\ l_2)\\
B \models termed(l_1),&\ \textit{if } s = (\mathbf{while}\ (\mathbf{cond}\ \psi)\ l_1)\\
B \models termed(l_1),&\ \textit{if } s = (l_1; \cdots ; l_m)\\
B \models \bigwedge_{i=1}^{m} termed(l_i),&\ \textit{if } s = (\mathbf{choice}\ l_1 \cdots l_m)\\
B \models \bigvee_{i=1}^{m} termed(l_i),&\ \textit{if } s = (\mathbf{par}\ l_1 \cdots l_m)\\
\nexists \tau \cdot B \models \psi\tau,&\ \textit{if } s = (\mathbf{agent\text{-}bind}\ \boldsymbol{v}\ \psi)\\
B \models \bigvee_{\tau \in \{\theta | B \models \psi\theta\}} termed(l_1\tau),&\ \textit{if } s = (\mathbf{forall}\ (\mathbf{cond}\ \psi)\ l_1)\\
B \models \bigvee_{\tau \in \{\theta | B \models \psi\theta\}} termed(l_1\tau),&\ \textit{if } s = (\mathbf{foreach}\ (\mathbf{cond}\ \psi)\ l_1)\\
B \models \bigvee_{i=1}^{m} termed(l_i),&\ \textit{if } s = (\mathbf{JointDo\ AND}\ (A_1\ l_1) \cdots (A_m\ l_m))\\
B \models \bigwedge_{i=1}^{m} termed(l_i),&\ \textit{if } s = (\mathbf{JointDo\ OR}\ (A_1\ l_1) \cdots (A_m\ l_m))\\
B \models \bigwedge_{i=1}^{m} termed(l_i),&\ \textit{if } s = (\mathbf{JointDo\ XOR}\ (A_1\ l_1) \cdots (A_m\ l_m))
\end{aligned}$$

Note that in Definition 4, the truth of *termed* in the clauses for **if** and **while** is independent from the condition ψ because the truth of ψ might have been changed during the execution of the sub-statements (say, l_1). Also, conjunction rather than disjunction is used in defining the **choice** clause because the semantics of choice allows re-try upon failures: a **choice** statement fails only when all the branches have failed.

Definition 5 (Goal selection).

$$\frac{\exists g = \langle \psi, A\rangle \in G,\ \exists(\rho\ \boldsymbol{v}) \in P,\ self \in A,\quad B \models pre(\rho)\theta\tau,\ post(\rho)\theta\tau \models \psi,\ \boldsymbol{v}\ \textit{is not free in}\ \theta\tau}{\langle B, G, \emptyset, \theta\rangle \rightarrow \langle B, G \setminus \{g\}, \{[(\psi, A) \leftarrow (\mathbf{Do}\ A\ (\rho\ \boldsymbol{v})\theta\tau)]\}, \theta\tau\rangle}, (\mathbf{G1})$$

$$\frac{\forall g = \langle \psi, A\rangle \in G, \forall(\rho\ \boldsymbol{v}) \in P\ \ \nexists\tau \cdot post(\rho)\theta\tau \models \psi}{\langle B, G, \emptyset, \theta\rangle \rightarrow \mathbf{STOP}}, (\mathbf{G2})$$

$$\frac{}{\langle B, \emptyset, \emptyset, \theta\rangle \rightarrow \mathbf{SUCCEED}}. (\mathbf{G3})$$

In Definition 5, Rule **G1** states that when the intention set is empty, the agent will choose one goal from its goal set and select an appropriate plan, if there exists such a plan, to achieve that goal. Rule **G2** states that an agent will stop running if there is no plan can be used to pursue any goal in G. Rule **G3** states that an agent terminates successfully if all the goals and intentions have been achieved. **G1** is the only rule introducing new intentions. It indicates that an agent can only have one intention in focus (it cannot commit to another intention until the current one has already been achieved or dropped). To allow intention shifting (i.e., pursue multiple top-level goals simultaneously), **G1** can be revised by replacing the empty intention set with H.

As defined in Definition 6, when the execution of the top intention slice is done (the body becomes ε), the corresponding achievement goal ψ_k will be checked. If succeed, the intention will be revised with the top slice popped, and the execution of this intention will proceed (**EI1**). Otherwise, the execution stops (**EI2**); this means something was wrong with the plan selection.. Rule **EI3** states that an intention is done successfully and dropped if the ultimate goal ψ_0 is satisfiable. If the agent believes the execution of s is terminated but ψ_0 is not satisfiable, it stops abnormally (**(EI4)**).

Definition 6 (End of intention/intention slice).

$$\frac{B \models \psi_k\theta}{\langle B, G, [\cdots\backslash\omega_{k-1}\backslash(\psi_k, A_k) \leftarrow \varepsilon], \theta\rangle \rightarrow \langle B, G, [\cdots\backslash\omega_{k-1}], \theta\rangle}, (\mathbf{EI1})$$

$$\frac{B \not\models \psi_k\theta}{\langle B, G, [\cdots\backslash\omega_{k-1}\backslash(\psi_k, A_k) \leftarrow \varepsilon], \theta\rangle \rightarrow \mathbf{STOP}}, (\mathbf{EI2})$$

$$\frac{B \models \psi_0\theta, \{[(\psi_0, A_0) \leftarrow s]\} \in H}{\langle B, G, H, \theta\rangle \rightarrow \langle B, G, H - \{[(\psi_0, A_0) \leftarrow s]\}, \theta\rangle}, (\mathbf{EI3})$$

$$\frac{B \not\models \psi_0\theta, B \models \mathit{termed}(s)}{\langle B, G, H \cup \{[(\psi_0, A_0) \leftarrow s]\}, \theta\rangle \rightarrow \mathbf{STOP}}. (\mathbf{EI4})$$

The successful execution of an agent-bind statement is to compose the substitution obtained from evaluating the constraint ϕ with θ (Rule **B1**). The agent stops if there is no solution to the constraints (Rule **B2**).

Definition 7 (Agent selection). *For intention*
$h = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{agent-bind}\ \ \boldsymbol{v}\ \phi); s]$,

$$\frac{B \models \phi\theta\tau}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s], \theta\tau\rangle}, (\mathbf{B1})$$

$$\frac{\nexists\tau \cdot B \models \phi\theta\tau}{\langle B, G, h, \theta\rangle \rightarrow \mathbf{STOP}}. (\mathbf{B2})$$

Given any configuration $\langle B, G, H, \theta\rangle$, for any instantiated plan ρ, variables in $\mathit{body}(\rho)$ are all bounded either by some binding τ where $B \models \mathit{pre}(p)\theta\tau$, or by some preceeding agent-bind statement in $\mathit{body}(\rho)$.

Definition 8 (Sequential execution). *For intention*
$h = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l_1; \cdots; l_m]$,

$$\frac{\langle B, \emptyset, [(\mathit{true}, A_k) \leftarrow l_1], \theta\rangle \rightarrow \langle B', \emptyset, [(\mathit{true}, A_k) \leftarrow \varepsilon], \theta'\rangle}{\langle B, G, h, \theta\rangle \rightarrow \langle B', G, [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l_2; \cdots; l_m], \theta'\rangle}. (\mathbf{SE})$$

Definition 9 (Individual operator execution). *For intention*
$h = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (Do\ a\ (\alpha\ \boldsymbol{t})); s]$,
$h_2 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\alpha\ \boldsymbol{t}); s]$, *where* $(\alpha\ \boldsymbol{v}) \in Ioper$, $\eta = \{\boldsymbol{v}/\boldsymbol{t}\}$,

$$\frac{self = a, B \models pre(\alpha)\theta\eta\tau, B' = BU(B, post(\alpha)\theta\eta\tau)}{\langle B, G, h, \theta\rangle \rightarrow \langle B', G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow l; s], \theta\rangle}, (\mathbf{I1})$$

$$\frac{self = a, \nexists\tau \cdot B \models pre(\alpha)\theta\eta\tau}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s'; s], \theta\rangle}, (\mathbf{I2})$$

$$\frac{self \neq a}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow l_2; s], \theta\rangle}, (\mathbf{I3}),$$

$$\frac{B \models pre(\alpha)\theta\eta\tau, B' = BU(B, post(\alpha)\theta\eta\tau)}{\langle B, G, h_2, \theta\rangle \rightarrow \langle B', G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s], \theta\rangle}, (\mathbf{I4})$$

$$\frac{\nexists\tau \cdot B \models pre(\alpha)\theta\eta\tau}{\langle B, G, h_2, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s''; s], \theta\rangle}, (\mathbf{I5}).$$

where $l = (\mathbf{Do}\ self\ (\mathbf{send}\ A_k \setminus \{self\}, \langle ctell, self, \psi_0, \alpha\rangle))$,
$l_2 = (\mathbf{wait\ until}\ ctell(a, \psi_0, \alpha) \in B)$,
$s' = (\mathbf{wait\ until}\ \exists\tau \cdot B \models pre(\alpha)\theta\eta\tau); (\mathbf{Do}\ self\ (\alpha\ \boldsymbol{t})))$,
$s'' = (\mathbf{wait\ until}\ \exists\tau \cdot B \models pre(\alpha)\theta\eta\tau); (\alpha\ \boldsymbol{t}))$.

Each agent in a team needs to evaluate the top intention slice. Suppose the intention is of form h. In case that an agent is the assigned doer, if the precondition of the individual operator is satisfiable wrt. the agent's belief base, then the execution of the operator is to update the belief base with the postcondition of the action (**I1**); otherwise, the agent has to wait until more information becomes available (**I2**). In case that an agent is not the assigned doer, since the intention is derived from part of a team process, before the agent can proceed, it has to wait until being told about the accomplishment of α (**I3**). Rules **I4** and **I5** are similar to **I1** and **I2** except that the intention is now of form h_2, which by default all the individual agents in A_k are the doers of α.

To execute a team operator, all the involved agents need to synchronize. Let $Y(\psi, \Gamma) = \{a' | sync(a', \psi, \Gamma) \in B\}$, which is a set of agent names who has already sent out synchronization message wrt. ψ and Γ.

In Definition 10, Rule **T1** states that if the agent itself is one of the assigned doers, the preconditions of the team operator holds, and the agent has not synchronized with other agents in A, it will first send out synchronization messages before executing Γ. Rule **T2** states that the agent itself has already synchronized with others, but has not received enough synchronization messages from others, then it continues waiting. Rule **T3** states that the execution of Γ will update B with the effects of the team operator, and before proceed, it has to retrack the sync messages regarding Γ (ensure correct agent behavior in case that Γ needs to be executed later) and inform the agents not in A of the accomplishment of Γ. Rule **T4** deals with the case when the preconditions of Γ does not hold, and Rule **T5** deals with the case when an agent does not belong to A; it has to wait until being informed.

Definition 10 (Team operator execution). *For intention* $h = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (Do\ A\ (\mathit{\Gamma}\ \boldsymbol{t})); s]$, *where* $(\mathit{\Gamma}\ \boldsymbol{v}) \in Toper$, $\eta = \{\boldsymbol{v}/\boldsymbol{t}\}$,

$$\frac{self \in A, B \models pre(\mathit{\Gamma})\theta\eta\tau, sync(self, \psi_0, \mathit{\Gamma}) \notin B}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^1; s], \theta\rangle}, (\mathbf{T1})$$

$$\frac{self \in A, B \models pre(\mathit{\Gamma})\theta\eta\tau, sync(self, \psi_0, \mathit{\Gamma}) \in B, |Y(\psi_0, \mathit{\Gamma})| < |\mathit{\Gamma}|}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^2; s], \theta\rangle}, (\mathbf{T2})$$

$$\frac{\begin{array}{c} self \in A, B \models pre(\mathit{\Gamma})\theta\eta\tau, \\ sync(self, \psi_0, \mathit{\Gamma}) \in B, |Y(\psi_0, \mathit{\Gamma})| \geq |\mathit{\Gamma}|, B' = BU(B, post(\mathit{\Gamma})\theta\eta\tau) \end{array}}{\langle B, G, h, \theta\rangle \rightarrow \langle B', G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^3; s], \theta\rangle}, (\mathbf{T3})$$

$$\frac{self \in A, \nexists\tau \cdot B \models pre(\mathit{\Gamma})\theta\eta\tau}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^4; s], \theta\rangle}, (\mathbf{T4})$$

$$\frac{self \notin A}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^5; s], \theta\rangle}. (\mathbf{T5})$$

where $s^1 = (\mathbf{Do}\ self\ \mathbf{send}(A, \langle sync, self, \psi_0, \mathit{\Gamma}\rangle)); (\mathbf{Do}\ A\ (\mathit{\Gamma}\ \boldsymbol{t}))$,
$s^2 = (\mathbf{wait\ until}\ (|Y(\psi_0, \mathit{\Gamma})| \geq |\mathit{\Gamma}|)); (\mathbf{Do}\ A\ (\mathit{\Gamma}\ \boldsymbol{t}))$,
$s^3 = (\mathbf{Do}\ self\ \mathbf{unsync}(\psi_0, \mathit{\Gamma})); (\mathbf{Do}\ self\ \mathbf{send}(A_k \setminus A, \langle ctell, self, \psi_0, \mathit{\Gamma}\rangle))$,
$s^4 = (\mathbf{wait\ until}\ \exists\tau \cdot B \models pre(\mathit{\Gamma})\theta\eta\tau); (\mathbf{Do}\ A\ (\mathit{\Gamma}\ \boldsymbol{t}))$,
$s^5 = (\mathbf{wait\ until}\ \forall a \in A \cdot ctell(a, \psi_0, \mathit{\Gamma}) \in B)$.

The semantics of joint-do is a little complicated. A joint-do statement implies agent synchronization both at the beginning and at the end of its execution. Its semantics is given in terms of basic constructs.

Definition 11 (Joint-Do). *For intentions*
$h_1 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\mathbf{joint\text{-}do\ AND}\ (A'_1\ l_1) \cdots (A'_n\ l_n)); s]$,
$h_2 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\mathbf{joint\text{-}do\ OR}\ (A'_1\ l_1) \cdots (A'_n\ l_n)); s]$,
$h_3 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\mathbf{joint\text{-}do\ XOR}\ (A'_1\ l_1) \cdots (A'_n\ l_n)); s]$,

$$\frac{\bigcap_{j=1}^{n} A'_j = \emptyset, self \in A'_i}{\langle B, G, h_1, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^1; s], \theta\rangle}, (\mathbf{J1})$$

$$\frac{\bigcap_{j=1}^{n} A'_j = \emptyset, self \in A'_i}{\langle B, G, h_2, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^0; s^{21}; s^{22}; s^0; s], \theta\rangle}, (\mathbf{J2})$$

$$\frac{self \in A'_i, isSelected(A'_i)}{\langle B, G, h_3, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^1; s], \theta\rangle}, (\mathbf{J3})$$

$$\frac{self \in A'_i, \neg isSelected(A'_i)}{\langle B, G, h_3, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^0; s^0; s], \theta\rangle}, (\mathbf{J4}), where$$

$s^0 = (\mathbf{Do}\ self\ (\mathbf{send}\ \bigcup_{j=1}^{n} A'_j, \langle sync, self, \psi_0, nil\rangle));$
$\quad (\mathbf{wait\ until}\ (\forall a \in \bigcup_{j=1}^{n} A'_j \cdot sync(a, \psi_0, nil) \in B)); (\mathbf{Do}\ self\ (\mathbf{unsync}\ \psi_0, nil))$;
$s^1 = s^0; (\mathbf{Do}\ A'_i\ l_i); s^0$,
$s^{21} = (\mathbf{If}(\mathbf{cond}\ \ \nexists l_x, a \cdot ctell(a, \psi_0, l_x, 0) \in B)$
$\quad (s^3; (\mathbf{Do}\ A'_i\ l_i); (\mathbf{Do}\ self\ (\mathbf{send}\ \bigcup_{j=1, j\neq i}^{n} A'_j, \langle ctell, self, \psi_0, l_i, 1\rangle))\ )\ )$,
$s^3 = (\mathbf{If}\ (\mathbf{cond}\ \ \nexists a \cdot cask(a, \psi_0, l_i) \in B)$
$\quad (\ (\mathbf{Do}\ self\ (\mathbf{send}\ \bigcup_{j=1, j\neq i}^{n} A'_j, \langle ctell, self, \psi_0, l_i, 0\rangle));$
$\quad\ (\mathbf{Do}\ self\ (\mathbf{send}\ A'_i \setminus \{self\}, \langle cask, self, \psi_0, l_i\rangle))\ )\ )$,

$s^{22} = (\textbf{while}(\textbf{cond}\ \exists \phi_x, a \cdot ctell(a, \psi_0, l_x, 0) \in B)$
$\quad (\textbf{wait until}\ \forall b \in A'_x \cdot ctell(b, \psi_0, l_x, 1) \in B); (\textbf{Do}\ (\textbf{untell}\ \psi_0, l_x))\ ).$

Rule **J1** defines semantics for joint-do with share type "AND". It states that before and after an agent does its task l_i, it needs to synchronize (i.e., s^0) with the other teammates wrt. l_i. A joint-do statement with share type "OR" requires that at least one sub-process has to be executed. In Rule **J2**, the joint-do statement is replaced by $s^0; s^{21}; s^{22}; s^0$. s^{21} states that if an agent has not received any message regarding the start of some sub-statement l_x (i.e., this agent itself is the first ready to execute the joint-do statement), it will sequentially do (a) s^3: if among A'_i this agent is the first ready to execute l_i, then tell all other agents not in A'_i regarding the start of l_i (i.e., $\langle ctell \cdots 0\rangle$) and request other agents in A'_i to execute l_i; (b) agents in A'_i together execute l_i; (c) tell other agents not in A'_i the accomplishment of l_i (i.e., $\langle ctell \cdots 1\rangle$). s^{22} states in case that this agent was informed of the start of some other sub-statement l_x, it needs to wait until being informed by all the doers that l_x has been completed. The semantics of joint-do with share type "XOR" is based on a function $isSelected()$[6]: if an agent belongs to the group of selected agents, it simply synchronizes and executes the corresponding sub-statement (Rule **J3**); otherwise, only synchronization is needed (Rule **J4**).

Definition 12 (Plan entering, executing and exiting). *Let*
$h_1 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\textbf{Do}\ A\ (\rho\ \boldsymbol{t})); s]$,
$h'_1 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\textbf{Do}\ A\ (\rho\ \boldsymbol{t}))\theta\eta\tau; s\theta]$,
$h''_1 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\textbf{Do}\ A\ (\rho\ \boldsymbol{t}))\theta\eta\tau; s\theta \backslash (post(\rho)\theta\eta\tau, A) \leftarrow \textbf{endp}]$,
$h'''_1 = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\textbf{Do}\ A\ (\rho\ \boldsymbol{t}))\theta\eta\tau; s\theta \backslash (post(\rho)\theta\eta\tau, A) \leftarrow l_1; \cdots; l_m; \textbf{endp}]$,
where $(\rho\ \boldsymbol{v}) \in Plan$, $\eta = \{\boldsymbol{v}/\boldsymbol{t}\}$,

$$\frac{self \notin A}{\langle B, G, h_1, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^3; s], \theta\rangle}, \ (\textbf{P1})$$

$$\frac{self \in A, \langle B, G, h_1, \theta\rangle \models isTermed(\rho), B' = BU(B, (termed\ \rho\ \boldsymbol{t}))}{\langle B, G, h_1, \theta\rangle \rightarrow \langle B', G, h_1, \theta\rangle}, \ (\textbf{P2})$$

$$\frac{self \in A, B \models pre(\rho)\theta\eta\tau}{\langle B, G, h_1, \theta\rangle \rightarrow \langle B, G, [h'_1 \backslash (post(\rho)\theta\eta\tau, A) \leftarrow s^1; \textbf{endp}], \theta\eta\tau\rangle}, \ (\textbf{P3})$$

$$\frac{self \in A, \langle B, G, h'''_1, \iota\rangle \models isTermed(\rho), B' = BU(B, (termed\ \rho\ \boldsymbol{t}))}{\langle B, G, h'''_1, \iota\rangle \rightarrow \langle B', G, h''_1, \iota\rangle}, \ (\textbf{P4})$$

$$\frac{self \in A, B \models termed(l_1), B' = BU(B, (termed\ \rho\ \boldsymbol{t}))}{\langle B, G, h'''_1, \iota\rangle \rightarrow \langle B', G, h''_1, \iota\rangle}, \ (\textbf{P5})$$

$$\frac{self \in A, B \not\models termed(\rho), B' = BU(B, post(\rho)\theta)}{\langle B, G, h''_1, \theta\rangle \rightarrow \langle B', G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s], \theta\rangle}, \ (\textbf{P6})$$

$$\frac{self \in A, B \models termed(\rho)}{\langle B, G, h''_1, \theta\rangle \rightarrow \langle B, G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s^2], \theta\rangle}, \ (\textbf{P7}), where$$

$s^1 = (\textbf{Do}\ self\ (\textbf{send}\ A, \langle sync, self, \psi_0, \rho\rangle)); (\textbf{wait until}\ (\forall a \in A \cdot sync(a, \psi_0, \rho) \in B));$
$(\textbf{Do}\ self\ (\textbf{unsync}\ \psi_0, \rho)); body(\rho)\theta\eta\tau;$

[6] Some negotiation strategy is required to define *isSelect*; this is left to the designers of MALLET interpreters.

$(\textbf{Do}\ \mathit{self}\ (\textbf{send}\ A_k, \langle \mathit{ctell}, \mathit{self}, \psi_0, \rho\rangle)); (\textbf{wait until}\ (\forall a \in A \cdot \mathit{ctell}(a, \psi_0, \rho) \in B)),$
$s^2 = (\textbf{Do}\ \mathit{self}\ (\textbf{send}\ A_k, \langle \mathit{unachievable}, \mathit{self}, \psi_0, \rho\rangle));$
$\quad (\textbf{wait until}\ (\forall a \in A \cdot \mathit{unachievable}(a, \psi_0, \rho) \in B)),$
$s^3 = (\textbf{wait until}\ (\forall a \in A \cdot \mathit{unachievable}(a, \psi_0, \rho) \in B \vee \forall a \in A \cdot \mathit{ctell}(a, \psi_0, \rho) \in B)).$

Plan execution is a process of hierarchical expansion of (sub-)plans. Rule **P1** states that if an agent is not involved, it simply waits until ρ is done. Before entering a plan, an agent first checks the corresponding pre-conditions. Rule **P2** applies when the preconditions does not hold and "wait" is specified as agents' response (rules can be given for other responses such as "fail" and "achieve", refer to syntax). Rule **P3** applies when the preconditions holds. s^1 states that on entering a plan, a new intention slice will be appended where the agent needs to synchronize with others (when everyone is ready the synchronization messages are dropped to ensure that this plan can be properly re-entered later), and then execute the plan body instantiated by the environment binding θ and local binding τ, which is followed by communications (tell other agents not involved in ρ about the accomplishment of ρ), synchronizations, and **endp**. Rule **P4** and **P5** applies when executing a plan. An agent will give up executing ρ in case that either $isTermed(\rho)$ is derivable from the current configuration (Rule **P4**); or the first statement of the top intention slice is terminated (Rule **P5**). In both cases, all the statements before **endp** are omitted. On exiting a plan (**endp** is the only statement in the body of the top intention slice), the top intention slice is popped. If ρ has been successfully executed, the DO statement will be dropped and B is updated with the effects of ρ (Rule **P6**); otherwise, inform agents in A_k of the inachievability of ρ (Rule **P7**). The semantics of plan invocation of form $(\rho\ \boldsymbol{t})$ (i.e., no doers are explicitly specified) can be similarly defined, except that A_k will be used as the doers of ρ.

The **choice** construct can be used to specify explicit choice points in a complex team process. For example, suppose a fire-fighting team is assigned to extinguish a fire caused by an explosion at a chemical plant. After collecting enough information (e.g., chemicals in the plant, nearby dangerous facilities, etc.), the team needs to decide how to put out the fire. They have to select one plan if there exist several options. The **choice** construct is composed of a list of branches, each of which specifies a plan (a course of actions) and may be associated with preference conditions and a priority information. The preference conditions of a branch is a collection of first-order formulas; the evaluation of their conjunction determines whether the branch is workable under that context. The priority information is used in selecting a branch in case that the preference conditions of more than one branch are satisfiable.

Given a configuration $\langle B, G, H, \theta\rangle$ and a statement $(\textbf{choice}\ Br_1\ Br_2 \cdots Br_m)$ where $Br_i = (\mathit{pref}_i\ \mathit{pro}_i\ (\textbf{DO}\ A_i\ (\rho_i\ t_i)))$, let $BR = \{Br_i | 1 \le i \le m\}$, $BR_- \subseteq BR$ be the set of branches in BR already considered. We assume B can track the changes of BR_-. Let $BR^+ = \{Br_k | \exists\tau \cdot B \models \mathit{pref}_k \cdot \theta\tau, 1 \le k \le m\} \setminus BR_-$, which is the set of branches that has not been considered and the associated preference conditions can be satisfied by B. In addition, let $BR^\oplus$ be the subset of BR^+

such that all the branches in $BR^{\oplus}$ have the maximal priority value among those in BR^{+}, and $ram(BR^{\oplus})$ can randomly select and return one branch from $BR^{\oplus}$.

Definition 13 (Choice construct). *Let*
$h = [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow (\mathbf{choice}\ Br_1\ Br_2 \cdots Br_m); s]$,
$h_1 = [h \backslash (true, A_k) \leftarrow (\mathbf{DO}\ A_i\ (\rho_i\ t_i)); \mathbf{cend}_i]$,

$$\frac{ram(BR^{\oplus}) = Br_i, B' = BU(B, BR_{-}.add(Br_i))}{\langle B, G, h, \theta\rangle \rightarrow \langle B', G, [h \backslash (true, A_k) \leftarrow (\mathbf{DO}\ A_i\ (\rho_i\ t_i)); \mathbf{cend}_i], \theta\rangle}, \mathbf{(C1)}$$

$$\frac{ram(BR^{\oplus}) = null}{\langle B, G, h, \theta\rangle \rightarrow \langle B, G, h, \theta\rangle}, \mathbf{(C2)}$$

$$\frac{self \in A_i, \langle B, G, h_1, \theta\rangle \models isTermed(\rho_i), B' = BU(B, (termed\ \rho_i\ \boldsymbol{t}_i))}{\langle B, G, h_1, \theta\rangle \rightarrow \langle B', G, h, \theta\rangle}, \mathbf{(C3)}$$

$$\frac{self \in A_i, B \models termed(\rho_i)}{\langle B, G, [h \backslash (true, A_k) \leftarrow \mathbf{cend}_i], \theta\rangle \rightarrow \langle B, G, h, \theta\rangle}, \mathbf{(C4)}$$

$$\frac{self \in A_i, B \not\models termed(\rho_i), B' = BU(B, post(\rho_i)\theta)}{\langle B, G, [h \backslash (true, A_k) \leftarrow \mathbf{cend}_i], \theta\rangle \rightarrow \langle B', G, [\omega_0 \backslash \cdots \backslash (\psi_k, A_k) \leftarrow s], \theta\rangle}. \mathbf{(C5)}$$

In Definition 13, Rule **C1** applies when there exists a workable branch. The intention h is appended with a new slice ended with $\mathbf{cend}_i$ so that the agents in A_k can backtrack to the latest choice point as ρ_i fails. An agent has to wait (e.g., for more information becomes available) if there is no workable branch (Rule **C2**). Rule **C3** applies when an agent starts to do ρ_i but the preconditions does not hold (i.e., $isTermed(\rho_i)$ is true on entering): it returns to the choice point (to try another branch). When an agent comes to statement $\mathbf{cend}_i$ and finds out that ρ_i is terminated abnormally (e.g., the performance does not result in the expected effects), then return to the choice point (Rule **C4**). In case that an agent comes to statement $\mathbf{cend}_i$ and the execution of ρ_i is successful, it proceeds to the next statement following the choice point (Rule **C5**).

Par is a construct that takes a list of processes and executes them in any order. For instance, an agent can safely execute walking and chewing gum in either order or at the same time with no conflict. When each process in the list has completed successfully, the entire **par** process is said to complete successfully. If at any point one of the process fails, then the entire **par** process returns failure and gives up executing any of the statements after that point.

Intuitively, a parallel statement with k branches requires the current process (transition) split itself into k processes. These spawned processes each will be responsible for the execution of exactly one parallel branch, and they have to be merged into one process immediately after all have completed their responsibility. To prevent the spawned processes from committing to other tasks, their initial transitions need to be established such that (1) the intention set only has one intention with one intention slice at its top; (2) the goal base is empty (so that the transition cannot proceed further after the unique intention has been completed). Because the original goal set and intention set have to be recovered after the execution of the parallel statement, we adopt an extra transition, which has the same components as the original transition except that # is pushed as the top intention slice. This indicates the intention is *suspended.* Note that the other intentions in the intention set may still be executable, which may change the belief base and substitution of the transition.

Definition 14 (Parallel construct). *Let* $h_0 = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s_k; s]$, $h = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s_k; s\backslash\#]$, *where* $s_k = (\textbf{par}\ l_1\ l_2 \cdots l_m)$, $T_j = \langle B, \emptyset, [(true, A_k) \leftarrow l_j], \theta\rangle \rightarrow^* \langle B_j, \emptyset, [(true, A_k) \leftarrow \varepsilon], \theta_j\rangle \wedge B_j \not\models termed(l_j)$, *and* $P_B = \langle B, G, h, \theta\rangle \parallel \langle B, \emptyset, [(true, A_k) \leftarrow l_1], \theta\rangle \parallel \cdots \parallel \langle B, \emptyset, [(true, A_k) \leftarrow l_m], \theta\rangle$,

$$\frac{B \not\models termed(s_k)}{\langle B, G, h_0, \theta\rangle \rightarrow P_B}, (\textbf{PA1})$$

$$\frac{\bigwedge_{j=1}^{m}(T_j), B' = BU(\bigcup_{j=1}^{m} B_j, B_0), \theta' = \theta_0\theta_1\cdots\theta_m}{\langle B_0, G, h, \theta_0\rangle \rightarrow \langle B', G, [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s], \theta'\rangle}, (\textbf{PA2})$$

$$\frac{\exists j, \langle B, \emptyset, [(true, A_k) \leftarrow l_j], \theta\rangle \rightarrow^* \langle B_j, \emptyset, [(true, A_k) \leftarrow\ l'_j], \theta_j\rangle, B_j \models termed(l'_j)}{\langle B_0, G, h, \theta_0\rangle \rightarrow \langle B_0, G, h_0, \theta_0\rangle}.(\textbf{PA3})$$

Now, it's easy to define semantics for composite processes. For instance, **forall** construct is an implied **par** over the condition bindings, whereas **foreach** is an implied **seq** over the condition bindings. The constructs **forall** and **foreach** are fairly expressive when the number of choices is unknown before runtime.

Definition 15 (Composite plans). *Let*
$h_1 = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{if}\ (\textbf{cond}\ \phi)\ l_1\ l_2); s]$,
$h_2 = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{while}\ (\textbf{cond}\ \phi)\ l); s]$,
$h_3 = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{foreach}\ (\textbf{cond}\ \phi)\ l); s]$,
$h_4 = [\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{forall}\ (\textbf{cond}\ \phi)\ l); s]$,

$$\frac{B \models \phi\theta\tau}{\langle B, G, \{h_1\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l_1\tau; s]\}, \theta\rangle}, (\textbf{S1})$$

$$\frac{\nexists\tau \cdot B \models \phi\theta\tau}{\langle B, G, \{h_1\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l_2; s]\}, \theta\rangle}, (\textbf{S2})$$

$$\frac{B \models \phi\theta\tau}{\langle B, G, \{h_2\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l\tau; (\textbf{while}\ (\textbf{cond}\ \phi)\ l); s], \theta\rangle}, (\textbf{S3})$$

$$\frac{\nexists\tau \cdot B \models \phi\theta\tau}{\langle B, G, \{h_2\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s], \theta\rangle}, (\textbf{S4})$$

$$\frac{\exists\tau_1, \cdots, \tau_k \cdot \bigwedge_{j=1}^{k} B \models \phi\theta\tau_j}{\langle B, G, \{h_3\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow l\tau_1; \cdots; l\tau_k; s]\}, \theta\rangle}, (\textbf{S5})$$

$$\frac{\nexists\tau \cdot B \models \phi\theta\tau}{\langle B, G, \{h_3\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s]\}, \theta\rangle}, (\textbf{S6})$$

$$\frac{\exists\tau_1, \cdots, \tau_k \cdot \bigwedge_{j=1}^{k} B \models \phi\theta\tau_j}{\langle B, G, \{h_4\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow (\textbf{par}\ l\tau_1\ \cdots l\tau_k); s]\}, \theta\rangle}, (\textbf{S7})$$

$$\frac{\nexists\tau \cdot B \models \phi\theta\tau}{\langle B, G, \{h_4\}, \theta\rangle \rightarrow \langle B, G, \{[\omega_0\backslash\cdots\backslash(\psi_k, A_k) \leftarrow s]\}, \theta\rangle}, (\textbf{S8})$$

5 Conclusion

MALLET is a language that organizes plans hierarchically in terms of different process constructs such as sequential, parallel, selective, iterative, or conditional.

It can be used to represent teamwork knowledge in a way that is independent of the context in which the knowledge is used. This paper defined an operational semantics for MALLET in terms of a transition system, which is important in further studying the formal properties of team-based agents specified in MALLET. The effectiveness of MALLET in encoding complex teamwork knowledge was already shown in the CAST system [8], which implemented an interpreter for MALLET using PrT nets as the internal representation of team process.

References

1. Cohen, P.R., Levesque, H.J.: Teamwork. Nous **25** (1991) 487–512
2. Cohen, P.R., Levesque, H.J., Smith, I.A.: On team formation. In Hintikka, J., Tuomela, R., eds.: Contemporary Action Theory. (1997)
3. Jennings, N.R.: Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence **75** (1995) 195–240
4. Grosz, B., Kraus, S.: Collaborative plans for complex group actions. Artificial Intelligence **86** (1996) 269–358
5. Tambe, M.: Towards flexible teamwork. Journal of AI Research **7** (1997) 83–124
6. Rich, C., Sidner, C.: Collagen: When agents collaborate with people. In: Proceedings of the International Conference on Autonomous Agents (Agents'97). (1997)
7. Giampapa, J., Sycara, K.: Team-oriented agent coordination in the RETSINA multi-agent system. Technical Report CMU-RI-TR-02-34, CMU (2002)
8. Yen, J., Yin, J., Ioerger, T., Miller, M., Xu, D., Volz, R.: CAST: Collaborative agents for simulating teamworks. In: Proceedings of IJCAI'2001. (2001) 1135–1142
9. Tidhar, G.: Team oriented programming: Preliminary report. In: Technical Report 41, AAII, Australia. (1993)
10. Pynadath, D.V., Tambe, M., Chauvat, N., Cavedon, L.: Toward team-oriented programming. In: Agent Theories, Architectures, and Languages. (1999) 233–247
11. Scerri, P., Pynadath, D.V., Schurr, N., Farinelli, A.: Team oriented programming and proxy agents: the next generation. In: Proc. of the 1st Inter. Workshop on Prog. MAS at AAMAS'03. (2003)
12. Rao, A.S., Georgeff, M.P., Sonenberg, E.A.: Social plans: A preliminary report. In Werner, E., Demazeau, Y., eds.: Decentralized AI 3 –Proceedings of MAAMAW-91), Elsevier Science B.V.: Amsterdam, Netherland (1992) 57–76
13. Kinny, D., Ljungberg, M., Rao, A.S., Sonenberg, E., Tidhar, G., Werner, E.: Planned team activity. In Castelfranchi, C., Werner, E., eds.: Artificial Social Systems (LNAI-830), Springer-Verlag: Heidelberg, Germany (1992) 226–256
14. Tidhar, G., Rao, A., Sonenberg, E.: Guided team selection. In: Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96). (1996)
15. Bordini, R., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agentspeak. In: Proceedings of AAMAS-2003. (2003) 409–416
16. Wooldridge, M., Fisher, M., Huget, M., Parsons, S.: Model checking multiagent systems with MABLE. In: Proceedings of AAMAS-2002. (2002)
17. Dastani, M., van Riemsdijk, B., Dignum, F., Meyer, J.J.C.: A programming language for cognitive agents: Goal directed 3APL. In: Proc. of the 1st Inter. Workshop on Prog. MAS at AAMAS'03. (2003)
18. Rao, A.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: MAAMAW'96, LNAI 1038, Springer-Verlag: Heidelberg, Germany (1996) 42–55
19. Giacomo, G.D., Lesperance, Y., Levesque, H.J.: ConGolog, a concurrent programming language based on the situation calculus. AI **121** (2000) 109–169