# Goals in the Context of BDI Plan Failure and Planning *

Sebastian Sardina
RMIT University
Melbourne, Australia
ssardina@cs.rmit.edu.au

Lin Padgham
RMIT University
Melbourne, Australia
linpa@cs.rmit.edu.au

## ABSTRACT

We develop a Belief-Desire-Intention (BDI) style agent-oriented programming language with special emphasis on the semantics of *goals* in the presence of the typical BDI failure handling present in many BDI systems and a novel account of hierarchical lookahead planning. The work builds incrementally on two existing languages and accommodates three type of goals: classical BDI-style event goals, declarative goals, and planning goals. We mainly focus on the *dynamics* of these type of goals and, in particular, on a kind of commitment scheme that brings the new language closer to the solid existing work in agent theory. To that end, we develop a semantics that recognises the usual hierarchical structure of active goals as well as their declarative aspects. In contrast with previous languages, the new language prevents an agent from blindly persisting with a (blocked) *subsidiary* goal when an alternative strategy for achieving a higher-level motivating goal exists. In addition, the new semantics ensures watchfulness by the agent to ensure that goals that succeed or are deemed impossible are immediately dropped, thus conforming to the requirements of basic rational commitment strategy. Finally, a mechanism for the proactive adoption of new goals, other than the mere reaction to events, and a formal account of interaction with the external environment are provided. We believe that the new language is an important step towards turning *practical* BDI programming languages more compatible with the established results in the area of agent theory.

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent Agents,Languages and structures*

## 1. INTRODUCTION

It is well accepted that goals are a central concept for intelligent agents. The BDI (Belief-Desire-Intention) model, based originally on the philosophical work of Bratman [3] and Dennet [9], has been developed into both formal and implemented agent programming languages such as PRS [19], AGENTSPEAK [25], 3APL [16], JACK [5], and CAN [33]. These BDI agent-oriented systems are extremely flexible and responsive to the environment, and as a result, well suited for complex applications with (soft) real-time reasoning and control requirements. However, partly due to practical concerns, the level of support for representing and reasoning about goals has not been commensurate with their importance. For example, AGENTSPEAK—one of the most frequently referenced BDI agent programming languages—has a relatively simplistic representation of goals based on the so-called *events*. The language does not (natively) account for declarative aspects of goals and fails to capture the typical behaviour of implemented BDI systems, where commitment to a goal ensures that if one approach to achieving a goal fails, other possible approaches are tried.

The language CAN [33] (partially) addressed the above two issues by enriching the notion of event-goal with some declarative content and by providing a "fail-and-retry" failure handling mechanism for failed events. Its semantics is then, at some level, consistent with some desired properties of (declarative) goals [7, 24]: persistent, possible, and unachieved. In turn, CANPLAN [27] refined and built on CAN to provide a new kind of goals, namely, *lookahead planning* goals. This was done by integrating an account of HTN-style planning within BDI-type agents. One can distinguish then three types of goals: classical event-goals, declarative event-goals, and planning goals. Nonetheless, a deep study on the properties of these goals has yet not been performed. Even worse, CAN(PLAN)[1] agents fall short on goals. For example, they may sometimes pursue achieved or impossible goals, or even overcommit to a declarative event-goal. Also, the differences between a goal and its subgoals are not sufficiently represented in either languages.

In this paper, we take goals seriously and develop CANPLAN2, a modular extension of CANPLAN which provides: (a) a more appropriate commitment strategy than that provided by AGENTSPEAK or CAN(PLAN); (b) a semantics which guarantees agent watchfulness regarding fortuitous goal achievement or failure; (c) a differentiation between event-goals (a "reactive" form of goal) and declarative goals (a more persistent type of goals); (d) a mechanism for proactively adopting new goals, other than a simple reaction to external events; and (e) an execution cycle compatible with Rao's well-known abstract architecture for rational agents [26].

Appropriate persistence of goals and commitment of an agent to its corresponding intentions is a hallmark of the intelligent agent paradigm. The widely accepted *single-minded* commitment strategy of Rao and Georgeff [24] requires an agent to maintain its commitment to a goal until this either succeeds or is believed impossible. AGENTSPEAK, however, allows to drop a whole intention as

*We would like to acknowledge the support of Agent Oriented Software and of the Australian Research Council under the grant "Learning and Planning in BDI Agents" (number LP0560702). We also thank the anonymous reviewers for their detailed comments.


*AAMAS'07,* May 14–18, 2007, Honolulu, Hawai'i, USA.


[1] CAN(PLAN) will denote both CAN and CANPLAN languages.



978-81-904262-7-5 (RPS) 

soon as it is not possible to make a step on it, i.e., if the intention is *blocked*. As the blocking may well be temporary or there could be alternative ways to achieve the goal in question, this is a low level type of commitment. CANPLAN, on the other hand, maintains the commitment to a goal until it either succeeds or is deemed impossible by virtue of a special failure condition. While this level of commitment may well be appropriate for top-level goals, it is not always so for a (sub)goal that has been adopted purely in the service of some other higher level *motivating goal*. For example suppose an agent has the goal to quench her thirst, and in the service of this goal, she adopts the (sub)goal to buy a can of soda. However, upon arrival to the store, she realises that all the cans of soda are sold out. Fortunately, though, the shop has bottles of water. In this situation, it is irrational for the agent to drop the whole goal of quenching her thirst just because soda is not available. An AGENTSPEAK agent may do so though. Similarly, we do not expect the agent to fanatically insist on her (sub)goal and just wait indefinitely for soda to be delivered. A CAN(PLAN) agent may indeed do. What we expect is the agent to merely drop her commitment to buy soda and adopt the alternative (sub)goal to buy a bottle of water, thereby achieving the *main* goal. This is the commitment semantics associated to goals and intentions that we shall develop in this paper.

## 2. BDI PROGRAMMING & GOALS

Generally speaking, BDI agent-oriented programming languages are built around an explicit representation of beliefs, desires, and intentions. A BDI architecture addresses how these components are represented, updated, and processed to determine the agent's actions. There are a number of agent programming languages in the BDI tradition, such as AGENTSPEAK and JASON [25, 1], PRS [19], JAM [17], JACK [5], and 3APL [16].

An agent consists, basically, of a belief base, a set of recorded pending events (goals), a plan library, and an intention base. The *belief base* encodes the agent's knowledge about the world. The *plan library* contains *plan rules* of the form $e : \psi \leftarrow P$ encoding a *plan-body program* $P$ for handling an event-goal $e$ when *context* condition $\psi$ is believed to hold. The *intention base* contains the current, partially instantiated, plans that the agent has already committed to in order to handle or achieve some event-goal.

A BDI system responds to *events*, the inputs to the system, by committing to handle one pending event-goal, selecting a plan rule from the library, and placing its plan-body program into the intention base. The execution of this program may, in turn, post new subgoal events to be achieved. If at any point a program fails, then an alternative plan rule is found and its plan-body is placed into the intention base for execution. This process repeats until a plan succeeds completely or until there are no more applicable plans, in which case failure is propagated to the event-goal.

BDI agent systems, such as PRS [12] and its various successors, were developed as a way of enabling *abstract plans* written by programmers to be combined and used in real-time, in a way that was both flexible and robust. In contrast with traditional planning, *execution* happens at each step. The assumption is that the use of plans' preconditions to make choices as late as possible, together with the built-in mechanism for retrying alternative options, ensures that a successful execution will eventually be obtained, if possible, even in the context of changes in the environment. Nonetheless, the benefits and feasibility of sometimes engaging in (restricted) *lookahead planning* within the BDI execution framework has recently been recognised too [27, 10]. In section 3, we shall discuss in detail one formal BDI language of this sort.

The concept of a *goal* is central to both agent theory and agent programming. Agents behave because they try to satisfy and bring about their goals, desires, and objectives. Goals explain and specify the agent's (proactive) behaviour.

In agent theory [7, 24, 13] and planning, goals are interpreted in a *declarative* way, as states of affairs to reach (e.g., not being thirsty). In contrast, most agent programming languages have taken a *procedural* perspective on goals in that these are *tasks* or *processes* the agent need to complete or execute (e.g., drink water). The need to conveniently accommodate declarative aspects of goals in these languages has recently attracted much attention [33, 28, 32, 18, 8]. As argued in [32], even a limited account of declarative goals can help decouple plan execution and goal achievement [15, 33], facilitate goal dynamics [31, 29] and sophisticated plan failure handling [33, 18], enable reasoning about goal and plan interaction [30], enhance goal and plan communication [20], etc.

When it comes to declarative goals there are some desired properties that have to be satisfied [7]: persistent, possible, and unachieved. In particular, the *persistence* of goals and intentions depend much on the agent's commitment strategy [7, 13], that is, the various ways in which the agent deals with goal and plan abandonment. For example, a *blindly* committed agent only drops a goal when this is believed to be achieved; a *single-minded* agent also abandons a goal if this is deemed impossible; and an *open-minded* agent would even drop a goal if its motivation is no longer valid.

Goal adoption is also an important issue [21, 15, 31, 32]: *when* and *which* goals are to be adopted by the agent. There are many reasons why an agent would adopt or generate a new goal. A new goal may be adopted due to communication [20] (e.g., another agent requesting to get the room cleaned), or due to social norms or obligations [4] (e.g., keeping your workplace in good condition). Internal motivations could also result in the adoption of new goals. For instance, an internal agent's desire (e.g., never be thirsty) may activate and generate a new goal to pursue, or a current active plan may require the adoption of a new *subgoal* (e.g., buy a drink) in the service of a more abstract goal (e.g., quench thirst) [32].

Below, we shall develop an agent programming language that addresses many, though not all, of the above issues. The long-term objective is an agent framework, with formal semantics, that take goals seriously by accommodating some typical aspects of goals generally recognised in agent theory, while still keeping it practical.

## 3. THE BASIC BDI LANGUAGE

In this section and the next one, we shall develop a BDI-type agent-oriented language, which we call CANPLAN2. The technical machinery we use to define our language is essentially that of [33] and [27]. However, some adaptation is necessary to suitably model goals, both declarative and procedural, in the context of BDI failure handling and hierarchical planning. In this section, we focus on CANPLAN2's language and its basic operational semantics capturing the execution of individual intentions. Then, in the next section, we will define the operational semantics of a whole agent who is pursing multiple goals and intentions concurrently.

So, the language we propose refines and modifies CANPLAN [27] to improve the semantics associated with goals. As a consequence, the revised language more accurately captures the widely accepted abstract agent interpreter as described by Rao and Georgeff in [26]. The new language also includes a proactive mechanism for adopting new goals as well as a more expressive semantics for sensing information. For legibility and coherence, we present the full CANPLAN2 language, and highlight and discuss the new features and changes with respect to its predecessor language CANPLAN.

An *agent configuration* is defined, as in CANPLAN, by a tuple consisting of an action description library $\Lambda$, a plan library $\Pi$, a belief base $\mathcal{B}$, the sequence of actions already performed by the

agent $\mathcal{A}$, and the set of current (active) intentions $\Gamma$. In addition, a CANPLAN2 agent configuration also includes a *motivation library* $\mathcal{M}$, which, as we shall see in section 4, allows for the proactive generation and potential adoption of new goals by the agent.

The *belief base* $\mathcal{B}$ is a set of formulae from any (knowledge representation) logical language that allows for operations to check whether a condition $\phi$—a logical formula over the agent's beliefs—follows from a belief set (i.e., $\mathcal{B} \models \phi$), and to add and delete a belief $b$ to and from a belief base (i.e., $\mathcal{B} \cup \{b\}$ and $\mathcal{B} \setminus \{b\}$, respectively).[2]

The agent's *plan library* $\Pi$ consists of a collection of plan rules of the form $e : \psi \leftarrow P$, where $e$ is an event and $\psi$ is the (belief) context condition which must be true for the plan-body $P$ to be applicable.[3] The *plan-body* or *program* $P$ is built from primitive actions $act$ that the agent can execute directly, operations to add $+b$ and delete $-b$ beliefs, tests for conditions $?\phi$, event-goals $!e$, and the important *declarative* event goal construct $\mathsf{Goal}(\phi_s, !e, \phi_f)$ that was introduced in CAN. Complex plans can be specified using sequencing $P_1; P_2$ and parallelism $P_1 \| P_2$. In addition to the above *user plan language* there are also a number of auxiliary plan forms which are used internally when assigning semantics to constructs: basic (terminating) program *nil*; and compound plans $P_1 \rhd P_2$, which executes $P_1$ and then executes $P_2$ only if $P_1$ has failed; $(\!|\{\psi_1 : P_1, \ldots, \psi_n : P_n\}|\!)$, which encodes a (finite) set of guarded plans; and general declarative goal-programs $\mathsf{Goal}(\phi_s, P, \phi_f)$. The *full plan language* is thus described by the following grammar:

$$P ::= \mathit{nil} \mid act \mid ?\phi \mid +b \mid -b \mid !e \mid P_1; P_2 \mid P_1 \rhd P_2 \mid P_1 \| P_2 \mid \mathsf{Goal}(\phi_s, P_1, \phi_f) \mid (\!|\{\psi_1 : P_1, \ldots, \psi_n : P_n\}|\!).$$

The *action description library* $\Lambda$ contains simple STRIPS-style rules of the form $act : \psi_{act} \leftarrow \Phi^-_{act}; \Phi^+_{act}$, one for each action type in the domain. Formula $\psi_{act}$ corresponds to the action's precondition, and $\Phi^+_{act}$ and $\Phi^-_{act}$ stand for the add and delete lists of atoms, respectively. The set *Act* denotes for the set of all domain actions.

The newly introduced *motivation library* $\mathcal{M}$ allows for agents to generate their own goals in a proactive manner. At this stage, the library only accounts for what has been elsewhere called *desires* [32] or *automatic events* [5]: goals that are conditionalised by beliefs. Hence, an agent may adopt a new goal on the basis of recognising a particular world state. In concrete, $\mathcal{M}$ consists of rules of the form:

$$\psi \rightsquigarrow \mathsf{Goal}(\phi_s, !e, \phi_f),$$

meaning that if the agent comes to believe $\psi$, she should *consider adopting* the declarative event goal $\mathsf{Goal}(\phi_s, !e, \phi_f)$. For example, Rao's cleaning robot example [25] could be written as follows:

$$RoomDirty \wedge \neg Busy \rightsquigarrow \mathsf{Goal}(\neg RoomDirty, !clean, HasWork).$$

That is, an idle agent may adopt the goal of getting the room cleaned when this is believed to be dirty.

This concludes the syntax of the language. Let us turn into its semantics, which is based on operational semantics [23]. A *transition relation* $\longrightarrow$ on so-called *configurations* is defined by a set of derivation rules. A transition $C \longrightarrow C'$ specifies that executing configuration $C$ a *single step* yields configuration $C'$. We write $C \longrightarrow$ to state that there exists $C'$ such that $C \longrightarrow C'$, and $\xrightarrow{*}$ to denote the usual reflexive transitive closure of $\longrightarrow$. A labelled transition is written as $C \xrightarrow{\mathtt{t}} C'$, where $\mathtt{t}$ is its label. When no label is stated, all labels apply. A *derivation rule* consists of a, possibly empty, set of premises, which are transitions together with some auxiliary conditions, and a single transition conclusion derivable from these premises (see [23] for more on operational semantics).

As in CANPLAN, two transition systems are used to define the semantics of CANPLAN2 agents. The first transition $\longrightarrow$ defines what it means to execute a single intention and is defined in terms of *basic configurations* of the form $\langle \mathcal{B}, \mathcal{A}, P \rangle$ consisting of the current belief base $\mathcal{B}$ of the agent, the sequence $\mathcal{A}$ of primitive actions executed so far, and the plan-body program $P$ being executed (i.e., the current intention).[4] Within basic transitions we use two type labels, namely $\mathsf{bdi}$ and $\mathsf{plan}$ labels, which stand for basic *execution* and *planning* steps, respectively. The second type of transition $\Longrightarrow$ is defined in terms of the first type and defines what it means to execute a complete agent, rather than just a particular intention. This agent-level transition will be defined in section 4.

Let us next describe the basic level semantics, with the rules grouped in three classes for legibility purposes.

## Event Handling

One of the contributions of the original CAN language was that it gave a detailed operational semantics for the kind of failure handling typical of implemented BDI systems, where if a plan fails, alternative plans for achieving the goal are tried, if possible. This was accomplished by using the construct $P_1 \rhd P_2$ which maintains the set of possible alternative plans to consider in $P_2$, while executing $P_1$. CANPLAN2 retains most of the event handling semantics of CANPLAN, but modifies slightly the approach when the current strategy $P_1$ is blocked and cannot continue.

The main derivation rules for handling events and selecting plans in CANPLAN2 are as follows:[5]

$$\frac{\Delta = \{\psi_i\theta : P_i\theta \mid e' : \psi_i \leftarrow P_i \in \Pi \wedge \theta = \mathsf{mgu}(e, e')\}}{\langle \mathcal{B}, \mathcal{A}, !e \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!) \rangle} \; Ev$$

$$\frac{\psi_i : P_i \in \Delta \quad \mathcal{B} \models \psi_i\theta}{\langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, P_i\theta \rhd (\!|\Delta \setminus \{\psi_i : P_i\}|\!) \rangle} \; Sel$$

$$\frac{P_1 \neq \mathit{nil} \quad \langle \mathcal{B}, \mathcal{A}, P_1 \rangle \not\xrightarrow{\mathsf{bdi}} \quad \langle \mathcal{B}, \mathcal{A}, P_2 \rangle \xrightarrow{\mathsf{bdi}} \langle \mathcal{B}', \mathcal{A}', P_2' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2 \rangle \xrightarrow{\mathsf{bdi}} \langle \mathcal{B}', \mathcal{A}', P_2' \rangle} \; \rhd_f^{\mathsf{bdi}}$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rhd P_2 \rangle} \rhd \qquad \frac{\mathcal{B} \models \phi\theta}{\langle \mathcal{B}, \mathcal{A}, ?\phi \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, \mathit{nil} \rangle} ?$$

$$\frac{a : \psi \leftarrow \Phi^-; \Phi^+ \in \Lambda \quad a\theta = act \quad \mathcal{B} \models \psi\theta}{\langle \mathcal{B}, \mathcal{A}, act \rangle \longrightarrow \langle (\mathcal{B} \setminus \Phi^-\theta) \cup \Phi^+\theta, \mathcal{A} \cdot act, \mathit{nil} \rangle} \; act$$

Rule $Ev$ handles event goals by collecting all *relevant* plans for the event in question. Rule $Sel$ selects one *applicable* plan $P_i$ from the set of (remaining) relevant plans $\Delta$. Program $P \rhd (\!|\Delta|\!)$ states that program $P$ should be tried first, falling back to the remaining alternatives in $\Delta$, if necessary. Such special behaviour is captured with rules $\rhd_f^{\mathsf{bdi}}$ and $\rhd$. Rule $\rhd_f^{\mathsf{bdi}}$, together with rule $Sel$, implements the details of the plan *failure handling*: if the current plan $P_i\theta$ for a goal is *blocked* (e.g., at some point an action's precondition or a test is not met), rules $\rhd_f^{\mathsf{bdi}}$ and $Sel$ may apply in sequence in order to select another *applicable* alternative—if any—for the event in question. Notice that all the rules, except for $\rhd_f^{\mathsf{bdi}}$, can be applied in both $\mathsf{bdi}$ and $\mathsf{plan}$ basic contexts.

The rule $\rhd_f^{\mathsf{bdi}}$ is slightly, but importantly, different from that used in CANPLAN in that the current strategy $P_1$ is dropped *only if* there is some other plan in $P_2$ which can be acted upon. If however there is no alternative applicable plan, it may make sense for the agent to simply "wait," rather than drop its current strategy $P_1$,

[2] In practice, the belief base contains ground belief *atoms* in a first-order language.

[3] Notice that $e$, $\psi$, and $P$ may contain free variables.

[4] Strictly speaking, the plan and action libraries $\Pi$ and $\Lambda$ should also be part of basic configurations. For legibility purposes, we omit them as they are assumed to be static entities. Configurations must also include a variable substitution $\theta$ for keeping track of all bindings done so far during the execution of a plan-body. Again, for legibility, we keep substitutions implicit in places where they need to be carried across multiple rules (e.g., in rule ?). See [16] on how substitutions are propagated across derivation rules for 3APL.

[5] Rules that are new or different from CANPLAN are underlined.

only to discover *after* that there is currently no better option. As we will later discuss and prove in section 5, this simple modification has important ramifications on the kind of commitment and goal persistence that the new language has.

Finally, rule ? deals with tests by checking that the condition follows from the current belief base, whereas rule $act$ handles the case of primitive actions by using the domain action description library $\Lambda$. The remaining basic core rules can be found in Figure 1. Rules $Seq$ and $Seq_t$, and $\|_1, \|_2$ and $\|_t$ handle sequencing and interleaved concurrency of programs in the usual way; whereas $+b$ and $-b$ handle belief update operations.

## Declarative Goals

A central feature of CAN(PLAN), in addition to the built-in failure handling mechanism, is the goal construct $\mathsf{Goal}(\phi_s, P, \phi_f)$, which provides a mechanism for representing both declarative and procedural aspects of goals. Intuitively, a *goal-program* $\mathsf{Goal}(\phi_s, P, \phi_f)$ states that we should achieve the (declarative) goal $\phi_s$ by using the (procedural) program $P$; failing if $\phi_f$ becomes true (e.g., the goal is impossible, not required anymore, etc.). Recall that, because BDI programmers can only write goal-programs of the form $\mathsf{Goal}(\phi_s, !e, \phi_f)$, the Goal construct can be seen as an enhancement of the typical achievement event-goal $!e$.

The execution of a goal-program is expected to be consistent with some desired properties of declarative goals: persistent, possible, and unachieved. For instance, if $P$ is fully executed but $\phi_s$ is still not true, $P$ will be retried (i.e., $P$ is restarted); and if $\phi_s$ becomes true during $P$'s execution, the whole goal succeeds.

The new language modifies the basic level semantics of goal-programs in two ways. First, the goal-program *initialisation* rule $\mathsf{G}_I$ is enhanced to allow the adoption of a goal *only if* the agent does have some relevant plan to eventually handle the goal. This ensures that agents behave as described in [22] and do not adopt goals for which there are no capabilities. Interestingly, the initialisation rule could be further developed to capture other constraints, such as a requirement that a goal be adopted only if it does not conflict with a goal already committed to (see discussion section).

The second modification involves changing the way in which goal-programs are retried. In CAN(PLAN), if the current plan for achieving a goal finishes or is blocked, but the declarative goal's success condition is not true, then the goal is re-instantiated from the beginning (unless the fail condition becomes true). This provides a persistence on declarative goals that normal BDI events lack. Nonetheless, this re-instantiation may sometimes lead to an overcommitment to a particular way of achieving a goal. An active goal could merely be a *subsidiary* subgoal (i.e., a goal that is part of a program) for another higher-level *motivating* goal (i.e., the goal which caused the selection of that program). Thus, an agent should not fanatically overcommit to a subsidiary goal, if alternative means for achieving the motivating goal do exist: it makes no sense to keep instantiating a goal to buy a can of soda to quench my thirst, if the store is out of soda and there is a viable alternative way by buying a bottle of water. To that end, CANPLAN2 modifies the retry rule $\mathsf{G}_R^{\mathsf{bdi}}$ to re-instantiate a goal *only if* restarting it does provide indeed an alternative way of addressing the goal, than that which is currently stuck or has finished without achieving success.

These are the corresponding derivation rules for goal-programs:

$$\frac{\mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, !e \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P \rangle}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, !e, \phi_f) \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', \mathsf{Goal}(\phi_s, P \rhd P, \phi_f) \rangle}\ \mathsf{G}_I$$

$$\frac{\mathcal{B} \models \phi_s}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, P, \phi_f) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil \rangle}\ \mathsf{G}_s$$

$$\frac{\mathcal{B} \models \phi_f}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, P, \phi_f) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, ?false \rangle}\ \mathsf{G}_f$$

$$\frac{P = P_1 \rhd P_2 \quad \mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P_1 \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, P, \phi_f) \rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', \mathsf{Goal}(\phi_s, P' \rhd P_2, \phi_f) \rangle}\ \mathsf{G}_S$$

$$\frac{P = P_1 \rhd P_2 \quad P_1 \neq P_2 \quad \mathcal{B} \not\models \phi_s \vee \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P_1 \rangle \overset{\mathsf{bdi}}{\not\longrightarrow}}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, P, \phi_f) \rangle \overset{\mathsf{bdi}}{\longrightarrow} \langle \mathcal{B}, \mathcal{A}, \mathsf{Goal}(\phi_s, P_2 \rhd P_2, \phi_f) \rangle}\ \mathsf{G}_R^{\mathsf{bdi}}$$

When a goal-program is first encountered during execution, rule $\mathsf{G}_I$ applies: $\mathsf{G}_I$ "initialises" the execution of a goal-program by setting the program in the goal to $P \rhd P$, where the first $P$ becomes the *current strategy* to be executed and the second $P$ is just used to store and carry the *original* program $P$ in case rule $\mathsf{G}_R^{\mathsf{bdi}}$ may eventually apply later on. Because, initially, the procedural part of a goal-program is an event-goal $!e$, it is not hard to see that $P$ will stand for the set of corresponding relevant plans $(\!|\Delta|\!)$ (see derivation rule $Ev$ above), that is, the set of all potential useful strategies for $e$. We say that the agent *adopts* the declarative goal $\mathsf{G}(\phi_s, \phi_f)$ when the derivation rule $\mathsf{G}_I$ is successfully applied.

The second and third rules handle the cases where either the success condition $\phi_s$ or the failure condition $\phi_f$ become true. The fourth rule $\mathsf{G}_S$ is the one responsible for performing a single step on the current strategy of an already initialised goal-program. Notice that the second part in the pair $P_1 \rhd P_2$, the original set of potentially useful strategies for the event-goal, remains constant.

Finally, the revised restart rule $\mathsf{G}_R^{\mathsf{bdi}}$, discussed above, restarts the original program (i.e., $P_2$ in $P_1 \rhd P_2$) whenever the current program $P_1$ is not able to achieve the goal. That is, it re-instates the original set of relevant plans for the original event. Observe that for a goal to be re-instantiated, the current strategy $P_1$ must be blocked and it must be different from the original set of possible strategies $P_2$. Therefore, if upon re-instantiation of the goal, there is no applicable strategy for achieving the goal in question, then the complete goal-program becomes *blocked*. This is important because it provides the possibility, due to rule $\rhd_f^{\mathsf{bdi}}$, for the agent to actually *drop* a (blocked) declarative goal-program (e.g., the goal to buy a can of soda) if there is an alternative way (e.g., buying a bottle of water) of achieving a motivating goal (e.g., quenching my thirst). Note that this mechanism relies on the assumption that whenever the context condition of a plan-rule applies, there are sufficient reasons to believe the goal in question will indeed be achieved. Otherwise, one could argue that rule $\mathsf{G}_R^{\mathsf{bdi}}$ may sometimes result in an overcommitment to a goal when there are always executable plans for the goal, but these never actually manage to realise the goal. Note too that a goal-program being blocked is different from $\phi_f$ being true, which would cause the goal-program to immediately *fail*.

## Local Hierarchical Planning

As expected, CANPLAN2 inherits from CANPLAN the uniform integration of *hierarchical planning* into the BDI architecture. Thus, a language construct Plan is used for (local) offline lookahead planning. Importantly, the HTN planning operator operates on the *same domain knowledge* as the BDI system, but allows lookahead to ensure success, prior to acting. The rules for planning, together with other rules for the language, are shown in figure 1.

Fortunately, CANPLAN2 retains unchanged all the original operational rules for the construct Plan, as well as the special basic transition plan label type. However, it also includes the following additional rule for dropping impossible planning goals:

$$\frac{\langle \mathcal{B}, \mathcal{A}, P \rangle \overset{\mathsf{plan}}{\not\longrightarrow}^{*} \langle \mathcal{B}', \mathcal{A}', nil \rangle}{\langle \mathcal{B}, \mathcal{A}, \mathsf{Plan}(P) \rangle \overset{\mathsf{bdi}}{\longrightarrow} \langle \mathcal{B}, \mathcal{A}, ?false \rangle}\ \mathsf{P}_f$$

In other words, a planning goal may completely *fail* if it has no solution. See that because of the bdi context of rules $\rhd_f^{\mathsf{bdi}}$ and $\mathsf{G}_R^{\mathsf{bdi}}$, failure handling and goal restarting is *not* available during planning; they are features of the BDI execution cycle only.

# 4. AGENT LEVEL EXECUTION

On top of the basic transition semantics, we define what it means to execute an agent. An *agent configuration* $\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle$ contains the set of all current intentions $\Gamma$, rather than simply one intention as in a basic configuration, as well as the newly introduced motivational library $\mathcal{M}$. It also includes an action description library $\Lambda$, a BDI plan library $\Pi$, the agent's belief base $\mathcal{B}$, and the sequence of actions $\mathcal{A}$ executed so far.

The agent-level transition semantics that we provide in this section addresses the remaining issues identified in section 1. In particular, the important aspects of the new semantics at this level are:

- The ability to proactively generate goals on the basis of what is believed about the world.
- Assurance that the agent will notice and react to situations where a goal has succeeded or become impossible.
- An improved ability to model interaction with the environment, including information obtained from sensors.

The semantics provided closely matches Rao and Georgeff's abstract interpreter for intelligent rational agents [26] which, roughly speaking, requires the following three steps: (i) select an intention and execute a step, (ii) incorporate any pending external events, (iii) update the set of goals and intentions.

The agent-level execution semantics in CAN(PLAN) was extremely simplistic and thus did not account for the above rational cycle of behavior.[6] Because of that, almost all rules below for CANPLAN2 are new. Four labels are used to define the agent-level transition $\Longrightarrow$. The main agent-labelled transition makes use of three (auxiliary) agent-level transitions, namely, int, event, and goal agent transitions. Let us now describe these transitions.

## Generating and Updating Goals

The goal-labelled agent transition $C \overset{\mathsf{goal}}{\Longrightarrow} C'$ states that the agent configuration $C$ evolves to configuration $C'$ due to a (declarative) goal update. We will provide three derivation rules of this type.

The first derivation rule accommodates a proactive mechanism for generating new goals, besides the classical purely reactive one. Generally speaking, it accounts for the so-called *automatic events* in JACK [5] and *desires as conditional goals* in [31]. Using the motivational library $\mathcal{M}$ discussed earlier, the following agent-level derivation rule provides this kind of self-motivating behaviour:

$$\frac{\psi \rightsquigarrow \mathsf{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi\theta \quad \mathsf{Goal}(\phi_s, P, \phi_f)\theta \notin \Gamma}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{goal}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{\mathsf{Goal}(\phi_s, !e, \phi_f)\theta\} \rangle} A^1_{\mathsf{goal}}$$

That is, if it is believed that $\psi$ holds, the above rule creates a completely new intention with the corresponding declarative goal as the top-level program to execute, provided such goal is not being already pursed as a top-level program. One could also imagine including other type of motivational attitudes in $\mathcal{M}$ (see discussion section).

Let us now focus on the issue of updating the set of goals currently being pursued—the set of active goals. In the original semantics of CAN, an agent configuration included a goal base $\mathcal{G}$ to keep track of currently *active* declarative goals. This goal base was explicitly updated at each transition step. This is not necessary or even possible in the presence of the new planning construct. In fact, the active goals are already *implicitly* represented in the intention base $\Gamma$ and, therefore, there is no need to carry them explicitly. The following definition extracts this (implicit) goal base.

[6]The reason for this was that the focus was on other aspects, namely, the introduction of goals and the semantics of individual intentions in CAN, and the integration of planning in CANPLAN.

DEFINITION 1 (DECLARATIVE ACTIVE GOALS). *The set of active goals in an intention $P$ is inductively defined as follows:*

$$\mathcal{G}(P) = \begin{cases} \mathcal{G}(P_1) & \text{if } P = P_1; P_2 \mid P_1 \rhd P_2 \mid \mathsf{Plan}(P_1), \\ \mathcal{G}(P_1) \cup \mathcal{G}(P_2) & \text{if } P = P_1 \| P_2, \\ \mathcal{G}(P_1) \cup \{\mathsf{G}(\phi_s, \phi_f)\} & \text{if } P = \mathsf{Goal}(\phi_s, P_1 \rhd P_2, \phi_f), \\ \emptyset & \text{otherwise}. \end{cases}$$

*If $\Gamma$ is a set of intentions, we define $\mathcal{G}(\Gamma) = \bigcup_{P \in \Gamma} \mathcal{G}(P)$.*

So, when $P$ is an active intention in $\Gamma$, $\mathcal{G}(P)$ is the set of declarative goals of the form $\mathsf{G}(\phi_s, \phi_f)$ that the agent has already adopted and is executing within $P$. Recall that for a goal to be active, it must have been previously *adopted* by means of basic derivation rule $\mathsf{G}_I$. Thus, a goal-program mentioned in the second part of a sequence $P_1; P_2$ cannot be active, as $P_2$ has not yet been started.

Next, we define rules $A^2_{\mathsf{goal}}$ and $A^3_{\mathsf{goal}}$ The former is responsible for dropping goals that are already fulfilled; the latter is responsible for dropping impossible or useless goals, i.e., goals whose failure conditions are believed true. We use the notation $C \underset{+\blacktriangle}{\longrightarrow} C'$, where $\longrightarrow$ is a transition relation and $\blacktriangle$ is a set of derivation rules for $\longrightarrow$, to denote that $C \longrightarrow C'$ holds and it involves at least one application of a rule in $\blacktriangle$.

$$\frac{P \in \Gamma \quad \mathsf{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \underset{+\{\mathsf{G}_s\}}{\overset{\mathsf{bdi}}{\longrightarrow}} \langle \mathcal{B}, \mathcal{A}, P' \rangle}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{goal}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle} A^2_{\mathsf{goal}}$$

$$\frac{P \in \Gamma \quad \mathsf{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \underset{+\{\mathsf{G}_f\}}{\overset{\mathsf{bdi}}{\longrightarrow}} \langle \mathcal{B}, \mathcal{A}, P' \rangle}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{goal}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle} A^3_{\mathsf{goal}}$$

## Incorporating External Events

We now centre our attention on how the external environment may affect the agent. The informal treatment of this issue in languages like AGENTSPEAK and CAN(PLAN) makes it difficult to prove any property relative to a particular external environment. In addition, CAN(PLAN) did not deal with *direct* updates of the belief base due to sensor input (only plans could change/update the agent's beliefs).

In our language, we shall distinguish three types of events: (i) $!e$ stands for an external (achievement) event; (ii) $+b$ stands for the sensor input that $b$ is true; and (iii) $-b$ stands for sensor input that $b$ is false. The set *Event* denotes the set of all possible events.

An (external) environment is defined as follows. (Recall that the set $Act$ is the set of all possible domain actions.)

DEFINITION 2 (ENVIRONMENT). *An environment is a total function $\mathcal{E} : Act^* \rightarrow 2^{Event}$ satisfying the following consistency property: for every sequence of action $\mathcal{A} \in Act^*$ and ground belief atom $b$, if $+b \in \mathcal{E}(\mathcal{A})$, then $-b \notin \mathcal{E}(\mathcal{A})$.*

The three rules for handling external events are as follows:[7]

$$\frac{\Gamma' = \{!e : !e \in \mathcal{E}(\mathcal{A})\}}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{event}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \Gamma' \rangle} A^1_{\mathsf{event}}$$

$$\frac{+b \in \mathcal{E}(\mathcal{A}) \quad \mathcal{B} \not\models b}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{event}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B} \cup \{b\}, \mathcal{A}, \Gamma \rangle} A^2_{\mathsf{event}}$$

$$\frac{-b \in \mathcal{E}(\mathcal{A}) \quad \mathcal{B} \models b}{\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\mathsf{event}}{\Longrightarrow} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B} \setminus \{b\}, \mathcal{A}, \Gamma \rangle} A^3_{\mathsf{event}}$$

[7]For simplicity, we keep the environment $\mathcal{E}$ implicit. Technically, though, one could follow [2] and define transitions between pairs of *system configurations* $\langle \mathcal{E}, C \rangle$, where $\mathcal{E}$ is an environment "circumstance" and $C$ is an agent configuration.

$$\frac{}{\langle \mathcal{B},\mathcal{A},(nil \rhd P')\rangle \longrightarrow \langle \mathcal{B},\mathcal{A},nil\rangle}\ \rhd_t \qquad \frac{}{\langle \mathcal{B},\mathcal{A},+b\rangle \longrightarrow \langle \mathcal{B}\cup\{b\},\mathcal{A},nil\rangle}\ +b \qquad \frac{}{\langle \mathcal{B},\mathcal{A},-b\rangle \longrightarrow \langle \mathcal{B}\setminus\{b\},\mathcal{A},nil\rangle}\ -b$$

$$\frac{\langle \mathcal{B},\mathcal{A},P_1\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},(P_1 \rhd P_2)\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',(P' \rhd P_2)\rangle}\ \rhd \qquad \frac{\langle \mathcal{B},\mathcal{A},P\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},(nil\,;P)\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}\ Seq_t \qquad \frac{\langle \mathcal{B},\mathcal{A},P_1\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},(P_1;P_2)\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',(P';P_2)\rangle}\ Seq$$

$$\frac{\langle \mathcal{B},\mathcal{A},P_1\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},(P_1 \parallel P_2)\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',(P' \parallel P_2)\rangle}\ \|_1 \qquad \frac{\langle \mathcal{B},\mathcal{A},P_2\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},(P_1 \parallel P_2)\rangle \longrightarrow \langle \mathcal{B}',\mathcal{A}',(P_1 \parallel P')\rangle}\ \|_2 \qquad \frac{}{\langle \mathcal{B},\mathcal{A},(nil \parallel nil)\rangle \longrightarrow \langle \mathcal{B},\mathcal{A},nil\rangle}\ \|_t$$

$$\frac{}{\langle \mathcal{B},\mathcal{A},\mathsf{Plan}(nil)\rangle \longrightarrow \langle \mathcal{B},\mathcal{A},nil\rangle}\ \mathsf{P}_t \qquad \frac{\langle \mathcal{B},\mathcal{A},P\rangle \xrightarrow{\mathsf{plan}} \langle \mathcal{B}',\mathcal{A}',P'\rangle \quad \langle \mathcal{B}',\mathcal{A}',P'\rangle \xrightarrow{\mathsf{plan}^*} \langle \mathcal{B}'',\mathcal{A}'',nil\rangle}{\langle \mathcal{B},\mathcal{A},\mathsf{Plan}(P)\rangle \xrightarrow{\mathsf{bdi}} \langle \mathcal{B}',\mathcal{A}',\mathsf{Plan}(P')\rangle}\ \mathsf{P} \qquad \frac{\langle \mathcal{B},\mathcal{A},P\rangle \xrightarrow{\mathsf{plan}} \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \mathcal{B},\mathcal{A},\mathsf{Plan}(P)\rangle \xrightarrow{\mathsf{plan}} \langle \mathcal{B}',\mathcal{A}',\mathsf{Plan}(P')\rangle}\ \mathsf{P}_\mathsf{P}$$

$$\Delta = \{Agent^1, Agent^2, A^{1,2}_{\mathsf{int}}, A^{1,2,3}_{\mathsf{event}}, A^{1,2,3}_{\mathsf{goal}}\} \cup \{Ev, Sel, +b, -b, act, ?, Seq, Seq_t, \rhd, \rhd_t, \|_1, \|_2, \|_t, \mathsf{G}_I, \mathsf{G}_s, \mathsf{G}_S, \mathsf{P}, \mathsf{P}_t, \mathsf{P}_f, \mathsf{P}_\mathsf{P}\} \cup \{\rhd^{\mathsf{bdi}}_f, \mathsf{G}^{\mathsf{bdi}}_R\}.$$

**Figure 1: CANPLAN's complete set of rules $\Delta$ is built from the rules described in the text plus the ones shown here.**

## Executing and Finishing Intentions

Three derivation rules are used to characterise what it means to evolve an active intention in the agent. An intention can evolve by either making a legal basic-level step or terminating. The first case is captured with the usual rule $A^1_{\mathsf{int}}$ (called $A_{step}$ in CANPLAN):

$$\frac{P \in \Gamma \quad \langle \mathcal{B},\mathcal{A},P\rangle \xrightarrow{\mathsf{bdi}} \langle \mathcal{B}',\mathcal{A}',P'\rangle}{\langle \Lambda,\Pi,\mathcal{M},\mathcal{B},\mathcal{A},\Gamma\rangle \xrightarrow{\mathsf{int}} \langle \Lambda,\Pi,\mathcal{M},\mathcal{B}',\mathcal{A}',(\Gamma\setminus\{P\})\cup\{P'\}\rangle}\ A^1_{\mathsf{int}}$$

An intention may also evolve by *terminating*, because it has successfully executed fully (i.e., $P = nil$) or it is blocked:

$$\frac{P \in \Gamma \quad \mathcal{G}(P) = \emptyset \quad \langle \mathcal{B},\mathcal{A},P\rangle \not\xrightarrow{\mathsf{bdi}}}{\langle \Lambda,\Pi,\mathcal{M},\mathcal{B},\mathcal{A},\Gamma\rangle \xrightarrow{\mathsf{int}} \langle \Lambda,\Pi,\mathcal{M},\mathcal{B},\mathcal{A},\Gamma\setminus\{P\}\rangle}\ A^2_{\mathsf{int}}$$

Observe that an intention is considered terminating only if it is a *reactive* intention, that is, it is currently not pursuing any declarative goal. There must be good reasons, though, to drop a declarative goal that is being pursued in an intention, besides being blocked. In short, a declarative goal may be abandoned if the goal has been achieved, is deemed impossible, or is not required anymore as a subsidiary goal for a higher-level motivating goal. We will formalise and prove this in the next section.

## Top-Level Agent Execution

We now have all the necessary machinery required to define the main top-level execution of an agent relative to an external environment. When $\longrightarrow$ is a transition relation, we use $C \longrightarrow\!| \ C'$ to compactly denote that $C \xrightarrow{*} C'$ and $C' \not\longrightarrow$. Informally, $\longrightarrow\!|$ evolves a configuration as much as possible w.r.t. transition $\longrightarrow$.

The following two agent-level derivation rules capture the well-known Rao's abstract BDI execution cycle for rational agents [26]:

$$\frac{C \xrightarrow{\mathsf{int}} C' \quad C' \xrightarrow{\mathsf{event}}\!| \ C'' \quad C'' \xrightarrow{\mathsf{goal}}\!| \ C'''}{C \xrightarrow{\mathsf{agent}} C'''}\ Agent^1$$

$$\frac{C \not\xrightarrow{\mathsf{int}} \quad C \xrightarrow{\mathsf{event}}\!| \ C' \quad C' \xrightarrow{\mathsf{goal}}\!| \ C''}{C \xrightarrow{\mathsf{agent}} C''}\ Agent^2$$

Initially, an intention is selected and evolved a single step, if possible. This may lead to the selected intention being executed or terminated. After that, *all* pending events are assimilated, including direct belief updates from sensors. Lastly, currently active goals are updated accordingly, by applying all legal goal-labelled transitions at the end of every cycle. The second agent derivation rule $Agent^2$ accounts for the cases in which no intention can be executed or terminated, but where changes in the world may still arise.

This concludes the specification of the proposed language.

# 5. PROPERTIES OF CANPLAN2

In this section, we prove that the language we defined above, in contrast with its predecessor versions and other similar BDI languages, enjoys some desired properties regarding goals. In concrete, we demonstrate that CANPLAN2 agents have a commitment on goals, and their corresponding intentions, such that the hierarchical structure of goals is respected and the goal base is correctly updated at every execution cycle as suggested in [26].

To begin, let us formally define the meaning of an agent execution relative to an environment.

DEFINITION 3 (BDI EXECUTION). *A BDI execution $E$ of an agent $C_0 = \langle \mathcal{N},\Lambda,\mathcal{M},\Pi,\mathcal{B}_0,\mathcal{A}_0,\Gamma_0\rangle$ relative to an environment $\mathcal{E}$ is a, possibly infinite, sequence of agent configurations $C_0 \cdot C_1 \cdot \ldots$ such that $C_i \xrightarrow{\mathsf{agent}} C_{i+1}$, for every $i \geq 0$. A terminating execution is a finite execution $C_0 \cdot \ldots \cdot C_n$ where $\Gamma_n = \{\}$. An environment-free execution is one where $\mathcal{E}(\mathcal{A}_i) = \emptyset$, for every $\mathcal{A}_i$.*

As argued in section 2, it is generally accepted that a rational agent should not insist on achieved or impossible goals. In the context of our language, we define an agent of that sort as follows.

DEFINITION 4 (SINGLE-MINDED AGENT). *A CANPLAN2 agent $\langle \Lambda,\Pi,\mathcal{M},\mathcal{B},\mathcal{A},\Gamma\rangle$ is single-minded if for every declarative goal $\mathsf{G}(\phi_s,\phi_f) \in \mathcal{G}(\Gamma)$, it is the case that $\mathcal{B} \not\models \phi_s$ and $\mathcal{B} \not\models \phi_f$.*

This definition provides a practical approximation of the notion of single-minded agents by taking advantage of the declarative information in goal-programs. Observe that it is not clear how one could define a similar notion for regular (procedural) BDI event-goals, as in principle, one do not know *why* those are "executed." In contrast with the standard definition of single-mindedness from [24], the above definition is not defined as a temporal property. The following theorem, though, states that the single-minded property is propagated through BDI executions.

THEOREM 1. *Let $C_0$ be a single-minded CANPLAN2 agent, $\mathcal{E}$ be an environment, and $C_0 \cdot \ldots \cdot C_n$ be a BDI execution of $C_0$ relative to $\mathcal{E}$. Then, $C_i$ is a single-minded agent, for $0 \leq i \leq n$.*

PROOF (SKETCH). *Direct from the agent-level derivation rules $Agent^1$, $Agent^2$, $A^1_{\mathsf{goal}}$ and $A^2_{\mathsf{goal}}$.* □

So no matter how an agent evolves relative to the environment, her goal base is correctly updated—she will never desire goals that are currently true or deemed unfeasible. Because of their simplistic agent-level execution semantics, neither CAN nor CANPLAN satisfies the above theorem (e.g., an achieved goal may still be pursed).

Besides a goal being dropped due to its success or failure, there are other reasons why a goal may be abandoned. Because of that,

we shall informally claim that our CANPLAN2 agents have a commitment strategy that we refer to as *flexible single-minded*: a goal may also be dropped because it might be reconsidered as an appropriate instrument for some higher-level motivating goal. We shall now try to characterise those cases. We first provide three technical definitions that will come handy to express our results.

We start by defining what are the three type of goal-intentions that could be actively executing in an agent.

DEFINITION 5 (ACTIVE GOAL). *An active event goal is a program of the form $G = P \rhd (\!|\Delta|\!)$; an active declarative goal is a program of the form $G = \mathsf{Goal}(\phi_s, P \rhd (\!|\Delta|\!), \phi_f)$; and, finally, an active planning goal is a program of the form $G = \mathsf{Plan}(P)$. Furthermore, program $P$ is referred to as the goal's current strategy and the set $\Delta$ as its alternative strategies.*

At any point, a single intention may be working on several goals simultaneously, and these will in turn be organised *hierarchically*: some goals are pursued as (mere) instruments for other higher-level goals. The following definition generalises Definition 1 to account for all three types of goals and their usual hierarchical structure.

DEFINITION 6 (ACTIVE GOAL TRACE). *An active goal trace $\lambda$ is a sequence $G_1 \cdot \ldots \cdot G_n$ of active goals. The set $\mathcal{G}^*(P)$ of all active goal traces in $P$ is inductively defined as follows:*

- *If $P = nil \mid act \mid ?\phi \mid +b \mid -b$, then $\mathcal{G}^*(P) = \{\}$.*
- *If $P = P_1 \rhd (\!|\Delta|\!) \mid \mathsf{Plan}(P_1) \mid \mathsf{Goal}(\phi_s, P_1 \rhd (\!|\Delta|\!), \phi_f)$, then $\mathcal{G}^*(P) = \{P \cdot \lambda' : \lambda' \in \mathcal{G}^*(P_1)\}$.*
- *If $P = P_1; P_2$, then $\mathcal{G}^*(P) = \mathcal{G}^*(P_1)$.*
- *If $P = P_1 \| P_2$, then $\mathcal{G}^*(P) = \mathcal{G}^*(P_1) \cup \mathcal{G}^*(P_2)$.*

*The $k$-th element of an active goal sequence $\lambda$, denoted $Nth(\lambda, k)$, is called the k-th (sub)goal in $\lambda$. A goal $G$ is an active goal in $P$ if $Nth(\lambda, n) = G$, for some $\lambda \in \mathcal{G}^*(P)$ and $n > 0$.*

An active goal trace represents a chain of goals and subgoals that are active in the intention—the $(n+1)$-th subgoal is a subsidiary goal for the motivating $n$-th subgoal. We say that the $n$-th subgoal $G$ in an active goal trace $\lambda$ is *part of a planning goal* if there exists an $n'<n$, such that the $n'$-th subgoal in $\lambda$ is an active planning goal.

To achieve a goal, an agent may be performing a particular strategy. However, the agent may eventually resort to alternative courses of action if such strategy cannot be continued further.

DEFINITION 7. *Let $\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle$ be an agent, $P$ be an intention in $\Gamma$, and $G$ be an active goal in $P$. The current strategy for $G$ is blocked, if either:*

- *$G \in \{\mathsf{Goal}(\phi_s, P \rhd (\!|\Delta|\!), \phi_f), P \rhd (\!|\Delta|\!)\}$ and $\langle \mathcal{B}, \mathcal{A}, P \rangle \overset{\text{bdi}}{\not\longrightarrow}$.*
- *$G = \mathsf{Plan}(P)$ and $\langle \mathcal{B}, \mathcal{A}, \mathsf{Plan}(P) \rangle \overset{\text{bdi}}{\not\longrightarrow}$.*

*In addition, we say that $G$ has an alternative applicable strategy if $G \in \{P \rhd (\!|\Delta|\!), \mathsf{Goal}(\phi_s, P \rhd (\!|\Delta|\!), \phi_f)\}$ and $\langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!) \rangle \overset{\text{bdi}}{\longrightarrow}$.*

Basically, an agent may consider dropping the current strategy for a goal and adopting an alternative strategy (by means of basic rule $\rhd_f^{\text{bdi}}$) when the former cannot be continued. The following result means that, in such cases, the failure handling mechanism respects the hierarchical structure of goals by preserving as much as possible what has been already performed in the world.

THEOREM 2. *Let $\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle$ be an agent and $P$ be an intention in $\Gamma$. Let $\lambda$ be an active goal trace in $P$, i.e., $\lambda \in \mathcal{G}^*(P)$, and let $G_k$ be the $k$-th subgoal in $\lambda$ such that its current strategy is blocked. Then, for every $k'$-th subgoal $G_{k'}$ in $\lambda$ such that $k' > k$, it is the case that one of the following cases applies:*

1. *the current strategy for $G_{k'}$ is blocked and $G_{k'}$ does not have an alternative strategy; or*
2. *$G_{k'}$ is part of a $k''$-th level planning goal in $\lambda$, with $k \leq k''$, whose current strategy is blocked.*

PROOF (SKETCH). *Suppose $G_{k'}$ is not part of a blocked planning goal. If $G_{k'}$'s current strategy is not blocked or $G_{k'}$'s current strategy is blocked but $G_{k'}$ has an alternative applicable strategy, then $G_k$'s current strategy cannot be blocked as it can be progressed one step by evolving its subgoal related to $G_{k'}$. In the latter case, it would do so by means of failure rule $\rhd_f^{\text{bdi}}$.* □

Three important points should be mentioned. First, the above theorem points out that the alternative strategies for goal $G_k$, if any, may be considered *only if* no alternative ways can be found for *all* the subgoals that are instrumental to $G_k$. Second, an active goal that is instrumental to a (higher) planning goal may be dropped if the whole planning goal cannot be resolved, that is, if the planning goal is blocked. Third, unlike in CAN and CANPLAN, an active declarative goal-program may indeed be blocked, and as a consequence, instrumental goal-programs could be abandoned for the sake of an alternative strategy within the hierarchy of active goals.

Finally, let us focus once again on *declarative* goals. The question is: what are the reasons why a CANPLAN2 agent may consider dropping a declarative goal? Theorem 1 suggests that a declarative goal is dropped because it has just been achieved or deemed unachievable. There could be other reasons, though, why a goal may be abandoned. A goal, for instance, is dropped if it is a subsidiary goal and a higher-level motivating goal is achieved or considered unachievable. A subsidiary goal can also be dropped because the agent does not envision any current way to act upon it, but an alternative strategy is found for a higher-level motivating goal.

The next theorem formally captures all the reasons why an agent would drop a declarative goal.

THEOREM 3. *Let $C$ and $C'$ be two agent configurations such that $C \overset{\text{agent}}{\Longrightarrow} C'$ and such that $\mathsf{G}(\phi_s, \phi_f) \in \mathcal{G}(\Gamma)$, but $\mathsf{G}(\phi_s, \phi_f) \notin \mathcal{G}(\Gamma')$. Then, one of the following cases must apply:*

1. *$\mathcal{B}' \models \phi_s$, i.e., the goal has been achieved;*
2. *$\mathcal{B}' \models \phi_f$, i.e., the goal is believed to be impossible; or*
3. *if $\lambda$ is an active goal trace in an intention in $\Gamma$ and $G_k = \mathsf{Goal}(\phi_s, P, \phi_f)$ is the $k$-th subgoal in $\lambda$, then there is a $k'$-th subgoal $G_{k'}$ in $\lambda$, where $k' < k$, such that either:*
   - *(a) $G_{k'} = \mathsf{Goal}(\phi'_s, P', \phi'_f)$ and $\mathcal{B}' \models \phi'_s \vee \phi'_f$;*
   - *(b) $\langle \mathcal{B}, \mathcal{A}, G_k \rangle \not\longrightarrow$, but $G_{k'}$ has an alternative applicable strategy, and all $k''$-th subgoals in $\lambda$, where $k' < k'' \leq k$, have their current strategy blocked and no alternative applicable strategy; or*
   - *(c) $G_{k'} = \mathsf{Plan}(P_1)$ and $\langle \mathcal{B}, \mathcal{A}, \mathsf{Plan}(P_1) \rangle \overset{\text{bdi}}{\not\longrightarrow}$.*

In words, a goal may be completely terminated in an agent transition if it has been achieved or aborted (cases 1 and 2), or the goal is not necessary or convenient anymore as an instrumental subgoal for some higher-level goal (case 3). The third case is the most involved one. Case $(a)$ states that a higher-level declarative goal $G_{k'}$ has just been achieved or failed and, thus, fully terminated, together with *all* its subgoals. Case $(b)$ states that the strategy to achieve the goal is indeed blocked (i.e., no transition is possible), but that there is an alternative way of addressing a higher-level goal for which the (sub)goal in question was just an instrument. Lastly, case $(c)$ accounts for the case where the goal is a subsidiary goal for a higher-level planning goal for which there is no solution.

To recap: we presented three technical results that demonstrate that the BDI-style language that we have developed provides a commitment strategy on goals and intentions that is more accurate of rational agents. CANPLAN2 agents would not pursue goals that are achieved or deemed unachievable (Theorem 1), and would always respect the hierarchical structure of active goals, in which some subgoals are mere instruments for achieving higher level goals (Theorem 2). Finally, we identified all the reasons why a declarative goal may be abandoned by an agent (Theorem 3). Roughly speaking, the "flexible single-minded" type of commitment that we have claimed for our agents lays between the simple single-minded and the sophisticated open-minded commitment strategies.

# 6. DISCUSSION

The framework presented here has a number of limitations: goals are adopted without checking for negative or positive interactions with current active goals; goals and motivations are restricted to achievement goals only; no account for "suspended" goals or intentions is provided; and planning goals are not repaired upon failure.

The new language provides support for further development on reasoning about goals, such as reasoning about conflicts or synergies among current goals within different intentions ([30]). For example, one could extend basic configurations to include the current agent's goal base, $\mathcal{G} = \mathcal{G}(\Gamma)$, and extend the goal adoption rule given in section 3 as follows:

$$\frac{P \neq P_1 \rhd P_2 \quad \mathcal{B} \not\models \phi_s \vee \phi_f \quad \nexists g \in \mathcal{G} : Conflict(g, \mathsf{G}(\phi_s, \phi_f))}{\langle \mathcal{B}, \mathcal{G}, \mathcal{A}, \mathsf{Goal}(\phi_s, P, \phi_f) \rangle \longrightarrow \langle \mathcal{B}, \mathcal{G}, \mathcal{A}, \mathsf{Goal}(\phi_s, P \rhd P, \phi_f) \rangle} \; \mathsf{G}_I$$

where relation $Conflict(g_1, g_2)$ characterises the conditions under which two declarative goals are in conflict (e.g., goal $g_1$ tautologically implies $\neg g_2$). The goal in question will then be adopted provided it does not conflict with current active goals. Similarly, it would be interesting to envision ways for avoiding adopting a goal if this is *already implied* by some other active goal-intention.

Rational agents may adopt goals of various sorts [8] and for various reasons, besides the ones dealt with here. For example, agent communication [20], social norms and obligations [4] are also common sources of motivations for agents. Our language does not currently support the temporary *suspension* of an intention. Test goals of the form $?\phi$ are bound to fail when false. Sometimes, however, an agent should just "suspend" the intention and wait for the test to become true (e.g., when waiting for some *expected* change in the environment). Finally, unlike in systems such as RAP [11], there is no explicit account of plan monitoring and recovery. The work in [14] on plan failure and abortion should be orthogonal to CANPLAN2. Further study of all the above issues is required.

Goals are an integral and core aspect of agents. We believe it is necessary to accommodate sophisticated accounts of goals that go beyond simple procedural reactions to the environment, as well as to study their role within these languages formally. The language CANPLAN2 presented in this paper is a step towards these objectives, by incrementally building on previous work.

# 7. REFERENCES


[1] R. H. Bordini and J. F. Hübner. BDI agent programming in AgentSpeak using Jason. In *Proc. of CLIMA*, pages 143–164, 2006.

[2] R. H. Bordini and Á. F. Moreira. Proving BDI properties of agent-oriented programming languages. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):197–226, 2004.

[3] M. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[4] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. der van Torre. The BOID architecture: Conflicts between beliefs, obligations, intentions and desires. In *Proc. of AGENTS*, pages 9–16, 2001.

[5] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents: Components for intelligent agents in Java. AgentLink News Letter, Jan 1999. AOS Pty. Ltd.

[6] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proc. of AAAI*, pages 495–502, 1999.

[7] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[8] M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. Goal types in agent programming. In *Proc. of AAMAS*, pages 1285–1287, 2006.

[9] D. Dennett. *The Intentional Stance*. MIT Press, 1987.

[10] J. Dix, H. Muñoz-Avila, D. S. Nau, and L. Zhang. IMPACTing SHOP: Putting an ai planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, 2003.

[11] J. R. Firby. An investigation into reactive planning in complex domains. In *Proc. of AAAI*, pages 202–206, 1987.

[12] M. Georgeff and F. Ingrand. Decision Making in an Embedded Reasoning System. In *Proc. of IJCAI*, pages 972–978, 1989.

[13] M. Georgeff and A. Rao. The semantics of intention maintenance for rational agents. In *Proc. of IJCAI*, pages 704–710, 1995.

[14] J. Thangarajah, D. Morley, N. Yorke-Smith, and J. Harland. Aborting tasks and plans in BDI agents. In *Proc. of AAMAS*, 2007. To appear.

[15] K. Hindriks, F. de Boer, W. van der Hoek, and J. Meyer. Agent programming with declarative goals. In *Proc. of ATAL*, 2001.

[16] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[17] M. J. Huber. JAM: a BDI-theoretic mobile agent architecture. In *Proc. of AGENTS*, pages 236–243, New York, NY, USA, 1999.

[18] J. Hubner, R. Bordini, and M. Wooldridge. Programming declarative goals using plan patterns. In *Proc. of DALT*, pages 65–81, 2006.

[19] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert: Intelligent Systems and Their Applications*, 7(6):34–44, 1992.

[20] Á. Moreira, R. Vieira, and R.Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication.In *Proc. of DALT*,2003.

[21] T. J. Norman and D. Long. Goal creation in motivated agents. In *Proc. of ATAL*, pages 277–290, 1994.

[22] L. Padgham and P. Lambrix. Formalisations of capabilities for bdi-agents. *Autonomous Agents and Multi-Agent Systems*, 10(3):249–271, May 2005.

[23] G. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Dept. of Computer Science, Aarhus University, Denmark, 1981.

[24] A. Rao and M. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. of KR*, pages 473–484, 1991.

[25] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*, volume 1038 of *LNAI*, pages 42–55. 1996.

[26] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In *Proc. of KR*, pages 438–449, 1992.

[27] S. Sardina, L. P. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proc. of AAMAS*, pages 1001–1008, 2006.

[28] S. Sardina and S. Shapiro. Rational action in agent programs with prioritized goals. In *Proc. of AAMAS*, pages 417–424, 2003.

[29] S. Shapiro, Y. Lespérance, and H. J. Levesque. Goal change. In *Proc. of IJCAI*, pages 582–588, 2005.

[30] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & Exploiting Positive Goal Interaction in Intelligent Agents. In *Proc. of AAMAS*, pages 401–408, 2003.

[31] B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. C. Meyer. Dynamics of declarative goals in agent programming. In *Proc. of DALT*, LNCS, pages 1–18. 2005.

[32] M. B. van Riemsdijk, M. Dastani, and J.-J. C. Meyer. Semantics of Declarative Goals in Agent Programming. In *Proc. of AAMAS*, 2005.

[33] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & Procedural Goals in Intelligent Agent Systems. In *Proc. of KR*, pages 470–481, 2002.