Under consideration for publication in Formal Aspects of Computing

Refining autonomous agents with declarative beliefs and desires

Qin Li[1] and Graeme Smith[2]

[1]Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China.
[2]School of Information Technology and Electrical Engineering, The University of Queensland, Australia

Abstract. An autonomous agent is one that is not only directed by its environment, but is also driven by internal motivation to achieve certain goals based on beliefs about the environmental behaviour. Design paradigms for autonomous agents such as BDI (Belief-Desire-Intention) take into account the agent's "mental" features when presenting its patterns of behaviour. In this paper we present an approach to modelling autonomous agents by introducing mental features to conventional transition system specifications. Mental features such as belief and desire are represented by declarative Linear Temporal Logic (LTL) formulas. Refinement is then proposed to define the correctness of the agent design and development. It turns out, however, that the introduction of these mental features is not monotonic with respect to refinement. We therefore introduce additional refinement proof obligations to enable the use of simulation rules when checking refinement.

Keywords: Refinement; Autonomous agents; Belief and desire; Object-Z; Temporal logic

1. Introduction

The design of autonomous agents is one of the central issues of the artificial intelligence community [Woo09]. An agent has the capability to manage its own resources and sense its environment. The behaviour of an agent is often determined dynamically based on its current perception of itself and the environment as well as a goal to achieve. This is the main difference between agents and conventional components.

Agents with the same capability may have different goals and be deployed in different environments. Therefore, the design of such agents must not depend on following specific goals or be based on particular assumptions about the environment. In order to support such flexibility, an autonomous agent is usually described in terms of not only its "physical" features such as state variables and actions, but also its "mental" features such as *beliefs* and *desires*.

An autonomous agent in the BDI (Belief-Desire-Intention) paradigm [RG95, WJ95] formulates a plan (its intention) based on its current beliefs about itself and its environment in order to achieve its desire. Its behaviour, therefore, is derived not only from what it is able to do, but also from what it "wants" to

Correspondence and offprint requests to: Qin Li e-mail: qli@sei.ecnu.edu.cn

do [Woo09]. In this sense, the mental features, belief and desire, regulate the behaviour of the agents by restricting their choice of available actions; the resulting action sequences form the intention of the agents.

In the multi-agent system community, there are various formalism for the high-level description of BDI agents using, for example, modal logics [Mey14, MBH15, Wob15]. Based on these, a number of verification approaches, mostly using model checking, have been developed for reasoning about agent decision making capabilities [BFVW06, FBO10, Fis11, RL07]. Operational semantics are also studied for agent programming languages [MB02, HMTYS14]. There is little work, however, on stepwise refinement of agents; in particular, on approaches which allow an agent to be refined by modifying its beliefs and desires. To employ established refinement techniques, such an approach requires a formal link between the "mental" and "physical" features of an agent.

In [LS13], we propose an extension of Object-Z [Smi00] supporting the specification and stepwise refinement of agents driven by *desires*. To an Object-Z class, capturing the state and actions of an agent, we add a desire — a sequence of goals each specified in terms of linear temporal logic (LTL) [Eme90]. Typical goals include getting a task done in the future, which can be specified with the *eventually* temporal operator $\Diamond$, or maximising a reward at each step, which can be specified with the *always* temporal operator $\Box$. A goal provides internal motivation for an agent to choose one action over another. An agent is called *rational* if every decision it makes benefits the achievement of its current goal.

At any moment of time, a BDI agent has a single goal which it chooses based on its beliefs and desire. By modelling the desire of an agent as a sequence of goals in [LS13], we assume that the specifier knows all goals the agent will ever follow, even those dynamically generated by the agent at runtime. The goals appear in the sequence in order of priority. The agent begins with the first goal in its sequence and, when that goal is no longer achievable, moves to the next goal. The realisation that a goal is not achievable arises from the agent's perception of itself and its environment, the latter derived from input variables of the agent's actions. When the final goal is reached and it is also not achievable, the agent acts in an arbitrary fashion within what is allowed by its specified actions. We call this *unmotivated behaviour*.

The approach of [LS13] does not allow an agent to return to earlier goals which, due to changes in the environment, may have become achievable again. For this reason, it does not reflect the many agent designs where *all* possible goals are evaluated when a change of goal is required. In this paper, we adapt the approach of [LS13] to have a more general notion of desire-driven behaviour which does take into account all goals at each decision point. The agent always chooses the *first* achievable goal in the sequence, *i.e.*, the achievable goal with the highest priority.

We also introduce *beliefs* to the approach. A belief of an agent is also specified as an LTL formula, reflecting the agent's perception of the environment. It captures the agent's knowledge about the current situation of the environment and how the agent believes the environment will interact with it in the future. The belief of an agent with no knowledge of the environment is captured by the LTL formula *true*.

As with desires, the choices made by an agent are affected by its current belief. The agent does not take the risk of performing a sequence of actions which relies on specific environmental inputs inconsistent with its belief. Note that the belief does not necessarily match the actual environmental behaviour but is only the agent's perspective. The agent updates its belief during its execution whenever the environment provides an input which is inconsistent with the current belief.

In our framework, the agent's physical capability is firstly specified with an action system specification using Object-Z notation. It specifies all actions the agent can perform when the environment is cooperative. The mental features, belief and desire, restrict an agent's decision making process which reduces the nondeterminism of its behaviour. The agent always chooses an action leading to its goal according to its belief about the environment interactions. In this sense, the introduction of mental features into the agent specification results in a refinement of its behaviour. The refinement theory we provide is able to justify the correctness of design and development paradigms for adapting to environments by:

(a) Refining the local actions of the agent to increase the feasibility of the desire under different environmental conditions.

(b) Improving the belief-updating mechanism to guide the agent in making more reliable decisions based on more precise beliefs.

(c) Introducing alternative goals to the desire to reduce unmotivated behaviour.

To provide flexible support for (a) we allow the introduction of both state variables and actions in the

concrete specification. The refinement theory and its simulation rules are based on event refinement in Event-B [Abr10], which in turn is based on refinement in action systems [Bac90].

To support (b), an order for the belief-updating functions is defined reflecting which updating mechanism is more reasonable for the agent.

To support (c), we allow unmotivated behaviour to be restricted by alternative goals.

However, as we show, restricting agent behaviour with beliefs and desires is not monotonic with respect to refinement. In order to refine autonomous agents, therefore, we provide additional refinement obligations. A refinement of an autonomous agent can be verified by checking both the standard simulation rules and the new proof obligations.

The remainder of the paper is organised as follows. We introduce the specification notation for agents and its semantics in Section 2; the refinement relation and simulation rules are introduced in Section 2.1. Section 3 then introduces the mental feature *desire* into an agent specification to refine the behaviour of the agent according to internal motivation. The non-monotonicity of the behaviour restriction is revealed, and the refinement obligation to aid the checking of refinement is proposed. Section 4 enriches the agent specification with the mental feature *belief* which further refines the behaviour of the agents. Again non-monoticity is discussed and a refinement obligation proposed. Section 5 provides an overview of related work, and we conclude in Section 6.

2. Agents

In this paper, we focus on the specification and refinement of single agents. Such agents have sole access to their own state variables, and interact with their environment through inputs and outputs. The environment may comprise other agents, and cooperation with these other agents may be necessary to achieve an agent's goal.

In many formalisms for BDI agents [Mey14, MBH15, Wob15], the mental features of an agent are directly modelled as system states that can change during the execution. The behaviour of an agent is then abstracted as the updating of these mental states according to the interactions with the environment. From those formalisms, it is usually difficult to determine the relations between the mental states and the program states, and therefore difficult to verify whether the design of the agent is correct with respect to a behavioural specification over program states.

Unlike those approaches, our framework considers the effect of the mental features of agents as a positive restriction of the "unmotivated" behaviour from a state-transition system perspective. The introduction of mental features does not compromise the conventional formalisation of software systems. From this point of view, we can adopt established formal techniques to analyse and verify behavioural properties of autonomous agents.

Syntactically, we represent the physical features of an agent — its state variables and actions – by a construct, based on the class construct of Object-Z [Smi00], which we will call a *module*.[1] A module includes a state schema declaring the state variables and an invariant constraining their values, an initial state schema and a set of actions modelling state transitions. As in Object-Z, primed variables, *e.g.*, x', denote the value of state variables in the post-state of an action, and actions include a Δ-list of state variables whose values they may change.

Unlike standard Object-Z, an action has both a guard and a precondition. The guard condition is stated explicitly in an action separated from the effect predicate describing the action's behaviour. The explicit guard is an extension to Object-Z aimed at allowing a more flexible notion of refinement similar to that of Event-B [Abr10]. Specifically, an action can be enabled in a state which is not included as a pre-state of the effect predicate; and the result is divergence.

An action has the following form, where y denotes those state variables not in the Δ-list.

[1] Our past work [SW12, SL14] has shown that the object-oriented structuring in Object-Z is ideal for modeling multi-agent systems (MAS): classes allow us to both describe an agent in isolation, and instantiate a large number of agents operating in parallel. We eventually want to incorporate the ideas in this paper into MAS development, and so have based the work on Object-Z.

Action

$\Delta(x)$	variables which action may change
$u? : Type\_of\_u$	input variables
$v! : Type\_of\_v$	output variables
$a : Type\_of\_a$	auxiliary variables
$guard(u?, a, x, y)$	
$effect(u?, a, x, y, x', v!)$	

In the standard semantics of Object-Z, the state variables are hidden (*i.e.*, executions are represented by sequences of actions) and the interaction variables (inputs and outputs) appear as part of the actions which occur. While such a semantics is suited to standard data refinement [DB14], to allow the introduction of actions we use a semantics in which the actions are hidden (*i.e.*, executions are represented by sequences of states), and hence embed the interaction variables in the states.

The interaction variables are implicitly added to the Δ-list of every action. Any reference in an action to an input variable is a reference to its pre-state value. Hence, actions cannot refer to or constrain their post-state values which represent the values of the inputs used by the next action. Any reference to an output variable is a reference to its post-state value. In the case that an action does not generate a value for a given output variable $v!$ then $v!$ is implicity assigned the special null (undefined) value ϵ.

Our semantics and definition of refinement of modules is based, like Event-B, on that of action systems [Bac90]. Semantically, a module is a tuple $M = (\Sigma, I, \mathcal{A})$ where

- Σ is the set of states of the module. Each state is a function mapping the state variables and interaction variables to values which satisfy the variables' types.

  $$\Sigma \ \widehat{=}\ ((Var \cup In) \rightarrow Val) \cup (Out \rightarrow (Val \cup \{\epsilon\}))$$

  where Var is the set of state variables declared in the module, In is the set of input variables appearing in any action of the module, and Out is the set of output variables appearing in any action of the module. Val is the set of all values and ϵ is the null value for output variables.
- $I \subseteq \Sigma$ is the set of states which satisfy the module's initial condition. The initial value of each output variable is ϵ.

  $$I \ \widehat{=}\ \{\sigma \mid \sigma \in \Sigma \land \sigma \models (init(Var) \land inv(Var)) \land \forall\, v! \in Out \bullet \sigma(v!) = \epsilon\}$$

  where $init(Var)$ is the initialisation condition and $inv(Var)$ is the invariant over the state variables. The input variables are not constrained initially. The values chosen represent the values of the inputs used by the first action to occur.
- $\mathcal{A} \subseteq \Sigma \times \Sigma^{\perp}$ is the transition relation specified by the actions, where $\Sigma^{\perp} = \Sigma \cup \{\perp\}$ and $\perp$ denotes a divergent state in which the values of the state variables are undefined. Divergence occurs when the current state enables an action but the effect of executing the action is undefined. Divergent behaviour is modelled as maximally nondeterministic behaviour allowing it to be refined by any other behaviour. Hence, divergence can be used to abstract the details of behaviour of interest only at some lower level of abstraction. When an action results in divergent behaviour, the divergent state $\perp$, as well as any other state in Σ, may result. In this way, divergent behaviour can be distinguished semantically from maximally nondeterministic terminating behaviour. Also, since any normal behaviour will be subsumed by the maximally nondeterministic behaviour associated with divergence, it is not possible to semantically distinguish behaviour which *may* diverge, from that which *must* diverge.
  Formally, an individual action named A is represented semantically as

  $$sem\ A \ \widehat{=}\ \{(\sigma, \sigma') \mid \sigma \in \Sigma \land \sigma' \in \Sigma^{\perp} \land \sigma \models A.guard \land ((\sigma, \sigma') \models E(A) \lor (\nexists \sigma'' \bullet (\sigma, \sigma'') \models E(A)))\}$$

  where $A.guard$ is the guard condition of action A and

  $$E(A) \mathrel{\widehat{=}} A.effect \land inv(Var) \land inv(Var') \land \forall\, v! \in (Out \backslash A.out) \bullet \sigma'(v!) = \epsilon$$

  where $A.effect$ is the effect predicate of action A, and $A.out$ is its set of output variables. Note the final disjunct in the definition of *sem* A corresponds to the case where the action is enabled in a state which is not a pre-state of the effect predicate.

Given that the set of all action names is *Actions*, we have

$$\mathcal{A} = \bigcup_{A \in Actions} sem\ A$$

For simplicity, we omit the notation *sem* in the rest of the paper.

The behaviour of an agent is the set of all possible traces of the associated module, *i.e.*, infinite sequences of states $\langle \sigma_1, \sigma_2, \ldots \rangle$ where every state is a member of $\Sigma^\perp$, $\sigma_1 \in I$, and for all $i \in \mathbb{N}_1$, (σ_i, σ_{i+1}) corresponds to the execution of an action A, or to agent inactivity. By allowing unlimited agent inactivity, we model the fact that an unmotivated autonomous agent can always choose to do nothing. This is not the case when the agent is motivated by a desire.

The set of traces of a module form a tree structure where branches correspond to nondeterministic choices by the agent and its environment: the agent controls the state variables and output variables by nondeterministically choosing one of the enabled actions, and the environment controls the input variables. Formally, the set of traces of a module is defined below, where for any $i \in \mathbb{N}_1$, $s[i]$ denotes the ith state in trace s.

Definition 2.1. (Module Traces) For a trace s and a set $m \subseteq \mathbb{N}_1$, let $non\_div(s, m)$ be true iff s does not diverge at indices in m, *i.e.*,

$$non\_div(s, m) \mathrel{\widehat{=}} \forall\, i \in m \bullet (s[i], s[i+1]) \in \mathcal{A} \cup Skip \wedge (s[i], \perp) \notin \mathcal{A}$$

where $Skip \mathrel{\widehat{=}} \{(\sigma, \sigma') \mid \forall\, x \in Var \bullet \sigma'(x) = \sigma(x) \wedge \forall\, v! \in Out \bullet \sigma'(v!) = \epsilon\}$

The behaviour described by any module M is modelled as a set of traces, divided into the following subsets distinguished by divergence.

$nml(M)$ denotes the set of all normal, *i.e.*, non-divergent, traces of M.

$$nml(M) \ \mathrel{\widehat{=}}\ \{s \mid s[1] \in I \wedge non\_div(s, \mathbb{N}_1)\}$$

For $n \in \mathbb{N}_1$, $div(M, n)$ denotes the set of traces of M that diverge after the nth state.

$$div(M, n) \ \mathrel{\widehat{=}}\ \{s \mid s[1] \in I \wedge non\_div(s, \{1 \,..\, n-1\}) \wedge (s[n], \perp) \in \mathcal{A}\}$$

Note that it is not possible to recover from divergence; the behaviour following the nth state is undefined and hence maximally nondeterministic (all behaviours including those with the divergent state $\perp$ are included).

$tr(M)$ denotes the set of all possible execution traces of M.

$$tr(M) \ \mathrel{\widehat{=}}\ nml(M) \cup (\bigcup_{n \geq 1} div(M, n))$$ □

Example 2.1.

Consider an agent driving a car. The agent receives information about the local traffic via an on-board navigation device. This information includes the time required to reach the destination on the current route, taking into account traffic congestion, and the time on any alternative routes which are faster than the current route. The agent can change the route it takes based on this information.

Let *Route* be a given type denoting the set of all routes leading to a destination. We use R_i $(i \in \mathbb{N})$ to denote a route in *Route*. The agent's state has two variables $current : Route$ denoting the current route and $time : Route \nrightarrow \mathbb{N}$ denoting the travel times of the current route and all alternative routes which the navigation device has suggested. For simplicity, we abstract away from the location of the car. We assume that the car can always change to another route if it decides to. Initially, the current route is the only available route. The action *Update* models the agent receiving route information from the navigator, and the agent choosing a route based on this information.

Agent

$current : Route$
$time : Route \pfun \mathbb{N}$

Init

$\mathrm{dom}\, time = \{current\}$

Update

$\Delta(current, time)$
$time? : Route \pfun \mathbb{N}$

$true$

$current \in \mathrm{dom}\, time?$
$\forall\, r : \mathrm{dom}\, time? \setminus \{current\} \bullet time?(r) < time?(current)$
$time' = time?$
$current' \in \mathrm{dom}\, time'$

The input *time*? in the action *Update* models a communication from the navigation device. The communication includes updated times for the current route and any faster routes. Only faster routes are suggested since it is assumed the agent will not change its current route for a slower one.

Let $R1, R2, R3 \in Route$. A normal trace of the agent is:

$\langle (current = R1, time = \{R1 \mapsto 50\}, time? = \{R1 \mapsto 40, R2 \mapsto 35\}),$
$(current = R1, time = \{R1 \mapsto 40, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R3 \mapsto 20\}),$
$(current = R3, time = \{R1 \mapsto 35, R3 \mapsto 20\}, time? = \{R3 \mapsto 20\}), \ldots \rangle$

A divergent trace of the agent is:

$\langle (current = R1, time = \{R1 \mapsto 50\}, time? = \{R1 \mapsto 40, R2 \mapsto 35\}),$
$(current = R1, time = \{R1 \mapsto 40, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R2 \mapsto 40\}),$
$\bot, \ldots \rangle$

In this case, the divergence is caused by the navigator providing an input violating the precondition $\forall\, r : \mathrm{dom}\, time? \setminus \{current\} \bullet time?(r) < time?(current)$. □

2.1. Refinement and simulation rules

A refinement of an agent specification guarantees that the changes of the state variables in the concrete specification are consistent with those in the abstract specification with respect to a *retrieve relation* R. Given two modules M_1 and M_2, a retrieve relation $R : \mathbb{P}(\Sigma_1^{\bot} \times \Sigma_2^{\bot})$ defines a correspondence between their states. Note that R maps $\bot$ only to itself, and no other state is mapped to $\bot$, *i.e.*, $R(\bot) = \{\bot\}$ and $R^{-1}(\bot) = \{\bot\}$. As well as allowing R to be applied as a function on sets of states (note that a single state argument is interpreted as the singleton set containing that state), we allow it to be applied as a function on traces, sets of traces and formulas. The results of the application of R to these constructs are based on its application to states. For instance, the application of R to a trace can be defined as:

$R(s) = \{t \mid \forall\, i \in \mathbb{N}_1 \bullet t[i] \in R(s[i])\}$

The application of R to a set of traces S can be defined based on its application to traces:

$R(S) \mathrel{\widehat{=}} \bigcup_{s \in S} R(s)$

The application of R to a formula P is also defined in terms of its application to traces:

$R(P) = \{Q \mid \forall\, s, t \bullet t \in R(s) \Rightarrow (s \models P \Leftrightarrow t \models Q)\}$

Definition 2.2. (Refinement) Let M_1 and M_2 be two modules. We say M_1 is refined by M_2 with respect to retrieve relation R, denoted $M_2 \sqsupseteq_R M_1$, iff $tr(M_2) \subseteq R(tr(M_1))$.

The subscript R in $\sqsupseteq_R$ may be omitted if R is the identity relation. □

Internal changes in the concrete specification may be hidden by the retrieve relation, making some of the concrete actions appear like inactivity at the abstract level. Such concrete actions are called 'stuttering actions'. Formally, action A defined in a concrete module is called a *stuttering action* if it behaves as inactivity in the abstract view, *i.e.*, R; $A \subseteq Skip$; R where ; is the relational composition operator.[2] Any concrete action not having that property is called a *change action*.

To prove refinement via Definition 2.2 is generally intractable, requiring analysis of all traces of the modules, and so as usual we consider simulations. The following simulation rules are inspired by those of Event-B [Abr10] which allow a single abstract state transition to be refined by a sequence of concrete transitions.

Theorem 2.1. (Forward Simulation) Let M_1 and M_2 be modules and R be a retrieve relation between their states. Then $M_2 \sqsupseteq_R M_1$ if

(1) $I_2 \subseteq R(I_1)$

(2) for any change action A_2 of M_2, there exists an action A_1 of M_1 where $R\,;\, A_2 \subseteq A_1\,;\, R$.

The proof is straightforward noting that any concrete trace which has stuttering states is related to an abstract trace with inactivity, *i.e.*, *Skip*, in the corresponding positions in the trace. □

Condition (2) allows the guard of a concrete change action to be stronger than that of the corresponding abstract action. This can result in the introduction of deadlock, *i.e.*, where no actions are enabled. While an agent can choose to be inactive, we would not usually want to refine an agent to one which is always inactive in certain circumstances. Hence, as in Event-B, we propose an additional condition to prevent the introduction of deadlock.

(3) The overall guard of M_2 is weaker than that of M_1, *i.e.*,

$$R(\{\sigma \mid \sigma \models g(M_1)\}) \;\subseteq\; \{\gamma \mid \gamma \models g(M_2)\}$$

where the overall guard $g(M)$ is the disjunction of all action guards declared in module M.

3. Autonomous Agents Driven by Desire

A module can specify the behaviour of an agent by referring to the variables it controls and the actions it can perform. In order to specify the autonomous behaviour of an agent, we need to specify its mental features which affect its behaviour. The advantage of separating the agent's physical capability from its mental features is that we can obtain required behaviour by solely modifying the mental features without changing its physical capability or vice versa.

In this section we first introduce the mental feature desire into the specification. It gives an internal motivation for an agent's execution: to fulfil its desire. The mental feature belief will be introduced in Section 4.

A desire of an agent can be represented as a sequence of goals reflecting the agent's motivations with priority. At any moment when the agent needs a new goal, it will begin the evaluation from the first goal of that sequence until it finds an achievable goal and sets it as the current goal. The goal will be held until it is achieved or the agent realises that it is unachievable due to the environment. The desire is a declarative specification of the agent's motivation throughout its execution.

A goal Φ is represented by a linear temporal logic (LTL) [Eme90] formula with the syntax below.

$$\Phi ::= \; \varphi \;\mid\; \Box\Phi \;\mid\; \Diamond\Phi \;\mid\; \bigcirc \Phi \;\mid\; \Phi_1 \mathbf{U} \Phi_2 \;\mid\; \Phi_1 \odot \Phi_2$$

where φ is a first order predicate over states, and the binary operator $\odot \in \{\wedge, \vee, \Rightarrow\}$ (from which other logical operators can be derived). The meanings of the temporal operators $\Box$, $\Diamond$, $\bigcirc$ and $\mathbf{U}$ are defined below.

Definition 3.1. (Satisfaction) Let Δ be the set of all sequences of states which do not include $\perp$. Let $\delta \in \Delta$ and Φ be an LTL formula. For any $i \in \mathbb{N}_1$, we define

[2] A; B is the relational composition of relations A and B, *i.e.*, $A; B \mathrel{\widehat{=}} \{(a, b) \mid \exists\, c \bullet (a, c) \in A \wedge (c, b) \in B\}$.

- $(\delta, i) \vdash \varphi$ iff $\varphi(\delta[i])$
- $(\delta, i) \vdash \bigcirc\Phi$ iff $(\delta, i+1) \vdash \Phi$
- $(\delta, i) \vdash \Box\Phi$ iff $\forall j \geq i \bullet (\delta, j) \vdash \Phi$
- $(\delta, i) \vdash \Diamond\Phi$ iff $\exists j \geq i \bullet (\delta, j) \vdash \Phi$
- $(\delta, i) \vdash \Phi_1 \mathbf{U} \Phi_2$ iff $\exists j \geq i \bullet (\delta, j) \vdash \Phi_2 \wedge (\forall k \in \{i, .., j-1\} \bullet (\delta, k) \vdash \Phi_1)$
- $(\delta, i) \vdash \Phi_1 \odot \Phi_2$ iff $((\delta, i) \vdash \Phi_1) \odot ((\delta, i) \vdash \Phi_2)$

For a trace s, we define $s \vDash \Phi$ iff $(s, 1) \vdash \Phi$. □

3.1. Goal-driven behaviour

We begin by considering an agent M driven by a single goal Φ. Such an agent is denoted by $M?\Phi$. To achieve the goal, the agent calculates execution paths based on its current beliefs about itself and the environment, and chooses to follow a path leading to the goal. If there is no path for the agent to achieve the goal, the agent has no motivation for the rest of its execution, *i.e.*, it chooses any enabled action.

The traces of $M?\Phi$ include:

- Any non-divergent trace of M which satisfies Φ.
- Any divergent trace of M which has satisfied Φ before divergence. After the desire has been satisfied, the agent may act in any manner available to it.
- Any trace of M in which, from a certain point, the agent has no opportunity to make a decision which will lead to its desire being satisfied while before that point the agent made acceptable choices all along the trace. Such a trace corresponds to unmotivated behaviour.

Note that a decision made by the agent is acceptable when there exists a path afterward which can satisfy the desire given a cooperative environment. That is, we do not insist that the decisions of the agent guarantee that every path afterward can satisfy the desire. This reflects the fact that such decisions would be based on the agent's beliefs about the future behaviour of the environment which may, or may not, turn out to be true. The effect of beliefs will be formalised in Section 4.

To formalise our notion of goal-driven behaviour, we first introduce some notation.

1. For traces s and t, let $s =_n t$ be true iff s and t share the same prefix of length n, *i.e.*,

   $$s =_n t \mathrel{\widehat{=}} \forall i \in 1..n \bullet s[i] = t[i]$$

2. For an autonomous agent $M?\Phi$ and a trace $s \in tr(M)$, we let $\Gamma(s, M, \Phi, i)$ denote a predicate that is true when the goal Φ is unachievable in trace s due to either (a) the value of inputs at point i, or (b) when i is 1, the trace's initial state. Since an agent does not control its initial state or inputs, these situations are ones in which the agent has no opportunity to make a decision which will lead to the goal being satisfied. For case (a), (1) there does not exist a trace $r \in nml(M)$ which shares the prefix of trace s up to point i and achieves the goal Φ, and (2) there exists a trace $u \in nml(M)$ which shares the prefix of s up to point $i-1$ and differs from s at point i by the input values and achieves the goal Φ. This case indicates that the goal is unable to be satisfied due to inputs from the environment.
   For case (b), there does not exist a trace $r \in nml(M)$ which shares the same initial state as s and achieves the goal Φ. This case indicates that the desire is unable to be satisfied due to the initialisation.
   Let $V \lhd \sigma$ denote the state σ restricted to variables in set V.

   $$\begin{aligned}\Gamma(s, M, \Phi, i) \mathrel{\widehat{=}}\ & (\forall r \in nml(M) \bullet r =_i s \Rightarrow r \not\models \Phi) \wedge \\ & (i > 1 \Rightarrow (\exists u \in nml(M) \bullet u =_{i-1} s \wedge u[i] \neq s[i] \wedge \\ & \qquad (Var \cup Out) \lhd u[i] = (Var \cup Out) \lhd s[i] \wedge u \models \Phi))\end{aligned}$$

   Such a trace s contains unmotivated behaviour after point i. It is kept in the autonomous behaviour for further refinement.

Definition 3.2. (Goal driven behaviour) Let M be a module and Φ be a goal. The goal driven behaviour of an autonomous agent $M?\Phi$ is modeled as a set of traces, divided into the following subsets distinguished by divergence.

The normal traces of $M?\Phi$ are those normal traces of M which either satisfy Φ (given by *nmlsucc*), or fail

to satisfy Φ due to input values or their initial state (given by *nmlfail*).

$nml(M?\Phi) \;\widehat{=}\; nmlsucc(M?\Phi) \cup nmlfail(M?\Phi)$, where

$nmlsucc(M?\Phi) \;\widehat{=}\; \{s \mid s \in nml(M) \wedge s \models \Phi\}$

$nmlfail(M?\Phi) \;\widehat{=}\; \{s \mid s \in nml(M) \wedge \exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, M, \Phi, i)\}$

The divergent traces of $M?\Phi$ are those divergent traces which satisfy Φ (*i.e.*, those where all non-diverging traces that do not differ from them before the point of divergence satisfy Φ), given by *divsucc*, or fail to satisfy Φ due to input values (given by *divfail*).

$div(M?\Phi, n) \;\widehat{=}\; divsucc(M?\Phi, n) \cup divfail(M?\Phi, n)$, where

$divsucc(M?\Phi, n) \;\widehat{=}\; \{s \mid s \in div(M, n) \wedge \forall\, t \in \Delta \bullet s =_n t \Rightarrow t \models \Phi\}$

$divfail(M?\Phi, n) \;\widehat{=}\; \{s \mid s \in div(M, n) \wedge \exists\, i \in 1..n \bullet \Gamma(s, M, \Phi, i)\}$

The set of all possible traces of $M?\Phi$ is

$$tr(M?\Phi) \;\widehat{=}\; nml(M?\Phi) \cup (\bigcup_{n \geq 1} div(M?\Phi, n)) \,. \qquad \square$$

Example 3.1.

Reconsider the driver agent *Agent* of Example 2.1. Let the desire Φ of the agent be that the time cost of the current route is always no greater than the previous current route. This can be specified in LTL as follows.

$$\Box(\exists\, t : \mathbb{N} \bullet t = time(current) \wedge \bigcirc(time(current) \leq t))$$

In this case, the following normal trace (where the agent does not change to the faster route $R2$) would no longer be allowed.

$\langle (current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 45, R2 \mapsto 35\}),$
$(current = R1, time = \{R1 \mapsto 45, R2 \mapsto 35\}, time? = \{R1 \mapsto 35, R3 \mapsto 40\}),$
$\ldots \rangle$

If all routes provided by the navigator take more time than the current route, the agent would have no choice but to violate its current goal. The behaviour after this point is considered to be unmotivated behaviour. Such a case is shown below where the input *time*? in the first state gives the agent no choice to satisfy its goal. The rest of the trace is unmotivated.

$\langle (current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 50, R2 \mapsto 45\}),$
$(current = R1, time = \{R1 \mapsto 50, R2 \mapsto 45\}, time? = \{R2 \mapsto 45, R3 \mapsto 40\}),$
$\ldots \rangle$ $\square$

3.2. Desire-driven behaviour

The desire Q of an agent is a fixed, finite sequence of goals. The agent chooses the first goal in Q as its initial goal. The decision making of the agent is then restricted by the current goal. At a moment when the agent realises that the current goal is no longer achievable due to environmental influence, it resets the current goal to the first achievable goal along the desire sequence. If no goal in the desire sequence is achievable, the agent's behaviour becomes unmotivated (choosing any enabled action). At a lower level of abstraction, the unmotivated behaviour might be refined by, for example, introducing a new goal to the desire sequence.

We define a function *FA* to retrieve the first achievable goal, if any, in the desire sequence Q for a module M.

$FA(M, Q) = \Phi$ iff

$$\Phi \in Q \wedge nmlsucc(M?\Phi) \neq \varnothing \wedge$$
$$\forall\, \Psi \bullet idx(\Psi, Q) < idx(\Phi, Q) \Rightarrow nmlsucc(M?\Psi) = \varnothing$$

where $idx(\Phi, Q)$ returns the index of the member Φ in the sequence Q.

Let the notation $M[\sigma]$ denote a module whose initial state is consistent with the state σ, and whose other components are the same as those of M. Formally, the initial state I of $M[\sigma]$ is defined as follows:

$$I \mathrel{\widehat{=}} \{\gamma \mid \gamma \in \Sigma \wedge Var \lhd \gamma = Var \lhd \sigma\}$$

The behaviour of the agent driven by a desire sequence is defined as follows.

Definition 3.3. (Autonomous Agent Driven by Desire)

Let M be a module and Q be a finite sequence of goals. We let an autonomous agent driven by desire be denoted by $M?Q$. The initial goal, denoted as Φ_0, is the first goal in the desire sequence. The behaviour of the autonomous agent can be defined as follows.

$tr(M?Q) \mathrel{\widehat{=}} nml(M?Q, \Phi_0) \cup (\bigcup_{n\geq 1} div(M?Q, \Phi_0, n))$

- The normal behaviour of $M?Q$ comprises three parts. The first part contains the non-divergent traces which can fulfil the current goal Φ. The second part includes the traces in which the current goal Φ is unachievable at a point but it is possible to fulfil another goal within the desire Q after that point. Note that the next achievable goal of the desire sequence only takes effect on the traces which contain unmotivated behaviours with respect to the current goal. The third part contains the traces where it is impossible to fulfil any goal in the desire after that point where the current goal is unachievable.

$$
\begin{aligned}
nml(M?Q, \Phi) \mathrel{\widehat{=}}\ & nmlsucc(M?\Phi) \cup \\
& \{s \mid s \in nmlfail(M?\Phi) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, M, \Phi, i) \wedge \\
& \quad (\exists\, \Psi \bullet \Psi = FA(M[s[i+1]], Q) \wedge (i\,..\,\#s \lhd s) \in nml(M[s[i+1]]?Q, \Psi)))\} \cup \\
& \{s \mid s \in nmlfail(M?\Phi) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, M, \Phi, i) \wedge \\
& \quad (\nexists \Psi \bullet \Psi = FA(M[s[i+1]], Q)))\}
\end{aligned}
$$

  where $i\,..\,\#s \lhd s$ denotes the postfix of s beginning with the ith state.
- The divergent behaviour of $M?Q$ also comprises three parts. The first part contains the divergent traces which can fulfil the current goal Φ before their divergent point. The second part includes the traces which cannot achieve Φ at a point before divergence but can fulfil another goal in the desire Q after that point and before divergence starts. The third part contains the traces where there is no achievable goal in the desire Q after the current goal is unachievable.

$$
\begin{aligned}
div(M?Q, \Phi, n) \mathrel{\widehat{=}}\ & divsucc(M?\Phi, n) \cup \\
& \{s \mid s \in divfail(M?\Phi, n) \wedge (\exists\, i \in 1..n \bullet \Gamma(s, M, \Phi, i) \wedge \\
& \quad (\exists\, \Psi \bullet \Psi = FA(M[s[i+1]], Q) \wedge (i+1\,..\,\#s \lhd s) \in div(M[s[i+1]]?Q, \Psi, n-(i+1))))\} \cup \\
& \{s \mid s \in divfail(M?\Phi, n) \wedge (\exists\, i \in 1..n \bullet \Gamma(s, M, \Phi, i) \wedge \\
& \quad (\nexists \Psi \bullet \Psi = FA(M[s[i+1]], Q)))\}
\end{aligned}
$$

□

According to the above definition, a goal reset happens at the beginning of the execution and whenever the current goal is unachievable due to the environment. This further restricts the behaviour of the autonomous agent by reducing its unmotivated behaviours.

3.3. Refinement obligations regarding desire

The desire-driven behaviour of an autonomous agent only removes a trace where it can choose another one with a shared prefix to achieve the desire. Otherwise, the behaviour is like a standard module without desires. This allows us to refine an agent by introducing additional goals to reduce the unmotivated behaviour of the abstract specification. However, such a restriction is non-monotonic with respect to the refinement order. An intuitive example follows.

Consider the situation shown in Figure 1 which shows a subset of the behaviours of two agents M_1 and M_2. In M_1, s is the only trace which shares a prefix with t of length k and satisfies a goal Φ.

Consider the case where the traces s and t of agent M_1 differ at point $k+1$ due to a local choice made by the agent. At point k, therefore, the agent has a chance to make a local choice and follow trace s to fulfil the goal Φ. So trace t will be removed from the behaviour of $M_1?\Phi$.

Agent M_2 has all traces of M_1 apart from s. It is obvious that $M_2 \sqsupseteq M_1$ according to Definition 2.2. However, with the same goal Φ, trace t will not be removed from the behaviour of $M_2?\Phi$ since, in this case, the agent has no opportunity to fulfil the desire. Hence the desire-driven behaviour $M_1?\Phi$ is not refined by $M_2?\Phi$.

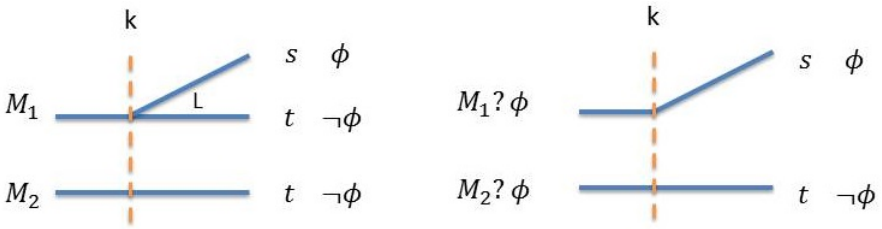

Fig. 1. Counter example for monotonicity

This situation arises whenever we disable a choice which can lead to an agent's desire and hence make it impossible for the refined agent to satisfy the desire. To avoid such refinements, we need an additional proof obligation that ensures that the concrete agent does not introduce more unmotivated behaviour. That is, if a concrete trace has unmotivated behaviour from a given point, then its corresponding abstract trace also has unmotivated behaviour from the same point.

This conclusion can be formalised by the following theorem. For simplicity, we first explore the case of introducing a goal to the empty desire sequence.

Theorem 3.1. (Refinement Obligations)

Let M_1 and M_2 be modules linked by a retrieve relation R and $\Phi_2 \in R(\Phi_1)$. We have $M_2?\Phi_2 \sqsupseteq_R M_1?\Phi_1$ if both of the following conditions hold.

1. $M_2 \sqsupseteq_R M_1$
2. For any trace of M_2 if it is impossible for the agent to make a local choice leading to the satisfaction of Φ_2 at a point in the trace, then from the same point in the corresponding trace of M_1 it is also impossible to satisfy Φ_1. That is, for all $s \in tr(M_1)$ and $t \in tr(M_2)$ where $t \in R(s)$, we have

$$\forall\, i \in \mathbb{N}_1 \bullet \Gamma(t, M_2, \Phi_2, i) \Rightarrow \Gamma(s, M_1, \Phi_1, i)$$

Proof:

Following Definition 2.2, we need to show that for any trace $t \in tr(M_2?\Phi_2)$, each of its corresponding traces s, *i.e.*, those traces where $t \in R(s)$, satisfies $s \in tr(M_1?\Phi_1)$. According to condition 1 and Definition 3.2, we get $s \in tr(M_1)$, since $M_2?\Phi_2 \sqsupseteq M_2 \sqsupseteq_R M_1$.

The proof proceeds by a case analysis of traces s and t based on whether or not they are in Δ (the set of traces without $\bot$) and, if so, on their satisfaction of the respective desires. Since $s \in R(t)$ and $R(\bot) = \{\bot\} \wedge R^{-1}(\bot) = \{\bot\}$, it follows that $t \in \Delta \Leftrightarrow s \in \Delta$.

Assume $s, t \in \Delta$. In this situation, there are two cases to consider:

1. If $t \models \Phi_2$ then its corresponding trace s satisfies Φ_1 since $\Phi_2 \in R(\Phi_1)$ (see the definition of R applied to formulas). Hence, s appears in $tr(M_1?\Phi_1)$.
2. If $t \not\models \Phi_2$ then, according to Definition 3.2, its appearance in $tr(M_2?\Phi_2)$ implies that there exists a point in t where the agent has no opportunity to make a choice leading to the goal Φ_2.

   $$\exists\, i \in \mathbb{N}_1 \bullet \Gamma(t, M_2, \Phi_2, i)$$

   Hence in the corresponding trace s, Γ holds at the same point i due to condition 2. This then leads to

   $$\exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, M_1, \Phi_1, i)$$

   Hence, according to Definition 3.2, s also appears in $tr(M_1?\Phi_1)$.

The remaining case to consider is when $s, t \notin \Delta$. According to Definition 2.1, we can find $n > 1$ so that t diverges at the nth step, *i.e.*, $t \in div(M_2?\Phi_2, n)$.

1. If $t \in divsucc(M_2?\Phi_2, n)$, which means $\forall\, u \in \Delta \bullet t =_n u \Rightarrow u \models \Phi_2$, then since $\Phi_2 \in R(\Phi_1)$ and $t \in R(s)$, we have $\forall\, r \in \Delta \bullet s =_n r \Rightarrow r \models \Phi_1$. Hence $s \in div(M_1?\Phi_1, n)$.
2. If $t \in divfail(M_2?\Phi_2, n)$, which means $\exists\, i \in 1..n \bullet \Gamma(t, M_2, \Phi_2, i)$, then according to condition 2 we can conclude that $\exists\, i \in 1..n \bullet \Gamma(s, M_1, \Phi_1, i)$. Hence we have $s \in div(M_1?\Phi_1, n)$.

In summary, we have $t \in tr(M_2?\Phi_2) \Rightarrow s \in tr(M_1?\Phi_1)$ which means $M_2?\Phi_2 \sqsupseteq_R M_1?\Phi_1$. □

The refinement obligations imply that the refined agent has no more unmotivated behaviour than the abstract agent. Reconsider the counter example of Figure 1. The trace s has unmotivated behaviour in $M_2?\Phi$ but not in $M_1?\Phi$.

Theorem 3.1 considered an agent with only a single goal. A general version of the refinement obligations for desires is as follows.

Theorem 3.2. (General Obligations) Let M_1 and M_2 be two modules linked by a retrieve relation R, and Q_1 and Q_2 be sequences of goals with the same length and for any index i within the domain of the sequences, $Q_2[i] \in R(Q_1[i])$. We have $M_2?Q_2 \sqsupseteq_R M_1?Q_1$ if both of the following conditions hold.

1. $M_2 \sqsupseteq_R M_1$
2. For all $s \in tr(M_1)$ and $t \in tr(M_2)$ where $t \in R(s)$, $GOB(s, t, Q_1, Q_2)$ holds, where

$$GOB(s, t, Q_1, Q_2) \mathrel{\widehat{=}} \forall\, n \in 1..\#Q_1, i \in \mathbb{N}_1 \bullet Q_2[n] = FA(M_2[t[1]], Q_2) \wedge \Gamma(t, M_2, Q_2[n], i) \Rightarrow$$
$$Q_1[n] = FA(M_1[s[1]], Q_1) \wedge \Gamma(s, M_1, Q_1[n], i) \wedge GOB(i{+}1..\#s \lhd s, i{+}1..\#t \lhd t, Q_1, Q_2)$$

Note that the notation GOB in the second condition is defined in a recursive manner. It says that if trace t in M_2 becomes unmotivated w.r.t the first achievable goal in desire sequence Q_2, then its corresponding trace s in M_1 also becomes unmotivated at the same point w.r.t. the corresponding goal in sequence Q_1; and the condition holds for the subtraces after that point. The recursion terminates when the subtrace can achieve a goal in the desire sequence or none of the goals in the sequence is achievable.

The proof can be partitioned based on the recursive unfolding of GOB. For every partition with a fixed goal (the first achievable goal), the conclusion can be obtained using a similar proof technique to that of Theorem 3.1. □

With the refinement obligations we obtain from Theorems 3.1 and 3.2, refinement checking between two autonomous agents can be done modularly and recursively. First we can check the refinement between their modules without desires. Then we check the obligation for each goal along their desire sequences.

3.4. Development strategy

The refinement obligations require the refined agent preserve the possibility of achieving the desire under any environmental inputs. A development strategy which satisfies the obligations is to introduce local mechanisms to adapt to environmental "hostility". The strategy includes the following three rules.

St1: Weaken the preconditions of actions to accept a larger range of inputs. This makes the agent handle more situations of the environment and hence reduces divergence.

This strategy can be justified with the theory of conventional data refinement where a divergent behaviour occurs when the inputs violate the precondition.

St2: Optimise the decision making algorithms so that the agent has a more deterministic way to achieve the desire under certain inputs than the specification does.

This strategy can be also justified with the conventional data refinement theory where a refined set of post states is a subset of the abstract post states.

For example, for the goal to arrive at the destination in the fastest time, the action *Update* of Example 2.1 could be refined to always choose the fastest route.

Update

$$\vdots$$
$$current' \in \mathrm{dom}\, time'$$
$$\forall\, r : \mathrm{dom}\, time' \bullet time'(r) \geq time'(current')$$

St3: Introduce additional goals to the desire sequence. An alternative goal can provide additional motivation for the agent which will further regulate its (unmotivated) behaviour.

This strategy is novel. It can be justified with the following theorem.

According to Definition 3.3, it is intuitive to obtain the conclusion that introducing a goal to the agent's desire sequence refines its behaviour.

Theorem 3.3. Let M be a module and Q_1 and Q_2 be desires such that Q_1 is subsequence of Q_2. Then we have $M?Q_2 \sqsupseteq M?Q_1$.

Proof

1. If $Q_1 = \langle\rangle$ and $Q_2 = \langle\Phi\rangle$, then $tr(M?Q_1) = tr(M)$, $tr(M?Q_2) = tr(M?\Phi)$. From Definition 3.2, all traces of $M?\Phi$ are traces of M, *i.e.*, $tr(M?\Phi) \subseteq tr(M)$. Hence, we have $tr(M?Q_2) \subseteq tr(M?Q_1)$ and therefore $M?Q_2 \sqsupseteq M?Q_1$.
2. If $Q_1 \neq \langle\rangle$ and Q_1 is a subsequence of Q_2, then $tr(M?Q_2)$ can be obtained recursively from Definition 3.3. It is straightforward to show that in each recursive step, any trace which does not satisfy the original goals nor the newly introduced goal but at some point has the opportunity to achieve the new goal is removed from the behaviour of the previous step. Hence we have $tr(M?Q_2) \subseteq tr(M?Q_1)$ and therefore $M?Q_2 \sqsupseteq M?Q_1$. □

According to Theorem 3.3, an agent with more goals is a refinement of one with the same module but a subsequence of the desire.

Example 3.2. Recall the driver agent $Agent?\Phi$ in Example 3.1. For the unmotivated behaviour where the navigator provides no faster route than the current one, we introduce an alternative goal to allow the agent to choose the fastest of the provided routes. Formally, let Φ' be

$$\Box(\nexists r : \mathrm{dom}\, time \bullet time(r) < time(current))$$

With this alternative goal, the following trace from Example 3.1 no longer results in unmotivated behaviour.

$\langle(current = R1, time = \{R1 \mapsto 40\}, time? = \{R1 \mapsto 50, R2 \mapsto 45\}),$
$(current = R2, time = \{R1 \mapsto 50, R2 \mapsto 45\}, time? = \{R2 \mapsto 45, R3 \mapsto 40\}),$
$\ldots\rangle$

Let $M?Q$ be an agent with $Q = \langle\Phi, \Phi'\rangle$. It will initially follow the goal Φ and when it is impossible to achieve Φ at any moment, it will change its goal to Φ' rather than running without motivation. This design is a refinement of the design $M?\langle\Phi\rangle$. □

In summary, **St1** is a conventional refinement rule. **St2** requires the designer to ensure the agent has a way to achieve its desire in the development. **St3** allows the designer to design a sequence of alternative goals when the agent fails to satisfy its current goal. Following the above strategy, the refined agent has more reliable local mechanisms to achieve the desire than it does in the abstract specification.

4. Autonomous Agents Driven by Belief and Desire

An agent's belief reflects its local perspective on its environment. In this paper, a belief of an agent at a given moment is an LTL formula indicating the agent's perspective of the environmental behaviour. For instance, the agent could believe that the environment will eventually provide a certain input, which can be specified as an eventually property; or that the environment will never provide certain input, which can be specified as an always property. During the execution, the belief of the agent evolves in reaction to the inputs from the environment. Hence a belief-updating process is often embedded in the agent design.

Given an execution trace s of an agent, we can determine whether s satisfies a belief β by checking $s \models \beta$. The belief affects the agent's *local choices*, *i.e.*, those based on its current belief about the environmental behaviour. For example, if the driver agent believes that a route to the destination will be blocked during the rush hour, it would not choose to take that route until the navigator informs it otherwise.

However, without a goal, the belief alone has no effect on the behaviour of the agent. In other words, without a motivation, the rationality of the agent's decision cannot be judged. The agent has no reason not to perform any enabled action since it does not know what to achieve. Therefore, in this paper, the belief is only considered when the agent has a goal to achieve and the reasoning following from the belief is determined based on its current goal.

4.1. Belief-driven behaviour

Let β be a belief of an agent M which is driven by a goal Φ. We denote this agent by $M?\Phi!\beta$. An unmotivated behaviour of an agent is a trace where the current goal cannot be achieved on every consequent path consistent with the current belief from some point, while it could be achieved if the environment provided different input. According to this perspective, the unmotivated behaviour based on both belief and desire is defined as follows:

$$\begin{aligned}\Upsilon(s, M, \Phi, \beta, i) \mathrel{\widehat{=}} & (\forall\, r \in nml(M) \bullet r =_i s \Rightarrow (r \models \beta \Rightarrow r \not\models \Phi)) \wedge \\ & (i > 1 \Rightarrow \\ & \quad (\exists\, u \in nml(M) \bullet u =_{i-1} s \wedge \\ & \qquad (Var \cup Out) \lhd u = (Var \cup Out) \lhd s \wedge u \models \beta \wedge u \models \Phi))\end{aligned}$$

According to this definition, those traces which cannot achieve the current goal under the current belief are classified as unmotivated traces and can be refined in the further design. Furthermore, a rational agent should choose a path where the goal is more likely to be achieved when the environment is consistent with the belief. Therefore, the agent would not choose a path in the following two scenarios:

1. *every* execution after choosing this path does not achieve the goal when the belief is satisfied, and there is another choice of path where there is the possibility of achieving the goal with the belief satisfied. Θ_1 captures this scenario:

$$\begin{aligned}\Theta_1(s, M, \Phi, \beta, i) \mathrel{\widehat{=}} & i > 1 \wedge \\ & (\forall\, r \in nml(M) \bullet r =_i s \Rightarrow (r \models \beta \Rightarrow r \not\models \Phi)) \wedge \\ & (\exists\, u \in nml(M) \bullet u =_{i-1} s \wedge In \lhd u[i] = In \lhd s[i] \wedge u \models \beta \wedge u \models \Phi)\end{aligned}$$

2. *there is* an execution after choosing this path that does not achieve the goal when the belief is satisfied, and there is another choice of path where the achievement of the goal is guaranteed. Θ_2 captures this scenario:

$$\begin{aligned}\Theta_2(s, M, \Phi, \beta, i) \mathrel{\widehat{=}} & i > 1 \wedge \\ & (\exists\, r \in nml(M) \bullet r =_i s \wedge r \models \beta \wedge r \not\models \Phi) \wedge \\ & (\exists\, u \in nml(M) \bullet u =_{i-1} s \wedge In \lhd u[i] = In \lhd s[i] \wedge \\ & \qquad (\forall\, w \in nml(M) \bullet w =_i u \Rightarrow (w \models \beta \Rightarrow w \models \Phi)))\end{aligned}$$

The traces satisfying the above two scenarios should be removed from the behaviour of the agent. Such traces can be identified with the following predicate:

$$\Theta(s, M, \Phi, \beta, i) \mathrel{\widehat{=}} \Theta_1(s, M, \Phi, \beta, i) \vee \Theta_2(s, M, \Phi, \beta, i)$$

If a trace s satisfies the above property Θ, we say s is an *irrational* path with respect to the belief β and the goal Φ. An agent should not follow an irrational path when it makes a local choice among enabled actions.

Definition 4.1. (Autonomous Agent Driven by Belief) Let $M?\Phi!\beta$ be an autonomous agent driven by the goal Φ and belief β. The behaviour of $M?\Phi!\beta$ prevents the agent choosing irrational paths.

$$tr(M?\Phi!\beta) \;\widehat{=}\; nml(M?\Phi!\beta) \cup (\bigcup_{n \geq 1} div(M?\Phi!\beta, n))$$

$$nml(M?\Phi!\beta) \mathrel{\widehat{=}} nmlsucc(M?\Phi!\beta) \cup nmlfail(M?\Phi!\beta)$$

$$div(M?\Phi!\beta, n) \mathrel{\widehat{=}} divsucc(M?\Phi!\beta, n) \cup divfail(M?\Phi!\beta, n)$$

- The normal successful behaviour of $M?\Phi!\beta$ contains the non-divergent traces which satisfies both β and Φ.

$$nmlsucc(M?\Phi!\beta) \mathrel{\widehat{=}} \{s \mid s \in nml(M) \wedge s \models \beta \wedge s \models \Phi\}$$

- The normal failed behaviour of $M?\Phi!\beta$ contains the non-divergent unmotivated traces but not irrational traces.

$$nmlfail(M?\Phi!\beta) \mathrel{\widehat{=}} \{s \mid s \in nml(M) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Upsilon(s, M, \Phi, \beta, i)) \;\wedge \neg(\exists\, j \in \mathbb{N}_1 \bullet \Theta(s, M, \Phi, \beta, j))\}$$

- The divergent successful behaviour of $M?\Phi!\beta$ contains the divergent traces which satisfy both β and Φ before they diverge.

$$divsucc(M?\Phi!\beta, n) \mathrel{\widehat{=}} \{s \mid s \in div(M, n) \land (\forall\, t \in \Delta \bullet s =_n t \Rightarrow (t \models \beta \land t \models \Phi))\}$$

  where Δ is the set of traces not containing $\bot$.
- The divergent failed behaviour of $M?\Phi!\beta$ contains the divergent unmotivated traces before they diverge but removes the irrational traces.

$$divfail(M?\Phi!\beta, n) \mathrel{\widehat{=}} \{s \mid s \in div(M, n) \land (\exists\, i \in 1..n \bullet \Upsilon(s, M, \Phi, \beta, i)) \land \neg(\exists\, j \in 1..n \bullet \Theta(s, M, \Phi, \beta, j))\}$$ □

Example 4.1. Reconsider the driver agent $Agent?\Phi$ in Example 3.1. Let the belief of the agent be that the travel time of route R_1 is always less than or equal to that of the current route. Formally, we define its belief β as follows:

$$\beta \mathrel{\widehat{=}} \Box(R_1 \in \mathrm{dom}\, time? \land time?(R_1) \leq time(current))$$

With this belief, the agent can always choose the route R_1 since it will guarantee that the goal Φ will be achieved when β is satisfied. A possible behaviour is

$\langle(current = R_1, time = \{R_1 \mapsto 40\}, time? = \{R_1 \mapsto 35, R_2 \mapsto 30\}),$
$(current = R_1, time = \{R_1 \mapsto 35, R_2 \mapsto 30\}, time? = \{R_1 \mapsto 30\}),$
$...\rangle$

In fact, with this belief the agent should not choose R_2 since we can find the following trace where the goal Φ is violated when the belief β still holds. Such traces will be ruled out since they are irrational according to Θ:

$\langle(current = R_1, time = \{R_1 \mapsto 40\}, time? = \{R_1 \mapsto 35, R_2 \mapsto 30\}),$
$(current = R_2, time = \{R_1 \mapsto 35, R_2 \mapsto 30\}, time? = \{R_1 \mapsto 20, R_2 \mapsto 40\}),$
$(current = R_2, time = \{R_1 \mapsto 20, R_2 \mapsto 40\}, time? = \{R_1 \mapsto 20, R_2 \mapsto 30\}),$
$...\rangle$

When the environment provides an input which violates its belief β, the trace will be considered as unmotivated behaviour. □

Note that the behaviours of *nmlfail* and *divfail* include the unmotivated traces where the current belief is violated (according to the definition of Υ). The reason for this is that the belief is not always consistent with the actual environment. The agent cannot control the inputs provided by the open environment. As an autonomous agent, it should be able to update its belief accordingly when an inconsistency occurs. In our framework, the belief-updating designs are guided by refinement as defined in the next section.

4.2. Belief-updating behaviour

So far we have considered an agent as having a single, fixed belief. In general, a belief is assigned to an agent initially and will be updated by the agent during its execution according to its local states and the interactions with the environment. The belief-updating mechanism specifies when the agent should update its belief and what the current belief will be updated to. In this paper, we abstract the belief-updating process as a function $BF : \Sigma^* \to B$ whose domain includes all interaction sequences between agent and environment so far. In principle, a belief-updating function should be a total function indicating the belief the agent should update to under any given interaction sequence. However, because of the open environment it is often impossible for a designer to consider all possible execution cases. Hence, a particular design of the belief-updating function is often an incomplete specification that needs to be refined further when the designer has more information about the environment. In order to allow such development, our model considers that an agent will have a default belief *true* after it updates its belief when the current execution situation is not included in the domain of its belief-updating function.

The initial belief of an agent is given by $BF(\langle\rangle)$. Once the belief is determined, the agent chooses a rational path towards the current goal according to the current belief. Whenever the agent realises that the current input from the environment violates its current belief, it will update its belief according to the

belief-updating function. The condition can be formalised by the predicate $\Gamma(s, M, \beta, i)$ where the execution trace s encounters an inconsistency between the current belief β and the environment input at step i. The new belief is determined by $BF(t)$ where t is the subsequence of s up to step $i-1$. If t is not covered in the domain of BF, the agent will set its belief to *true*.

Whenever the agent realises that the current goal cannot be achieved while the environment behaves as it believes, it changes its goal to the first achievable goal in the desire sequence. Hence, we redefine the function FA to retrieve the first achievable goal, if any, among the desire sequence according to belief β:

$FA(M, Q, \beta) = \Phi$ iff

$$\Phi \in Q \wedge nmlsucc(M?\Phi!\beta) \neq \varnothing \wedge$$
$$(\forall \Psi \bullet idx(\Psi, Q) < idx(\Phi, Q) \Rightarrow nmlsucc(M?\Psi!\beta) = \varnothing)$$

In summary, the behaviour of an autonomous agent driven by both desire and belief is defined as follows.

Definition 4.2. (Autonomous Agent Driven by Beliefs and Desires)

Let M be a module, Q be a sequence of goals and BF be a belief-updating function. An autonomous agent driven by desire Q and with belief-updating function BF is denoted by $M?Q!BF$. The behaviour of this autonomous agent can be defined as follows.

Base case: If the desire sequence is empty, *i.e.* $Q = \langle\rangle$, the agent has no motivation to guide its action choice. Hence the belief has no effect on the behaviour of $M?Q!BF$, *i.e.*,

$$M?\langle\rangle!BF \mathrel{\widehat{=}} M$$

General case: If the desire sequence is non-empty, *i.e.* $Q \neq \langle\rangle$, the agent firstly follows the initial belief $\beta_0 = BF(\langle\rangle)$ and the first goal of the desire sequence. Whenever the inputs provided by the environment violate the current belief, the agent updates the current belief with the belief-updating function BF. Whenever the environment makes the current goal unachievable, the agent updates the current goal to the first achievable goal in the desire sequence.

$$tr(M?Q!BF) \mathrel{\widehat{=}} nml(M?Q!BF, \Phi_0, \beta_0) \cup (\bigcup_{n\geq 1} div(M?Q!BF, \Phi_0, \beta_0, n)$$

- The normal behaviour of the agent can be divided into four kinds of traces: (1) the traces which satisfy the current belief and goal; (2) the traces containing unmotivated behaviour with respect to the current goal from some point after which the agent changes its goal to the next one in the desire sequence if possible; (3) the traces satisfying the similar condition to (2) but where the agent has no achievable goal in its desire sequence and hence the consequent behaviour is without any restriction; (4) the traces which violate the current belief from some point after which the agent changes its belief based on the belief-updating function. If a belief cannot be generated by that function, the agent will set the belief to *true*. (Note that we reuse the function Γ of Section 3 in this and the following definition to indicate that a belief does not hold on a trace after some state.)

  $nml(M?Q!BF, \Phi, \beta) \mathrel{\widehat{=}} nmlsucc(M?\Phi!\beta) \cup$

  $$\{s \mid s \in nmlfail(M?\Phi!\beta) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Upsilon(s, M, \Phi, \beta, i) \wedge$$
  $$(\exists\, \Psi \bullet \Psi = FA(M[s[i]], Q, \beta) \wedge (i\,..\,\#s \lhd s) \in nml(M[s[i]]?Q!BF, \Psi, \beta)))\} \cup$$
  $$\{s \mid s \in nmlfail(M?\Phi!\beta) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Upsilon(s, M, \Phi, \beta, i) \wedge$$
  $$(\nexists \Psi \bullet \Psi = FA(M[s[i]], Q, \beta)))\} \cup$$
  $$\{s \mid s \in nml(M?\Phi!\beta) \wedge (\exists\, i \in \mathbb{N}_1 \bullet \Gamma(s, M, \beta, i) \wedge$$
  $$((1\,..\,i-1 \lhd s) \in \mathrm{dom}\, BF \Rightarrow (i\,..\,\#s \lhd s) \in nml(M[s[i]]?Q!BF, \Phi, BF(1\,..\,i-1 \lhd s))) \wedge$$
  $$((1\,..\,i-1 \lhd s) \notin \mathrm{dom}\, BF \Rightarrow (i\,..\,\#s \lhd s) \in nml(M[s[i]]?Q!BF, \Phi, true)))\}$$

  where $1\,..\,i-1 \lhd s$ is the subsequence of s containing its first $i-1$ elements.
- The divergent behaviour of the agent contains similar kinds of traces to the normal behaviour. The difference is that we only consider the behaviour before divergence.

  $div(M?Q!BF, \Phi, \beta, n) \mathrel{\widehat{=}} divsucc(M?\Phi!\beta) \cup$

  $$\{s \mid s \in divfail(M?\Phi!\beta, n) \wedge (\exists\, i \in 1..n \bullet \Upsilon(s, M, \Phi, \beta, i) \wedge$$

$$
\begin{aligned}
&(\exists\, \Psi \bullet \Psi = FA(M[s[i]], Q, \beta) \land (i+1\,..\,\#s \lhd s) \in div(M[s[i]]?Q!BF, \Psi, \beta, n-i)))\} \cup \\
&\{s \mid s \in divfail(M?\Phi!\beta, n) \land (\exists\, i \in 1..n \bullet \Upsilon(s, M, \Phi, \beta, i) \land \\
&(\nexists \Psi \bullet \Psi = FA(M[s[i]], Q, \beta)))\} \cup \\
&\{s \mid s \in div(M?\Phi!\beta, n) \land (\exists\, i \in 1..n \bullet \Gamma(s, M, \beta, i) \land \\
&((1\,..\,i-1 \lhd s) \in \operatorname{dom} BF \Rightarrow (i\,..\,\#s \lhd s) \in div(M[s[i]]?Q!BF, \Phi, BF(1\,..\,i-1 \lhd s)) \land \\
&((1\,..\,i-1 \lhd s) \notin \operatorname{dom} BF \Rightarrow (i\,..\,\#s \lhd s) \in div(M[s[i]]?Q!BF, \Phi, true, n-i)))\} \qquad \square
\end{aligned}
$$

From the definition of the behaviour of an autonomous agent, having a belief would further refine the behaviour by removing irrational paths. Whenever an agent has a specific belief other than *true*, its behaviour is regulated based on it.

Example 4.2.

Reconsider the driver agent specified by $Agent?Q$ with the desire sequence $Q = \langle \Phi, \Phi' \rangle$ where

$$\Phi \mathrel{\widehat{=}} \Box(\exists\, t : \mathbb{N} \bullet t = time(current) \land \bigcirc(time(current) \leq t))$$

$$\Phi' \mathrel{\widehat{=}} \Box(\nexists r : \operatorname{dom} time \bullet time(r) < time(current))$$

as is defined in Section 3.

Let the initial belief of the agent be that the travel time of route R_1 is always less than or equal to that of the current route. Let $i \in \{1, 2, 3\}$. We define

$$\beta_i \mathrel{\widehat{=}} \Box(R_i \in \operatorname{dom} time? \land time?(R_i) \leq time(current))$$

Then the initial belief can be specified by β_1. Whenever the agent gets an input *time*? such that its current belief is not true but, for some i, $time?(R_i)$ is less than or equal to $time(current)$, the agent will update its belief to β_i.

Let BF be the belief updating function of the agent,

$$BF(s \frown \langle \sigma \rangle) \mathrel{\widehat{=}} \beta_i$$

where $\sigma.time?(R_i) \leq \sigma.time(current) \land \forall\, r \neq R_i \bullet \sigma.time?(r) > \sigma.time(current)$.

Note that we have purposefully not considered the case where more than one route requires less time than the current route. Such cases are introduced as part of a development step guided by refinement notation in Example 4.3.

Whenever the environment evolves to a configuration where the travel time of route R_1 becomes greater than the current route, the agent will update its belief according to the belief-updating function. Consider the following trace s:

$\langle (current = R_1, time = \{R_1 \mapsto 40\}, time? = \{R_1 \mapsto 35, R_2 \mapsto 30\}),$
$(current = R_1, time = \{R_1 \mapsto 35, R_2 \mapsto 30\}, time? = \{R_1 \mapsto 50, R_2 \mapsto 30, R_3 \mapsto 40\}),$
$(current = R_2, time = \{R_1 \mapsto 50, R_2 \mapsto 30, R_3 \mapsto 40\}, time? = \{R_2 \mapsto 25\}),$
$...\rangle$

At the second entry of s, the agent realises that route R_2 has less travel time than the current route and will update its belief to β_2, *i.e.*,

$$BF(s^2) = \beta_2$$

Whenever the environment provides an input where none of the routes has less travel time than the current one, it will neither satisfy the agent's current belief nor its current goal Φ. In this case, the agent $Agent?Q!BF$ will change both its belief and goal. The belief will be updated to the default belief *true* (since this situation is not specified in BF) and the goal will be changed to Φ'. $\square$

4.3. Proof obligation regarding belief

Similarly to the introduction of goals, the introduction of beliefs to a goal-driven agent is non-monotonic with respect to the refinement relation. Consider the following counter-example shown in Figure 2. $tr(M_1?\Phi)$

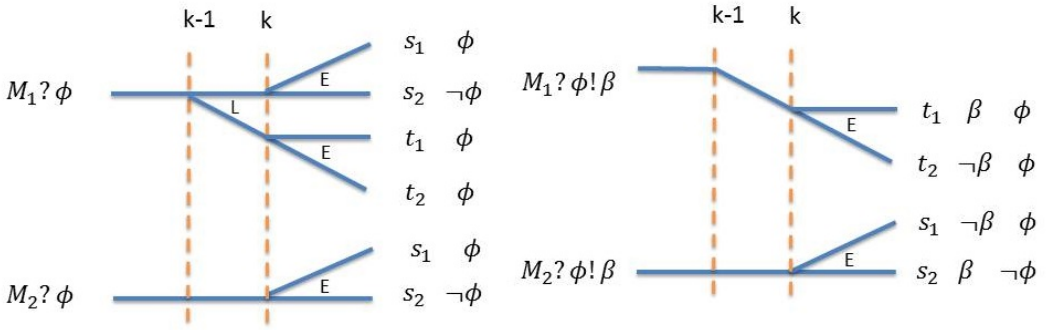

Fig. 2. Counter example for belief monotonicity

contains traces s_1, s_2, t_1 and t_2 where $s_1 \models \Phi$, $s_2 \not\models \Phi$ and $t_1, t_2 \models \Phi$. s_1 differs from s_2 at point k by an environmental choice. t_1 and t_2 differs from s_1 and s_2 at point $k-1$ by a local choice. $tr(M_2?\Phi)$ only contains the traces s_1 and s_2. It is obvious that $M_2?\Phi \sqsupseteq M_1?\Phi$.

Consider a belief β where $s_1 \not\models \beta$, $s_2 \models \beta$, $t_1 \models \beta$ and $t_2 \not\models \beta$. That is, the belief indicates the final trace will be either s_2 or t_1. The agent $M1?\Phi$ extended with this belief would not make the choice resulting in s_2 since this trace does not satisfy its desire. Hence, the agent $M_1?\Phi!\beta$ would not have traces s_1 and s_2, leaving only t_1 and t_2. The agent $M_2?\Phi$, on the other hand, has no choice to make and $M_2?\Phi!\beta$ would have traces s_1 and s_2. Therefore $M_1?\Phi!\beta$ is not refined by $M_2?\Phi!\beta$.

In order to refine an agent driven by both desire and belief, a further refinement obligation should be fulfilled.

Theorem 4.1. (Refinement obligation regarding belief) Let $M_1?\Phi_1$ and $M_2?\Phi_2$ be two goal driven agents satisfying $M_2?\Phi_2 \sqsupseteq_R M_1?\Phi_1$ where R is a retrieve relation. Let β_1 and β_2 be beliefs satisfying $\beta_2 \in R(\beta_1)$. Then we have $M_2?\Phi_2!\beta_2 \sqsupseteq_R M_1?\Phi_1!\beta_1$ if the following condition holds:

For any trace s of the abstract agent M_1, if s is an irrational path with respect to β_1 and Φ_1, then its corresponding trace t of the concrete agent M_2 is also an irrational path with respect to β_2 and Φ_2.

$$(\exists\, i : \mathbb{N}_1 \bullet \Theta(s, M_1, \Phi_1, \beta_1, i)) \Rightarrow (\exists\, j : \mathbb{N}_1 \bullet \Theta(t, M_2, \Phi_2, \beta_2, j))$$

Proof:

With the fact that $M_2?\Phi_2 \sqsupseteq_R M_1?\Phi_1$, we have

$$tr(M_1?\Phi_1) \subseteq R(tr(M_2?\Phi_2))$$

The obligation implies that for any trace s belonging to $\{s \mid \exists\, i \in \mathbb{N}_1 \bullet \Theta(s, M_1, \Phi_1, \beta_1, i)\}$, its corresponding trace t belongs to $\{t \mid \exists\, j \in \mathbb{N}_1 \bullet \Theta(t, M_2, \Phi_2, \beta_2, j)\}$. According to the definition of *nmlfail* in Definition 4.1, the belief rules out the traces which satisfy the Θ condition. Hence, for any trace t belonging to $tr(M_2?\Phi_2!\beta_2)$, its corresponding trace s belongs to $tr(M_1?\Phi_1!\beta_1)$, which leads to the conclusion that $M_2?\Phi_2!\beta_2 \sqsupseteq_R M_1?\Phi_1!\beta_1$. □

For two autonomous agents $Agent_1$ and $Agent_2$ whose desire sequences and belief-updating functions are related by a retrieve relation R, if their modules are related by refinement, then the refinement of the entire agent holds when the abstract agent changes its belief or goal whenever the concrete one does, and for every configuration of current belief and goal during their executions the refinement relation is preserved. Formally, we have the following refinement obligation to preserve the monotonicity of module development.

Theorem 4.2. (Refinement obligation for autonomous agents) Let Q_1 and Q_2 be sequences of goals with the same length and every goal within them is related by R, *i.e.*, for any index i within the domain of sequences, $Q_2[i] \in R(Q_1[i])$. Let BF_1 and BF_2 be belief-updating functions related by R, *i.e.*, for any $s \in dom(BF_1)$, $t \in dom(BF_2)$ and $t \in R(s)$, we have $BF_2(t) \in R(BF_1(s))$. Let $M_1?Q_1!BF_1$ and $M_2?Q_2!BF_2$ be two autonomous agents satisfying $M_2 \sqsupseteq_R M_1$. Then we have $M_2?Q_2!BF_2 \sqsupseteq_R M_1?Q_1!BF_1$ if for any traces $s \in tr(M_1?Q_1!BF_1)$, $t \in tr(M_2?Q_2!BF_2)$ satisfying $t \in R(s)$, the condition $GOB(s, t, M_1, \Phi_1, \beta_1, M_2, \Phi_2, \beta_2)$ holds, where

$$GOB(s,t,M_1,\Phi_1,\beta_1,M_2,\Phi_2,\beta_2) \mathrel{\widehat{=}} M_2[t[1]]?\Phi_2!\beta_2 \sqsupseteq_R M_1[s[1]]?\Phi_1!\beta_1 \wedge$$
$$(\forall\, i \in \mathbb{N}_1 \bullet \Upsilon(t,M_2,\Phi_2,\beta_2,i) \Rightarrow \Upsilon(s,M_1,\Phi_1,\beta_1,i) \wedge$$
$$GOB(i\,..\,\#s \lhd s, i\,..\,\#t \lhd t, M_1[s[i]], FA(M_1[s[i]],Q_1,\beta_1),\beta_1, M_2[t[i]], FA(M_2[t[i]],Q_2,\beta_2),\beta_2)) \wedge$$
$$(\forall\, j \in \mathbb{N}_1 \bullet \Gamma(t,M_2,\beta_2,j) \Rightarrow \Gamma(s,M_1,\beta_1,j) \wedge$$
$$GOB(j\,..\,\#s \lhd s, j\,..\,\#t \lhd t, M_1[s[j]], \Phi_1, BF_1(1\,..\,j-1 \lhd s), M_2[t[j]], \Phi_2, BF_2(1\,..\,j-1 \lhd t)))$$

The condition *GOB* says that for any fragment of execution where the current mental features are Φ_2 and β_2, $Agent_2$ is a refinement of $Agent_1$ with the linked features Φ_1 and β_1, and when $Agent_2$ needs to change its goal or update its belief, $Agent_1$ also does the corresponding changes. Since within every partition based on the recursive unfolding of *GOB*, the refinement for the fixed belief and goal is guaranteed, the conclusion for the entire execution is obtained. □

4.4. Development strategy

According to Definition 4.1, the behaviour of the agent is regulated by its current belief and its belief-updating function. If the belief updating function covers more scenarios, the behaviour will become more deterministic, which results in a refinement of the system. Based on this perspective, we define an order over the belief-updating functions.

Definition 4.3. (Order of BF) Given two belief-updating functions BF_1 and BF_2. We say that BF_2 is at least as strong as BF_1, denoted as $BF_2 \sqsupseteq BF_1$, if and only if BF_2 is a conservative extension of BF_1, *i.e.*,

(1) $\mathrm{dom}(BF_1) \subseteq \mathrm{dom}(BF_2)$ and

(2) $\forall\, s \in \mathrm{dom}(BF_1) \bullet BF_1(s) = BF_2(s)$ □

The belief generated by the belief-updating function BF is the criteria for the agent when it makes a choice among enabled local actions. It follows that a belief-updating function covering more execution sequences can result in a refinement of an autonomous agent since it can rule out more traces.

Theorem 4.3. Let $M?Q_1!BF_1$ and $M?Q_2!BF_2$ be autonomous agents with $BF_2 \sqsupseteq BF_1$ and Q_1 is a subsequence of Q_2. Then we have $M?Q_2!BF_2 \sqsupseteq M?Q_1!BF_1$.

Proof: For the base case where Q_1 is empty, the conclusion holds due to the base case definitions in Definition 4.2 and Theorem 3.3. For the general case, the complementary goal and updated belief further rule out some of the unmotivated traces according to the former goal and belief. It is straightforward that every trace of $M?Q_2!BF_2$ is a trace of $M?Q_1!BF_1$. □

According to Theorem 4.3, a refinement of an agent $M?Q!BF$ can be obtained by extending its belief-updating function BF to cover more of the cases when the belief of the agent is inconsistent with the real environment. It supports the following development strategy.

St4: Design and optimise the belief-updating mechanism of an agent by letting it cover more configurations of the environment.

Example 4.3. Reconsider the agent in Example 4.2. Its belief-updating function BF does not cover the cases where more than one route requires less time than the current one, *e.g.*

$$time?(R_1) > time(current) \wedge time?(R_2) \leq time(current) \wedge time?(R_3) \leq time(current) \qquad (*)$$

Whenever the environment provides such an input *time*?, the agent would have the weakest belief *true* which places no restriction on its behaviour. To solve this problem, we can define a new belief-updating function BF' which enriches the domain of BF by adding the cases mentioned above, *e.g.*,

$$BF'(s \frown \langle\sigma\rangle) = \beta_2$$

where $\sigma \models (*)$.

With this new belief-updating function, the agent will update its belief to β_2 rather than *true* when both R_2 and R_3 require less time than the current route. □

5. Related work

Much research has focussed on the design of the decision-making process of an agent to fulfil its goals based on its belief about the environment. Winikoff *et al.* provide a unified goal framework to represent varieties of goal types and their combinations [WDvR10]. The teleo-reactive approach [Nil94, BH04], as a typical design paradigm, specifies the agent as a set of prioritised reactive rules which continuously sense the environment and trigger agent actions whose effect will lead to the achievement of the goal. This teleo-reactive approach takes account of the environment's influence as well as the agent's goal. However, it does not provide a methodology to derive these rules from a high-level specification which only refers to the belief and desire of the agent. It is often a non-trivial task to justify the correctness of the design, *i.e.*, to prove that following these rules can result in the achievement of the goal. Furthermore, there is no theoretical criteria for judging whether one design is better than another. Our framework could support development from an abstract agent specification with declarative mental features to an agent implementation in the teleo-reactive paradigm. The priority of the action conditions corresponds to the priority of goal selection in our framework.

Another school of autonomous agent design employs declarative beliefs and desires in terms of a knowledge description language [HBHM01]. These mental features are maintained by the agents as mental states which evolve during the execution. An agent is then specified with a declarative specification focussing on the updates of its mental states. Hindriks [Hin09] proposes an agent programming language called "GOAL" based on this perspective. Hindriks and Riemsdijk [HR09] also introduce temporal logic formulas to integrate goals and qualitative preferences with agent programming. The agent is called rational if its selection of actions is determined by its current mental states. However, it is difficult to determine the correctness of an imperative implementation of an agent with respect to its declarative specification. Our framework does not introduce mental features as states but as constraints on the executions of the agent. An agent is considered as rational until it makes irrational choices which makes the goal unachievable.

Several formalisms have been developed to specify interactions between an agent and its environment. Alternating transition systems (ATS) proposed by Alur *et al.* [AHKV98] treat an agent and its environment as the opponents in a game. Agents choose their own transitions to update the current state and the final result is the intersection of their choices. The action-based alternating transition system studied by Atkinson *et al.* [ABC07] provides reasoning techniques to determine which action should be chosen by an agent in particular situations. Zhu [Zhu01] proposes a formal notation for specifying agent behaviour. The autonomous behaviour of the agent is formalised by a set of rules designed for various environmental scenarios. While these approaches are able to specify agent-environment interactions, they do not specify the behaviour of agents as being driven by its "mental" states (*e.g.*, desire).

Rao *et al.* [RG95] use a possible world model to interpret the semantics of BDI logic for autonomous agents. This is suitable for representing the belief, desire and intention of agents by assigning each of them a set of accessible worlds. Kaile Su *et al.* [SSW$^+$05, SYS$^+$06] propose an observation-based modal logic for BDI agents (OBDI logic). Its interpretation model associates the mental features of an agent to properties of computing paths. It facilitates model checking a BDI agent program with related mental properties. A multi-dimensional modal logic called QCLBDI is then proposed to reason about the evolution of agents' mental attitudes [CLSL14]. However, unlike our approach, their modal logics lack a theory to justify the correctness of the development of autonomous agents by introducing mechanisms to adapt to the environment.

Aştefănoaei and de Boer [AdB10] define a notion of refinement for BDI agents. This notion is based on trace inclusion where traces are sequences of actions. It is well known (specifically from the literature on CSP [Hoa85]) that such refinement is unable to detect potential deadlock in implementations. This is in contrast to the notion of refinement we have adopted from action systems, which is also based on trace inclusion but where traces are sequences of states. Furthermore, abstract and concrete specifications in [AdB10] are not in the same notation. Therefore, the approach allows only a single refinement step from an abstract to a concrete representation of an agent, not the incremental development of an agent.

The model of BDI agents in [AdB10] is also more restrictive. In particular, the goal of an agent is fixed. While they allow environmental hostility to be dealt with by changing plans, they do not allow the goal of an agent to be changed, nor the possibility of an agent not fulfilling its goal.

Other work on specifying agents using Z and Object-Z [dL97, GHKC02] has considered agent mental features but not refinement. Hence, there is no obvious way to relate such a specification to one where the mental states are captured implicitly via carefully engineered actions. There are recent approaches to refinement of agents in Object-Z and Event-B [SL14, LTG$^+$14]. Refinement in these approaches is also based

on action systems refinement and so is identical to that in this paper. However, these approaches have not considered refining mental features as part of the development process.

6. Conclusion

This paper specifies the behaviour of an autonomous agent from the view of restricting its standard action-based behaviour with mental features in terms of temporal properties. The framework specifies an agent in terms of three components: a state-transition system representing the capability of the agent; a sequence of goals in terms of LTL formulas denoting the agent's desire; and a belief-updating function capturing the agent's belief about the environment and its updates. Mental features such as belief and desire serve as a guide and motivation for the agent to make local choices among enabled actions. When the current mental feature is not appropriate, the agent acts in the standard manner (chooses enabled actions arbitrarily), which we call unmotivated behaviour. A refinement theory is applied to justify the development process of autonomous agents by reducing such unmotivated behaviours.

However, the inclusion of unmotivated behaviour is not monotonic with respect to the refinement relation. Hence, we introduce additional design obligations serving as a complement to the simulation rules for checking refinement. With the obligations, the correctness of the development of the agent can be proved step-by-step as long as it reduces the unmotivated behaviour. The following kinds of development strategies are justified by the refinement theory.

1. Refining the agent module with more deterministic decision-making protocols so that the agent will make the right choice leading to its current goal.
2. Introducing complementary goals so that the agent will have less chance to become unmotivated.
3. Improving the belief updating mechanism so that the agent will make a better choice based on a more precise belief.

There are three areas of future work we would like to pursue. Firstly, we would like to incorporate the modelling of mental states and their refinement into our work on multi-agent system development [SL14]. That approach is also based on Object-Z and action system refinement making such an integration relatively straightforward. This would allow us to use desires and beliefs to not only refine the behaviour of individual agents, but also, through their interactions, the global behaviour of the systems to which they belong.

Secondly, we would like to develop tool support for our approach. For the refinement checking of modules, this could be accomplished by a translation between Object-Z and Event-B to enabled the use of the Rodin toolkit [ABH+10]. The additional proof obligations required for desires and beliefs are properties over state traces, and hence the use of model checking to discharge them could be explored.

Finally, we would like to formally link our temporal logic descriptions of beliefs and desires to higher level descriptions used in the multi-agent systems community. This would provide an important link between the state-of-the-art design of agents, and the stepwise refinement of their designs to implementations.

Acknowledgement This work was supported by Australian Research Council (ARC) Discovery Grant DP110101211, National Science Foundation of China NSFC 61402176, and Science and Technology Commission of Shanghai Municipality Projects (No. 15ZR1410400 and No. 15511104700).

References

[ABC07] K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artif. Intell.*, 171(10-15):855–874, 2007.

[ABH+10] J.-R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 12(6):447–466, 2010.

[Abr10] J. R. Abrial. *Modelling in Event-B.* Cambridge University Press, 2010.

[AdB10] L. Aştefănoaei and F.S. de Boer. The refinement of multi-agent systems. In M. Dastani, K.V. Hindriks, and J.-J.C. Meyer, editors, *Specification and Verification of Multi-agent Systems*, chapter 2, pages 35–65. Springer-Verlag, 2010.

[AHKV98] R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *International Conference on Concurrency Theory (CONCUR '98)*, volume 1466 of *LNCS*, pages 163–178. Springer-Verlag, 1998.

[Bac90] R.-J.R. Back. Refinement calculus, part II: Parallel and reactive programs. In *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, pages 67–93. Springer, 1990.

[BFVW06] R.H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous agents and multi-agent systems*, 12(2):239–256, 2006.

[BH04] K. Broda and C.J. Hogger. Designing and simulating individual teleo-reactive agents. In *Annual German Conference on Artificial Intelligence (KI 2004)*, volume 3238 of *Lecture Notes in Artificial Intelligence*, pages 1–15. Springer-Verlag, 2004.

[CLSL14] Q. Chen, Q. Li, K. Su, and X. Luo. Quantified coalition logic for BDI-Agents: Completeness and complexity. In D.-N. Pham and S.-B. Park, editors, *Pacific Rim International Conference on Artificial Intelligence (PRICAI 2014)*, volume 8862 of *Lecture Notes in Computer Science*, pages 871–876. Springer-Verlag, 2014.

[DB14] J. Derrick and E. Boiten. *Refinement in Z and Object-Z, Foundations and Advanced Applications*. Springer-Verlag, 2nd edition, 2014.

[dL97] M. d'Inverno and M. Luck. Development and application of a formal agent framework. In *International Conference on Formal Engineering Methods (ICFEM '97)*, pages 222–231. IEEE Press, 1997.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072. Elsevier, 1990.

[FBO10] M.S. Fagundes, H. Billhardt, and S. Ossowski. Reasoning about norm compliance with rational agents. In *European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, pages 1027–1028, 2010.

[Fis11] M. Fisher. Agent deliberation in an executable temporal framework. *Journal of Applied Logic*, 9(4):223–238, 2011.

[GHKC02] P. Gruer, V. Hilaire, A. Koukam, and K. Cetnarowicz. A formal framework for multi-agent systems analysis and design. *Expert System Applications*, 23(4):349–355, 2002.

[HBHM01] K.V. Hindriks, F.S. de Boer, W. Hoek, and J.C. Meyer. Agent programming with declarative goals. In *International Workshop on Intelligent Agents VII: Agent Theories Architectures and Languages (ATAL '00)*, pages 228–243. Springer-Verlag, 2001.

[Hin09] K.V. Hindriks. Programming rational agents in GOAL. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-agent programming: Languages, platforms and applications*, volume 2, chapter 4, pages 119–157. Springer-Verlag, 2009.

[HMTYS14] J. Harland, D.N. Morley, J. Thangarajah, and N. Yorke-Smith. An operational semantics for the goal life-cycle in bdi agents. *Autonomous agents and multi-agent systems*, 28(4):682–719, 2014.

[Hoa85] C.A.R. Hoare. *Communicating sequential processes*. Prentice Hall, 1985.

[HR09] K.V. Hindriks and M.B. Riemsdijk. Using temporal logic to integrate goals and qualitative preferences into agent programming. In M. Baldoni, T. Son, M.B. Riemsdijk, and M. Winikoff, editors, *Declarative Agent Languages and Technologies VI (DALT 2008)*, volume 5397 of *Lecture Notes in Computer Science*, pages 215–232. Springer-Verlag, 2009.

[LS13] Q. Li and G. Smith. A refinement framework for autonomous agents. In J. Iyoda and L. de Moura, editors, *Brazilian Symposium on Formal Methods (SBMF 2013)*, volume 8195 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 2013.

[LTG+14] L. Laibinis, E. Troubitsyna, Z. Graja, F. Migeon, and A. Hadj Kacem. Formal modelling and verification of cooperative ant behaviour in Event-B. In D. Giannakopoulou and G. Salaün, editors, *Software Engineering and Formal Methods (SEFM 2014)*, volume 8702 of *Lecture Notes in Computer Science*, pages 363–377. Springer-Verlag, 2014.

[MB02] Á.F. Moreira and R.H. Bordini. An operational semantics for a bdi agent-oriented programming language. In *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02), held in conjunction with the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), April*, pages 45–59, 2002.

[MBH15] J.-J. Meyer, J.M. Broersen, and A. Herzig. BDI logics. pages 453–498, 2015.

[Mey14] J.-J. Meyer. Logics for intelligent agents and multi-agent systems. 2014.

[Nil94] N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[RG95] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *1st International Conference of Multi-agent Systems (ICMAS-95)*, pages 312–319. MIT Press, 1995.

[RL07] F. Raimondi and A. Lomuscio. Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *Journal of Applied Logic*, 5(2):235–251, 2007.

[SL14] G. Smith and Q. Li. MAZE: An extension of Object-Z for multi-agent systems. In Y. Ait Ameur and K.-D. Schewe, editors, *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2014)*, volume 8477 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, 2014.

[Smi00] G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publishers, 2000.

[SSW+05] K. Su, A. Sattar, K. Wang, X. Luo, G. Governatori, and V. Padmanabhan. Observation-based model for BDI-Agents. In M.M. Veloso and S. Kambhampati, editors, *National Conference on Artificial Intelligence (AAAI 2005)*, pages 190–195. AAAI Press, 2005.

[SW12] G. Smith and K. Winter. Incremental development of multi-agent systems in Object-Z. In *IEEE Software Engineering Workshop, (SEW 2012)*, pages 120–129. IEEE Press, 2012.

[SYS+06] K. Su, W. Yue, A. Sattar, M.A. Orgun, and X. Luo. Observation-based logic of knowledge, belief, desire and intention. In J. Lang, F. Lin, and J. Wang, editors, *Knowledge Science, Engineering and Management (KSEM 2006)*, volume 4092 of *Lecture Notes in Computer Science*, pages 366–378. Springer-Verlag, 2006.

[WDvR10] M. Winikoff, M. Dastani, and M.B. van Riemsdijk. A unified interaction-aware goal framework. In *European Conference on Artificial Intelligence (ECAI 2010)*, pages 1033–1034. IOS Press, 2010.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, **10**:115–152, 1995.

[Wob15] W. Wobcke. A logic of intention and action for regular bdi agents based on bisimulation of agent programs. *Autonomous Agents and Multi-Agent Systems*, 29(4):569–620, 2015.

[Woo09] M. Wooldridge. *An Introduction to MultiAgent Systems (2nd Edition)*. Wiley, 2009.

[Zhu01] H. Zhu. Formal specification of agent behaviour through environment scenarios. In J.L. Rash, W. Truszkowski, M.G. Hinchey, C.A. Rouff, and D. Gordon, editors, *Formal Approaches to Agent-Based Systems (FAABS 2000)*, volume 1871 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, 2001.