Resource-Bounded KARO

Author:
Tim Harteveld

Supervisor:
prof. dr. J-J.Ch. Meyer

June 8, 2017
45 ECTS

Abstract:

Usually, a logic is not concerned with computing power. This is a problem for logic-based agents that need to react fast to their environment. Especially in complex situations and/or when the agent has little computing power. The agent must also take into account the time it takes to complete an action. If the agent needs to change its environment fast it is not useful to react with an action that takes a long time to complete. Thus there is a constraint on the time a reaction takes to complete and the time it takes to find a suitable reaction.
In the KARO framework agents can plan a sequence of actions that will fulfil the goals of the agent. The KARO framework is extended to take into account the two types of time constraints. A goal of the agent must be fulfilled before a certain point in time. The resource-bounded KARO framework doesn't allow action sequences that finish after the end time of a goal. It is also not allowed to reason about the right action sequence for so long that the found action sequence cannot be finished in time. The KARO framework is also extended with parallel actions. A goal can be reached faster if the actions to reach the goal can be performed in parallel.

Contents

1 Introduction

Agents perform actions to change their environment. The agents must perform the right actions in order to achieve their goals. The actions must also be performed in the right order to have the desired effect. KARO allows agents to plan action sequences in order to achieve their goals. The framework is introduced in a series of papers. In the basic framework [6,11] the knowledge, abilities and opportunities of the agent are formalised as well as the results of the actions performed by the agent. Other papers introduce concepts like observations [7, 8], default reasoning [10] and belief revision [9].

There is often a time constraint for achieving a goal. When dealing with these time constraints there are two things the agent needs to take into account. Not only does performing the action takes time but also finding a suitable sequence of actions to achieve the goal. In this paper, the KARO framework is extended to take into account these two time constraints. The goal is to allow an agent to plan and perform actions to achieve its goal before a certain time. The KARO framework is also extended with parallel actions. A goal can be reached faster if the actions to reach the goal can be performed in parallel.

The KARO framework is reintroduced in section 2. In section 3 the KARO logic is extended so that actions take a certain amount of time and wishes are fulfilled before a certain point in time. In section 4 the KARO logic is extended with parallel actions. And in section 5 the logic is extended to also take into account the time the agent needs for finding solutions.

2 KARO

The KARO framework allows agents to plan action sequences to achieve their goals. The agent has the ability to do certain actions and depending on the environment has the opportunity to execute them. The agent also has knowledge about the state of its environment and the effect actions have on it. The following definition is based on the definition given in [4].

2.1 Syntax

The syntax of KARO is defined with two languages, $\mathcal{L}$ and $\mathcal{L}^{\mathcal{C}}$. This is to prevent the agents from applying higher level operations on themselves. This prevents statements like: I wish to wish to commit myself to action α and it is implementable to select committing to action α. The higher level operations in $\mathcal{L}^{\mathcal{C}}$ can only be applied to statements in $\mathcal{L}$.

2.1.1 Language $\mathcal{L}$

The language $\mathcal{L}$ is based on propositional logic and is extended with modal operators for knowledge, the opportunity to do actions and the capability to do actions. If $\mathrm{p} \in \Pi$ where Π is the set of propositional atoms, $i \in \mathrm{A}$ where A is the set of agents and $\alpha \in \mathrm{Ac}$ where Ac is a set of actions defined later, then the following grammar defines the language $\mathcal{L}$:

$$\varphi, \psi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{K}_i\varphi \mid \mathbf{A}_i\alpha \mid \langle \mathrm{do}_i(\alpha) \rangle \varphi$$

The operators for negation, conjunction and disjunction have their standard meaning. The epistemic modal operator $\mathbf{K}_i\varphi$ indicates that agent i knows φ. $\mathbf{A}_i\alpha$ denotes the capability of agent i to perform the action α. $\langle \mathrm{do}_i(\alpha) \rangle \varphi$ is the notation used in dynamic logic [2] for the effect of doing action α. It denotes that when agent i does action α it can terminate with φ holding. $[\mathrm{do}_i(\alpha)]\varphi$ is defined to be $\neg \langle \mathrm{do}_i(\alpha) \rangle \neg\varphi$, it denotes that the agent is guaranteed to ends up in an state where φ holds afther executing action α. This is different from $\langle \mathrm{do}_i(\alpha) \rangle \varphi$ where the agent is guaranteed to have an opportunity to do α but not the guarantee that this will result in a state where φ holds. The tautology $\top$ is defined as $\mathrm{p} \vee \neg\mathrm{p}$ and the contradiction $\bot$ is defined as $\mathrm{p} \wedge \neg\mathrm{p}$. The implication operator $\varphi \rightarrow \psi$ is defined as $\neg\varphi \vee \psi$.

If a ∈ At where At is the set of atomic actions and $\varphi \in \mathcal{L}$, then the following grammar defines the actions that are in the set Ac:

$$\alpha, \beta ::= \mathrm{a} \mid \varphi? \mid \alpha; \beta \mid \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta \texttt{ fi} \mid \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od}$$

The action $\varphi?$ verifies whether φ holds. The next action is executed if φ holds. Execution fails if φ doesn't hold, in that case the action results in a failure state. The action $\alpha;\beta$ is composed of the actions α and β which are done in sequence. The action `if` φ `then` α `else` β `fi` executes α if φ holds and executes β otherwise. The action `while` φ `do` α `od` executes α as long as φ holds. The empty action Λ is an action that doesn't do anything, it has the property $\Lambda;\alpha = \alpha;\Lambda = \alpha$.

$$\begin{aligned}
\mathbf{O}_i\alpha &=^{\text{def}} \langle \mathrm{do}_i(\alpha)\rangle \top && \text{Def. 2.1}\\
\mathbf{PracPoss}_i(\alpha,\varphi) &=^{\text{def}} \langle \mathrm{do}_i(\alpha)\rangle \varphi \wedge \mathbf{A}_i\alpha && \text{Def. 2.2}\\
\mathbf{Can}_i(\alpha,\varphi) &=^{\text{def}} \mathbf{K}_i\mathbf{PracPoss}_i(\alpha,\varphi) && \text{Def. 2.3}\\
\mathbf{Cannot}_i(\alpha,\varphi) &=^{\text{def}} \mathbf{K}_i\neg\mathbf{PracPoss}_i(\alpha,\varphi) && \text{Def. 2.4}
\end{aligned}$$

It is practically possible for an agent i to make φ true by performing the action α if the agent has the opportunity and the capability to do action α and doing the actions α results in φ being true. An agent i can do an action if it knows that it is practically possible. The agent cannot do an action if it knows that it is not practically possible. $\mathbf{O}_i\alpha$ denotes that agent i has the opportunity to do action α. The ability of an agent describes what actions the agent can perform and the opportunity describes in what situation the agent can perform the actions.

In this paper it is assumed that all actions are deterministic. This guarantees that the action α results in a state where φ holds, making $\langle \mathrm{do}_i(\alpha)\rangle \varphi$ stronger than $[\mathrm{do}_i(\alpha)]\varphi$. If actions are deterministic it holds that $\langle \mathrm{do}_i(\alpha)\rangle \varphi \leftrightarrow [\mathrm{do}_i(\alpha)]\varphi \wedge \mathbf{O}_i\alpha$.

2.1.2 Language $\mathcal{L}^{\mathcal{C}}$

The language $\mathcal{L}^{\mathcal{C}}$ has the same operators as $\mathcal{L}$ and has four additional higher level operators. If $\mathrm{p} \in \Pi$, $i \in \mathrm{A}$, $\alpha \in \mathrm{Ac}^{\mathcal{C}}$ where $\mathrm{Ac}^{\mathcal{C}}$ is a set of actions defined later, $\beta \in \mathrm{Ac}$ and $\phi \in \mathcal{L}$, then the following grammar defines the language $\mathcal{L}^{\mathcal{C}}$:

$$\varphi, \psi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{K}_i\varphi \mid \mathbf{A}_i\alpha \mid \langle \mathrm{do}_i(\alpha)\rangle \varphi \mid \mathbf{W}_i\phi \mid \Diamond_i\phi \mid \mathbf{C}_i\phi \mid \mathbf{Com}_i\beta$$

$\mathbf{W}_i\phi$ is the wish operator, it indicates the desire of agent i to make ϕ true. $\Diamond_i\phi$ is the implementability operator, indicating that ϕ is implementable. $\mathbf{C}_i\phi$ is the selection operator. $\mathbf{Com}_i\beta$ is the commitment operator indicating the actions agent i has committed itself to doing β. The definitions in $\mathcal{L}^{\mathcal{C}}$ for tautology, contradiction, implication, $[\mathrm{do}_i(\alpha)]\varphi$, practically possible, can and cannot are the same as in $\mathcal{L}$.

The set of actions $\mathrm{Ac}^{\mathcal{C}}$ is a superset of Ac. If a ∈ At, $\varphi \in \mathcal{L}$ and $\gamma \in$ Ac, then the following grammar defines the actions that are in the set $\mathrm{Ac}^{\mathcal{C}}$:

$$\alpha, \beta ::= \mathrm{a} \mid \varphi? \mid \alpha;\beta \mid \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta \texttt{ fi} \mid \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od} \mid \texttt{select}\varphi \mid \texttt{commit}\gamma \mid \texttt{uncommit}\gamma$$

`select`φ is an action that adds φ to the selected wishes in $\mathbf{C}$. `commit`γ is an action that commits the agent to do the action γ. `uncommit`γ is an action that uncommits the agent to do the action γ. The other actions have the same meaning as in Ac. The set of semi-atomic actions At^+ is the combination of all the atomic actions and verify actions, $At^+ = At \cup \{\varphi? \mid \varphi \in \mathcal{L}\}$.

$$\begin{aligned}
\mathbf{Goal}_i\varphi &=^{\text{def}} \mathbf{W}_i\varphi \wedge \neg\varphi \wedge \Diamond_i\varphi \wedge \mathbf{C}_i\varphi && \text{Def. 2.5}\\
\mathbf{PossIntend}_i(\alpha,\varphi) &=^{\text{def}} \mathbf{Can}_i(\alpha,\varphi) \wedge \mathbf{K}_i\mathbf{Goal}_i\varphi && \text{Def. 2.6}
\end{aligned}$$

The goal operator $\mathbf{Goal}_i\varphi$ indicates that the agent i has the goal of reaching a state in which φ holds. An agent i can have a goal φ if it has the wish to bring about φ, φ doesn't already hold, φ is implementable and selected. $\mathbf{PossIntend}_i(\alpha, \varphi)$ is the possibly intends operator, it indicates that agent i has the possible intention to do action α in order to bring about φ. The agent can intend to do α to bring about φ if it can do the action α to bring about φ and it knows that it has the goal of bringing about φ.

2.2 Semantics

2.2.1 Language $\mathcal{L}$

The language $\mathcal{L}$ is formally interpreted as a Kripke model. The class $\mathbf{M}$ of models contains all $M = \langle S, \pi, R, r_0, c_0\rangle$. S is a non-empty set of possible world states. $\pi : \Pi \times S \to \{0, 1\}$ is the function that assigns a truth value to propositional symbols Π in state $s \in S$. $R : A \to \mathcal{P}(S \times S)$ is the function that gives the epistemic accessibility relation for every agent. R returns for agent $i \in A$ a set of relations between states in S. The relation must be **S5**, thus reflexive and euclidean. The $R(i)$-equivalence class $[s]_{R(i)}$ is defined as $\{s' \in S | (s, s') \in R(i)\}$. $r_0 : A \times \mathrm{At} \to (S \cup \{\varepsilon\}) \to \mathcal{P}(M, S)$ gives the set of states agent $i \in A$ can be in after the state transition resulting from preforming the action $\alpha \in \mathrm{At}$ in the state $s \in S$. There is a special case where the function returns an empty set as a result of an impossible action. In that case the agent will go to a special error state ε. It is not possible to preform actions from the failure state ε. It is assumed in this paper that actions are deterministic, thus the resulting set of function r_0 will always contain just one state. $c_0 : A \times \mathrm{At} \to (S \cup \{\varepsilon\}) \to \{0, 1\}$ is the capability function. c_0 assigns a truth value to the action $\alpha \in \mathrm{At}$ performed by agent $i \in A$ in the state $s \in S$. This indicates whether agent i is capable of preforming action α in state s.

Definition 2.7. *$(M, s) \models \varphi$ denotes that φ is true in the state $s \in S$ and in the model $M \in \boldsymbol{M}$. This is defined in the following way, where $\alpha \in Ac$:*

$$
\begin{aligned}
M, s &\models \varphi && \Leftrightarrow \pi(s)(\varphi) = 1 \\
M, s &\models \neg\varphi && \Leftrightarrow M, s \not\models \varphi \\
M, s &\models \varphi \wedge \psi && \Leftrightarrow M, s \models \varphi \text{ and } M, s \models \psi \\
M, s &\models \varphi \vee \psi && \Leftrightarrow M, s \models \varphi \text{ or } M, s \models \psi \\
M, s &\models \boldsymbol{K}_i\varphi && \Leftrightarrow \forall s' \in S((s, s') \in R(i) \Rightarrow M, s' \models \varphi) \\
M, s &\models \langle do_i(\alpha)\rangle\, \varphi && \Leftrightarrow \exists M', s'(M', s' \in r(i, \alpha)(M, s) \wedge M', s' \models \varphi) \\
M, s &\models \boldsymbol{A}_i\alpha && \Leftrightarrow c(i, \alpha)(s) = 1
\end{aligned}
$$

Definition 2.8. *The result function r is defined in the following way, where* $\mathrm{a} \in At$ *and* $r(i, \alpha)(M, \varepsilon) = M, \varepsilon$.

$$
\begin{aligned}
&r(i, \mathrm{a})(M, s) && = r_0(i, \mathrm{a})(M, s) \\
&r(i, \varphi?)(M, s) && = (M, s) \text{ if } M, s \models \varphi \text{ otherwise } (M, \varepsilon) \\
&r(i, \alpha_1; \alpha_2)(M, s) && = r(i, \alpha_2)(r(i, \alpha_1)(M, s)) \\
&r(i, \texttt{if } \varphi \texttt{ then } \alpha_1 \texttt{ else } \alpha_2 \texttt{ fi})(M, s) && = r(i, \alpha_1)(M, s) \text{ if } M, s \models \varphi \text{ and} \\
& && \quad r(i, \alpha_2)(M, s) \text{ otherwise} \\
&r(i, \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od})(M, s) && = (M', s') \text{ such that } \exists k \in \mathbb{N}\ \exists (M_0, s_0) \ldots \exists (M_k, s_k) \\
& && \quad ((M_0, s_0) = (M, s) \text{ and } (M_k, s_k) = (M', s') \text{ and} \\
& && \quad \forall j < k((M_{j+1}, s_{j+1}) = r(i, \varphi?; \alpha)(M_j, s_j)) \text{ and} \\
& && \quad M', s' \models \neg\varphi)
\end{aligned}
$$

Definition 2.9. *The capability function c is defined in the following way where* $\mathrm{a} \in At$ *and* $c(i,\alpha)(M,\varepsilon) = 0$.

$$
\begin{aligned}
&c(i,\mathrm{a})(M,s) && = c_0(i,\mathrm{a})(s)\\
&c(i,\varphi?)(M,s) && = 1 \textit{ if } M,s \models \varphi \textit{ and } 0 \textit{ otherwise}\\
&c(i,\alpha_1;\alpha_2)(M,s) && = c(i,\alpha_1)(M,s) \textit{ and } c(i,\alpha_2)(r(i,\alpha_1)(M,s))\\
&c(i,\texttt{if } \varphi \texttt{ then } \alpha_1 \texttt{ else } \alpha_2 \texttt{ fi})(M,s) && = c(i,\varphi?;\alpha_1)(M,s) \textit{ or } c(i,\neg\varphi?;\alpha_2)(M,s)\\
&c(i,\texttt{while } \varphi \texttt{ do } \alpha \texttt{ od})(M,s) && = 1 \textit{ if for some } k \in \mathbb{N} c(i,(\varphi?;\alpha)^k;\neg\varphi?)(M,s)\\
& && \quad \textit{and } 0 \textit{ otherwise}
\end{aligned}
$$

2.2.2 Language $\mathcal{L}^{\mathcal{C}}$

The language $\mathcal{L}^{\mathcal{C}}$ is formally interpreted as a Kripke model in a similar way to language $\mathcal{L}$. The class $\mathbf{M}^{\mathcal{C}}$ of models contains all $M = \langle S, \pi, R, r_0, c_0, W, C, \mathrm{Agenda}\rangle$. The set S and the functions π, R, r_0 and c_0 are the same as in the model for Language $\mathcal{L}$. $W : A \to \mathcal{P}(S \times S)$ is a function that gives the desirability relation for every agent. W returns for agent $i \in A$ a set of relations between states in S. $C : A \times S \to \mathcal{P}(\mathcal{L})$ is a function that for a given agent $i \in A$ and state $s \in S$ returns a set of expressions from the language $\mathcal{L}$ that represent the choices made by i in s. Agenda : $A \times S \to \mathcal{P}(Ac)$ is a function that for a given agent $i \in A$ and state $s \in S$ returns the set of actions i is committed to in s.

Before the semantics of language $\mathcal{L}^{\mathcal{C}}$ can be given, a transition system is needed. This is mainly needed for defining the `commit` action. The Structural Operational Semantics method [5] is used for this. A transition is denoted by $\langle \alpha, s\rangle \rightharpoonup^{M}_{i,\mathrm{a}^+} \langle \alpha', s'\rangle$ where $\alpha, \alpha' \in Ac$, $i \in A$, $\mathrm{a}^+ \in \mathrm{At}^+$ with $\varphi \in \mathcal{L}$ and $s, s' \in S$. This transition denotes that in state s, agent i has to perform the action α and after performing action a^+ the agent is in state s' and still has to do action α'. The state resulting from doing action a^+ state in state s can be denoted with s_{a^+}.

Definition 2.10. *If the model $M \in \boldsymbol{M}^{\mathcal{C}}$, $\alpha, \beta \in Ac$, $s \in S$ and the function π_2 simply returns the second element of a pair, in this case the state s of the model state pair (M,s), then is the transition system T_M is given with the following axioms:*

$$
\begin{aligned}
&\langle \alpha, s\rangle \rightharpoonup^{M}_{i,\alpha} \langle \Lambda, s'\rangle \textit{ with } s' = \pi_2(r^{\mathcal{C}}(i,\alpha)(M,s)) \textit{ if } \alpha \in At^+ \textit{ and } r^{\mathcal{C}}(i,\alpha)(M,s) \neq \emptyset\\
&\langle \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta \texttt{ fi}, s\rangle \rightharpoonup^{M}_{i,\varphi?} \langle \alpha, s_{\varphi?}\rangle \textit{ if } s \models \varphi\\
&\langle \texttt{if } \varphi \texttt{ then } \alpha \texttt{ else } \beta \texttt{ fi}, s\rangle \rightharpoonup^{M}_{i,\neg\varphi?} \langle \beta, s_{\neg\varphi?}\rangle \textit{ if } s \not\models \varphi\\
&\langle \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od}, s\rangle \rightharpoonup^{M}_{i,\varphi?} \langle \alpha; \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od}, s_{\varphi?}\rangle \textit{ if } s \models \varphi\\
&\langle \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od}, s\rangle \rightharpoonup^{M}_{i,\neg\varphi?} \langle \Lambda, s_{\neg\varphi?}\rangle \textit{ if } s \not\models \varphi
\end{aligned}
$$

And the rule:

$$\frac{\langle \alpha, s\rangle \rightharpoonup^{M}_{i,\mathrm{a}^+} \langle \alpha', s'\rangle}{\langle \alpha;\beta, s\rangle \rightharpoonup^{M}_{i,\mathrm{a}^+} \langle \alpha';\beta, s'\rangle}$$

Definition 2.11. *A computation run is the sequence of semi-atomic actions needed to perform an action. If $M \in \boldsymbol{M}^{\mathcal{C}}$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in Ac$, $\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_m \in At^+$ and $s_1, \ldots, s_n \in S$, then computation run is defined as:*

$$
\begin{aligned}
CR^{\mathcal{C}}_M(i,\alpha,s) = \{\mathrm{a}_1;\mathrm{a}_2;\ldots;\mathrm{a}_n\} \textit{ iff } &\langle \alpha, s\rangle \rightharpoonup^{M}_{i,\mathrm{a}_1} \langle \alpha_1, s_1\rangle \rightharpoonup^{M}_{i,\mathrm{a}_2} \langle \alpha_2, s_2\rangle\\
&\rightharpoonup^{M}_{i,\mathrm{a}_3} \cdots \rightharpoonup^{M}_{i,\mathrm{a}_n} \langle \alpha_n, s_n\rangle \textit{ such that } \alpha_n = \Lambda
\end{aligned}
$$

Definition 2.12. *The relation $\models^{\mathcal{C}}$ contains all relations $\models$ of language $\mathcal{L}$. It also contains relations for $\boldsymbol{W}_i\varphi$, $\Diamond_i\varphi$, $\boldsymbol{C}_i\varphi$ and $\boldsymbol{Com}_i\alpha$. These are defined in the following way, where $M \in \boldsymbol{M}^{\mathcal{C}}$, $s \in S$ and Prefix(α, β) indicate that action sequence α is a prefix of action sequence β.*

$$
\begin{aligned}
M, s \models^{\mathcal{C}} \varphi \quad & \Leftrightarrow M, s \models \varphi \\
M, s \models^{\mathcal{C}} \boldsymbol{W}_i\varphi \quad & \Leftrightarrow \forall s' \in S((s, s') \in W(i) \Rightarrow M, s' \models^{\mathcal{C}} \varphi) \\
M, s \models^{\mathcal{C}} \Diamond_i\varphi \quad & \Leftrightarrow \exists k \exists \alpha_1, \ldots, \alpha_k \in At(M, s \models^{\mathcal{C}} \boldsymbol{PracPoss}_i(\alpha_1; \ldots; \alpha_k, \varphi)) \\
M, s \models^{\mathcal{C}} \boldsymbol{C}_i\varphi \quad & \Leftrightarrow \varphi \in C(i, s) \\
M, s \models^{\mathcal{C}} \boldsymbol{Com}_i\alpha \quad & \Leftrightarrow \forall s' \in [s]_{R(i)} \exists \alpha_1 \in CR^{\mathcal{C}}_M(i, \alpha, s') \exists \alpha_2 \in Agenda(i, s') \\
& \qquad \exists \alpha_2' \in CR^{\mathcal{C}}_M(i, \alpha_2, s')(Prefix(\alpha_1, \alpha_2'))
\end{aligned}
$$

An agent $i \in A$ has the wish to fulfil φ, $\mathbf{W}_i\varphi$, if φ holds in all the states reachable from state s with the desirability relation $W(i)$. φ is implementable for agent $i \in A$, $\Diamond_i\varphi$, if there is a sequence of actions that make it practically possible to make φ true. Thus the agent has the opportunity and the capability to make φ true. The agent $i \in A$ has selected φ, $\mathbf{C}_i\varphi$, if φ is in the set of selected actions given by the function $C(i, s)$ for the state s. An agent i is committed to action α, $\mathbf{Com}_i\alpha$, if the computation run of the action is a prefix of a computation run of an action that is in the agent's Agenda. Thus the agent is committed to the first part of an action that is in the agenda, for example, if $\alpha; \beta; \gamma \in Agenda(i, s)$ then agent i is committed to the action α. This must hold in all the states reachable with the $R(i)$-equivalence class $[s]_{R(i)}$ so that if the agent is committed to an action it also knows that it is committed to the action.

Definition 2.13. *If $M = \langle S, \pi, R, r_0, c_0, W, C, Agenda\rangle$ then*

$$
\begin{aligned}
r^{\mathcal{C}}(i, \alpha)(M, s) \quad & = r(i, \alpha)(M, s) \\
r^{\mathcal{C}}(i, \texttt{select}\ \varphi)(M, s) \quad & = (\langle S, \pi, R, r_0, c_0, W, \texttt{choose}(i, \varphi)(M, s), Agenda\rangle, s) \\
& \quad \textit{if } M, s \models^{\mathcal{C}} \boldsymbol{W}_i\varphi \textit{ and } \varepsilon \textit{ otherwise} \\
r^{\mathcal{C}}(i, \texttt{commit}\ \alpha)(M, s) \quad & = (\langle S, \pi, R, r_0, c_0, W, C, \texttt{agenda}^{+}(i, \alpha)(M, s)\rangle, s) \\
& \quad \textit{if } M, s \models^{\mathcal{C}} \boldsymbol{PossIntend}_i(\alpha, \varphi) \textit{ for some } \varphi \in \boldsymbol{C}(i, s) \\
& \quad \textit{and } \varepsilon \textit{ otherwise} \\
r^{\mathcal{C}}(i, \texttt{uncommit}\ \alpha)(M, s) & = (\langle S, \pi, R, r_0, c_0, W, C, \texttt{agenda}^{-}(i, \alpha)(M, s)\rangle, s) \\
& \quad \textit{if } M, s \models^{\mathcal{C}} \boldsymbol{Com}_i\alpha \textit{ and } \varepsilon \textit{ otherwise}
\end{aligned}
$$

When performing the action $\texttt{select}\varphi$ the agent i reaches a new model where φ is added to the selected actions if i wishes φ to be true. If agent i does not wish for φ to be true the action $\texttt{select}\varphi$ result in the error state. The result of the action $\texttt{commit}\alpha$ is that the agenda is updated with the action α if i possibly intends to do α to make some selected proposition true. If the agent does not possibly intend action α to make the selected propositions true the action $\texttt{commit}\alpha$ results in the error state. The result of the action $\texttt{uncommit}\alpha$ is that if i was committed to α it is removed from its agenda. If the agent is not committed to α the action $\texttt{uncommit}\alpha$ results in the error state.

Definition 2.14. *If C is the current choice function in $M^{\mathcal{C}}$, then:*

$$
\begin{aligned}
\texttt{choose}(i, \varphi)(M, s) = C' \textit{ where } C'(i', s') = C(i', s') \textit{ if } i \neq i' \textit{ or } s \neq s' \\
\textit{and } C'(i, s) = C(i, s) \cup \{\varphi\}
\end{aligned}
$$

$\texttt{choose}(i, \varphi)(M, s)$ returns a choice function where φ is added to the result of the function C for the agent i and the state s, but the result stays the same for the other agents and states. The result of the action $\texttt{select}\varphi$ is thus that φ is added to the selected propositions if φ is a wish of agent i.

Definition 2.15. *If Agenda is the current agenda in $M^{\mathcal{C}}$, then:*

$$\begin{aligned}\texttt{agenda}^{+}(i,\alpha)(M,s) = \mathit{Agenda}' \ &\mathit{where\ for\ all}\ s' \in [s]_{R(i)}, \mathit{Agenda}'(i,s') = \\ &\mathit{Agenda}(i,s') \cup \{\alpha\}\ \mathit{and\ for\ all}\ s', s'', s''' \in S, \\ &\alpha' \in \mathit{Agenda}'(i,s')\ \mathit{such\ that,for\ some\ semi\text{-}atomic} \\ &\mathit{action}\ \mathrm{a}, \langle \alpha', s' \rangle \rightharpoonup_{i,\mathrm{a}} \langle \alpha'', s'' \rangle\ \mathit{and}\ s''' \in [s'']_{R(i)}, \\ &\mathit{Agenda}'(i,s''') = \mathit{Agenda}(i,s''') \cup \{\alpha''\}\end{aligned}$$

$\texttt{agenda}^{+}(i,\alpha)(M,s)$ returns an agenda that is updated with α. Not only the agenda corresponding to the agent's current state s is updated but also all the states that are epistemically equivalent with s. Thus the commitment to α is known by the agent i. The agenda is also updated in the states that are visited when α is executed. The agenda is then updated with the remainder of the action α and also all states that are epistemically equivalent are updated. Figure 1 is an example showing the propositions that are added to the agenda returned by the function $\texttt{agenda}^{+}$.

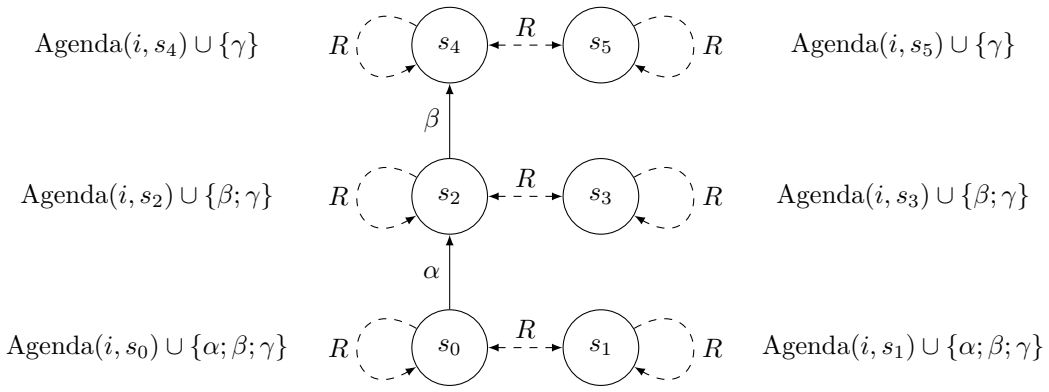


Figure 1: The changed agenda's in the model that is return by $\texttt{agenda}^{+}(i,\alpha;\beta;\gamma)(M,s_0)$

Definition 2.16. *If Agenda is the current agenda in $M^{\mathcal{C}}$, then:*

$$\begin{aligned}\texttt{agenda}^{-}(i,\alpha)(M,s) = \mathit{Agenda}' \ &\mathit{where\ for\ all}\ s' \in [s]_{R(i)}, \mathit{Agenda}'(i,s') = \\ &\mathit{Agenda}(i,s') \setminus \left\{\beta \mid \mathit{Prefix}(CR^{\mathcal{C}}_M(i,\alpha,s'), CR^{\mathcal{C}}_M(i,\beta,s'))\right\} \\ &\mathit{and\ for\ all}\ s', s'', s''' \in S, \alpha' \in \mathit{Agenda}'(i,s')\ \mathit{such\ that,} \\ &\mathit{for\ some\ semi\text{-}atomic\ action}\ \mathrm{a}, \langle \alpha', s' \rangle \rightharpoonup_{i,\mathrm{a}} \langle \alpha'', s'' \rangle \\ &\mathit{and}\ s''' \in [s'']_{R(i)}, \mathit{Agenda}'(i,s''') = \mathit{Agenda}(i,s''') \setminus \\ &\left\{\beta \mid \mathit{Prefix}(CR^{\mathcal{C}}_M(i,\alpha'',s'''), CR^{\mathcal{C}}_M(i,\beta,s'''))\right\}\end{aligned}$$

$\texttt{agenda}^{-}(i,\alpha)(M,s)$ returns an agenda where α is removed. Not only the agenda corresponding to the agent's current state s is updated but also all the states that are epistemically equivalent with s. Thus the commitment to α is no longer known by i. The agenda is also updated in the states that are visited when α is executed. The remained of the action α is removed from the agenda and also from the agenda of all states that are epistemically equivalent.

Definition 2.17. *The capability function $c^{\mathcal{C}}$ contains all functions c*

$$
\begin{aligned}
c^{\mathcal{C}}(i,\alpha)(M,s) &= c(i,\alpha)(M,s)\\
c^{\mathcal{C}}(i,\texttt{select}\ \varphi)(M,s) &= 1 \textit{ if } M,s \models^{\mathcal{C}} \neg\varphi \wedge \Diamond_i\varphi \textit{ and } 0 \textit{ otherwise}\\
c^{\mathcal{C}}(i,\texttt{commit}\ \alpha)(M,s) &= 1 \textit{ if } Agenda(i,s) = \emptyset \textit{ and } 0 \textit{ otherwise}\\
c^{\mathcal{C}}(i,\texttt{uncommit}\ \alpha)(M,s) &= 1 \textit{ if } M,s \models^{\mathcal{C}} \neg\boldsymbol{PossIntend}_i(\alpha,\varphi) \textit{ for all } \varphi \in \boldsymbol{C}(i,s)\\
&\quad \textit{and } 0 \textit{ otherwise}
\end{aligned}
$$

The agent i is capable to perform the $\texttt{select}\varphi$ action if φ doesn't already hold and φ is implementable for agent i. The agent is capable to perform the $\texttt{commit}\alpha$ action if the Agenda is empty. The agent i is capable to perform the $\texttt{uncommit}\alpha$ action if i does not possibly intend to do α to make one of the selected propositions true.

2.3 Block World Example

The blocks world has different sized blocks which can be stacked on top of each other, as given in [4]. This world provides a simple environment for KARO to plan in. The blocks in the example come in three different sizes. Blocks of type A are bigger then blocks of type B, blocks of type B are bigger then blocks of type C and blocks of type A are bigger then blocks of type C. Blocks can lay on the floor or be stacked on top of each other. Bigger blocks cannot be stacked on smaller blocks.

When block Y is on top of block X it's denoted as $\texttt{is\_on}(X,Y)$ and $\texttt{is\_clear}(X)$ denotes that there is nothing on block X. $\texttt{floor}(X)$ denotes that block X rests on the foor. The formula $\texttt{type}(X,A)$ denotes that block X is of type A. An agent in the blocks world is capable of perform two actions. $\texttt{drop}(X)$ is the action that drops block X on the floor. $\texttt{put}(X,Y)$ is the action that places block Y on top of block X. The following rules describe the blocks world:

$$
\begin{aligned}
A_1 &\quad \mathbf{A}_i\texttt{put}(X,Y)\\
A_2 &\quad \texttt{clear}(X) \leftrightarrow \mathbf{A}_i\texttt{drop}(X)\\
O_1 &\quad \texttt{clear}(X) \wedge X \neq Y \wedge \neg(X < Y) \leftrightarrow \mathbf{O}_i(\texttt{put}(X,Y))\\
O_2 &\quad \mathbf{O}_i\texttt{drop}(X)\\
E_1 &\quad [\texttt{do}_i(\texttt{put}(X,Y))]\,(\texttt{is\_on}(X,Y) \wedge \neg\texttt{is\_clear}(X))\\
E_2 &\quad \texttt{is\_on}(X,Y) \rightarrow [\texttt{do}_i(\texttt{drop}(Y))]\,(\texttt{is\_clear}(X) \wedge \texttt{floor}(Y))\\
N_1 &\quad (\texttt{is\_clear}(Z) \wedge Z \neq X) \rightarrow [\texttt{do}_i(\texttt{put}(X,Y))]\,\texttt{is\_clear}(Z)\\
N_2 &\quad (\texttt{is\_on}(V,Z) \wedge Z \neq Y) \rightarrow [\texttt{do}_i(\texttt{put}(X,Y))]\,\texttt{is\_on}(V,Z)\\
N_3 &\quad (\texttt{is\_on}(X,Y) \wedge \texttt{is\_on}(U,V) \wedge X \neq U) \rightarrow [\texttt{do}_i(\texttt{drop}(Y))]\,\texttt{is\_on}(U,V)\\
N_4 &\quad \texttt{is\_clear}(Z) \rightarrow [\texttt{do}_i(\texttt{drop}(Y))]\,\texttt{is\_clear}(Z)\\
N_5 &\quad (X = Y \wedge U \neq V) \rightarrow [\texttt{do}_i(\alpha)]\,(X = Y \wedge U \neq V)\\
C_1 &\quad (\texttt{type}(X,A) \wedge \texttt{type}(Y,B) \wedge \texttt{type}(Z,C)) \rightarrow ((X > Y) \wedge (Y > Z) \wedge (X > Z))\\
C_2 &\quad \mathbf{Goal}_e\varphi \rightarrow \mathbf{K}_i\mathbf{Goal}_i\varphi\\
C_3 &\quad \mathbf{A}_i\alpha \rightarrow \mathbf{K}_i\mathbf{A}_i\alpha\\
C_4 &\quad \langle\texttt{do}_i(\alpha)\rangle\,\varphi \rightarrow \mathbf{K}_i\,\langle\texttt{do}_i(\alpha)\rangle\,\varphi\\
C_5 &\quad \texttt{tower}(X,Y,Z) \leftrightarrow \texttt{floor}(X) \wedge \texttt{is\_on}(X,Y) \wedge \texttt{is\_on}(Y,Z) \wedge \texttt{is\_clear}(Z)
\end{aligned}
$$

The A rules describe the abilities of the agent.[1] Constraint A_1 describes that the agent always has the ability to put a block on another block. Constraint A_2 describes the ability of the agent to drop a block on the floor if there is no block on top of it. The O rules describe opportunities for the agent. Constraint O_1 describes the opportunity to put a block of type Y onto a block of type X when blocks of type Y is smaller than blocks of type X. O_2 describes the opportunity to drop a block. The E rules describe effects of certain actions. Rule E_1 makes sure that after a block is put on top of another block, this block isn't clear anymore. E_2 describes that dropping a block puts the block on the floor and the block on which it was stacked is now clear. The N rules describe the non-effects of actions.[2] N_1 makes sure that blocks that are clear stay clear if there is a block put on another block. N_2 makes sure that two blocks stay on top of each other if another block is put somewhere else. Constraint N_3 makes sure that blocks that are stacked on top of each other stay on top of each other if other blocks are dropped. N_4 makes sure that clear blocks stay clear if a block is dropped and N_5 makes sure that blocks stay of the same type if an action is performed. The C rules introduce additional constraints for this example. Constraint C_1 defines the size difference between the different types of blocks. C_2 makes sure that agent i knows what it's goals are and C_3 makes sure that agent i knows what it's abilities are. Constraint C_4 makes sure that agent i knows the results of his actions. Finally, constraint C_5 gives a definition of a tower tree blocks high.

Let there be an agent e in the initial state s_0 as given in Figure 2 and the wish to build a tower of three blocks, $\mathbf{W}_e\mathtt{tower}(X,Y,Z)$. The initial selected wishes and agenda are empty, $C(e,s_0)=\emptyset$ and $\mathit{Agenda}(e,s_0)=\emptyset$. The agent can make building a tower it's goal if in the current state s_0 it wishes to make a tower, $\mathbf{W}_e\mathtt{tower}(X,Y,Z)$, there isn't already a tower, $\neg\mathtt{tower}(X,Y,Z)$, making a tower is implementable, $\Diamond_e\mathtt{tower}(X,Y,Z)$ and the agent has selected the wish to build a tower, $C_e\mathtt{tower}(X,Y,Z)$. It is given that the agent has the wish to build a tower and it is clear from Figure 2 that $M,s_0 \models \neg\mathtt{tower}(X,Y,Z)$. It's given that the agent hasn't selected any wish. Whether the agent is able to select the wish depends on that the wish isn't already true and that the wish is implementable. Again, there isn't a tower yet in state s_0. Whether the wish is implementable depends on whether there is an action sequence that makes the wish practically possible.

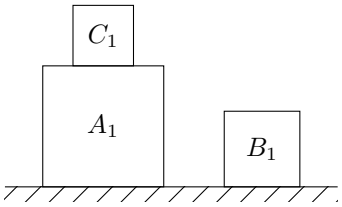


Figure 2: The staring position of the blocks in the example.

To see that making a tower is implementable consider the action sequence $\alpha = \mathtt{drop}(C_1);$ $\mathtt{put}(A_1,B_1);\mathtt{put}(B_1,C_1)$. According to definition 2.7 and 2.9 an agent is able to perform an action sequence if the agent is able to perform the atomic actions in the action sequence in the intermediate states. According to rule A_1 an agent is able to perform the atomic actions $\mathtt{put}(X,Y)$ in all states if Y is not of type A. According to rule A_2 an agent is able to perform the atomic action $\mathtt{drop}(X)$ in the state where X is clear and X is not of type A. Thus agent e is able to perform α in the initial state s_0 because e is able to perform the atomic actions in α in the intermediate states. $\langle \mathtt{do}_e(\alpha)\rangle\,\varphi$ holds in a state s_0 according to definition 2.7 if the agent i has the opportunity to preform α and φ holds after preforming α. According to rule O_1 an agent has the opportunity to perform the atomic action $\mathtt{put}(X,Y)$ in the state where X is clear and X is not smaller than Y. According to rule O_2 an agent always has the opportunity to perform

[1] The abilities slightly differ from the abilities given in [4]. In this paper blocks of type A are not too heavy and the agent is able to move them.

[2] [4] gives an additional non-effect rule (there it's rule N_3). This rule is, however, the same as rule E_2 and not a non-effect. It is omitted in this paper.

the atomic action $\texttt{drop}(X)$. Thus agent e has the opportunity to perform α in the initial state s_0 because e has the opportunity to perform the atomic actions in α in the intermediate states. Using the E, N and C_5 rules it is easy to see that $\texttt{tower}(A_1, B_1, C_1)$ holds after performing action α. It follows that $M, s_0 \models \langle \texttt{do}_e(\alpha) \rangle \texttt{tower}(A_1, B_1, C_1)$. Using definition 2.2 it follows that $\mathbf{PracPoss}_e(\alpha, \texttt{tower}(A_1, B_1, C_1))$ Thus there is an action sequence that is practically possible and that results in a tower, thus by definition 2.12, building a tower is implementable. The wish of agent e is implementable and doesn't already hold, according to definition 2.17 the agent is now capable to select the wish to build a tower.

The agent performs the action $\texttt{select}(\texttt{tower}(X, Y, Z))$, resulting in a new state s_1 where $M', s \models^{\mathcal{C}} \mathbf{C}_e \texttt{tower}(X, Y, Z)$. It follows from definition 2.13 that s_1 only differs from s_0 in the set of selected wishes $\mathbf{C}(e, s_1)$. Thus in s_1 building a tower is still implementable, $\Diamond_e \texttt{tower}(X, Y, Z)$, not already realised, $\neg\texttt{tower}(X, Y, Z)$, and a wish of e, $\mathbf{W}_e \texttt{tower}(X, Y, Z)$. Now that in s_1 building a tower is also selected if follows from definition 2.5 that building a tower is a goal for e, $\mathbf{Goal}_e \texttt{tower}(X, Y, Z)$. From rule $C2$ it follows that e also knows it has the goal to build a tower, $\mathbf{K}_e \mathbf{Goal}_e \texttt{tower}(X, Y, Z)$. It was already shown that e has the ability and the capability to build a tower an thus that it is practically possible for e to build a tower. It follows from rule C_3 and C_4 that e also knows that it is practically possible to build a tower,$\mathbf{K}_e \mathbf{PracPoss}_e(\alpha, \texttt{tower}(X, Y, Z))$. Thus from definition 2.3 it follows that e can build a tower with action sequence α, $\mathbf{Can}_e(\texttt{tower}(X, Y, Z), \alpha)$. The agent can build a tower with action sequence α and knows that it's goal is to build a tower, from definition 2.6 it now follows that e possibly intends to do action sequence α, $\mathbf{PossIntend}_e(\alpha, \texttt{tower}(X, Y, Z))$. It follows from definition 2.13 that e now has the opportunity to commit to action sequence α to fulfil its goal. Since the agenda of the agent is empty it follows from definition 2.17 that agent e also has the ability to commit to action sequence α to fulfil its goal.

Committing to action sequence α leads to state s_2 where α is added to the Agenda of e according to definition 2.13. According to definition 2.12 the agent is now committed to the prefix of α, $\mathbf{Com}_e\, \texttt{drop}(C_1)$. According to definition 2.17 the agent is not able to uncommit an action as long as it possibly intends to do the action. The act of committing only changes the agenda and thus the agent still possibly intends to do α. Let's suppose that the agent executes the action $\texttt{drop}(C_1)$, resulting in state s_3.

Dropping block C_1 has the effect according to rule E_2 that in state s_3 block C_1 is on the floor and block A_1 is clear. The rules N_3, N_4 and N_5 make sure that dropping the block doesn't have other effects. The agent is thus still possibly intends to build a tower with action sequence α and is thus unable to uncommit. The Agenda of e in s_3 contains $\texttt{put}(A_1, B_1); \texttt{put}(B_1, C_1)$ because e committed to α in state s_2. The agent is now committed to the prefix of the action in its agenda, $\mathbf{Com}_e\, \texttt{put}(A_1, B_1)$. Let's suppose that the agent executes the action $\texttt{put}(A_1, B_1)$, resulting in state s_4.

Putting block B_1 on top of block A_1 has the effect according to rule E_1 that in state s_4 block B_1 is on top of A_1 and block A_1 is not clear. The rules N_1, N_2 and N_5 make sure that putting one block on the other doesn't have other effects. The agent still possibly intends to build a tower with action sequence α and thus is unable to uncommit. The Agenda of e in state s_4 contains $\texttt{put}(B_1, C_1)$ because e committed to α in state s_2. The agent is now committed to the prefix of the action in its agenda. The action in the agents agenda is an semi-atomic action an thus the agent is committed to this action, $\mathbf{Com}_e\, \texttt{put}(B_1, C_1)$. Let's suppose that the agent executes the action $\texttt{put}(B_1, C_1)$, resulting in state s_4.

The agents puts block C_1 on bock B_1, which has similar results as in the previous state. But now it holds that $\texttt{Floor}(A_1) \wedge \texttt{is\_on}(A_1, B_1) \wedge \texttt{is\_on}(B_1, C_1) \wedge \texttt{is\_clear}(C_1)$ and thus $\texttt{tower}(A_1, B_1, C_1)$ according to rule C_5. The agent fulfilled its wish to build a tower, $\texttt{tower}(X, Y, Z)$. The agent no longer has the goal to build a tower because in this state there is already a tower.

3 Adding Time to Actions

In this section, KARO is extended with actions that take a certain amount of time to complete. This enables an agent to plan actions that fulfil a wish in a limited amount of time. It also enables the agent to have conflicting wishes at differed moments in time. For now it is assumed that the actions `select`, `commit` and `uncommit` take no time to perform. In section 5 a language is given where these actions do take time to perform.

3.1 Syntax

3.1.1 Language $\mathcal{L}^{at}$

The language $\mathcal{L}^{at}$ is similar to the language $\mathcal{L}$. $\mathcal{L}^{at}$ has in addition to the operators of language $\mathcal{L}$ also a $\mathbf{Before}_i(\alpha, \tau_{\text{end}})$ operator. This operator denotes that agent i is able to perform action α before the time τ_{end} is reached. Thus if $\text{p} \in \Pi$ where Π is the set of propositional atoms, $i \in \text{A}$ where A is the set of agents, $\alpha \in \text{Ac}$ and $\tau_{\text{end}} \in \mathbb{R}$, then the following grammar defines the language $\mathcal{L}^{at}$:

$$\varphi, \psi ::= \text{p} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{K}_i\varphi \mid \langle \text{do}_i(\alpha) \rangle \varphi \mid \mathbf{A}_i\alpha \mid \mathbf{Before}(\alpha, \tau_{\text{end}})$$

$\mathbf{CanBefore}_i(\alpha, \tau_{\text{end}})$ denotes that agent i knows that it is able to do action α before time τ_{end}.

$$\mathbf{CanBefore}_i(\alpha, \tau_{\text{end}}) =^{\text{def}} \mathbf{K}_i\mathbf{Before}_i(\alpha, \tau_{\text{end}}) \qquad \text{Def. 3.1}$$

3.1.2 Language $\mathcal{L}^{at^{\mathcal{C}}}$

The language $\mathcal{L}^{at^{\mathcal{C}}}$ is similar to language $\mathcal{L}^{\mathcal{C}}$. Wishes now also have an end time associated with them, $\mathbf{W}_i(\varphi, \tau_{\text{end}})$. This expresses that the agent is motivated to fulfil the wish before the end time. Implementability is extended with an end time, $\Diamond_i(\varphi, \tau_{\text{end}})$ indicating that it is possible to make φ true before the end time τ_{end}. **Goal** is also extended with an end time before which the goal must be reached. **PossInend** is extended with an end time indicating that the agent intends to do the actions before the end time. The agent shouldn't intent to do an action sequence that finishes after the end time. Thus the agent can only possibly intend to do an action if $\mathbf{CanBefore}_i(\alpha, \tau_{\text{end}})$ is true.

$$\mathbf{Goal}_i(\varphi, \tau_{\text{end}}) =^{\text{def}} \mathbf{W}_i(\varphi, \tau_{\text{end}}) \wedge \neg\varphi \wedge \Diamond_i(\varphi, \tau_{\text{end}}) \wedge \mathbf{C}_i\varphi \qquad \text{Def. 3.2}$$

$$\mathbf{PossIntend}_i(\alpha, \varphi, \tau_{\text{end}}) =^{\text{def}} \mathbf{Can}_i(\alpha, \varphi) \wedge \mathbf{K}_i\mathbf{Goal}_i(\varphi, \tau_{\text{end}}) \wedge \mathbf{CanBefore}_i(\alpha, \tau_{\text{end}}) \qquad \text{Def. 3.3}$$

KARO doesn't say anything about when the agent might perform its actions, only the order in which they must be performed. But when actions take a certain amount of time to complete and the goal must be reached before a certain end time it is important when the actions are executed. It can be assumed that the agent starts performing the actions as soon as it's committed to execute an action.

There are however always unforeseen external factors that prevent the action from starting immediately or that can slow down the action. Planning the actions is thus always done with the estimated time an action takes. It's better to overestimate the time it takes for an action to resolve than to underestimate it. When the time is overestimated, actions probably take shorter than estimated and thus the chance that the goal is reached before the end time is larger. But if the estimate is too large the agent might not consider certain actions because it thinks the actions take too long.

3.2 Semantics

3.2.1 Language $\mathcal{L}^{at}$

The class $\mathbf{M}^{at}$ of models contains all $M = \langle S, \pi, R, r_0, c_0, \tau_0 \rangle$. The set S and the functions π, R, r_0 and c_0 are the same as in the model for language $\mathcal{L}$. $\tau_0 : A \times At^+ \to (S \cup \{\emptyset\}) \to \mathbb{R}$ gives the time an action takes in a certain state.

The time an action α takes is just the sum of all semi-atomic actions needed to perform the action. It is calculated in the result function and given to the next state. In this paper, the time is in seconds, but the time can have any unit depending on the problem.

Definition 3.4. *$M, s, \tau \models^{at} \varphi$ denotes that φ is true in the state $s \in S$ in the model $M \in \boldsymbol{M}^{at}$ and τ is a real number indicating the time. $M, s, \tau \models^{at} \varphi$ is mostly defined in the same way as $M, s \models \varphi$,*

$$
\begin{aligned}
&M, s, \tau \models^{at} \varphi && \Leftrightarrow \pi(s)(\varphi) = 1 \\
&M, s, \tau \models^{at} \neg\varphi && \Leftrightarrow M, s, \tau \not\models^{at} \varphi \\
&M, s, \tau \models^{at} \varphi \wedge \psi && \Leftrightarrow M, s, \tau \models^{at} \varphi \text{ and } M, s, \tau \models^{at} \psi \\
&M, s, \tau \models^{at} \varphi \vee \psi && \Leftrightarrow M, s, \tau \models^{at} \varphi \text{ or } M, s, \tau \models^{at} \psi \\
&M, s, \tau \models^{at} \boldsymbol{K}_i\varphi && \Leftrightarrow \forall s' \in S((s, s') \in R(i) \Rightarrow M, s', \tau \models^{at} \varphi) \\
&M, s, \tau \models^{at} \langle do_i(\alpha) \rangle \varphi && \Leftrightarrow \exists M', s', \tau' (M', s', \tau' \in r^{at}(i, \alpha)(M, s) \wedge \\
& && \quad M', s', \tau' \models^{at} \varphi) \\
&M, s, \tau \models^{at} \boldsymbol{A}_i\alpha && \Leftrightarrow c(i, \alpha)(s) = 1 \\
&M, s, \tau \models^{at} \boldsymbol{Before}_i(\alpha, \tau_{end}) && \Leftrightarrow \pi_3(r^{at}(i, \alpha)(M, s, \tau)) \leq \tau_{end}
\end{aligned}
$$

The function π_3 returns the third element of a triple, in this case the time τ as returned by the result function. The definition for $M, s, \tau \models^{at} \langle \mathrm{do}_i(\alpha) \rangle \varphi$ differs from the definition in 2.7, because doing an action must result in a state where time has progressed. $M, s, \tau \models^{at} \mathbf{Before}_i(\alpha, \tau_{\mathrm{end}})$ compares the time resulting from executing action α with the end time τ_{end}.

Definition 3.5. *The result function r^{at} is defined in a similar way to function r, where* $\mathrm{a} \in At$ *and $r^{at}(i, \alpha)(M, \varepsilon) = M, \varepsilon$.*

$$
\begin{aligned}
&r^{at}(i, \mathrm{a})(M, s, \tau) && = (M', s', \tau') \text{ such that } (M', s') = r_0(i, \mathrm{a})(M, s) \\
& && \quad \text{and } \tau' = \tau + \tau_0(i, \mathrm{a})(M, s) \\
&r^{at}(i, \varphi?)(M, s, \tau) && = (M, s, \tau) \text{ if } M, s, \tau \models^{at} \varphi \text{ otherwise } M, \varepsilon \\
&r^{at}(i, \alpha_1; \alpha_2)(M, s, \tau) && = r^{at}(i, \alpha_2,)(r^{at}(i, \alpha_1)(M, s, \tau)) \\
&r^{at}(i, \texttt{if } \varphi \texttt{ then } \alpha_1 \texttt{ else } \alpha_2 \texttt{ fi})(M, s, \tau) && = r^{at}(i, \alpha_1)(M, s, \tau) \text{ if } M, s, \tau \models^{at} \varphi \text{ and} \\
& && \quad r^{at}(i, \alpha_2)(M, s, \tau) \text{ otherwise} \\
&r^{at}(i, \texttt{while } \varphi \texttt{ do } \alpha \texttt{ od})(M, s, \tau) && = (M', s', \tau') \text{ such that } \exists k \in \mathbb{N} \\
& && \quad \exists (M_1, s_1, \tau_1) \ldots \exists (M_k, s_k, \tau_k)((M_1, s_1, \tau_1) = \\
& && \quad (M, s, \tau) \text{ and } (M_k, s_k, \tau_0) = (M', s', \tau') \text{ and} \\
& && \quad \forall j < k((M_{j+1}, s_{j+1}, \tau_{j+1}) = \\
& && \quad r^{at}(i, \varphi?; \alpha)(M_j, s_j, \tau_j)) \text{ and } M', s', \tau' \models \neg\varphi)
\end{aligned}
$$

Definition 3.6. *The capability function c^{at} is defined in a similar way to function c, where $c^{at}(i,\alpha)(M,\varepsilon) = 0$.*

$$
\begin{aligned}
&c^{at}(i,\alpha)(M,s,\tau) && = c_0(i,a)(s)\\
&c^{at}(i,\varphi?)(M,s,\tau) && = 1 \textit{ if } M,s,\tau \models^{at} \varphi \textit{ and } 0 \textit{ otherwise}\\
&c^{at}(i,\alpha_1;\alpha_2)(M,s,\tau) && = c^{at}(i,\alpha_1)(M,s,\tau) \textit{ and}\\
& && \quad c^{at}(i,\alpha_2)(r^{at}(i,\alpha_1)(M,s,\tau))\\
&c^{at}(i,\texttt{if } \varphi \texttt{ then } \alpha_1 \texttt{ else } \alpha_2 \texttt{ fi})(M,s,\tau) && = c^{at}(i,\varphi?;\alpha_1)(M,s,\tau) \textit{ or}\\
& && \quad c^{at}(i,\neg\varphi?;\alpha_2)(M,s,\tau)\\
&c^{at}(i,\texttt{while } \varphi \texttt{ do } \alpha \texttt{ od})(M,s,\tau) && = 1 \textit{ if for some } k \in \mathbb{N}\\
& && \quad c^{at}(i,(\varphi?;\alpha)^k;\neg\varphi?)(M,s,\tau) \textit{ and}\\
& && \quad 0 \textit{ otherwise}
\end{aligned}
$$

3.2.2 Language $\mathcal{L}^{at^{\mathcal{C}}}$

The class $\mathbf{M}^{at^{\mathcal{C}}}$ of models contains all $M = \langle S,\pi,R,r_0,c_0,\tau_0,W,C,\text{Agenda}\rangle$. The set S and the functions π, R, r_0, c_0, W and Agenda are the same as in language $\mathcal{L}^{\mathcal{C}}$. The function τ_0 is the same as in language $\mathcal{L}^{at}$. The function C is changed so that an proposition and an end time are returned.

Definition 3.7. *The relation $\models^{at^{\mathcal{C}}}$ contains all relations $\models^{at}$ of the language $\mathcal{L}^{at}$. Similar to $\models^{\mathcal{C}}$ it also contains the relations for $\boldsymbol{W}_i(\varphi,\tau_{end})$, $\Diamond_i\varphi$, $\boldsymbol{C}_i(\varphi,\tau_{end})$ and $\boldsymbol{Com}_i\alpha$. These are defined in the following way, where $M \in \boldsymbol{M}^{\mathcal{C}}$, $s \in S$ and Prefix(α,β) indicates that action sequence α is a prefix of action sequence β.*

$$
\begin{aligned}
&M,s,\tau \models^{at^{\mathcal{C}}} \varphi && \Leftrightarrow M,s,\tau \models^{at} \varphi\\
&M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{W}_i(\varphi,\tau_{end}) && \Leftrightarrow \forall s' \in S((s,s') \in W(i) \Rightarrow M,s',\tau' \models^{at^{\mathcal{C}}} \varphi \textit{ and } \tau' \leq \tau_{end})\\
&M,s,\tau \models^{at^{\mathcal{C}}} \Diamond_i(\varphi,\tau_{end}) && \Leftrightarrow \exists k \exists \alpha_1,\ldots,\alpha_k \in At((M,s,\tau) \models^{at^{\mathcal{C}}} \boldsymbol{PracPoss}_i(\alpha_1;\ldots;\alpha_k,\varphi)) \wedge\\
& && \quad \boldsymbol{Before}_i(\alpha_1;\ldots;\alpha_n,\tau_{end})\\
&M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{C}_i(\varphi,\tau_{end}) && \Leftrightarrow (\varphi,\tau_{end}) \in C(i,s)\\
&M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{Com}_i\alpha && \Leftrightarrow \forall s' \in [s]_{R(i)} \exists \alpha_1 \in CR^{\mathcal{C}}_M(i,\alpha,s') \exists \alpha_2 \in Agenda(i,s')\\
& && \quad \exists \alpha_2' \in CR^{\mathcal{C}}_M(i,\alpha_2,s')(Prefix(\alpha_1,\alpha_2'))
\end{aligned}
$$

Definition 3.8. *If $M = \langle S,\pi,R,r_0,c_0,\tau_0,W,C,Agenda\rangle$ then*

$$
\begin{aligned}
&r^{at^{\mathcal{C}}}(i,\alpha)(M,s,\tau) && = r^{at}(i,\alpha)(M,s,\tau)\\
&r^{at^{\mathcal{C}}}(i,\texttt{select}(\varphi,\tau_{end}))(M,s,\tau) && = \left(\left\langle S,\pi,R,r_0,c_0,\tau_0,W,\texttt{choose}^{at}(i,\varphi,\tau_{end})(M,s),Agenda\right\rangle,s,\tau\right)\\
& && \quad \textit{if } M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{W}_i(\varphi,\tau_{end}) \textit{ and } \emptyset \textit{ otherwise}\\
&r^{at^{\mathcal{C}}}(i,\texttt{commit } \alpha)(M,s,\tau) && = \left(\left\langle S,\pi,R,r_0,c_0,\tau_0,W,C,\texttt{agenda}^{+}(i,\alpha)(M,s)\right\rangle,s,\tau\right)\\
& && \quad \textit{if } M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{PossIntend}_i(\alpha,\varphi,\tau_{end}) \textit{ for some}\\
& && \quad (\varphi,\tau_{end}) \in C(i,s) \textit{ and } \emptyset \textit{ otherwise}\\
&r^{at^{\mathcal{C}}}(i,\texttt{uncommit } \alpha)(M,s,\tau) && = \left(\left\langle S,\pi,R,r_0,c_0,\tau_0,W,C,\texttt{agenda}^{-}(i,\alpha)(M,s)\right\rangle,s,\tau\right)\\
& && \quad \textit{if } M,s,\tau \models^{at^{\mathcal{C}}} \boldsymbol{Com}_i\alpha \textit{ and } \emptyset \textit{ otherwise}
\end{aligned}
$$

Definition 3.9. *If C is the current choice function in $\boldsymbol{M}^{at^{\mathcal{C}}}$, then:*

$$\texttt{choose}^{at}(i,\varphi,\tau_{end})(M,s) = C' \text{ where } C'(i',s') = C(i',s') \text{ if } i \neq i' \text{ or } s \neq s' \\ \text{and } C'(i,s) = C(i,s) \cup \{(\varphi,\tau_{end})\}$$

Definition 3.10. *The capability function $c^{at^{\mathcal{C}}}$ contains all functions c^{at}.*

$$\begin{aligned}
&c^{at^{\mathcal{C}}}(i,\alpha)(M,s,\tau) && = c^{at}(i,\alpha)(M,s,\tau)\\
&c^{at^{\mathcal{C}}}(i,\texttt{select}(\varphi,\tau_{end}))(M,s,\tau) && = 1 \text{ if } M,s,\tau \models^{at^{\mathcal{C}}} \neg\varphi \wedge \Diamond_i(\varphi,\tau_{end}) \text{ and } 0 \text{ otherwise}\\
&c^{at^{\mathcal{C}}}(i,\texttt{commit } \alpha)(M,s,\tau) && = 1 \text{ if } Agenda(i,s) = \emptyset \text{ and } 0 \text{ otherwise}\\
&c^{at^{\mathcal{C}}}(i,\texttt{uncommit } \alpha)(M,s,\tau) && = 1 \text{ if } M,s,\tau \models^{at^{\mathcal{C}}} \neg\boldsymbol{PossIntend}_i(\alpha,\varphi,\tau_{end}) \text{ for all}\\
& && \quad (\varphi,\tau_{end}) \in C(i,s) \text{and } 0 \text{ otherwise}
\end{aligned}$$

Theorem 3.1. *The agent can only put actions successfully in its agenda if it is expected that the actions fulfil a wish before the end time of the wish.*

Proof. It follows from the truth axiom $\mathbf{K}_i\varphi \Rightarrow \varphi$ that for arbitrary M, s and τ it holds that $M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{K}_i\mathbf{Before}_i(\alpha,\tau_{\text{end}}) \Rightarrow M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{Before}_i(\alpha,\tau_{\text{end}})$. It follows from definition 3.1 that $M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{CanBefore}_i(\alpha,\tau_{\text{end}}) \Rightarrow M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{Before}_i(\alpha,\tau_{\text{end}})$. It thus also holds that $M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{Can}_i(\alpha,\varphi) \wedge \mathbf{K}_i\mathbf{Goal}_i(\varphi,\tau_{\text{end}}) \wedge \mathbf{CanBefore}_i(\alpha,\tau_{\text{end}}) \Rightarrow M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{Before}_i(\alpha,\tau_{\text{end}})$ for arbitrary φ. It follows from definition 3.3 that $M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{PossIntend}_i(\alpha,\varphi,\tau_{\text{end}}) \Rightarrow M,s,\tau \models^{at^{\mathcal{C}}} \mathbf{Before}_i(\alpha,\tau_{\text{end}})$. It follows from definition 3.8 that an agent i is only able to successfully put an action α in its agenda if for some selected proposition φ with end time τ_{end} the agent possibly intends to do the action α with result φ and end time τ_{end}. And thus the action α can only successfully put in the agenda if the action α can be finished before the end time. It also follows from definition 3.8 that all successfully selected propositions are wishes. Since the action α needs to be selected in order to put φ successfully in the agenda φ must also be a wish. Thus the wish φ is fulfilled before τ_{end} with action α. □

3.3 Block World Example

In the next examples moving a block of type A takes 3 second, moving a block of type B takes 2 seconds and moving a block of type C 1 second. If the agent can perform an action before the end time it knows that it can perform the action before the end time. Or in a constraint rule:

$$\begin{aligned}
C_6 \quad & \forall i \in A, (\texttt{type}(X,A) \wedge \texttt{type}(Y,B) \wedge X\texttt{type}(Z,C)) \rightarrow (\tau_0(i,\texttt{drop}(X)) = 3 \wedge\\
& \tau_0(i,\texttt{drop}(Y)) = 2 \ \wedge \tau_0(i,\texttt{drop}(Z)) = 1 \wedge \tau_0(i,\texttt{put}(X,U)) = 3 \wedge\\
& \tau_0(i,\texttt{put}(Y,U)) = 2 \wedge \tau_0(i,\texttt{put}(Z,U)) = 1)\\
C_7 \quad & \mathbf{Before}(\alpha,\tau_{\text{end}}) \rightarrow \mathbf{K}_i\mathbf{Before}_i(\alpha,\tau_{\text{end}})
\end{aligned}$$

Let there be an agent e in the initial state s_0 as given in Figure 3 where $\tau(s_0) = 0$. The agent e has the wish to make a block of type A clear in 1 second, $\mathbf{W}_e(\texttt{type}(X,A) \wedge \texttt{is\_clear}(X),1)$. The initial selected wishes and agenda are empty, $C(e,s_0) = \emptyset$ and $Agenda(e,s_0) = \emptyset$. Just like in the example in section 2.3, to make the agents wish a goal the wish must not already be fulfilled, implementable and selected. In state s_0 there isn't a block of type A that is clear. To be able to select the wish it must thus be implementable.

To see that the wish is implementable consider the action $\texttt{drop}(C_1)$. The action takes 1 second according to rule C_6, $\tau_{\text{end}} = 1$. According to definition 3.4 $M,s_0,0 \models^{at} \mathbf{Before}_e(\texttt{drop}(C_1),1)$, because $\tau_0(e,\texttt{drop}(C_1))(M,s_0)+0 \leq \tau_{\text{end}}$. The agent is able to drop block C_1 according to rule A_2 because block C_1 has no other blocks on top of it in state s_0. Agent e always has the opportunity

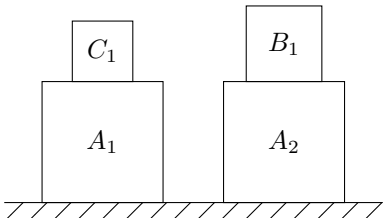


Figure 3: The staring position of the blocks in the example.

to drop block according to rule O_2. The effect of dropping block C_1 is according to rule E_2 that block C_1 is on the floor and block A_1 is clear. Thus $M, s_0, 0 \models^{at} \langle \mathrm{do}_e(\mathtt{drop}(C_1)) \rangle \, \mathtt{is\_clear}(A_1) \wedge \mathbf{A}_e \mathtt{drop}(C_1)$. It follows from definition 2.2 that $M, s_0, 0 \models^{at} \mathbf{PracPoss}_e(\mathtt{drop}(C_1), \mathtt{is\_clear}(A_1))$. Thus there is an action sequence that is practically possible and that results in a state where a block of type A to be clear, thus by definition 3.7 making a block of type A clear is implementable. The wish of agent e is implementable and doesn't already hold, according to definition 3.10 the agent is now capable to select the wish to make a block of type A clear.

The agent performs the action $\mathtt{select}(\mathtt{type}(X, A) \wedge \mathtt{is\_clear}(X), 1)$. This results in a new state s_1 where $M, s_1, 0 \models^{at^C} \mathbf{C}_e(\mathtt{type}(X, A) \wedge \mathtt{is\_clear}(X), 1)$. The time τ in state s_1 is still 0 because the select action does not takes any time to be performed. It follows from definition 3.2, in a similar way as in the example in section 2.3, that making a block of type A clear is now a goal of agent e, $M, s_1, 0 \models^{at^C} \mathbf{Goal}_e(\mathtt{type}(X, A) \wedge \mathtt{is\_clear}(X), 1)$. From using rule C_2 it follows that e also knows that it has the goal to clear a block of type A, $M, s_1, 0 \models^{at^C} \mathbf{K}_e \mathbf{Goal}_e(\mathtt{type}(X, A) \wedge \mathtt{is\_clear}(X), 1)$.

The agent might however consider the action $\mathtt{drop}(B_1)$ instead of $\mathtt{drop}(C_1)$. Similarly, to dropping block C_1, dropping block B_1 result in a state where a block of type A is clear. The agent is able and has the capability to perform the action to drop block C_1, thus it is practically possible. The action can however not be done before τ_{end} because the action takes 2 seconds according to rule C_6 and $\tau_{\mathrm{end}} = 1$, $\neg \mathbf{Before}_e(\mathtt{drop}(B_1), 1)$. It follows from definition 3.1 that $\neg \mathbf{CanBefore}_e(\mathtt{drop}(B_1), 1)$ and from definition 3.3 that $\neg \mathbf{PossIntend}_e(\mathtt{drop}(B_1), 1)$. The agent is thus unable to commit to the action $\mathtt{drop}(B_1)$.

The agent e is able to commit to the action $\mathtt{drop}(C_1)$. It is shown before that $\mathbf{Before}_e(\mathtt{drop}(C_1), 1)$ in state s_0, it also holds in state s_1 because the time is still 0. According to rule C_7 the agent also knows that it can do the action before the end time. According to definition 3.1 it holds in state s_1 that $\mathbf{CanBefore}_e(\mathtt{drop}(C_1), 1)$. It was already shown that e is able and capable to do $\mathtt{drop}(C_1)$, from using rules C_3 and C_4 and definitions 2.2 and 2.3 it follows that $M, s_1, 0 \models^{at} \mathbf{Can}_e(\mathtt{drop}(C_1), \mathtt{clear}(A_1))$. It was already shown that e knows that it has the goal to clear a block of type A, it can do an action that clears block A_1 before the end time, it now follows from definition 3.3 that $M, s_1, 0 \models^{at^C} \mathbf{PossIntend}_e(\mathtt{drop}_e(C_1), \mathtt{clear}(A_1), 1)$. It follows from definition 3.8 that e now has the opportunity to commit to action $\mathtt{crop}(C_1)$ to fulfil its goal. Since the agenda of the agent is empty it follows from definition 3.10 that agent e also has the ability to commit to action $\mathtt{drop}(C_1)$ to fulfil its goal.

Committing to action $\mathtt{drop}(C_1)$ leads to state s_2 where $\mathtt{drop}(C_1)$ is added to the Agenda of e according to definition 3.8. According to definition 3.7 the agent is now committed to drop the block C_1. Let's suppose that the agent executes the action $\mathtt{drop}(C_1)$, resulting in state s_3.

Dropping block C_1 has the effect according to rule E_2 that in state s_3 block C_1 is on the floor and block A_1 is clear. According to rule C_6 the action of dropping block C_1 took 1 second. Thus by definition 3.4 the time at state s_3 is 1. The agent e fulfilled its wish to make a block of type A clear within one second. The agent no longer has the goal to make a block of type A clear because a block of type A is already clear in s_3.

4 Parallel Actions

Performing actions in parallel becomes important if there is a limited amount of time. When two actions are done in serial the times it takes is the sum of the time the individual actions take. Parallel actions only take the time of the longest action. Thus it gives a huge time advantage when an agent is able to plan and perform actions in parallel. Also, Agents are usually able to perform actions and think at the same time. Since reasoning steps like committing and selecting are viewed as actions, it must be possible to do other actions in parallel with this reasoning action. The languages $\mathcal{L}, \mathcal{L}^{\mathcal{C}}, \mathcal{L}^{at}$ and $\mathcal{L}^{at^{\mathcal{C}}}$ can be extended to allow for parallel actions. The extension works in a similar way for all the languages and is given in this section.

4.1 Syntax

The search-space for parallel actions, that can happen at any time, is huge. By assuming that there are only two moments when an action can be performed the search space is kept small. An action either starts after another action or it starts at the same time as another action. An action can only start at the same time as another action when the two actions can be performed in parallel. This results in a tree structure where every node is an action which is connected to all actions that are performed in parallel after this action.

There is already a notation for actions that happen after each other, $\alpha; \beta$. $[\alpha_1 \| \ldots \| \alpha_n]$ denotes that the set of actions $\{\alpha_1, \ldots, \alpha_n\}$ are done in parallel. For the Languages $\mathcal{L}$ and $\mathcal{L}^{\mathcal{C}}$ it is assumed that the actions need to be atomic. Actions don't take time in these languages thus the order in which the actions are done must follow from the notation. In the languages $\mathcal{L}^{at}$ and $\mathcal{L}^{at^{\mathcal{C}}}$ the actions do not have to be atomic. It follows from the time the actions take what the order is in which they are performed. To determine the order of the atomic actions the actions are placed on a timeline. Figure 4 is a visual representation of the time line for the actions $[\alpha; \gamma \| \beta; [\beta \| \delta]]$, where $\forall i \in A, s \in S(\tau_0(i, \alpha)(M, s) = 1, \tau_0(i, \beta)(M, s) = 2, \tau_0(i, \gamma)(M, s) = 3$ and $\tau_0(i, \delta)(M, s) = 1)$.

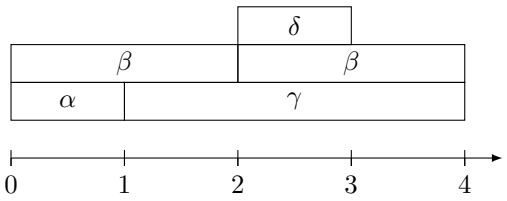


Figure 4: A visual representation of the timeline of the action $[\alpha; \gamma, \beta; [\beta \| \delta]]$.

From Figure 4 it's easy to see that the actions α and β, β and δ but also γ and δ are done simultaneously. That these actions can be done independently from each other and thus in parallel is denoted by $I(\alpha, \beta), I(\beta, \delta), I(\gamma, \beta)$ and $I(\gamma, \delta)$. Note that α and δ are not done simultaneously and thus don't need to be independent.

The following grammar defines the parallel actions that are in the set Ap, where a $\in \Pi$:

$$\alpha, \beta ::= \text{a} \mid \varphi? \mid \alpha; \beta \mid \text{if } \varphi \text{ then } \alpha \text{ else } \beta \text{ fi} \mid \text{while } \varphi \text{ do } \alpha \text{ od} \mid [\alpha_1 \| \ldots \| \alpha_n]$$

The following paragraph will define the same functions for Ap as the functions for Ac. This makes it is possible to use Ap in place of Ac in the languages $\mathcal{L}, \mathcal{L}^{\mathcal{C}}, \mathcal{L}^{at}$ and $\mathcal{L}^{at^{\mathcal{C}}}$, making them capable of using parallel actions.

4.2 Semantics

The actions in KARO are based on the state-transition paradigm. Trace theory [3] is used to allow for parallel actions in this paradigm. A trace is for parallel actions what a sequence of actions is for performing the actions in series. Two atomic actions, α and β, can form two sequences, $\alpha\beta$ and $\beta\alpha$. Trace theory states that if α and β can be performed in parallel the order of the actions doesn't matter. This means that $\alpha\beta$ and $\beta\alpha$ end in the same state. Performing the actions in parallel will also end in this state, both paths towards this state are valid. The trace in this example is the set $\{\alpha\beta, \beta\alpha\}$.

$$r(i, [\alpha_1 \| \ldots \| \alpha_n])(M, s) = r(i, \alpha_1; \ldots; \alpha_n)(M, s) \quad \text{Def. 4.1}$$

$$c(i, [\alpha_1 \| \ldots \| \alpha_n])(M, s) = c(i, \alpha_1; \ldots; \alpha_n)(M, s) \wedge \forall \alpha_x, \alpha_y \in \{\alpha_1, \ldots, \alpha_n\} (I(\alpha_x, \alpha_y)) \quad \text{Def. 4.2}$$

The reachability function given in definition 4.1 can be combined with the function given in definition 2.8 and the capability function given in definition 4.2 can be combined with the function given in definition 2.9. Both functions are about atomic actions. The state reached after performing the atomic actions in parallel is the state the agent would end if all the actions are done in serial. An agent i is capable of performing the atomic actions in parallel if i is capable of performing the actions in sequence and all the atomic actions are independent of each other.

The axiom[3] given in definition 4.3 is added to the transition system T_M as defined in definition 2.10:

$$\langle [\alpha_1 \| \ldots \| \alpha_n], s \rangle \rightharpoonup^{M}_{i,[\alpha_1 \| \ldots \| \alpha_n]} \langle \Lambda, s' \rangle \text{ with } s' = \pi_2(r^{\mathcal{C}}(i, [\alpha_1 \| \ldots \| \alpha_n])(M, s)) \text{ if } \alpha_n \in At^+ \text{ for all } n \text{ and } r^{\mathcal{C}}(i, [\alpha_1 \| \ldots \| \alpha_n])(M, s) \neq \emptyset \quad \text{Def. 4.3}$$

In the language $\mathcal{L}^{at}$ actions take a certain amount of time to complete. When actions take a certain amount of time the transitions of parallel actions cannot be in an arbitrary order. To see this consider the action $[\alpha \| \beta; \gamma]$, where $t(\alpha) = 1$, $t(\beta) = 2$, $t(\gamma) = 1$, $I(\alpha, \beta)$ and $I(\beta, \gamma)$. A visual representation of the action is shown in Figure 5. The order of actions α and $\beta; \gamma$ does matter, because α and γ cannot be done independently. The sequence $\alpha\beta\gamma$ is not the same as the sequence $\gamma\alpha\beta$, because the order of α and γ is different. The order of the sequence must be so that actions that cannot be done in parallel occur in the order in which they are on the timeline, thus α before γ. The trace in this example would be the set $\{\alpha\beta\gamma, \alpha\gamma\beta, \beta\alpha\gamma\}$. Figure 6 shows that doing γ before α leads to another state, $s_{3.2}$ and that all the action sequences in the trace lead to the same state $s_{3.1}$.

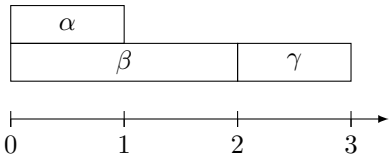


Figure 5: A visual representation of the timeline of the action $[\alpha \| \beta; \gamma]$.

[3]Note that the definition given in [3] is different from the definition 4.3. The definition of parallel actions given in [3] does not allow two parallel actions to start from the same state, multiple states can hold from witch actions can be performed. Definition 4.3 allows parallel actions to start from the same state because the actions must be independent to be performed in parallel. If two actions are independent they manipulate different things in the world. The state can be divided into two states with the different propositions the actions manipulate. The parallel actions can then start separately from the different states.

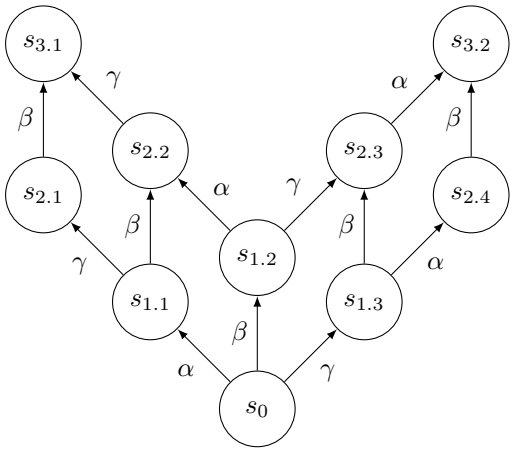
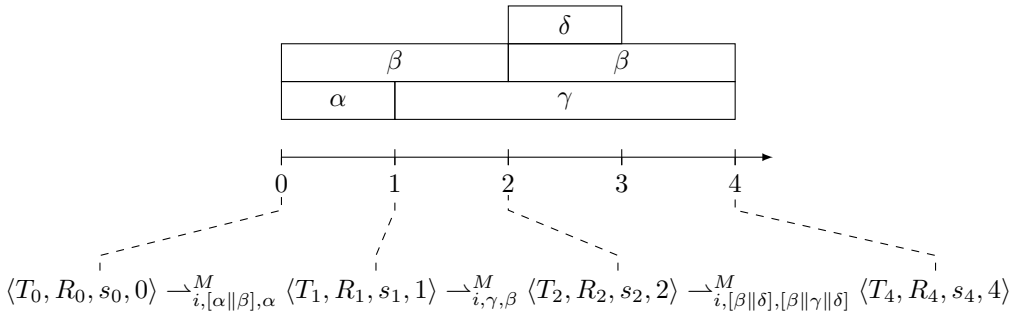


Figure 6: Transition system for the actions α, β and γ, where $I(\alpha, \beta)$ and $I(\beta, \gamma)$

When the agent is multitasking the transition is not clearly defined by one action. A transition starts with the execution of a number of actions in parallel and stops when some actions are finished. The actions that were finished are not necessarily the actions that were started in the transition. A transition is thus indicated with two types actions, the actions with which the transition is started and the action with which the transition is finished. These actions can be parallel actions. $\langle T, R, \alpha, s, \tau \rangle \rightharpoonup^{M}_{i,\mathrm{a,b}} \langle T', R', \alpha', s', \tau' \rangle$ denotes the transition from state s to state s' that starts with action a and ends after action b is finished. As before in state s action α needs to be done and in state s' action α' needs to be done. T is a set of tuples, (t, α), where α is an action that needs to be done at time t. R is a set of tuples, (t, α), where α is an action that is running and will end at time t. An example of the relation between the timeline and these transitions is visualised in Figure 7.

The transition system for parallel actions is given in definition 4.4. The function $first(T)$ returns all actions in T with the lowest time associated with them. The function $start(T, \tau)(M, s)$ returns a set of tuples, (t, α), where every tuple denotes an semi-atomic action α that is to start in the next transition and ends at time t. It is the set of all the first semi-atomic actions in T with the lowest start time. Thus all the actions start at the same time. The function $running(T, R, \tau)(M, s)$ returns a set of tuples, (t, α), where every tuple denotes a semi-atomic action that is running in the next transition. It is the union of the set actions that are already running before the next transition and the actions that started in the next transition. The function $next(T, \tau)$ returns a set of tuples, (t, α), where every tuple denotes an action α that needs to be done after the next transition. The action α should start at time t. It is the union of the set of all the actions after the first semi-atomic action in T with the lowest start time and the other actions in T. The function $stop(T, R, \tau)$ returns a set of tuples, (t, α), where every tuple denotes an action that stops running after the next transition. It is the set of all the actions that stop running before the next action starts. The function $act(T)$ returns an action where all the actions in T are done in parallel. If T only contains one tuple a single action is returned. The function $time(T)$ returns the time associated with the first action in T.



t	T_t	R_t
0	$\{(0,\alpha;\gamma),(0,\beta;[\beta\|\delta])\}$	$\emptyset$
1	$\{(1,\gamma),(2,[\beta\|\delta])\}$	$\{(2,\beta)\}$
2	$\{(2,\beta),(2,\delta)\}$	$\{(4,\gamma)\}$
4	$\emptyset$	$\emptyset$

Figure 7: A visual representation of the timeline corresponding to $[\alpha;\gamma\|\beta;[\beta\|\delta]]$ and the corresponding state transitions.

Definition 4.4. *The transitions system $T_M^{\mathcal{P}}$ is similar to the transition system T_M but has for every transition a starting and stopping action. Where $\alpha, \beta \in Ap$ and $s \in S$.*

$$\langle T, R, s, \tau\rangle \rightharpoonup^{M}_{i,\alpha,\beta} \langle T', R', s_\beta, \tau'\rangle$$

Where:

$$\begin{aligned}
\alpha &= act(start(T,\tau)(M,s))\\
\beta &= act(stop(T,R,\tau)(M,s))\\
T' &= next(T,\tau)(M,s)\\
R' &= running(T,R,\tau)(M,s) \setminus stop(T,R,\tau)(M,s)\\
\tau' &= time(stop(T,R,\tau)(M,s))
\end{aligned}$$

The rule:

$$\frac{\langle\{(t_1,\alpha_1),\ldots,(t_k,[\alpha_k\|\ldots\|\alpha_{k+m}]),\ldots,(t_n,\alpha_n)\},R,s,\tau\rangle}{\langle\{(t_1,\alpha_1),\ldots,(t_k,\alpha_k),\ldots,(t_k,\alpha_{k+m}),\ldots,(t_n,\alpha_n)\},R,s,\tau\rangle}$$

And Where:

$$\begin{aligned}
first(T) &= \{\alpha|\neg\exists(t',\alpha') \in T \text{ where } (t,\alpha)\in T, t > t'\}\\
start(T,\tau)(M,s) &= \{(t,\mathrm{a}^+)|\alpha \in first(T),\ \text{for all } s' \in S, \mathrm{a}^+ \in At^+\\
&\quad \text{that for some } \alpha' \in Ac\,\langle\alpha,s\rangle \rightharpoonup^{M}_{i,\mathrm{a}^+} \langle\alpha',s'\rangle\\
&\quad \alpha' \neq \Lambda \text{ and } t = \tau + \tau_0(i,\mathrm{a}^+)(M,s)\}\\
running(T,R,\tau)(M,s) &= start(T,\tau)(M,s) \cup R\\
next(T,\tau)(M,s) &= \{(t,\alpha')|\alpha \in first(T),\ \text{for all } s' \in S, \alpha' \in Ac\\
&\quad \text{that for some } \mathrm{a}^+ \in At^+\,\langle\alpha,s\rangle \rightharpoonup^{M}_{i,\mathrm{a}^+} \langle\alpha',s'\rangle\\
&\quad \alpha' \neq \Lambda \text{ and } t = \tau + \tau_0(i,\mathrm{a}^+)(M,s)\} \cup T \setminus first(T)\\
stop(T,R,\tau)(M,s) &= \{(t,\alpha)|\,(\neg\exists(t',\alpha') \in next(T,\tau)(M,s) \text{ where } t > t') \text{ and}\\
&\quad (t,\alpha) \in running(T,R,\tau)(M,s)\}
\end{aligned}$$

$$
\begin{aligned}
&act(\{(t,\alpha)\}) && = \alpha \\
&act(\{(t_1,\alpha_1),\ldots,(t_n,\alpha_n)\}) && = [\alpha_1 \| \ldots \| \alpha_n] \\
&time(\{(t,\alpha_1),\ldots,(t,\alpha_n)\}) && = t \\
&time(\{(t_1,\alpha_1),\ldots,(t_n,\alpha_n)\}) && = \{t | \neg\exists(t',\alpha') \in \{(t_1,\alpha_1),\ldots,(t_n,\alpha_n)\} \text{ where} \\
& && \quad (t,\alpha) \in \{(t_1,\alpha_1),\ldots,(t_n,\alpha_n)\}, t < t'\}
\end{aligned}
$$

The computation run, $\texttt{agenda}^+$ and $\texttt{agenda}^-$ and are redefined to uses the transition system with the axiom of definition 4.3. The transition actions don't need to be semi-atomic but can also be parallel semi-atomic actions, thus the actions in the agenda and in the computation run can also be parallel semi-atomic actions.

Definition 4.5. *The computation run is the sequence of actions needed to be started in order to execute an action. If $M \in \boldsymbol{M}^{\mathcal{C}}$, $\alpha_1, \alpha_2, \ldots, \alpha_n \in Ac$, $\beta_1, \beta_2, \ldots, \beta_m \in Ac$ and $s_1, \ldots, s_n \in S$, then the computation run is defined as:*

$$
\begin{aligned}
CR^{\mathcal{P}}_M(i,\alpha,s,\tau) = \{\alpha_1;\alpha_2;\ldots;\alpha_n\} \text{ iff } &\langle \{(0,\alpha)\}, \emptyset, s, \tau\rangle \rightharpoonup^M_{i,\alpha_1,\beta_1} \langle T_1, R_1, s_1, \tau_1\rangle \rightharpoonup^M_{i,\alpha_2,\beta_2} \langle T_2, R_2, s_2, \tau_2\rangle \\
&\rightharpoonup^M_{i,\alpha_3,\beta_2} \cdots \rightharpoonup^M_{i,\alpha_n,\beta_n} \langle T_n, R_n, s_n, \tau_n\rangle \text{ such that } T_n = \emptyset, R_n = \emptyset
\end{aligned}
$$

Definition 4.6. *If Agenda is the current agenda in M, then:*

$$
\begin{aligned}
\texttt{agenda}^+(i,\alpha)(M,s,\tau) = Agenda' \text{ where for all } T_k \text{and } s_k \text{ in } &\langle \{(0,\alpha)\}, \emptyset, s, \tau\rangle \rightharpoonup^M_{i,\alpha_1,\beta_1} \langle T_1, R_1, s_1, \tau_1\rangle \\
&\rightharpoonup^M_{i,\alpha_2,\beta_2} \langle T_2, R_2, s_2, \tau_2\rangle \rightharpoonup^M_{i,\alpha_3,\beta_3} \cdots \rightharpoonup^M_{i,\alpha_n,\beta_n} \langle T_n, R_n, s_n, \tau_n\rangle \\
&\text{such that } T_n = \emptyset, R_n = \emptyset, \\
&s'_k \in [s_k]_{R(i)}, Agenda'(i,s'_k) = Agenda(i,s'_k) \cup act(first(T_k))
\end{aligned}
$$

Definition 4.7. *If Agenda is the current agenda in M, then:*

$$
\begin{aligned}
\texttt{agenda}^-(i,\alpha)(M,s,\tau) = Agenda' \text{ where for all } T_k \text{and } s_k \text{ in } & \\
&\langle \{(0,\alpha)\}, \emptyset, s, \tau\rangle \rightharpoonup^M_{i,\alpha_1,\beta_1} \langle T_1, R_1, s_1, \tau_1\rangle \rightharpoonup^M_{i,\alpha_2,\beta_2} \langle T_2, R_2, s_2, \tau_2\rangle \\
&\rightharpoonup^M_{i,\alpha_3,\beta_3} \cdots \rightharpoonup^M_{i,\alpha_n,\beta_n} \langle T_n, R_n, s_n, \tau_n\rangle \text{ such that } T_n = \emptyset, R_n = \emptyset, \\
&s'_k \in [s_k]_{R(i)}, Agenda'(i,s'_k) = Agenda(i,s'_k) \backslash \\
&\left\{\beta \mid Prefix(CR^{\mathcal{C}}_M(i, act(first(T_k)), s'_k), CR^{\mathcal{C}}_M(i,\beta,s'_k,\tau'))\right\}
\end{aligned}
$$

For serial actions, the time an action takes is the sum of all the time the atomic actions take. This is no longer true when performing parallel actions. If two actions are done in parallel they take the same amount of time to complete as the longest atomic action. The result of performing parallel actions is the result of performing one of the action sequences in the trace of the action. Theorem 4.1 states that the computation run is always one of the action sequences in the trace of the action. Thus the computation run can be used in the definition of the result function. The agent is capable of doing a parallel action if it is capable of doing the semi-atomic actions that occur at the same time in parallel.

$$
\begin{aligned}
r^{at}(i,[\alpha_1\|\ldots\|\alpha_n])(M,s,\tau) = {} & (M',s',\tau') \text{ such that } (M',s') = && \text{Def. 4.8}\\
& r(i,\mathrm{CR}^{\mathcal{P}}_M(i,[\alpha_1\|\ldots\|\alpha_n],s,\tau))(M,s)\\
& \text{and } \tau' = \tau + \mathtt{MAX}(\tau_0(i,\alpha_1)(M,s),\ldots,\tau_0(i,\alpha_n)(M,s)))\\
c^{at}(i,[\alpha_1\|\ldots\|\alpha_n])(M,s,\tau) = {} & c(i,act(running(\{(0,\alpha)\},\emptyset,\tau)(M,s)\wedge && \text{Def. 4.9}\\
& c(i,act(running(T_1,R_1,\tau_1)(M,s)\wedge\cdots\wedge\\
& c(i,act(running(T_n,R_n,\tau_n)(M,s) \text{ iff}\\
& \langle\{(0,\alpha)\},\emptyset,s,\tau\rangle \rightharpoonup^M_{i,\alpha_1,\beta_1} \langle T_1,R_1,s_1,\tau_1\rangle\\
& \rightharpoonup^M_{i,\alpha_2,\beta_2} \cdots \rightharpoonup^M_{i,\alpha_n,\beta_n} \langle T_n,R_n,s_n,\tau_n\rangle
\end{aligned}
$$

Theorem 4.1. *$CR^{\mathcal{P}}_M(i,\alpha,s,\tau)$ returns a sequence of actions that is one of the action sequences in the trace of action α.*

Proof. It follows from definition 4.5 that the sequence of actions returned by $CR^{\mathcal{P}}_M(i,\alpha,s,\tau)$ are all the starting actions in the transition system $T^{\mathcal{P}}_M$ for action α. If follows from definition 4.4 that the starting actions are given by $act(start(T,\tau)(M,s))$. Thus doing all the actions in the tuple given by $start(T,\tau)(M,s)$ in parallel. It follows from definition 4.4 that $start(T,\tau)(M,s)$ returns a tuple of all the actions in T with the lowest time associated with them. The time associated with the action b is the moment in time that the action a is finished if in the previous state it held that a; b. Thus an action is only started if the previous action is finished. Thus the action b is always started after action a. Thus by definition of the trace the sequence of actions returned by $CR^{\mathcal{P}}_M(i,\alpha,s,\tau)$ is one of the action sequences in the trace of action α. □

4.3 Block World Example

Let there be an agent e in the initial state s_0 as given in Figure 8 where $\tau(s_0)=0$. The agent e has the wish to put block B_1 on to block A_1 in 3 seconds, $\mathbf{W}_e(3)\ \mathtt{on}(A_1,B_1)$. The initial selected wishes and agenda are empty, $C(e,s_0)=\emptyset$ and $\mathrm{Agenda}(e,s_0)=\emptyset$. It follows from definition 2.5 that the agent has a goal if it wishes something to be true, it is not already true, it is implementable and selected. In state s_0 block $B1$ is not on $A1$. To be able to select the wish it must be implementable.

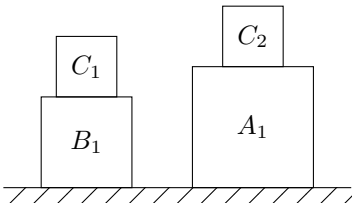


Figure 8: The starting position of the blocks in the example.

To see that the wish is implementable consider the action $\alpha = [\mathtt{drop}(C_1)\|\mathtt{drop}(C_2)]\,;\mathtt{put}(A_1,B_1)$, the timeline for this action is shown in figure 9. It follows from rule C_6 that α takes 3 seconds to perform. 1 second to move the blocks C_1 and C_2 simultaneously and 2 seconds to move block B_1. . It follows from definition 3.4 that $M,s_0,0 \models^{at} \mathbf{Before}_e(\alpha,3)$. The agent is able to drop block C_1 and C_2 and put block B_1 on to block A_1 according to rule A_1 and A_2. The agent has the opportunity to drop block C_1 and C_2 and put block B_1 on to block A_1 afterwards according to rule O_1 and O_2. The effect of dropping block C_1 and C_2 and putting block B_1 on A_1 is of course that block B_1 is on top of block A_1. It follows from definition 2.2 that $M,s_0,0 \models \mathbf{PracPoss}(\alpha,\mathtt{on}(A_1,B_1))$. Thus there is an action sequence that is practically possible and results in $\mathtt{on}(A_1,B_1)$, by definition 3.7 $\mathtt{on}(A_1,B_1)$ is implementable. The wish of agent e is implementable and doesn't already hold, according to definition 3.10 the agent is now capable to select the wish to put bock B_1 on block A_1.

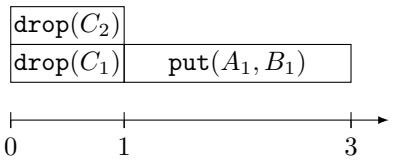


Figure 9: A visual representation of the timeline of the action $[\texttt{drop}(C_1) \| \texttt{drop}(C_2)] \,;\texttt{put}(A_1, B_1)$.

The agent performs the action $\texttt{select}(\texttt{on}(A_1, B_1), 3)$. This results in a new state s_1 where $\texttt{on}(A_1, B_1)$ is selected and the time is still 0. It follow from definition 3.2, in a similar way as in the example in section 2.3, that putting block B_1 on block A_1 is now a goal of agent e, $M, s_1, 0 \models^{at^C} \mathbf{Goal}_e(\texttt{on}(A_1, B_1))$. From using rule C_2 it follows that e also knows that it has the goal to put block B_1 on block A_1, $M, s_1, 0 \models^{at^C} \mathbf{K}_e\mathbf{Goal}_e(\texttt{on}(A_1, B_1))$.

The agent might consider action $\beta = \texttt{drop}(C_1); \texttt{drop}(C_2); \texttt{put}(A_1, B_1)$. In a similar way as shown with α, β is practically possible for agent e and will result in a state where block B_1 is on block A_1. This can however not be done within 3 seconds since the drop actions are not done in parallel, $\neg\mathbf{Before}_e(\beta, 3)$. Thus the agent e cannot possibly intend to reach its goal with action β within 3 seconds. The agent is thus unable to commit to β.

The agent is able to commit to the action α. It was already shown that $M, s_0, 0 \models^{at^C} \mathbf{Before}_e(\alpha, 3)$, it also holds in state s_1 because the time is still 0. According to rule C_7 the agent also knows that it can do the action before the end time. According to definition 3.1 it holds in state s_1 that $\mathbf{CanBefore}_e(\alpha, 3)$. It was already shown that e is able and capable to do action α, from using rules C_3 and C_4 and definitions 2.2 and 2.3 it follows that $M, s_1, 0 \models^{at} \mathbf{Can}_e(\alpha)$. It was already shown that e knows that it has the goal to put block B_1 on block A_1, it can do an action that does this before the end time, it follows from definition 3.3 that $M, s_1, 0 \models^{at^C} \mathbf{PossIntend}_e(\alpha, 3)$. It now follows from definition 3.8 that e now has the opportunity to commit to action α to fulfil its goal. Since the agenda of the agent is empty it follows from definition 3.10 that agent e also has the ability to commit to action α to fulfil its goal.

Committing to α leads to a state s_2 where α is added to the Agenda of e according to definition 4.6. According to definition 3.7 the agent is now committed to drop blocks C_1 and C_2 simultaneously. Let's suppose that the agent starts executing $[\texttt{drop}(C_1) \| \texttt{drop}(C_2)]$.

When the agent is finished dropping blocks C_1 and C_2 the agent is in state s_3. According to rule E_2 the effect of dropping the blocks is that they are on the floor and that block B_1 and block A_1 are clear. Dropping a block of type C takes according to rule C_6 1 second. The two blocks are dropped in parallel, thus the time to do them both at the same time is also 1 second. The time in state s_3 is 1 according to definition 3.4. The agent still possibly intend to put block B_1 on block A_1 with action sequence α and is thus unable to uncommit. The Agenda of e in state s_3 contains $\texttt{put}(A_1, B_1)$ because e committed to α in state s_2. According to definition 3.7 the agent is now committed to $\texttt{put}(A_1, B_1)$. Let's suppose that the agent starts to execute the action $\texttt{put}(A_1, B_1)$.

When the agent is finished with putting block B_1 on block A_1 the agent is in state s_4. According to rule E_1 the effect of $\texttt{Put}(A_1, B_1)$ is that block B_1 is on block A_1 and that block A_1 is not clear any more. The action takes according to rule C_6 2 second. By definition 3.4 the time at state s_4 is 3 second. The agent fulfilled its wish to put block B_1 on block A_1 within 3 seconds. The agent no longer has the goal to put block B_1 on block A_1 because this is already the case in state s_4.

5 Adding Time to Reasoning

Performing the planned actions is not the only thing that costs time. The hardware on which the agent performs its logical derivations also needs some time to perform these derivations. Thus all reasoning done by the agent costs time which needs to be taken into account if the agent is to achieve its wishes in time. The speed at which the agent can perform its derivations depends on the speed of the hardware the agent is running on. The same derivation can thus take a different

amount of time for different agents. Thus we need KARO with the ability to reason for a limited amount of time.

5.1 Non-standard Logic

Logical omniscience states that once an agent learns a logical statement it simultaneously knows all logically equivalent statements. Deriving all equivalent statements requires a lot of derivations, thus it takes time for the hardware to calculate all these equivalent statements. There can even be an endless amount of logically equivalent statements, deriving all these takes an endless amount of time. Only a subset of all the equivalent classes can be derived infinite time. A non-standard logic which is not logically omniscient takes a number of steps before deriving logically equivalent statements. These steps can be seen as time steps, thus deriving all the equivalent states takes a certain amount of time. The derivation process can also be stopped before all possible derivations are done. Thus the reasoning takes a certain amount of time and can be stopped after the desired result is found.

5.2 Timed Reasoning Logic

Timed Reasoning Logic(TRL) [1] gives a method to define the set of formulas the agent considers to be true at a certain time for any logic, making the logic thus not logically omniscient. This is done by dividing the deliberation into time-steps. Every state in the model is indexed by the agent and a natural number indicating the time-step to which the state belongs.

The agent has some rules to produce a new state. There are many ways in which these rules can be applied in every step. The agent can apply all rules of which the conditions are met at every step. This is called all rules at each cycle. The agent can also apply only a single rule each time-step, called single rule each cycle. This adds the difficulty of selecting the right rule from all the rules of which the conditions are met. The rules can also be continuously applied until there are no rules left for which the conditions are met. This is then repeated for each time-step. This way of applying the rules is called all rules to quiescence.

Let $\mathcal{L}_t^i$ be a set of languages where $i \in A$ and $t \in \mathbb{N}$. A TRL model M is a tuple $M = \langle obs, inf_i, \mathcal{M} \rangle$. obs is a function that for a given step gives a set of formulas in language $\mathcal{L}_t^i$ that the agent i observed at step t. inf_i is a transition function from formulas in $\mathcal{L}_t^i$ to formulas in $\mathcal{L}_{t+1}^i$. $\mathcal{M}$ is a set of a finite set of formulas, m_t^i, in $\mathcal{L}_t^i$ such that $m_{t+1}^i = inf_i(m_t^i) \cup obs(i, t+1)$.

There is a meta-logic that describes the agent's reasoning in time. If t is a time point and $i_1, \ldots, i_n, i$ are names of agents, then the meta-logic rules have the following form:

$$\frac{(i_1, t) : \phi_1, \ldots, (i_n, t) : \phi_n}{(i, t+1) : \psi}$$

5.3 TRL(KARO)

Until now it was assumed that derivations in KARO don't take up any time. Some precondition must hold before actions can be performed. Derivations are needed to check whether these preconditions hold. Thus the time to perform an action consists of two parts. The time needed to check the preconditions and the time to perform the action itself. Especially checking the preconditions of the select action can take a long time because a viable action sequence must be found.

KARO is redefined with TRL-type rules[4], making it not logically omniscient. This makes it possible to take into account the time it takes to check the preconditions of the actions needed to reach a goal. An action is actually performed by the agent when it is committed to the action and is capable of doing the action or when it is capable of doing one of the actions added in

[4]The definition given in this paper is the *all rules at each cycle* type. This results in a simple breadth search through all the derivations. For more control on the order of the derivations the *single rule each at each cycle* type can be implemented as given in [1].

the language $\mathcal{L}^{at^C}$. The actions that are added in the language $\mathcal{L}^{at^C}$ are the select, commit and uncommit actions.

The syntax of TRL(KARO) is the same as in language $\mathcal{L}^{at}$ and $\mathcal{L}^{at^C}$. Since the $\varphi?$, `if` φ `then` α `else` β `fi` and `while` φ `do` α `od` actions are not important for finding action sequences that work for one specific case they are left out of the following description.

5.3.1 Language $\mathcal{L}^{TRL}$

The class $\mathbf{M}^{TRL}$ of models contains all $M = \langle S, \pi, R, r_0, c_0, \tau_0, obs, inf_i, \mathcal{M}\rangle$. t^* indicates the time the derivation actually took to perform.

Definition 5.1. *The result function r^{at} is defined in a similar way to function r, where* $\mathrm{a} \in At$, $m_t^i \in \mathcal{M}$ *and* $r^{at}(i,\alpha)(M,\varepsilon) = M, \varepsilon$.

$$\begin{aligned}
r^{TRL}(i,\mathrm{a})(M,s,\tau) &= (M',s',\tau') \textit{ such that } (M',s') = r_0(i,\mathrm{a})(M,s)\\
&\quad \textit{and } \tau' = \tau + \tau_0(i,\mathrm{a})(M,s)\\
r^{TRL}(i,\varphi?)(M,s,\tau) &= (M,s,\tau) \textit{ if } \exists t \in \mathbb{N}((M,s,\tau)(i,t) : \varphi)\\
&\quad \textit{otherwise } M,\varepsilon\\
r^{TRL}(i,\alpha_1;\alpha_2)(M,s,\tau) &= r^{at}(i,\alpha_2,)(r^{at}(i,\alpha_1)(M,s,\tau))
\end{aligned}$$

$$\frac{(M,s,\tau)(i,t) : \varphi}{(M,s,\tau+t^*)(i,t+1) : \varphi} \quad \text{if } \varphi \text{ is not time dependent} \qquad \text{Def. 5.2}$$

$$\frac{(M,s,\tau)(i,t) : \top}{(M,s,\tau+t^*)(i,t+1) : \varphi} \text{ if } \pi(s)(\varphi) = 1 \qquad \text{Def. 5.3}$$

$$\frac{(M,s,\tau)(i,t) : \top}{(M,s',\tau+t^*)(i,t+1) : c(i,\mathrm{a})(M,s')} \quad \text{if } c_0(M,s')(i,\mathrm{a}), \mathrm{a} \in At \text{ and } (s,s') \in R \qquad \text{Def. 5.4}$$

$$\frac{(M,s,\tau)(i,t) : c(i,\alpha_1)(M,s), (M,s,\tau)(i,t) : c(i,\alpha_2)(r(i,\alpha_1)(M,s))}{(M,s,\tau+t^*)(i,t+1) : c(i,\alpha_1;\alpha_2)(M,s)} \qquad \text{Def. 5.5}$$

$$\frac{\begin{array}{c}(M,s,\tau)(i,t) : c(i, act(running(\{(0,\alpha)\},\emptyset,\tau)(M,s),\\ (M,s,\tau)(i,t) : c(i, act(running(T_1,R_1,\tau_1)(M,s),\ldots,\\ (M,s,\tau)(i,t) : c(i, act(running(T_n,R_n,\tau_n)(M,s)\end{array}}{(M,s,\tau+t^*)(i,t+1) : c(i,[\alpha_1\|\ldots\|\alpha_n])(M,s)} \qquad \text{Def. 5.6}$$

$$\text{if } \langle\{(0,\alpha)\},\emptyset,s,\tau\rangle \rightarrow^M_{i,\alpha_1,\beta_1} \langle T_1,R_1,s_1,\tau_1\rangle \rightarrow^M_{i,\alpha_2,\beta_2} \cdots \rightarrow^M_{i,\alpha_n,\beta_n} \langle T_n,R_n,s_n,\tau_n\rangle$$

$$\frac{(M,s,\tau)(i,t) : c(i,\alpha)(M,s,\tau)}{(M',s',\tau')(i,t+1) : \top} \quad \text{if } (M',s',\tau') \in r(i,\alpha)(M,s,\tau) \qquad \text{Def. 5.7}$$

The rule given in definition 5.2 describes that the reasoning of agent i is monotonic. If the agent derives something at time t it also holds at times after t. This only holds for formulas that are not dependent on completing actions before the end time and thus are derived from $\mathbf{Before}_i(\alpha, \tau_{\text{end}})$. The rules that are dependent on completing actions before the end time are $\mathbf{Before}_i(\alpha, \tau_{\text{end}})$, $\Diamond_i(\varphi, \tau_{\text{end}})$, $\mathbf{CanBefore}_i(\alpha, \tau_{\text{end}})$, $\mathbf{Goal}_i(\varphi, \tau_{\text{end}})$ and $\mathbf{PossInntend}_i(\alpha, \varphi, \tau_{\text{end}})$. The rule given in definition 5.3 describes that atomic propositional truths do not need to be derived. The rule given in definition 5.4 describes that every state in the epistemic equivalence class is capable of performing the atomic action. Definition 5.5 describes that if the agent is capable of performing two actions, the agent is capable to perform the actions in sequence. Definition 5.6 describes the definition 4.9. Definition 5.7 describes that the agent is able to make derivations in the state resulting from performing an action the agent is capable of in the current state. The rules of definitions 5.4 to 5.7 have the effect that the agent is searching breadth first through the possible action sequences.

$$\frac{(M, s, \tau)(i, t) : \langle \text{do}_i(\alpha) \rangle \top}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{O}_i \alpha} \quad \text{Def. 5.8}$$

$$\frac{(M, s, \tau)(i, t) : c(i, \alpha)(M, s)}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{A}_i \alpha} \quad \text{Def. 5.9}$$

$$\frac{(M, s, \tau)(i, t) : c(i, \alpha)(M, s), (M', s', \tau')(i, t)\varphi}{(M, s, \tau + t^*)(i, t + 1) \langle \text{do}_i(\alpha) \rangle \varphi} \quad \text{if } (M', s', \tau') \in r^{TRL}(i, \alpha)(M, s, \tau) \quad \text{Def. 5.10}$$

$$\frac{(M, s, \tau)(i, t) : c(i, \alpha)(M, s), (M, s, \tau)(i, t) : \mathbf{W}_i(\varphi, \tau_{\text{end}})}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{Before}_i(\alpha, \tau_{\text{end}})} \quad \text{if } (\tau + \pi_3(r^{TRL}(i, \alpha)(M, s, \tau)) \leq \tau_{\text{end}} \quad \text{Def. 5.11}$$

$$\frac{(M, s_1, \tau)(i, t) : \varphi, \ldots, (M, s_n, \tau)(i, t) : \varphi}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{K}_i \varphi} \quad \text{if } \{s_1, \ldots, s_n\} = [s]_{R(i)} \quad \text{Def. 5.12}$$

$$\frac{(M, s, \tau)(i, t) : \langle \text{do}_i(\alpha) \rangle \varphi, (M, s, \tau)(i, t) : \mathbf{A}_i \alpha}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{PracPoss}_i(\alpha, \varphi)} \quad \text{Def. 5.13}$$

$$\frac{(M, s, \tau)(i, t) : \mathbf{K}_i \mathbf{PracPoss}_i(\alpha, \varphi)}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{Can}_i(\alpha, \varphi)} \quad \text{Def. 5.14}$$

$$\frac{(M, s, \tau)(i, t) : \mathbf{K}_i \neg \mathbf{PracPoss}_i(\alpha, \varphi)}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{Cannot}_i(\alpha, \varphi)} \quad \text{Def. 5.15}$$

$$\frac{(M, s, \tau)(i, t) : \mathbf{K}_i \mathbf{Before}_i(\alpha, \tau_{\text{end}})}{(M, s, \tau + t^*)(i, t + 1) : \mathbf{CanBefore}_i(\alpha, \tau_{\text{end}})} \quad \text{Def. 5.16}$$

The rule in definition 5.8 describes the opportunity rule as defined in definition 2.1. The rule in definition 5.9 describes the ability rule as defined in definition 3.4. The rule in definition 5.10 describes the do rule as defined in definition 3.4. The rule in definition 5.11 reflex the before rule as defined in definition 3.4. The rule in definition 5.12 describes the knowledge rule as defined in definition 3.4. The rule in definition 5.13 describes the practically possible rule as defined in

definition 2.2. The rule in definition 5.14 describes the can rule as defined in definition 2.3. The rule in definition 5.15 describes the cannot rule as defined in definition 2.4. The rule in definition 5.16 describes the can before rule as defined in definition 3.1. The rules in definitions 5.8 to 5.16 are the rules of the language $\mathcal{L}^{at}$.

5.3.2 Language $\mathcal{L}^{TRL^{\mathcal{C}}}$

The class $\mathbf{M}^{TRL^{\mathcal{C}}}$ of models contains all $M = \langle S, \pi, R, r_0, c_0, \tau_0, W, C, \text{Agenda}, obs, inf_i, \mathcal{M}\rangle$.

Definition 5.17. *If $m_t^i \in \mathcal{M}$ and $M = \langle S, \pi, R, r_0, c_0, \tau_0, W, C, Agenda, obs, inf_i, \mathcal{M}\rangle$ then*

$$
\begin{aligned}
r^{TRL^{\mathcal{C}}}(i,\varphi)(M,s,\tau) &= r^{TRL}(i,\varphi)(M,s,\tau)\\
r^{TRL^{\mathcal{C}}}(i,\texttt{select}(\varphi,\tau_{end}))(M,s,\tau) &= (\langle S,\pi,R,r_0,c_0,\tau_0,W,\texttt{choose}(i,\varphi,\tau_{end})(M,s),\\
&\quad Agenda, obs, inf_i, \mathcal{M}\rangle, s, \tau)\\
&\quad \text{if } \exists t \in \mathbb{N}((M,s,\tau)(i,t) : \boldsymbol{W}_i(\varphi,\tau_{end})) \text{ and}\\
&\quad \emptyset \text{otherwise}\\
r^{TRL^{\mathcal{C}}}(i,\texttt{commit}\ \alpha)(M,s,\tau) &= (\langle S,\pi,R,r_0,c_0,\tau_0,W,C,\texttt{agenda}^{+}(i,\alpha)(M,s),\\
&\quad obs, inf_i, \mathcal{M}\rangle, s, \tau) \text{ if}\\
&\quad \exists t \in \mathbb{N}((M,s,\tau)(i,t) : \boldsymbol{PossIntend}_i(\alpha,\varphi,\tau_{end}))\\
&\quad \text{for all } (\varphi,\tau_{end}) \in C(i,s) \text{ and } \emptyset \text{ otherwise}\\
r^{TRL^{\mathcal{C}}}(i,\texttt{uncommit}\ \alpha)(M,s,\tau) &= (\langle S,\pi,R,r_0,c_0,\tau_0,W,C,\texttt{agenda}^{-}(i,\alpha)(M,s),\\
&\quad obs, inf_i, \mathcal{M}\rangle, s, \tau) \text{ if}\\
&\quad \exists t \in \mathbb{N}((M,s,\tau)(i,t) : \boldsymbol{Com}_i\alpha) \text{ and } \emptyset \text{ otherwise}
\end{aligned}
$$

$$\frac{(M,s,\tau)(i,t) : \mathbf{W}_i\varphi, (M,s,\tau)(i,t) : \neg\varphi, (M,s,\tau+t^*)(i,t) : \Diamond_i(\varphi,\tau_{\text{end}})}{(M,s,\tau)(i,t+1) : c(i,\texttt{select}(\varphi,\tau_{\text{end}}))} \quad \text{Def. 5.18}$$

$$\frac{(M,s,\tau)(i,t) : \mathbf{PosIntend}_i(\alpha,\varphi,\tau_{\text{end}})}{(M,s,\tau+t^*)(i,t+1) : c(\texttt{commit}\alpha)(M,s)} \quad \text{if } \exists(\varphi,\tau_{\text{end}}) \in C(i,s) \quad \text{Def. 5.19}$$

$$\frac{(M,s,\tau)(i,t) : \neg\mathbf{PossIntend}_i(\alpha,\varphi,\tau_{\text{end}}), (M,s,\tau)(i,t) : \mathbf{Com}_i\alpha}{(M,s,\tau+t^*)(i,t+1) : c(\texttt{uncommit}\alpha)(M,s)} \quad \forall(\varphi,\tau_{\text{end}}) \in C(i,s) \quad \text{Def. 5.20}$$

Definition 5.18 describes when the agent is capable of performing the select action. Definition 5.19 describes when the agent is capable of performing the commit action. Definition 5.20 describe when the agent is capable of performing the uncommit action

$$\frac{(M,s,\tau)(i,t) : \top}{(M,s,\tau+t^*)(i,t+1) : \mathbf{W}_e} \quad \text{if } \forall s' \in S((s,s') \in W(i) \Rightarrow \pi(s')(\varphi) = 1) \quad \text{Def. 5.21}$$

$$\frac{(M,s,\tau)(i,t) : \mathbf{PracPoss}_i(\alpha,\varphi), (M,s,\tau) : \mathbf{Before}_i(\alpha,\tau_{\text{end}})}{(M,s,\tau+t^*)(i,t+1) : \Diamond_i(\varphi,\tau_{\text{end}})} \quad \text{Def. 5.22}$$

$$\frac{(M,s,\tau)(i,t) : \top}{(M,s,\tau+t^*)(i,t+1) : \mathbf{C}_i(\varphi,\tau_{\text{end}})} \quad \text{if } (\varphi,\tau_{\text{end}}) \in C(i,s) \quad \text{Def. 5.23}$$

$$\frac{(M,s,\tau)(i,t):\top}{(M,s,\tau+t^*)(i,t+1):\mathbf{Com}_i\varphi} \quad \begin{array}{l}\text{if } \forall s'\in[s]_{R(i)}\,\exists\alpha_1\in \mathrm{CR}^{\mathcal{C}}_M(i,\alpha,s')\exists\alpha_2\in\mathrm{Agenda}(i,s')\\ \exists\alpha_2'\in\mathrm{CR}^{\mathcal{C}}_M(i,\alpha_2,s')(\mathrm{Prefix}(\alpha_1,\alpha_2'))\end{array} \quad \text{Def. 5.24}$$

$$\frac{(M,s,\tau)(i,t):\mathbf{W}_i(\varphi,\tau_{\mathrm{end}}),(M,s,\tau)(i,t):\neg\varphi,(M,s,\tau)(i,t):\Diamond_i(\varphi,\tau_{\mathrm{end}}),(M,s,\tau)(i,t):\mathbf{C}_i(\varphi,\tau_{\mathrm{end}})}{(M,s,\tau+t^*)(i,t+1):\mathbf{Goal}_i(\varphi,\tau_{\mathrm{end}})} \quad \text{Def. 5.25}$$

$$\frac{(M,s,\tau)(i,t):\mathbf{Can}_i(\alpha,\varphi),(M,s,\tau)(i,t):\mathbf{K}_i\mathbf{Goal}_i(\varphi,\tau_{\mathrm{end}}),(M,s,\tau)(i,t):\mathbf{CanBefore}_i(\alpha,\tau_{\mathrm{end}})}{(M,s,\tau+t^*)(i,t+1):\mathbf{PossIntend}_i(\alpha,\varphi,\tau_{\mathrm{end}})} \quad \text{Def. 5.26}$$

The rule definition 5.21 describes the wish rule as defined in definition 2.12. The rule in definition 5.22 describes the implementability rule as defined in definition 3.7. The rule in definition 5.23 describes the selected rule as defined 3.7. The rule in definition 5.24 describes the commitment rule as defined in definition 3.7. The rule in definition 5.25 describes the goal rule as defined in definition 3.2. The rule in definition 5.26 describes the possibly intends rule as defined in definition 3.3. The rules in definitions 5.21 to 5.26 are the rules of the language $\mathcal{L}^{at^{\mathcal{C}}}$.

Following the distribution axiom of epistemic logic the following rule can also be derived from 5.13:

$$\frac{(M,s,\tau)(i,t):\mathbf{K}_i\mathbf{A}_i\alpha,(M,s,\tau)(i,t):\mathbf{K}_i\langle\mathrm{do}_i(\alpha)\rangle\varphi}{(M,s,\tau+t^*)(i,t+1):\mathbf{K}_i\mathbf{PracPoss}_i(\alpha,\varphi)} \quad \text{Def. 5.27}$$

5.4 Block World Example

In this example TRL(KARO) is used to plan actions in the block world as described in 2.3. Some translations of the rules given in section 2.3 and 3.3 are given below:

$$\frac{(M,s,\tau)(i,t):\mathbf{Goal}_i\varphi}{(M,s,\tau+t^*)(i,t+1):\mathbf{K}_i\mathbf{Goal}_i\varphi} \quad \text{Def. 5.28}$$

$$\frac{(M,s,\tau)(i,t):\mathbf{A}_i\alpha}{(M,s,\tau+t^*)(i,t+1):\mathbf{K}_i\mathbf{A}_i\alpha} \quad \text{Def. 5.29}$$

$$\frac{(M,s,\tau)(i,t):\langle\mathrm{do}_i(\alpha)\rangle\varphi}{(M,s,\tau+t^*)(i,t+1):\mathbf{K}_i\langle\mathrm{do}_i(\alpha)\rangle\varphi} \quad \text{Def. 5.30}$$

$$\frac{(M,s,\tau)(i,t):\mathbf{Before}_i(\alpha,\tau_{\mathrm{end}})}{(M,s,\tau+t^*)(i,t+1):\mathbf{K}_i\mathbf{Before}_i(\alpha,\tau_{\mathrm{end}})} \quad \text{Def. 5.31}$$

Let there be an agent e in the initial state s_0 as given in figure 10 where $\tau(s_0) = 0$. The agent e has the wish to make a block A_1 clear within three seconds, $\mathbf{W}_e(\texttt{is\_clear}(A_1), 3)$. The initial selected wishes and agenda are empty, $C(e, s_0) = \emptyset$ and $\text{Agenda}(e, s_0) = \emptyset$. There are no observations and the inference rules are the rules given above. It is assumed that every derivation takes 0.1 seconds, $t^* = 0.1$.

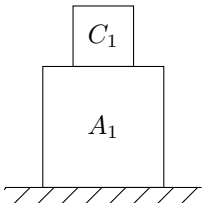


Figure 10: The staring position of the blocks in the example.

Figure 11 gives the derivations needed to show that the agent e is capable to select $\texttt{is\_clear}(A_1)$. For readability, the process is divided into smaller parts, connected with arrows and only the relevant formulas are shown. The arrows can also represent multiple applications of the monotonicity rule given in definition 5.2.

The agent is capable to do the action $\texttt{select}(\texttt{is\_clear}(A_1), 3)$. The agent executes the select action resulting in a model where $(M, s_1, 0.7)(e, 7) : \mathbf{C}_e(\texttt{is\_clear}(A_1), 3)$. In this model the agent can make new derivations, as shown in figure 12.

The agent is capable to commit execute the action drop block C_1 in order to make block A_1 clear, $\texttt{commit drop}(C_1)$. The agent performs the commit action resulting in a model where the agenda is updated with the action of dropping block C_1, according to definition 5.17. The agent can now apply de rule given in definition 5.24 as shown in figure 13.

The agent is now committed to dropping block C_1 and is also capable of dropping the block. The agent will thus execute the dropping action resulting in a model where the wish of the agent is fulfilled. As defined in rule C_6, the action of dropping a block of type C takes one second. The reasoning took 1.9 seconds, thus block A_1 will be clear at 2.9 seconds. This is before the end time of 3 seconds.

The agent would fail to commit itself to the wish of clearing block A_1 before 2.9 seconds. One of the aplications of the before rules given in definition 5.11 would fail. Without taking the (in this example slow) reasoning into account the agent would be able to drop block C_1 and clear block A_1 in one second. But the reasoning takes too long to allow for an action to be performed in time.

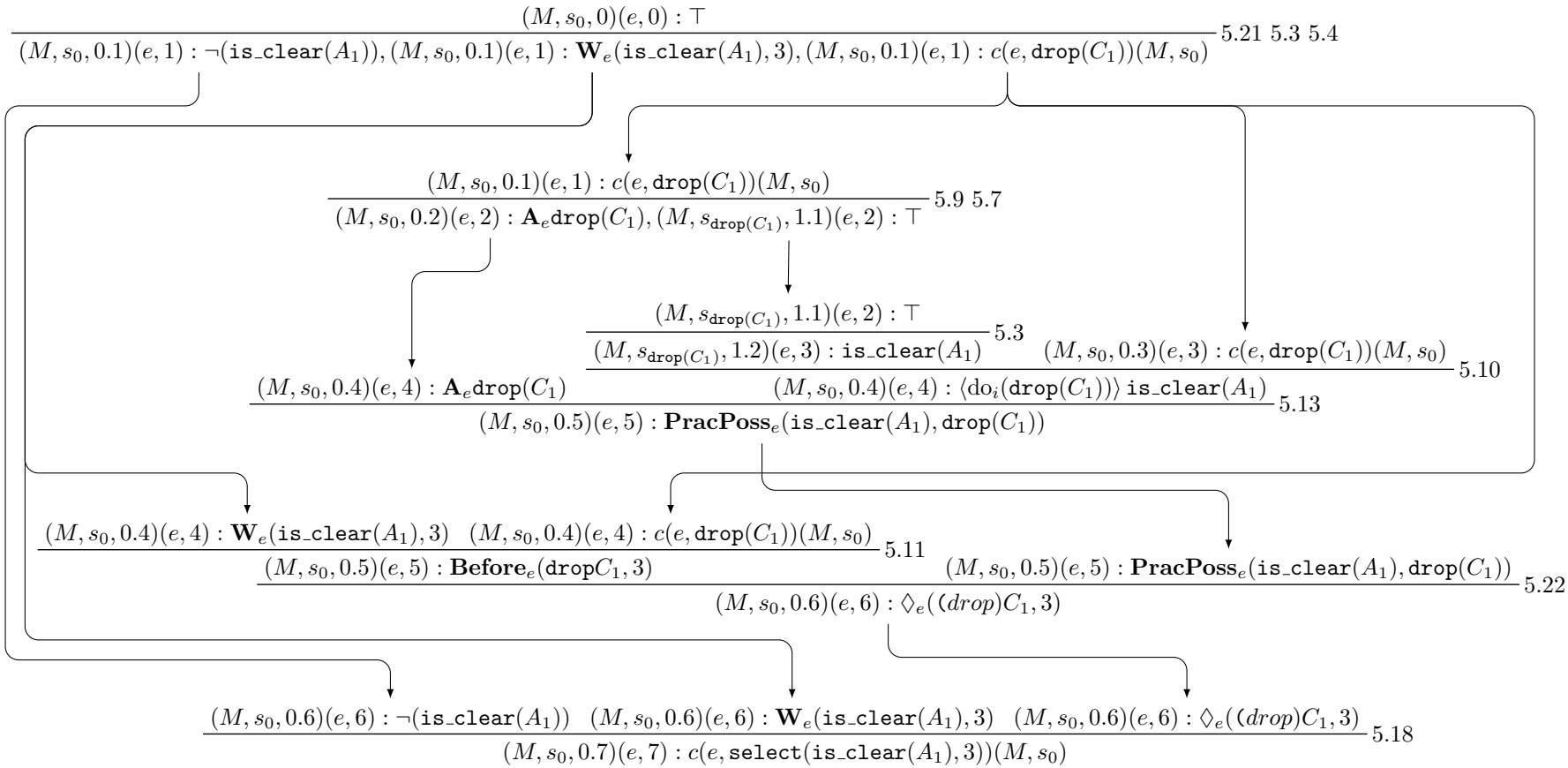


Figure 11: The rules used in TRL(KARO) to derive that agent e is capable to perform the action $\texttt{select}(\texttt{is\_clear}(A_1), 3)$.

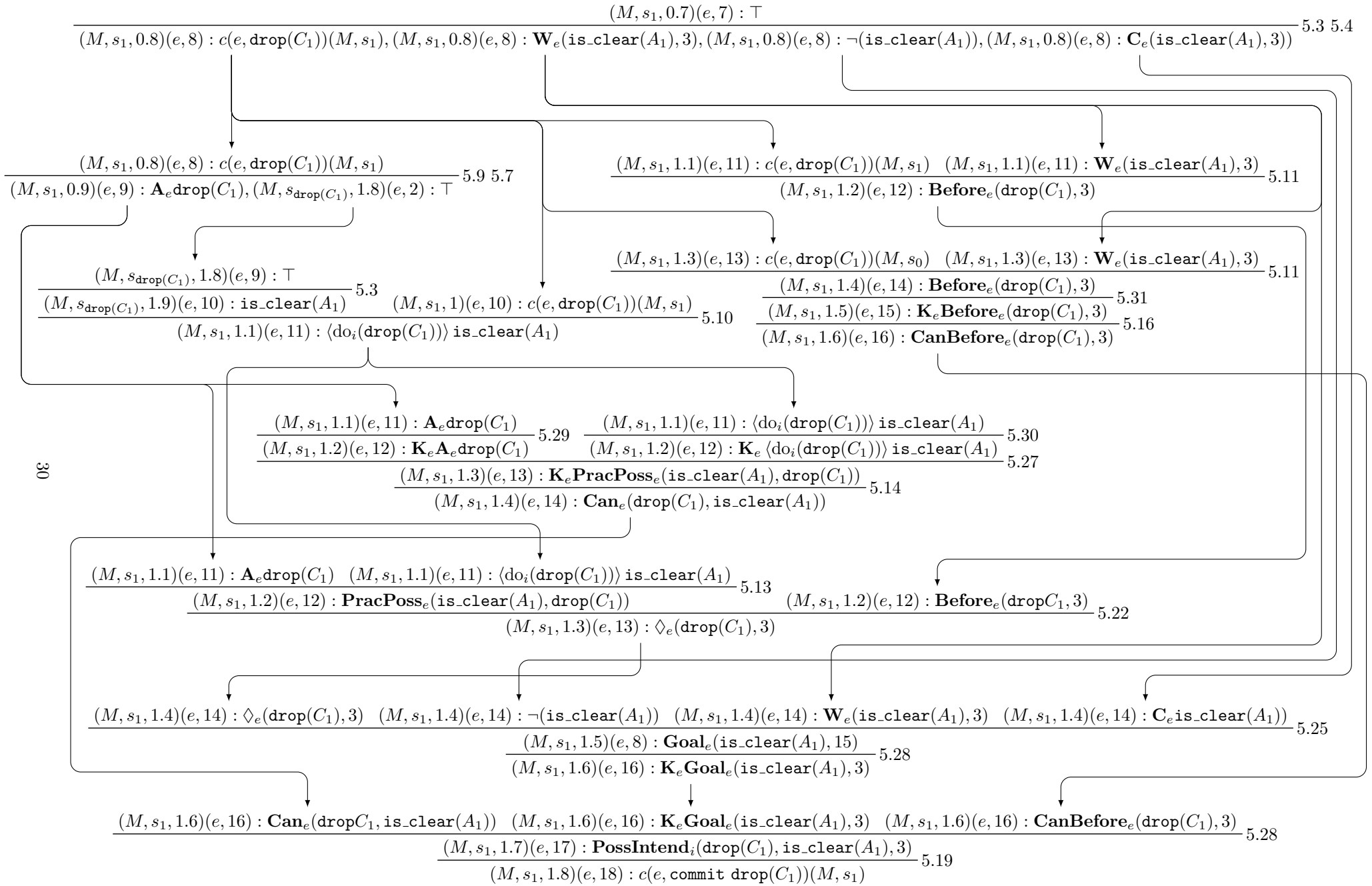


Figure 12: The rules used in TRL(KARO) to derive that agent e is capable to perform the action $\texttt{commit drop}(C_1)$.

$$\frac{(M, s_2, 1.8)(e, 18) : \top}{(M, s_2, 1.9)(e, 19) : \mathbf{Com}_e\texttt{drop}(C_1), (M, s_2, 1.9)(e, 19) : c(e, \texttt{drop}(C_1))(M, s_2)}\ 5.24\ 5.4$$

Figure 13: The rules used in TRL(KARO) to derive that agent e is committed and is capable to perform the action $\texttt{drop}(C_1)$.

6 Discussion

The goal operator of the KARO framework was in this paper extended with an end time before which the goal must be reached. Every action takes a certain amount of time to complete. The action sequence planned by the agent must thus not take so long that it finishes after the end time of the goal. A before operator was introduced to indicate that an action can be performed before a certain time. Parallel actions were also introduced to the KARO framework. Performing actions in parallel can be a lot faster than doing the actions after each other. This gives the agent more options for achieving the goal within the given time. There are also actions in KARO that are not about changing the environment but changes in the state of the agent. For example, the commit action that commits an agent to a certain action. These actions also take a certain amount of time to complete, because the agent needs to reason whether the preconditions of the actions hold. Especially finding a suitable action sequence to fulfil the goal can take a lot of time. Timed reasoning logic is used let KARO make only a limited number of derivations every time step, making KARO not logically omniscient. This allows derivations to be done in a finite amount of time.

The given extensions to the KARO framework make it possible to build a real world agent that is able to perform with a limited amount of time and computing power.

References

[1] Natasha Alechina, Brian Logan, and Mark Whitsey. A complete and decidable logic for resource-bounded agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 606–613. IEEE Computer Society, 2004.

[2] Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

[3] Antoni Mazurkiewicz. Introduction to trace theory. *The Book of Traces*, pages 3–41, 1995.

[4] J-J Ch Meyer, Wiebe van der Hoek, and Bernd van Linder. A logical approach to the dynamics of commitments. *Artificial Intelligence*, 113(1):1–40, 1999.

[5] Gordon D Plotkin. *A structural approach to operational semantics*. Computer Science Department, Aarhus University Aarhus, Denmark, 1981.

[6] Wiebe van der Hoek, Bernd van Linder, and J-J Ch Meyer. A logic of capabilities. In *International Symposium on Logical Foundations of Computer Science*, pages 366–378. Springer, 1994.

[7] Wiebe van der Hoek, Bernd van Linder, and John-Jules Ch Meyer. An integrated modal approach to rational agents. In *Foundations of rational agency*, pages 133–167. Springer, 1999.

[8] B Van Linder, W van der Hoek, and John-Jules Ch Meyer. *Tests as epistemic updates: Pursuit of knowledge*. Utrecht University, Department of Computer Science, 1994.

[9] Bernd Van Linder, Wiebe van der Hoek, and J-J Ch Meyer. Actions that make you change your mind. In *Annual Conference on Artificial Intelligence*, pages 185–196. Springer, 1995.

[10] Bernd van Linder, Wiebe van der Hoek, and J-J Ch Meyer. The dynamics of default reasoning. *Data & knowledge engineering*, 21(3):317–346, 1997.

[11] Bernd van Linder, Wiebe van der Hoek, and J-J Ch Meyer. Formalising abilities and opportunities of agents. *Fundamenta Informaticae*, 34(1, 2):53–101, 1998.