KRR Project - Documentation

Dmytro Narepekha
Jianhao Luo
Suzzane Abiri
Sabrila Trifebi Shina
Warsaw University of Technology
Faculty of Computer Science and Information Systems

April 22, 2024

Abstract

The aim of the task was to define an action description language $\mathcal{ADL}_{\mathcal{AGP}}$ and corresponding query language $\mathcal{QL}_{\mathcal{AGP}}$ that will allow describing a specific group of dynamic systems that takes into account agent-action-program related aspects. Accordingly, this contribution consists of two parts - the theoretical description of the task and the documentation of its practical implementation. In the first part, the introduction provides a brief overview of the task, including the assumptions applied to the corresponding group of dynamic systems. Then, the action description language is defined by means of its syntax and semantics. Following that, the query language is discussed in terms of proposed queries and their satisfiability conditions. Lastly, the theory is concluded with descriptions of various examples that provide an understanding of how the proposed languages can be applied in practice. The second part focuses on the description of the technical aspects of the implemented solution – i.e. used technology, implemented interface and the performed tests.

Contents

1 Theoretical description

1.1 Introduction

Let DS_{AGP} be a class of dynamic systems that are the primary focus of this task. DS_{AGP} fulfills the following assumptions:

A1. **Inertia law** – the observed changes, after performing a given action, are only the ones that are directly or indirectly induced by the aforementioned action

A2. **Complete information about all actions and all fluents** – there are no unknown effects of the actions

A3. **Determinism** – there is only one possible outcome of an action

A4. **Only sequential actions are allowed** – the parallel actions are out of consideration

A5. **Characteristics of actions**:

- ▷ precondition (a set of literals) reflecting a condition under which the action starts and leads to some effect. If a precondition does not hold, then the action is executed with an empty effect
- ▷ postcondition (a set of literals) reflecting the effect of the action
- ▷ agent who performs an action

A6. **State-Dependent Action Constraints for Agents** – In some states (specified by a propositional formula) some actions cannot be performed by specific agents.

A7. **Partial descriptions** – Partial descriptions of any state of the system (including the initial one) are allowed.

1.1.1 Task description

The task of this project is stated as follows:

> *Define an action description language* ADL$_{\text{AGP}}$ *for representing dynamic systems of the class specified above, and define the corresponding query language* QL$_{\text{AGP}}$ *which would allow getting answers for the following queries:*
>
> *Q1. Does a condition γ (set of literals) hold after performing a program $\mathcal{P}$?*
>
> *Q2. Is an agent ag active in executing a program $\mathcal{P}$?*

1.2 Action description language

Let $\mathcal{ADL}_{\mathcal{AGP}}$ be an action description language defined to represent a class of dynamic systems that satisfy the assumptions outlined in the previous section. The following terminology will be used to specify the syntax of the $\mathcal{ADL}_{\mathcal{AGP}}$ language:

$\Upsilon = (\mathcal{F}, \mathcal{A}_c, \mathcal{A}_g)$ - signature of the language, where:

- $\mathcal{F}$ is a set of fluents
- $\mathcal{A}_c$ is a set of actions
- $\mathcal{A}_g$ is a set of agents

$\bar{f}$ - literal corresponding to the fluent $f \in \mathcal{F}$ or its negation $\neg f$

$\mathcal{P} = ((A_1, X_1), \ldots, (A_n, X_n))$ - program, sequence of actions corresponding their agents.

1.2.1 Syntax

Two types of statements were defined for the $\mathcal{ADL}_{\mathcal{AGP}}$ language. The first ones are the **value statements**. They describe the state (more precisely, fluents) that initially holds in the system or holds in the system after performing a particular sequence of actions. The second types of statement are the **effect statements**. They describe how the system's state changes after performing a given action, namely, what state will result from performing a given action under specified preconditions.

1. Value statements

It is important to stress that the second value statement is, in fact, an abbreviation of the first one (i.e. it is not a separate value statement).

1. $\bar{f}_1, \ldots, \bar{f}_k$ **after** $(A_1, X_1), \ldots, (A_n, X_n)$ (where $f_1, \ldots, f_k \in \mathcal{F}$, $A_1, \ldots, A_n \in \mathcal{A}_c$, $X_1, \ldots, X_n \in \mathcal{A}_g$)

   Meaning: $\bar{f}_1, \ldots, \bar{f}_k$ holds after performing a set of actions $(A_1, X_1), \ldots, (A_n, X_n)$ in the initial state

2. **initially** $\bar{f}_1, \ldots, \bar{f}_k$

   Meaning: $\bar{f}_1, \ldots, \bar{f}_k$ holds in the initial state

2. Effect statements

Similarly to the previous section, the second effect statement is an abbreviation of the first one. The second emphasizes that the action is executed with no specific preconditions that have to be met before its execution.

1. A **causes** $\bar{f}_1, \ldots, \bar{f}_k$ by agent **X** if $\bar{g}_1, \ldots, \bar{g}_k$ where $f_1, \ldots, f_k \in \mathcal{F}$, $g_1, \ldots, g_k \in \mathcal{F}$ $A \in \mathcal{A}_c$ $a \in \mathcal{A}_g$

   Meaning: If A is performed in any state satisfying $\bar{f}_1, \ldots, \bar{f}_k$ by agent a, then in the resulting state $\bar{f}_1, \ldots, \bar{f}_k$ holds.

2. A by agent **X causes** $\bar{f}_1, \ldots, \bar{f}_k$ where $f_1, \ldots, f_k \in \mathcal{F}$

   Meaning: The performance of action A by agent a leads to the state for which α holds

1.2.2 Semantics

1. Transition function

A **transition function** is defined as a mapping $\Psi\colon \mathcal{A}_c \times \mathcal{A}_g \times \Sigma \to \Sigma$.
$\Psi(A, X, \sigma)$ defined for any $A \in \mathcal{A}_c$, for any $\sigma \in \Sigma$ and for any $X \in \mathcal{A}_g$ gives a resulting state after performing the action A from the state σ by agent X.

The aforementioned function can also be generalized to the mapping $\Psi^*\colon (\mathcal{A}_c \times \mathcal{A}_g)^* \times \Sigma \to \Sigma$ in a following way:

1. $\Psi^*(\epsilon, \sigma) = \sigma$

2. $\Psi^*(((A_1, X_1), \ldots, (A_n, X_n)), \sigma) = \Psi((A_n, X_n), \Psi^*(((A_1, X_1), \ldots, (A_{n-1}, X_{n-1})), \sigma))$

However, for the sake of simplification, the Ψ^* will be further denoted as Ψ. The **structure** of a language $\mathcal{ADL}_{\mathcal{CST}}$ is a pair $S = (\Psi, \sigma_0)$ where Ψ is a transition function and

$\sigma_0 \in \Sigma$ is the initial state.

A **state** is defined as a mapping $\sigma\colon \mathcal{F} \to \{0,1\}$, for any $f \in \mathcal{F}$.
If $\sigma(f) = 1$, then it means that f holds in the state σ and it is denoted by $\sigma \models f$.
If $\sigma(f) = 0$, then it means that f doesn't hold in the state σ and it is denoted $\sigma \models \neg f$.
Furthermore, let Σ denote a set of all states.

2. Satisfiability of language statements

Let $S = (\Psi, \sigma_0)$ be a structure for a language $\mathcal{ADL}_{\mathcal{AGP}}$. A statement s is true in S (denoted by $S \models s$), if and only if:

1. s is of the form: $\bar{f}_1, \dots, \bar{f}_k$ **after** $(A_1, X_1), \dots, (A_n, X_n)$

   $S \models s$ if and only if for an initial state $\sigma_0 \in \Sigma$ and the set of actions $A_1, \dots, A_n \in \mathcal{A}_c$, the set of agents $X_1, \dots, X_n \in \mathcal{A}_g$ the following holds: $\Psi^*(((A_1, X_1), \dots, (A_n, X_n)), \sigma_0) \models \bar{f}_1, \dots, \bar{f}_k$

2. s is of the form: A **causes** $\bar{f}_1, \dots, \bar{f}_k$ by agent X if $\bar{g}_1, \dots, \bar{g}_k$

   $S \models s$ if and only if for every state $\sigma \in \Sigma$ such that $\sigma \models \bar{g}_1, \dots, \bar{g}_k$ and the action $A \in \mathcal{A}_c$, the agent $X \in \mathcal{A}_g$, the following holds: $\Psi^*(A, X, \sigma) \models \bar{f}_1, \dots, \bar{f}_k$

3. Model of a language

Let D be an action domain (non-empty set of value or effect statements) in the language $\mathcal{ADL}_{\mathcal{AGP}}$ over a signature $\Upsilon = (\mathcal{F}, \mathcal{A}_c, \mathcal{A}_g)$. A structure for this language $S = (\Psi, \sigma_0)$ is a model of D if and only if:

1. for every $s \in D$, $S \models s$

2. for every $A \in \mathcal{A}_c$, for every $\bar{f}_1, \dots, \bar{f}_k, g_1, \dots, g_k \in \mathcal{F}$, for every $\sigma \in \Sigma$, for every $X \in \mathcal{A}_\}$ if one of the following conditions holds:

   (a) there is a statement in D of the form A **causes** $\bar{f}_1, \dots, \bar{f}_k$ by agent X if $\bar{g}_1, \dots, \bar{g}_k$ such that $\sigma \not\models \bar{f}_i$ for some $i = 1, \dots, k$

   (b) D does not contain an effect statement A **causes** $\bar{f}$ if $\bar{g}_1, \dots, \bar{g}_k$

   then $\sigma \models \bar{f}_1, \dots, \bar{f}_k$ if and only if $\Psi(A, a, \sigma) \models \bar{f}_1, \dots, \bar{f}_k$

1.3 Query Language

Let $\mathcal{QL}_{\mathcal{AGP}}$ be the query language corresponding to the $\mathcal{ADL}_{\mathcal{AGP}}$.

1.3.1 Query statements

There were two types of queries defined for $\mathcal{QL}_{\mathcal{CST}}$, namely the **value queries** and **active queries**. The value queries answer the question of whether a given condition is satisfied after performing a sequence of actions. The active queries evaluate if a given agent is active for execution of sequence of actions.

1. Value queries

The value query is defined in the following way:

$\bar{f}_1, \dots, \bar{f}_k$ **after** $(A_1, X_1), \dots, (A_n, X_n)$

Meaning: does the condition $\bar{f}_1, \dots, \bar{f}_k$ hold after executing the sequence of actions

$(A_1, X_1), \dots, (A_n, X_n)$

2. Active queries

active X in $(A_1, X_1), \dots, (A_n, X_n)$

Meaning: is the agent X active in the program $(A_1, X_1), \dots, (A_n, X_n)$?

1.3.2 Satisfiability of queries

Let D be an action domain. A query Q is a consequence of D, denoted by $D \models Q$, if and only if Q is true in every model S=(Ψ, σ_0) of D:

1. Q is of the form: $\bar{f}_1, \dots, \bar{f}_k$ **after** $(A_1, X_1), \dots, (A_n, X_n)$

   $D \models Q$ if and only if for any model $S = (\Psi, \sigma_0)$ of D, the following holds $\Psi(((A_1, X_1), \dots, (A_n, X_n)), \sigma_0) \models \bar{f}_i$ for every i $= 1, \dots, k$

2. Q is of the form: **active** X in $(A_1, X_1), \dots, (A_n, X_n)$

   $D \models Q$ if and only if for any model $S = (\Psi, \sigma_0)$ of D there exists i =1, 2, 3 ..., j, such that

   (a) X $= X_i$

   (b) $\sigma_i \neq \sigma_{i-1}$ and where $\sigma_j = \Psi((A_1, X_1), \dots, (A_j, X_j), \sigma_0)$ $j = 1, \dots, n$

2 Examples of dynamic systems

2.0.1 Online store Scenario

An online store sells housing materials.The following literals can describe the status of the product: ordered, shipped and delivered. Initially, the product is neither ordered nor shipped. Purchasing the product makes it ordered, shipping the product makes it shipped, and, lastly, delivering the product makes it delivered. Each action has an associated agent attached if the initial conditions for performing a specific action are met.

Fluent	**literal**	**nagation**
f_1	Ordered	¬Ordered
f_2	Shipped	¬Shipped
f_3	Delivered	¬Delivered

Actions are defined as:

1. Purchase
2. Ship
3. Deliver

Agents:

1. **Buyer**: Can only perform action: **Purchase**
2. **Seller**: Can only perform action: **Ship** and **Deliver**

Agents perform undefined actions will have no effect. The set of all considered states in this scenario is defined as:

$\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$, where:

$$\sigma_0 = \{\neg\text{ordered}, \neg\text{shipped}, \neg\text{delivered}\}$$
$$\sigma_1 = \{\text{ordered}, \neg\text{shipped}, \neg\text{delivered}\}$$
$$\sigma_2 = \{\neg\text{ordered}, \text{shipped}, \neg\text{delivered}\}$$
$$\sigma_3 = \{\text{ordered}, \text{shipped}, \text{delivered}\}$$
$$\sigma_4 = \{\neg\text{ordered}, \neg\text{shipped}, \text{delivered}\}$$
$$\sigma_5 = \{\neg\text{ordered}, \text{shipped}, \text{delivered}\}$$
$$\sigma_6 = \{\text{ordered}, \neg\text{shipped}, \text{delivered}\}$$
$$\sigma_7 = \{\neg\text{ordered}, \text{shipped}, \neg\text{delivered}\}$$

The structure of the scenario is as follows:

initially $\neg$*ordered*
initially $\neg$*shipped*
initially $\neg$*delivered*
Purchase by *Buyer* **causes** *ordered* **if** $\neg$*ordered* $\neg$*shipped* $\neg$*delivered*
Ship by *Seller* **causes** *shipped* **if** *ordered*, $\neg$*shipped* $\neg$*delivered*
Deliver by *Seller* **causes** *delivered* **if** *shipped*, *ordered*, $\neg$*delivered*

Hence, the following states are considered in this case:

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_0) = \sigma_1$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_0) = \sigma_0$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_0) = \sigma_0$
$\Psi((\text{Ship}, \text{Seller}), \sigma_0) = \sigma_0$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_0) = \sigma_0$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_0) = \sigma_0$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_1) = \sigma_1$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_1) = \sigma_1$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_1) = \sigma_1$
$\Psi((\text{Ship}, \text{Seller}), \sigma_1) = \sigma_2$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_1) = \sigma_1$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_1) = \sigma_1$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_2) = \sigma_2$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_2) = \sigma_2$
$\Psi((\text{Ship}, \text{Seller}), \sigma_2) = \sigma_2$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_2) = \sigma_2$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_2) = \sigma_2$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_2) = \sigma_3$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_3) = \sigma_3$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_3) = \sigma_3$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_3) = \sigma_3$
$\Psi((\text{Ship}, \text{Seller}), \sigma_3) = \sigma_3$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_3) = \sigma_3$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_3) = \sigma_3$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_4) = \sigma_4$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_4) = \sigma_4$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_4) = \sigma_4$
$\Psi((\text{Ship}, \text{Seller}), \sigma_4) = \sigma_4$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_4) = \sigma_4$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_4) = \sigma_4$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_5) = \sigma_5$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_5) = \sigma_5$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_5) = \sigma_5$
$\Psi((\text{Ship}, \text{Seller}), \sigma_5) = \sigma_5$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_5) = \sigma_5$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_5) = \sigma_5$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_6) = \sigma_6$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_6) = \sigma_6$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_6) = \sigma_6$
$\Psi((\text{Ship}, \text{Seller}), \sigma_6) = \sigma_6$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_6) = \sigma_6$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_6) = \sigma_6$

$\Psi((\text{Purchase}, \text{Buyer}), \sigma_7) = \sigma_7$
$\Psi((\text{Purchase}, \text{Seller}), \sigma_7) = \sigma_7$
$\Psi((\text{Ship}, \text{Buyer}), \sigma_7) = \sigma_7$
$\Psi((\text{Ship}, \text{Seller}), \sigma_7) = \sigma_7$
$\Psi((\text{Deliver}, \text{Buyer}), \sigma_7) = \sigma_7$
$\Psi((\text{Deliver}, \text{Seller}), \sigma_7) = \sigma_7$

Figure 1 outlines all of the aforementioned transitions.

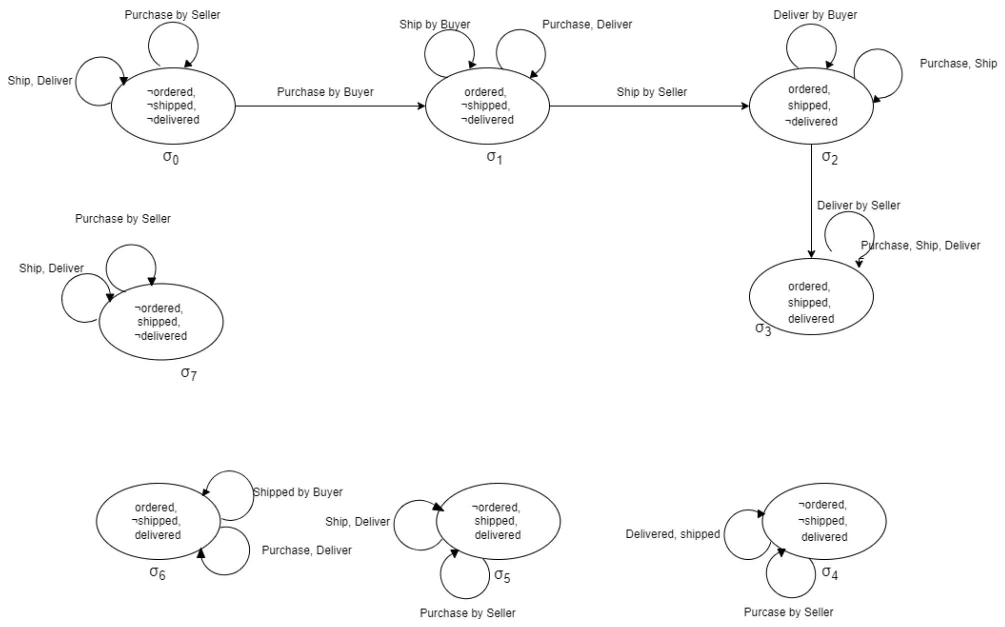

Figure (1) Shipping orders - all possible transitions

Let us demonstrate an example of asking queries using the structure of the query language $\mathcal{QL}_{\mathcal{CST}}$. Assume, that the initial state is σ_0 and the program $\mathcal{P}$ = ((*Purchase*, *Buyer*),(*Ship*, *Seller*), (*Deliver*, *Seller*). Let us consider the following query statements:

1. ¬*delivered* **after** ((*Purchase*, *Buyer*),(*Ship*, *Seller*), (*Deliver*, *Seller*)).
2. **Active** Seller **in** ((*Purchase*, *Buyer*),(*Ship*, *Seller*), (*Deliver*, *Seller*)).

The program execution will consist of the following transitions:

$$\Psi((\text{Purchase}, Buyer), \sigma_0) = \sigma_1$$
$$\Psi((\text{Ship}, Seller), \sigma_1) = \sigma_2$$
$$\Psi((\text{Deliver}, Seller), \sigma_2) = \sigma_3$$

As it can be observed, after the program execution, the state σ_3 will hold. Due to that, query number one will retrieve a negative response (i.e. *false*), as in the final state *delivered* holds. On the other hand, the second query will respond with *true* as action performed by Agent *Seller* did changed the state of σ_1 into σ_2 and the σ_2 into σ_3.

2.0.2 Coffee Order Scenario

In this scenario, the status of the order can be describe by the following fluents: received, brewed, and served. Initially the order is not received, the coffee is not brewed, and is not served. There will be 3 actions that can be executed: receive order, brew and serve. Receive order makes it received, brew an order makes it brewed, and serve makes it served the order. Each of those aforementioned actions has associated agent attached in case the initial conditions performing a specific action are met.

Fluent	**literal**	**nagation**
f_1	Received	¬Received
f_2	Brewed	¬Brewed
f_3	Served	¬Served

Actions are defined as:

1. Receive Order
2. Brew
3. Serve

Agents:

1. **Cashier**: Can only perform action: **Receive Order**
2. **Barista**: Can only perform action: **Brew** and **Serve**

Agents perform undefined actions will have no effect. The set of all considered states in this scenario is defined as:

$\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7\}$, where:

$$\begin{gathered}
\sigma_0 = \{\neg\text{received, } \neg\text{brewed, } \neg\text{served}\} \\
\sigma_1 = \{\text{received, } \neg\text{brewed, } \neg\text{served}\} \\
\sigma_2 = \{\text{received, brewed, } \neg\text{served}\} \\
\sigma_3 = \{\text{received, brewed, served}\} \\
\sigma_4 = \{\neg\text{received, } \neg\text{brewed, served}\} \\
\sigma_5 = \{\neg\text{received, brewed, served}\} \\
\sigma_6 = \{\text{received, } \neg\text{brewed, served}\} \\
\sigma_7 = \{\neg\text{received, brewed, } \neg\text{served}\}
\end{gathered}$$

The structure of the scenario is as follows:

$$\begin{gathered}
\textbf{initially } \neg \textit{received} \\
\textbf{initially } \neg \textit{brewed} \\
\textbf{initially } \neg \textit{served} \\
\textit{Receive Order} \text{ by } \textit{Cashier} \textbf{ causes } \textit{received} \textbf{ if } \neg\textit{received} \ \neg\textit{brewed} \ \neg\textit{served} \\
\textit{Brew} \text{ by } \textit{Barista} \textbf{ causes } \textit{brewed} \textbf{ if } \textit{received}, \neg\textit{brewed} \ \neg\textit{served} \\
\textit{Serve} \text{ by } \textit{Barista} \textbf{ causes } \textit{served} \textbf{ if } \textit{received}, \textit{brewed}, \neg\textit{served}
\end{gathered}$$

Hence, the following states are considered in this case:

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_0) = \sigma_1$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_0) = \sigma_0$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_0) = \sigma_0$
$\Psi((\text{Brew}, \text{Barista}), \sigma_0) = \sigma_0$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_0) = \sigma_0$
$\Psi((\text{Serve}, \text{Barista}), \sigma_0) = \sigma_0$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_1) = \sigma_1$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_1) = \sigma_1$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_1) = \sigma_1$
$\Psi((\text{Brew}, \text{Barista}), \sigma_1) = \sigma_2$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_1) = \sigma_1$
$\Psi((\text{Serve}, \text{Barista}), \sigma_1) = \sigma_1$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_2) = \sigma_2$

$\Psi((\text{Receive Order}, \text{Barista}), \sigma_2) = \sigma_2$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_2) = \sigma_2$
$\Psi((\text{Brew}, \text{Barista}), \sigma_2) = \sigma_2$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_2) = \sigma_2$
$\Psi((\text{Serve}, \text{Barista}), \sigma_2) = \sigma_3$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_3) = \sigma_3$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_3) = \sigma_3$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_3) = \sigma_3$
$\Psi((\text{Brew}, \text{Barista}), \sigma_3) = \sigma_3$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_3) = \sigma_3$
$\Psi((\text{Serve}, \text{Barista}), \sigma_3) = \sigma_3$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_4) = \sigma_4$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_4) = \sigma_4$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_4) = \sigma_4$
$\Psi((\text{Brew}, \text{Barista}), \sigma_4) = \sigma_4$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_4) = \sigma_4$
$\Psi((\text{Serve}, \text{Barista}), \sigma_4) = \sigma_4$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_5) = \sigma_5$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_5) = \sigma_5$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_5) = \sigma_5$
$\Psi((\text{Brew}, \text{Barista}), \sigma_5) = \sigma_5$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_5) = \sigma_5$
$\Psi((\text{Serve}, \text{Barista}), \sigma_5) = \sigma_5$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_6) = \sigma_6$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_6) = \sigma_6$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_6) = \sigma_6$
$\Psi((\text{Brew}, \text{Barista}), \sigma_6) = \sigma_6$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_6) = \sigma_6$
$\Psi((\text{Serve}, \text{Barista}), \sigma_6) = \sigma_6$

$\Psi((\text{Receive Order}, \text{Cashier}), \sigma_7) = \sigma_7$
$\Psi((\text{Receive Order}, \text{Barista}), \sigma_7) = \sigma_7$
$\Psi((\text{Brew}, \text{Cashier}), \sigma_7) = \sigma_7$
$\Psi((\text{Brew}, \text{Barista}), \sigma_7) = \sigma_7$
$\Psi((\text{Serve}, \text{Cashier}), \sigma_7) = \sigma_7$
$\Psi((\text{Serve}, \text{Barista}), \sigma_7) = \sigma_7$

Figure 2 outlines all of the aforementioned transitions.

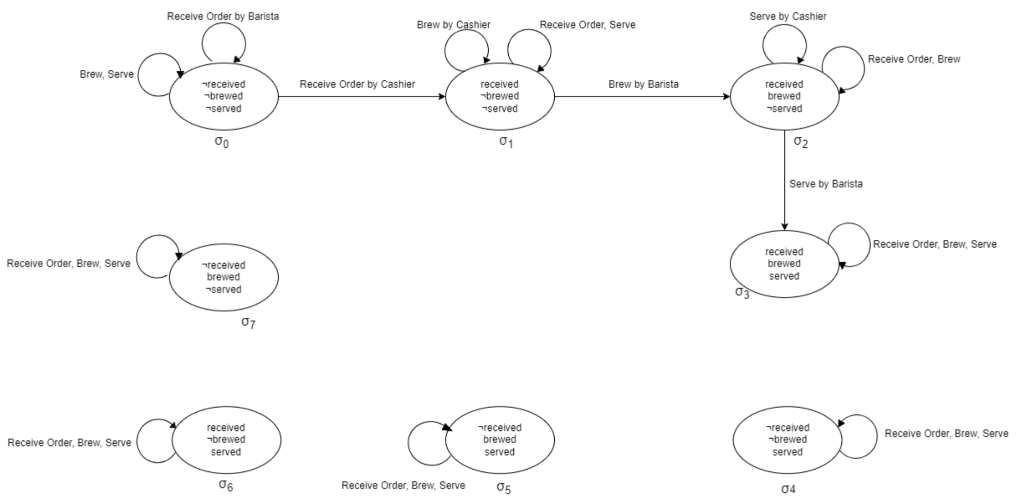

Figure (2) Coffee orders - all possible transitions

Let us demonstrate an example of asking queries using the structure of the query language $\mathcal{QL}_{\mathcal{CST}}$. Assume, that the initial state is σ_0 and the program $\mathcal{P}$ = ((*Receive Order*, *Cashier*),(*Brew*, *Barista*), (*Serve*, *Barista*)). Let us consider the following query statements:

1. *served* **after** ((*Receive Order*, *Cashier*),(*Brew*, *Barista*), (*Serve*, *Barista*)).
2. **Active** Cashier **in** ((*Receive Order*, *Cashier*),(*Brew*, *Barista*), (*Serve*, *Barista*)).

The program execution will consist of the following transitions:

$\Psi((\text{Receive Order}, Cashier), \sigma_0) = \sigma_1$
$\Psi((\text{Brew}, Barista), \sigma_1) = \sigma_2$
$\Psi((\text{Serve}, Barista), \sigma_2) = \sigma_3$

As it can be observed, after the program execution, the state σ_3 will hold. Due to that, query number one will retrieve a positive response (i.e. *true*), as in the final state *served* holds. On the other hand, the second query will respond with *true* as action performed by Agent *Cashier* did changed the state of σ_0 into σ_1.

3 Individual contribution

1. **Jianhao Luo** – Theoretical part: 70% Example part: 30%
2. **Dmytro Narepekha** – Theoretical part: 60% Example part: 40%
3. **Sabrila Trifebi Shina** – Theoretical part: 50% Example part: 50%
4. **Suzzane Abiri** – Theoretical part: 50% Example part: 50%