# Towards Legally and Ethically Correct Online HTN Planning for Data Transfer

Hisashi Hayashi and Ken Satoh

August 1, 2022

# Towards Legally and Ethically Correct Online HTN Planning for Data Transfer

Hisashi Hayashi[1,*], Ken Satoh[2]

[1]Advanced Institute of Industrial Technology, 1-10-40 Higashi-Ooi, Shinagawa-ku, Tokyo, 140-0011, Japan

[2]National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan

**Abstract**

Data transfer among servers is crucial for distributed data mining because many databases are distributed around the world. However, as data privacy is becoming more legally and ethically protected, it is necessary to abide by the laws and respect the ethical guidelines when transferring and utilizing data. Because information affecting legal/ethical decision making is often distributed, the data-transfer plan must be updated online when new information is obtained while transferring data among servers. In this study, we propose a dynamic hierarchical task network (HTN) planning method that considers legal and ethical norms while planning multihop data transfers and data analyses/transformations. In our knowledge representation, we show that data-transfer tasks can be represented by the task-decomposition rules of total-order HTN planning. We also show that legal norms can be expressed as the preconditions of tasks and actions, and ethical norms can be expressed as the costs of tasks and actions where legal norms cannot be violated, but ethical norms can be violated if necessary following the ethical theory of utilitarianism. In the middle of the plan execution, the online planner dynamically updates the plan based on new information obtained in accordance with laws and ethical guidelines.

**Keywords**

Data Transfer, Legal and Ethical Norms, Online HTN Planning, Logic Programming, Application of Knowledge Representation

## 1. Introduction

Because data privacy is respected worldwide, many laws and ethical guidelines governing the transfer and usage of collected data have been established. Some data can only be transferred within a country or a company. Some data can only be used for specific purposes.

Because the laws and ethical guidelines for collected data are complicated and different in each country, some researches have been conducted on the automated compliance check of norms in data transfers. In [1, 2, 3, 4, 5], the policy presentation of European general data protection regulation (GDPR) is studied to automate compliance checks.

Planning for data transfer in accordance with legal/ethical norms is a new field of research. In the studies of [5, 6], data-transfer planners and legal/ethical checkers are separate. These are good frameworks considering that the logic of legal/ethical checkers is complicated and should be separated from the logic of planning. However, dynamic replanning was not achieved in these studies.

Considering real international data transfers among distributed servers, dynamic replanning is crucial because the latest information necessary for planning is also distributed and not available when initially planning. In other words, the data-transfer plan must be dynamically checked and updated if necessary, even in the middle of the plan execution when new information is found on distributed servers, which may affect the validity of the plan.

In this paper, we present a new knowledge representation for dynamic HTN planning on transferring and utilizing distributed data considering legal and ethical norms. We use an extended algorithm of Dynagent [7] which is an *online* total-order HTN planner. Total-order HTN planning algorithms [7, 8, 9, 10, 11] are simple, easy to use, and used for representing the domain control heuristics by task-decomposition rules.

In our knowledge representation, we show that data-transfer tasks can be represented by the task-decomposition rules of total-order HTN planning. We also show that legal norms can be expressed as the preconditions of tasks and actions, and ethical norms can be expressed as the costs of tasks and actions where legal norms cannot be violated, but ethical norms can be violated if necessary following the ethical theory of utilitarianism. Using this knowledge and an online planning algorithm, the plan of data transfer and utilization is dynamically adapted to the new information obtained at local servers, abiding by the laws and following the ethical guidelines.

We assume that the data-transfer planners and legal/ethical checkers are separate as in [5, 6]. We focus on planning and replanning rather than legal/ethical checks. Because we use an online planning algorithm, the validity of the plan is checked and the plan is updated

*NMR 2022: 20th International Workshop on Non-Monotonic Reasoning, August 07–09, 2022, Haifa, Israel*

[*] Corresponding author.

hayashi-hisashi@aiit.ac.jp (H. Hayashi); ksatoh@nii.ac.jp (K. Satoh)

in the middle of the execution with the help of external legal and ethical checkers.

The rest of this paper is organized as follows: In Section 2, related work is discussed. In Section 3, the algorithm of online HTN planning is explained. In Section 4, the system architecture of the planning agent with external legal and ethical checkers is presented. In Section 5, the problem of data transfer and utilization is defined. In Section 6, the knowledge representation method to solve the problem is shown as a case study based on a specific scenario. In Section 7, the knowledge representation presented in the case study is discussed. In Section 8, the paper is summarized.

## 2. Related Work

HTN planners create plans by decomposing abstract tasks into more concrete subtasks. The first HTN planners were created in the late 1970s [12, 13]. Other previous HTN planners were created around 1990 [14, 15].

The most popular and well-established HTN planner is simple hierarchical ordered planner (SHOP) [8], which is a simple forward-chaining total-order planner. This forward-chaining planner decomposes the subtasks in the same order of execution. Domain control heuristics can be expressed easily by the task-decomposition rules (*methods*) in a manner similar to the Horn clauses of the Prolog programming language, which are used for goal/literal decomposition.

SHOP is still standard in HTN planning. For example, HDDL [16] was used in the HTN planning track of the international planning competition held in 2020, however, a translator from HDDL to (J)SHOP2 [17] was provided. (Note that SHOP2 is a partial-order-planner version of SHOP, and that JSHOP2 is the Java version of SHOP2.) SHOP-like total-order HTN planners are still being studied to improve computational efficiency [9, 10, 11].

Dynagent [7] is a simple SHOP-like total-order forward-chaining HTN planner. In contrast to SHOP, Dynagent is an online HTN planner. When the current assumption is updated, the Dynagent planner modifies the plan, even in the middle of plan execution. Dynagent was applied to real robot manipulation, such as online path planning [18] and online pick-and-place planning (arm manipulation) [19, 20]. In this study, we adopted and slightly modified the online HTN planning algorithm of Dynagent.

Another interesting online forward-chaining HTN-like planning is also studied in [21, 22]. This online planner never backtracks and cannot change the plan in the middle of execution. However, it delays the subtask decomposition until it becomes necessary and changes the way to decompose the subtasks according to the current situation. Interestingly, this planner conducts Monte Carlo tree search, which is often used for game tree search. This technique is known to be effective when the search space is very large, such as in chess or Go. This planner can also represent complicated control processes such as “if-then” and “repetition” as in standard procedural programming languages.

In the studies [5, 6], the knowledge representation for data-transfer planning is expressed by logic programs that represent the simplified version [23] of the event calculus [24]. These planners are implemented by the answer set programming (ASP [25]) solver, which makes stable models through forward reasoning. However, they are not online planners. The idea of using the simplified version of the event calculus for planning was first introduced in [26]. A planner based on the event-calculus was implemented in [27] using the Prolog programming language.

In [28, 29], event calculus is used for representing causalities in computational ethics. Another work on ethical principles on planning is found in [30].

## 3. Online HTN Planning

In this section, we define the syntax and sketch of the algorithm of online total-order forward-chaining HTN planning based on the algorithm of Dynagent [7]. Dynagent [7] is similar to SHOP [8]. However, in contrast to SHOP, Dynagent is an online planner.

### 3.1. Syntax

In this subsection, we define the syntax of the *belief* and *planning knowledge* that are used by the planner. Because we implemented the algorithm in Prolog, the syntax follows its representation.

In the following definition, fluents (predicates whose truth value can change) and belief rules (corresponding to Horn clauses in Prolog) are defined using **constants**, **variables**, **functions** (=function symbols), and **predicates** (=predicate symbols). As in Prolog, *constants*, *functions*, and *predicates*, are represented by alphanumeric characters starting with a lowercase alphabet and *Variables* are represented by alphanumeric characters starting with an uppercase alphabet or “_”.

**Definition 1.** *A* **term** *is one of the following: a constant, a variable, or a complex term. A* **complex term** *is of the following form:* $\mathsf{F}(\mathsf{T}_1, \cdots, \mathsf{T}_n)$ *where* $n \geq 0$*,* $\mathsf{F}$ *is an n-ary function, and each* $\mathsf{T}_i$ *(*$1 \leq i \leq n$*) is a term. A* **fluent** *is of the following form:* $\mathsf{P}(\mathsf{T}_1, \cdots, \mathsf{T}_n)$ *where* $n \geq 0$*,* $\mathsf{P}$ *is an n-ary predicate, and each* $\mathsf{T}_i$ *(*$1 \leq i \leq n$*) is a term. When* $\mathsf{P}$ *is a 0-ary predicate, the fluent* $\mathsf{P}()$ *can be abbreviated to* $\mathsf{P}$*. A fluent is either* **derived** *or* **primitive**.

In the following definition, belief rules are defined in the same way as in Prolog. Fluents are used to represent the states.

**Definition 2.** *A* **belief rule** *is of the following form:* $\mathtt{belief}(\mathsf{F}, [\mathsf{F}_1, \cdots, \mathsf{F}_n])$[1] *where* $n \geq 0$*,* $\mathsf{F}$ *is a* **derived** *fluent called the* **head***, each* $\mathsf{F}_i$ *(*$1 \leq i \leq n$*) is a fluent, and the set of fluents* $\mathsf{F}_1, \cdots, \mathsf{F}_n$ *is called the* **body***. When* $n > 0$*,* $\mathsf{F}$ *is a derived fluent. When* $n = 0$*, the belief rule* $\mathtt{belief}(\mathsf{F}, [])$ *can be expressed as* $\mathtt{belief}(\mathsf{F})$ *and* $\mathsf{F}$ *is called a* **fact***. The belief rule* $\mathtt{belief}(\mathsf{F}, [\mathsf{F}_1, \cdots, \mathsf{F}_n])$ **defines** *the fluent* $\mathsf{G}$ *if* $\mathsf{F}$ *is unifiable with* $\mathsf{G}$*. Fluent* $\mathsf{F}$ *is regarded as dynamic iff it is declared as* `dy(F)`*. The belief rule* $\mathtt{belief}(\mathsf{F})$ *can be asserted to or retracted from the belief after observation or action execution iff* $\mathsf{F}$ *is dynamic.*

We define the syntax of tasks, actions, and (total-order) plans as follows: *task symbols* are represented by alphanumeric characters starting with a lowercase alphabet.

**Definition 3.** *A* **task** *is of the following form:* $\mathsf{T}(\mathsf{X}_1, \cdots, \mathsf{X}_n)$ *where* $n \geq 0$*,* $\mathsf{T}$ *is an n-ary task symbol, and each* $\mathsf{X}_i$ *(*$1 \leq i \leq n$*) is a term. When* $\mathsf{T}$ *is a 0-ary task symbol, the task* $\mathsf{T}()$ *can be abbreviated to* $\mathsf{T}$*. A task is either* **abstract** *or* **primitive***. An* **action** *is a primitive task. The* **cost** $\mathsf{C}$ *of the task* $\mathsf{T}$*, where* $C$ *is a number (real number or integer), is represented as* $\mathtt{cost}(\mathsf{T}, \mathsf{C})$*.*

*A* **plan** *is a list of tasks of the following form:* $[\mathsf{T}_1, \cdots, \mathsf{T}_n]$ *where* $n \geq 0$ *and each* $\mathsf{T}_i$ *(*$1 \leq i \leq n$*) is a task, which is called the i-th element of the plan. The* **cost** *of the plan* $[\mathsf{T}_1, \cdots, \mathsf{T}_n]$ *is the sum of each cost of* $\mathsf{T}_i$ *(*$1 \leq i \leq n$*).*

To represent the effect of an action, we use the following action rules.

**Definition 4.** *An* **action rule** *is of the following form:* $action(\mathsf{A}, \mathsf{C}, \mathsf{E})$*, where* $\mathsf{A}$ *is an action,* $\mathsf{C}$ *is a list of fluents called* **preconditions***,* $\mathsf{E}$ *is a list of* **effects***, an effect is either of the following forms:* $\mathtt{initiates}(\mathsf{F})$ *or* $\mathtt{terminates}(\mathsf{F})$*, and* $\mathsf{F}$ *is a fluent.*

Intuitively, in the aforementioned definition, $\mathtt{initiates}(\mathsf{F})$ (or $\mathtt{terminates}(\mathsf{F})$) represents that the truth value of $\mathsf{F}$ becomes true (respectively, false) after the action execution, if all the preconditions hold.

To represent a method to decompose a task into subtasks, we use the following task-decomposition rules. Note that task decomposition rules are called *methods* in SHOP [8].

**Definition 5.** *A* **task-decomposition rule** *is of the following form:* $htn(\mathsf{H}, \mathsf{C}, \mathsf{B})$ *where* $\mathsf{H}$ *is an abstract task called the* **head***,* $\mathsf{C}$ *is a list of fluents called* **preconditions***, and* $\mathsf{B}$ *is a plan called the* **body***.*

The **planning agent** has *belief* and *planning knowledge*, which are used for planning. Furthermore, belief represents the current state, whereas planning knowledge represents the effects of actions and the methods to decompose tasks into subtasks.

**Definition 6.** **Belief** *is of the following form:* $\langle \mathsf{D}, \mathsf{S} \rangle$ *where* $\mathsf{D}$ *is a set of dynamic fluents, and* $\mathsf{S}$ *is a set of belief rules.*

**Planning knowledge** *is of the following form:* $\langle \mathsf{AR}, \mathsf{TDR}, \mathsf{COST} \rangle$ *where* $\mathsf{AR}$ *is a set of action rules,* $\mathsf{TDR}$ *is a set of task-decomposition rules, and* $\mathsf{COST}$ *is a set of the cost of each task.*

## 3.2. Semantics

Standard semantics of a plan can be used if all the tasks in the plan are actions. See the simplified version [23] of the event calculus for example.

## 3.3. Sketch of the Algorithm

In this subsection, we show the sketch of the algorithm we used in this study. We used the algorithm of Dynagent, which is defined in detail in [7]. However, the replanning method after cost updates is not shown in [7]. We modified the algorithm to handle cost updates, which is crucial for reflecting ethical norms in plan selection. Because the algorithm is implemented in Prolog, it can handle rules of predicate logic by unification.

### 3.3.1. Initial Planning

The planning agent has the belief and planning knowledge defined in the previous subsection. Belief represents the current state (the truth value of each fluent) of the world, which the planning agent believes. Planning knowledge includes action rules, task-decomposition rules, and cost information of tasks.

The planner recursively decomposes the task into subtasks that become primitive tasks (= actions) before execution. The HTN planning algorithm is forward-chaining and the task decomposition is conducted in the same order as task execution. As shown in Figure 1, when *taskA* in a plan is decomposed, all the previous tasks before *taskA* are primitive. Therefore, it is easy to evaluate the truth value of fluents in the state shortly before task execution. The preconditions (*precond2* and *precond3*) of the task decomposition, which are added to the preconditions of the first subtask (*taskA1*), must be satisfied before the task execution.

In general, there are several ways to decompose a task. For example, in the case of the data transfer problem, there are several routes for data transfer. When decomposing a task in a plan, multiple plans are created using

[1] This syntax reflects our implementation in Prolog. This belief rule can be understood as $\mathsf{F} \Leftarrow \mathsf{F}_1, \cdots, \mathsf{F}_n$.

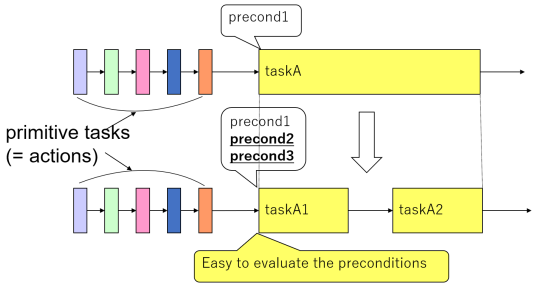

**Figure 1:** Task Decomposition

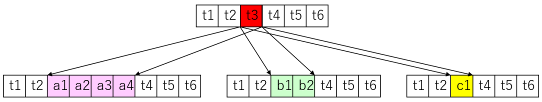

**Figure 2:** Multiple Subplans

multiple task-decomposition rules. For example, in Figure 2, the task *t3* in a plan is decomposed into three subplans [*a1, a2, a3, a4*], [*b1, b2*], and [*c1*]. Therefore, the search space of HTN planning is an or-search-tree of plans.

When each task has the cost information, the best-first search can be conducted. In the algorithm of Dynagent, to conduct the best-first search, the planning agent maintains frontiers (alternative plans) in the or-search-tree of plans, sorts the plans in ascending order of cost, and decomposes the first abstract task in the plan with the lowest cost. If the cost of a task is always lower than or equal to the cost of its primitive subplans (subplans that have only actions), the first found plan has the lowest cost.

### 3.3.2. Replanning after Belief Deletion

In the planning algorithm of Dynagent, each precondition (a dynamic fluent) of a task in a plan is recorded in association with the task in the plan if its truth value is subject to change. As shown in Figure 3, this fluent recorded as a precondition of a task serves as a *protected link* which must be true before the execution of the task.

Following a belief update, if the protected link in a plan is violated, the plan becomes invalid. Then, the invalid plan is removed from the frontiers of the or-search tree. As shown in Figure 4, if the current plan becomes invalid, the planning agent changes the current plan to the plan with the next-lowest cost, and continues the best-first search using the frontiers of valid plans.

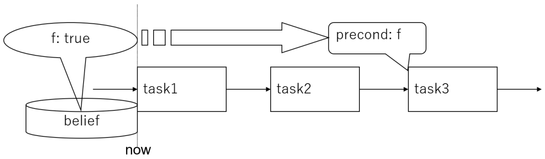

**Figure 3:** Protected Link

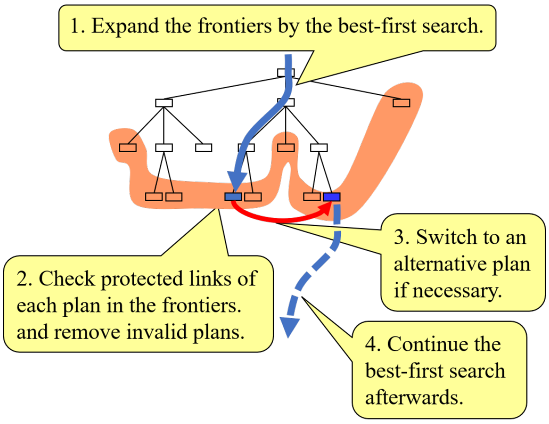
1. Expand the frontiers by the best-first search.
2. Check protected links of each plan in the frontiers. and remove invalid plans.
3. Switch to an alternative plan if necessary.
4. Continue the best-first search afterwards.

**Figure 4:** Switching to an Alternative Plan

### 3.3.3. Replanning after Belief Addition

When evaluating a precondition of a task in a plan in the planning algorithm of Dynagent, if the precondition is a dynamic fluent, the planning agent records the plan separately from the frontiers even if the fluent is false. During the plan execution, if the belief is updated and the precondition becomes true, the recorded plan is asserted to the frontiers as a new valid plan. Because the plans in frontiers are always sorted, if the new plan has the lowest cost, the planning agent stops the current plan execution, switches to the new plan, and continues the best-first search, which may lead to a better plan.

### 3.3.4. Replanning after Cost Update

In the planning algorithm of Dynagent, replanning after a cost update is not explicitly shown. However, this is crucial in our planning with an ethical checker because the costs of unethical actions are dynamically set higher after the ethical check. Therefore, we added a new replanning procedure to the algorithm.

Following the cost update of an action (or a task), we reevaluate the cost of each plan in the frontiers and sort the plans in ascending order of cost. When the current plan becomes less attractive in terms of costs after the g update, the planning agent stops the plan execution, changes the plan, and continues the best-first search, which may lead to a better plan.

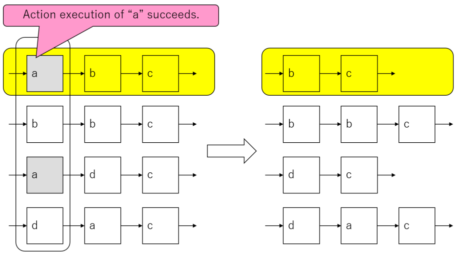

Figure 5: Plan Update after Action Execution

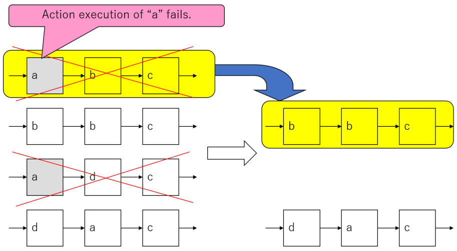

Figure 6: Replan after Action Failure

### 3.3.5. Replanning after Action Execution

Dynagent is an online planner that updates each plan after execution of each action. It maintains all the alternative plans so that any plan can be started from the current state.

As shown in Figure 5, when the execution of an action succeeds, if the executed action is unifiable with the first action in a plan, it is removed from the plan. Sometimes an action execution in a plan invalidates other alternative plans. Therefore, protected links are checked and invalid alternative plans are removed after a successful action execution.

As shown in Figure 6, when the execution of an action fails, if the executed action is unifiable with the first action in a plan, the plan is removed from the alternative plans recorded in the frontiers. In this case, the planning agent stops the plan execution and restarts the best-first search using the valid plans in the frontiers until it finds the plan.

## 4. Online HTN Planning Agent Architecture with External Legal and Ethical Checkers

In Figure 7, we show the overall system architecture of our online HTN planning agent with external legal and ethical checkers.

In this study, we focused on the knowledge representation of beliefs and planning knowledge, which is used for planning and replanning in the online HTN planner.

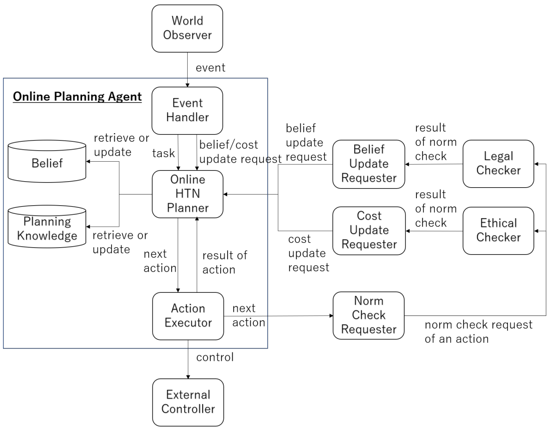

Figure 7: System Architecture

In addition, the online planning agent has an event handler that inputs a task or a belief/cost update request to the online HTN planner when receiving an event from an external world observer that obtains new information. Given a task or a belief/cost update request, the planner starts planning or replanning.

Note that the user interface that receives a command from the user can be regarded as a world observer. An example of the event handler is explained in [18].

The online planning agent also has an action executor that receives an action execution command from the online HTN planner and controls the external controller to execute the action.

To utilize external legal and ethical checkers, we need a norm check requester that inputs the next action to the legal and ethical checkers. Because we need to check the legal and ethical norms before executing an action, the action executor sends the next action to this norm check requester.

If the next action is not changed after checking the legal and ethical norms, the action executor executes the action as usual. If there is a legal or ethical problem, the belief update requester or the cost update requester sends the belief update request or the cost update request to the online HTN planner, which triggers replanning.

In this study, we only designed and implemented the knowledge (belief and planning knowledge) and the algorithm of the online planning agent. In the future, we would consider to connect the planning agent to the legal and ethical checkers.

## 5. Problem

In this section, we define the planning problem of legally and ethically correct data transfer and utilization.

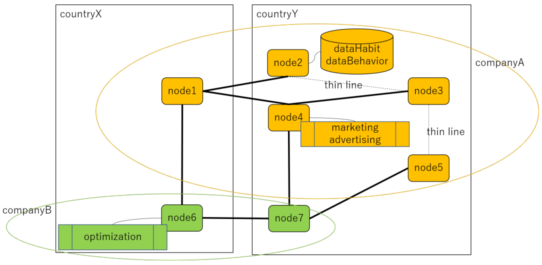

**Figure 8:** Connection of Servers

**Nodes** (servers) are connected by **arcs** (network lines). The data stored in the **database** at a node can be retrieved from the same node. Data at a node can be transferred to an adjacent node that is connected by an arc. An **analyzer** at a node can analyze data for a **specific purpose** at the same node. **Analysis output** is also data and can be transferred to an adjacent node connected by an arc.

There are **legal and ethical norms** for data transfers. Some data can only be transferred within specific countries. Some data can only be transferred within a company. Some data can only be analyzed for specific purposes. Legal norms must be satisfied. Ethical norms should be respected if possible.

The objective is to deliver the analysis output of specified data to a specified node for a specific purpose.

# 6. Case Study

In this section, we consider a specific network and data transfer to study the feasibility of our dynamic HTN planning framework for the planning problem of legally and ethically correct data transfer and utilization .

Figure 8 shows the whole network to be considered as a test case. This example is adopted and modified from the example written in [5]. In the following subsections, we explain the details of Figure 8 while showing how to express the domain knowledge, actions, and task decomposition rules.

## 6.1. Domain Knowledge

In this subsection, we show how to represent the domain knowledge that is used as a belief by the planning agent. This domain knowledge includes node connection, location of database, location of analyzers, allowed purposes for data analysis, region of nodes, allowed regions for data transfer and analysis, owners of nodes, and allowed companies for data transfer.

### 6.1.1. Node Connection

In Figure 8, there are seven nodes that represent servers. The arcs that connect nodes represent the network lines. These network connections are represented as follows:

```
belief(arc(node1,node2)). belief(arc(node1,node4)).
belief(arc(node1,node6)). belief(arc(node2,node3)).
belief(arc(node3,node4)). belief(arc(node3,node5)).
belief(arc(node4,node7)). belief(arc(node5,node7)).
belief(arc(node6,node7)).
```

To represent that each network connection is bidirectional, we define the “connected” predicate as follows:

```
belief(connected(Node1,Node2),[arc(Node1,Node2)]).
belief(connected(Node1,Node2),[arc(Node2,Node1)]).
```

The efficiency of data transfer changes according to the line and time.

### 6.1.2. Location of Database and Retrieved Data

There is a database at `node2` that contains data about the habits and behaviors of people, which are represented as follows:

```
belief(dbAt(dataHabit,node2)).
belief(dbAt(dataBehavior,node2)).
```

The location of the retrieved data from the database is subject to change, which is represented as follows:

```
belief(dataAt(_,_)).
```

### 6.1.3. Location of Analyzers

There are three analyzers of data on the habits and behaviors of people. The analyzer at `node6` is used for optimization. The analyzer at `node4` is used for marketing. Another analyzer at `node4` is used for advertising. This can be represented as follows:

```
belief(analyzableAt(dataHabit,marketing,node4)).
belief(analyzableAt(dataBehavior,marketing,node4)).
belief(analyzableAt(dataHabit,advertising,node4)).
belief(analyzableAt(dataBehavior,advertising,node4)).
belief(analyzableAt(dataHabit,optimizing,node6)).
belief(analyzableAt(dataBehavior,optimizing,node6)).
```

### 6.1.4. Allowed Purposes for Data Analysis

Initially, we assumed that all data were allowed to be analyzed for any purpose. However, this assumption is subject to change and may be corrected by the legal checker. This is represented as follows:

```
dy(allowedPurpose(_,_)).
belief(allowedPurpose(dataHabit,marketing)).
belief(allowedPurpose(dataHabit,advertising)).
belief(allowedPurpose(dataHabit,optimizing)).
belief(allowedPurpose(dataBehavior,marketing)).
belief(allowedPurpose(dataBehavior,advertising)).
belief(allowedPurpose(dataBehavior,optimizing)).
```

### 6.1.5. Regions of Nodes

The region (country) of each node can be represented as follows:

```
belief(nodeRegion(node1,countryX)).
belief(nodeRegion(node2,countryY)).
belief(nodeRegion(node3,countryY)).
belief(nodeRegion(node4,countryY)).
belief(nodeRegion(node5,countryY)).
belief(nodeRegion(node6,countryX)).
belief(nodeRegion(node7,countryY)).
```

### 6.1.6. Allowed Regions for Data Transfer and Analysis

Initially, we assumed that all data were allowed to be transferred in any region. However, this assumption is subject to change and may be corrected by the legal checker. Note that the analyzed data are also data. This can be represented for the case of `countryX` as follows:

```
dy(allowedRegion(_,_)).
belief(allowedRegion(dataHabit,countryX)).
belief(allowedRegion(dataBehavior,countryX)).
belief(allowedRegion(analysisOutput(
   dataHabit,marketing),countryX)).
belief(allowedRegion(analysisOutput(
   dataBehavior,marketing),countryX)).
belief(allowedRegion(analysisOutput(
   dataHabit,advertising),countryX)).
belief(allowedRegion(analysisOutput(
   dataBehavior,advertising),countryX)).
belief(allowedRegion(analysisOutput(
   dataHabit,optimizing),countryX)).
belief(allowedRegion(analysisOutput(
   dataBehavior,optimizing),countryX).
```

The case of countryY is expressed in the same way.

### 6.1.7. Owners of Nodes

The owner (company) of each node can be represented as follows:

```
belief(nodeOwnedBy(node1,companyA)).
belief(nodeOwnedBy(node2,companyA)).
belief(nodeOwnedBy(node3,companyA)).
belief(nodeOwnedBy(node4,companyA)).
belief(nodeOwnedBy(node5,companyA)).
belief(nodeOwnedBy(node6,companyB)).
belief(nodeOwnedBy(node7,companyB)).
```

### 6.1.8. Allowed Companies for Data Transfer

Initially, we assumed that all data were allowed to be transferred in any company. However, this assumption is subject to change and may be corrected by the legal checker. Note that the analyzed data are also data. This can be expressed for the case of companyA as follows:

```
dy(allowedCompany(_,_)).
belief(allowedCompany(dataHabit,companyA)).
belief(allowedCompany(dataBehavior,companyA)).
belief(allowedCompany(analysisOutput(
   dataHabit,marketing),companyA)).
belief(allowedCompany(analysisOutput(
   dataBehavior,marketing),companyA)).
belief(allowedCompany(analysisOutput(
   dataHabit,advertising),companyA)).
belief(allowedCompany(analysisOutput(
   dataBehavior,advertising),companyA)).
belief(allowedCompany(analysisOutput(
   dataHabit,optimizing),companyA)).
belief(allowedCompany(analysisOutput(
   dataBehavior,optimizing),companyA)).
```

The case of companyB is expressed in the same way.

## 6.2. Actions

The agent can execute three actions (primitive tasks): one action is to retrieve the specified data from a database, another action is to transfer the specified data to the specified adjacent node, and the other action is to analyze the specified data for the specified purpose.

### 6.2.1. Data Retrieval from DB

The action `getDataFromDB` retrieves the specified data from the DB at a node and store it at the same node. Subsequently, the data can be transferred to another node or analyzed for a specific purpose.

```
action(getDataFromDB(Data,Node),[
   dbAt(Data,Node)
],[
   initiates(dataAt(Data,Node))
]).
```

The aforementioned rule specifies that the precondition of the action is that the database of `Data` is at `Node`, and that it initiates `dataAt(Data,Node)`.

### 6.2.2. Data Transfer to an Adjacent Node

The action `transfer` transfers the specified data to the specified adjacent node.

```
action(transfer(Data,NodeFrom,NodeTo),[
   dataAt(Data,NodeFrom),
   connected(NodeFrom,NodeTo),
   allowedTransfer(Data,NodeTo)
],[
   initiates(dataAt(Data,NodeTo)),
   terminates(dataAt(Data,NodeFrom))
]).
```

The aforementioned rule specifies that the preconditions of the action are `dataAt(Data,NodeFrom)`, `connected(NodeFrom,NodeTo)`, and `allowedTransfer(Data,NodeTo)`. It also specifies that the effects of the action are to initiate `dataAt(Data,NodeTo)` and to terminate `dataAt(Data,NodeFrom)`.

The last precondition is defined as follows:

```
belief(allowedTransfer(Data,Node),[
   nodeRegion(Node,Region),
   allowedRegion(Data,Region),
   nodeOwnedBy(Node,Company),
   allowedCompany(Data,Company)
]).
```

This indicates that the transfer of `Data` to `Node` is allowed if the node is in an allowed region and is owned by an allowed company.

### 6.2.3. Data Analysis

The action analyze analyzes the specified data at the specified node for the specified purpose. The data must be at the same location as the analyzer and the purpose of the data analysis must be allowed. The analysis output is obtained as new data after the data analysis,

and the original data is erased.

```
action(analyze(Data,Node,Purpose),[
   analyzableAt(Data,Purpose,Node),
   allowedPurpose(Data,Purpose),
   dataAt(Data,Node)
],[
   initiates(dataAt(analysisOutput(Data,Purpose),Node)),
   terminates(dataAt(Data,Node))
]).
```

The aforementioned rule specifies that the preconditions of the action are Data analyzable at Node for Purpose, Data is allowed for Purpose, and Data is at Node. It also specifies that the effects of the action are to initiate dataAt(analysisOutput(Data,Purpose),Node), and to terminate dataAt(Data,Node).

## 6.3. Task Decomposition

The agent needs two abstract tasks to recursively decompose to primitive tasks (actions) before execution. One task is for transferring the specified data to the specified node via multiple nodes. The other task is the top-level task for delivering the analysis output of the data for the specific purpose to the specified node.

### 6.3.1. Multi-Step Transfer

The task multiStepTransfer is a compound task for transferring data to another node via multiple nodes. This task is recursively decomposed until the decomposed subtasks include only the transfer actions.

```
htn(multiStepTransfer(Data,Node,Node),[
   dataAt(Data,Node)
],[]).

htn(multiStepTransfer(Data,NodeFrom,NodeTo),[
   dataAt(Data,NodeFrom),
   connected(NodeFrom,Node)
],[
   transfer(Data,NodeFrom,Node),
   multiStepTransfer(Data,Node,NodeTo)
]).
```

The first rule specifies that no action is required for the transfer task multiStepTransfer(Data,Node,Node) when Data is already at the destination (dataAt(Data,Node)). The second rule specifies that when the data is at NodeFrom and Node is an adjacent node, the transfer task multiStepTransfer(Data,NodeFrom,NodeTo) can be executed by first transferring Data to the adjacent Node, then transferring Data to the destination NodeTo via multiple steps.

### 6.3.2. Delivery of Analytics

The task deliverAnalytics is the top-level task for obtaining the specified data from a DB at a node and delivering the analysis output for a specific purpose to the recipient at another node.

```
htn(deliverAnalytics(Data,NodeFrom,NodeTo,Purpose),[
   dbAt(Data,NodeFrom)
],[
   getDataFromDB(Data,NodeFrom),
   multiStepTransfer(Data,NodeFrom,NodeAnalysis),
   analyze(Data,NodeAnalysis,Purpose),
   multiStepTransfer(analysisOutput(Data,Purpose),
      NodeAnalysis,NodeTo)
]).
```

This rule specifies that to deliver the analysis result of Data for Purpose to the destination (NodeTo), the agent obtains Data from the DB at NodeFrom, transfers the data to NodeAnalysis via multiple steps, analyzes the data for the purpose, and transfer the analysis output to the destination via multiple steps.

## 6.4. Costs of Tasks and Actions

We set a specific value for each task and cost. The cost information is used for planning. The best-first search will always find the lowest-cost plan if the cost of each abstract task is less than or equal to the total cost of its primitive subtasks (actions), which we obtain by task decomposition.

### 6.4.1. Static Cost

We assume that the costs of abstract tasks are static and set at the minimum value of 1.

```
cost(deliverAnalysis(_,_,_,_),1).
cost(multiStepTransfer(_,_,_),1).
```

Furthermore, we assume the costs of the getDataFromDB action and analyze action are static and the values are set at 1.

```
cost(getDataFromDB(_,_),1).
cost(analyze(_,_,_),1).
```

### 6.4.2. Dynamic Cost

The data transfer costs are subject to change. We assume that the agent is aware that the line between node2 and node3 and the line between node3 and node5 are normally slow. The data transfer costs become double if these lines are used. These costs are set at 2.

```
cost(transfer(_,node2,node3),2).
cost(transfer(_,node3,node5),2).
cost(transfer(_,node3,node2),2).
cost(transfer(_,node5,node3),2).
```

The data transfer costs of the other lines are set at 1. This cost information is expressed in the same way.

## 6.5. Specific Task for Case Study

The specific task we consider in this case study is `deliverAnalytics(dataHabit, node2,node5,marketing)`. As shown in Figure 9, the objective of this task is to deliver the analysis output of `dataHabit`, which is stored in the database at `node2`, to `node5`. The purposed of the analysis is `marketing`.

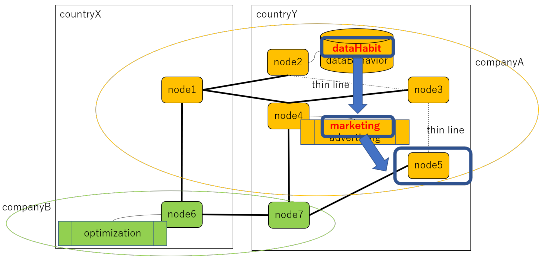

**Figure 9:** Given Task

## 6.6. Initial Planning

Considering the task, the planner creates the initial plan as follows:

```
1. getDataFromDB(dataHabit,node2)
2. transfer(dataHabit,node2,node1)
3. transfer(dataHabit,node1,node4)
4. analyze(dataHabit,node4,marketing)
5. transfer(analysisOutput(dataHabit,marketing),node4,node7)
6. transfer(analysisOutput(dataHabit,marketing),node7,node5)
```

As shown in Figure 10, according to the aforementioned plan, `dataHabit` is retrieved from the database at `node2`, transferred from `node2` to `node4` via `node1`, and analyzed for the purpose of `marketing` at `node4`. The analyzed output is transferred from `node4` to `node5` via `node3`.

There are two data-transfer routes from `node2` to `node4`. Similarly, there are two data-transfer routes from `node4` to `node5`. The planner selects the route with the lowest cost using the best-first search. Note that the data-transfer cost from `node2` to `node3` is set higher because the transfer speed is slow.

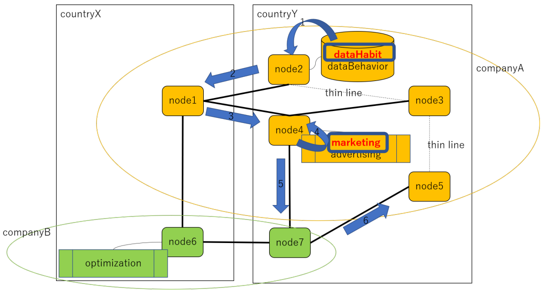

**Figure 10:** Initial Plan

## 6.7. Dynamic Replanning after Legal Check

We assume that `dataHabit` has been retrieved from the database at `node2`. The next action is to transfer the data to `node1`. Then, the legal checker indicates that it is illegal to transfer `dataHabit` to `countryX`. Subsequently, the agent removes the following from its belief:

```
belief(allowedRegion(dataHabit,countryX)).
```

Because the precondition of the next action becomes false, the planner modifies the plan as follows:

```
1. transfer(dataHabit,node2,node3)
2. transfer(dataHabit,node3,node4)
3. analyze(dataHabit,node4,marketing)
4. transfer(analysisOutput(dataHabit,marketing),node4,node7)
5. transfer(analysisOutput(dataHabit,marketing),node7,node5)
```

The modified plan is shown in Figure 11. We can confirm that `dataHabit` is transferred only within `countryY`, rather than via `countryX`. This indicates that the legal norm is satisfied. Note that the action `getDataFromDB(dataHabit,node2)` is erased from the plan because it has been executed.

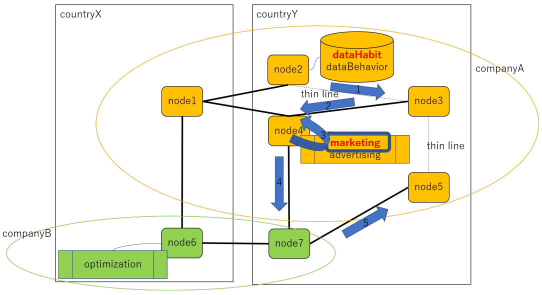

**Figure 11:** Legally Modified Plan

## 6.8. Dynamic Replanning after Ethical Check

We assume that `dataHabit` has been analyzed at `node4` for the purpose of `marketing`. The next action is to transfer the analysis output to `node7`. We assume that the ethical checker indicates that it is not ethical to transfer the analysis output to `companyB`. Then, the agent takes the position of utilitarianism and updates the cost of the next action as follows:

```
cost(transfer(analysisOutput(dataHabit,marketing),node4,node7),
   10).
```

The cost of the next action is now set at 10. Because the cost of the next action has become much higher, the planner dynamically modifies the plan as follows:

```
1. transfer(analysisOutput(dataHabit,marketing),node4,node3)
2. transfer(analysisOutput(dataHabit,marketing),node3,node5)
```

The modified plan is shown in Figure 12. In the modified plan, we can confirm that the analysis output (analysisOutput(dataHabit,marketing)) is transferred within companyA, rather than via companyB. This indicates that the ethical norm is respected, although it is not illegal to transfer it via companyB. Note that the planning agent does not abandon the plan to transfer the analysis output via companyB. It is maintained as an alternative plan and will be used only if there are no other options.

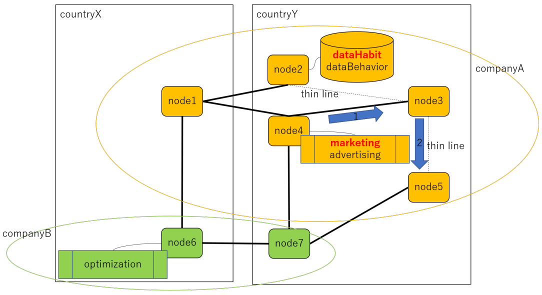

**Figure 12:** Ethically Modified Plan

### 6.9. Evaluation on Computation Time

We implemented the planning agent, belief, and planning knowledge in SWI Prolog for Windows 64-bit, version 8.2.4, which was installed on the Windows 10 Home PC equipped with Intel(R) Core(TM) i7-1065G7 CPU and the 32GB of RAM. We measured the CPU times for initial planning, dynamic replanning after legal check, and dynamic replanning after ethical check five times each, and the average CPU times were 0.003, 0.006, and 0.003 seconds, respectively.

Therefore, this planner is adequate for practical use for the test case scenario in this study. In future, we would like to evaluate the scalability for different types and sizes of networks. One way to tackle the scalability problem is to use stratified multi-agent HTN planning techniques [31, 32] where the parent agent first tries to find a rough data transfer route between the regions, and then its child agent tries to finds a detailed data transfer route inside the current region.

### 6.10. Evaluation on Compliance with Legal and Ethical Norms

In the test case scenario, in initial planning, neither legal norms nor ethical norms were ignored. In other words, dataHabit was planned to be transferred to countryX, which is against the legal norm, and analysisOutput(dataHabit,marketing) was planned to be transferred to companyB, which is against the ethical norm.

When the dynamic replanning algorithm was applied after legal check, dataHabit was planned to be transferred only within countryY. Therefore, the legal norm was complied with. However, analysisOutput(dataHabit,marketing) was still planned to be transferred to companyB, which is against the ethical norm.

When the dynamic replanning algorithm was applied both after legal check and ethical check, dataHabit was planned to be transferred only within countryY, and analysisOutput(dataHabit,marketing) was planned to be transferred only within companyA. Therefore, not only the legal norm was complied with but also the ethical norm was respected.

## 7. Discussion

From the case study, we can understand that the legal norm of an action can be expressed as the precondition of the action. This indicates that the illegal action cannot be executed because its legal norm (precondition) is not satisfied.

By contrast, the ethical norm of an action can be expressed as the cost of the action. The higher the cost is, the more unethical the action is. Even if an action is unethical, it is still legal to execute the action. If the planner can find the lower-cost plan, the agent can avoid unethical action execution if possible. However, unethical actions can still be executed if there is no other option. Even in that case, it is possible to stop the action execution when its cost is too high, which means that the action is too unethical.

Ethical norms of action can be expressed as the soft constraints of the precondition, which should be satisfied if possible but are not required. However, many planners do not support soft constraints. Therefore, it is easier to express ethical norms of action as the costs of the actions.

It is not always possible to collect all the necessary information at the time of initial planning, especially when the latest information is distributed across multiple servers. In the case study scenario in this paper, the planning agent obtains new information regarding the next action shortly before its execution. Legal and ethical norms are checked at this time. Therefore, it is important to dynamically check and update the plan while executing it. Therefore, an *online* planning algorithm is used in this paper.

## 8. Conclusion

We have shown how to represent knowledge about legal and ethical norms using an online total-order forward-chaining HTN planning algorithm in the domain of data transfer and utilization in multiagent systems. The precondition of an action was used for its legal check, however, the cost of an action was used for its ethical check. Dynamic adaptation to legal/ethical norms was achieved by dynamic replanning after legal/ethical check. These techniques are extremely important when the latest information, which may affect legal/ethical norms, is distributed across multiple servers. Experiment results confirmed that this planner is adequate for practical use in terms of computation time in our case study.

Furthermore, we have designed a system architecture that combines the online planning agent with external legal and ethical checkers. External legal and ethical checkers will be useful when the laws and ethical guidelines are complicated. In the future, we would consider implementing the overall system. In addition, we would consider designing an explainable dynamic planner that can explain the reason for plan modification to the users in terms of legal and ethical norms.

## Acknowledgments

This work was supported by JST, AIP Trilateral AI Research, Grant No. JPMJCR20G4 and JSPS KAKENHI, Grant No. JP19H05470 and JP21K12144.

## References

[1] Agarwal, S. Steyskal, F. Antunovic, S. Kirrane, Legislative compliance assessment: Framework, model and GDPR instantiation, in: Annual Privacy Forum, 2018, pp. 131–149.

[2] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, L. Robaldo, Legal ontology for modelling GDPR concepts and norms, Legal Knowledge and Information Systems (2018) 91–100.

[3] M. D. Vos, S. Kirrane, J. Padget, K. Satoh, ODRL policy modelling and compliance checking, in: International Joint Conference on Rules and Reasoning, 2019, pp. 36–51.

[4] P. A. Bonatti, S. Kirrane, I. M. Petrova, L. Sauro, Machine understandable policies and GDPR compliance checking, KI - Künstliche Intelligenz 34 (2020) 303–315.

[5] Y. Taheri, G. Bourgne, J.-G. Ganascia, A compliance mechanism for planning in privacy domain using policies, in: International Workshop on Jurisinformatics, JSAI International Symposia on AI, 2021.

[6] K. Satoh, J.-G. Ganascia, G. Bourgne, A. Paschke, Overview of RECOMP project, in: International Workshop on Computational Machine Ethics, International Conference on Principles of Knowledge Representation and Reasoning, 2021. https://www.cse.unsw.edu.au/~cme2021/CME2021_paper_Satoh.pdf (Accessed on 07 Feb. 2022).

[7] H. Hayashi, S. Tokura, T. Hasegawa, F. Ozaki, Dynagent: An incremental forward-chaining HTN planning agent in dynamic domains, in: Declarative Agent Languages and Technologies III, number 3904 in LNAI, Springer, 2006, pp. 171–187.

[8] D. Nau, Y. Cao, A. Lotem, H. Mũnoz-Avila, SHOP: simple hierarchical ordered planner, in: International Joint Conference on Artificial Intelligence, 1999, pp. 968–975.

[9] G. Behnke, D. Höller, S. Biundo, totSAT — totally-ordered hierarchical planning through SAT, in: International Conference on Autonomous Agents and Multiagent Systems, 2018, pp. 6110–6118.

[10] M. C. Magnaguagno, F. Meneguzzi, L. Silva, HyperTensioN: A three-stage compiler for planning, in: 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network Planning Track, 2021, pp. 5–8.

[11] D. Schreiber, Lilotane: A lifted sat-based approach to hierarchical planning, Journal of Artificial Intelligence Research 70 (2021) 1117–1181.

[12] E. Sacerdoti, A Structure for Plans and Behavior, Elsevier, 1977.

[13] A. Tate, Generating project networks, in: International Joint Conference on Artificial Intelligence, 1977, pp. 888–893.

[14] D. Wilkins, Practical Planning, Morgan Kaufmann, 1988.

[15] K. Currie, A. Tate, O-plan: The open planning architecture, Artificial Intelligence 52 (1991) 49–86.

[16] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford, HDDL: An extension to PDDL for expressing hierarchical planning problems, in: AAAI Conference on Artificial Intelligence, 2020, pp. 9883–9891.

[17] D. Nau, H. Mũnoz-Avila, Y. Cao, A. Lotem, S. Mitchell, Total-order planning with partially ordered subtasks, in: International Joint Conference on Artificial Intelligence, 2001, p. 425–430.

[18] H. Hayashi, S. Tokura, F. Ozaki, M. Doi, Background sensing control for planning agents working in the real world, International Journal of Intelligent Information and Database Systems 3 (2009) 483–501.

[19] H. Hayashi, H. Ogawa, N. Matsuhira, HTN planning for pick-and-place manipulation, in: International Conference on Agents and Artificial Intelligence, 2013, pp. 383–388.

[20] H. Hayashi, H. Ogawa, N. Matsuhira, Comparing repair-task-allocation strategies in MAS, in: International Conference on Agents and Artificial Intelligence, 2015, pp. 17–27.

[21] S. Patra, M. Ghallab, D. Nau, P. Traverso, Acting and

planning using operational models, in: AAAI Conference on Artificial Intelligence, 2019, pp. 7691–7698.

[22] S. Patra, J. Mason, A. Kumar, M. Ghallab, P. Traverso, D. Nau, Integrating acting, planning, and learning in hierarchical operational models, in: International Conference on Automated Planning and Scheduling, 2020, pp. 478–487.

[23] M. Shanahan, Prediction is deduction but explanation is abduction, in: International Joint Conference on Artificial Intelligence, 1989, pp. 1055–1060.

[24] R. Kowalski, M. Sergot, A logic-based calculus of events, New Generation Computing 4 (1985) 67–95.

[25] V. Lifschitz, Answer Set Programming, Springer, 2019.

[26] R. Miller, Notes on deductive and abductive planning in the event calculus, in: AISB Workshop on Practical Reasoning and Rationality, 1997.

[27] M. Shanahan, An abductive event calculus planner, The Journal of Logic Programming 44 (2000) 207–239.

[28] F. Berreby, G. Bourgne, J.-G. Ganascia, A declarative modular framework for representing and applying ethical principles, in: International Conference on Autonomous Agents and Multiagent Systems, 2017, p. 96–104.

[29] F. Berreby, G. Bourgne, J.-G. Ganascia, Event-based and scenario-based causality for computational ethics, in: International Conference on Autonomous Agents and Multiagent Systems, 2018, pp. 147–155.

[30] F. Lindner, R. Mattmüller, B. Nebel, Evaluation of the moral permissibility of action plans, Artificial Intelligence 287 (2020) 1–14.

[31] H. Hayashi, Stratified multi-agent htn planning in dynamic environments, in: KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications, 2007, pp. 189–198.

[32] H. Hayashi, Towards real-world htn planning agents, in: Knowledge Processing and Decision Making in Agent-Based Systems, 2009, pp. 13–41.