Advances in Smart Systems Research : ISSN 2050-8662 : http://nimbusvault.net/publications/koala/assr/
Vol. 3. No. 3 : pp.55-60 : isrp13-006

Plan State Representation Using Heterogeneous Data Sources

Santa Maiti, Plaban Bhowmick and Debnath Mukherjee

TCS Innovation Lab, Kolkata, India,
santa.maiti@tcs.com, plaban@gmail.com, debnath.mukherjee@tcs.com

Abstract

In this paper, we propose a framework that facilitates the use of heterogeneous data sources to represent a plan state. The traditional planners use monolithic predicate based schema for plan state representation where the world state is described as a set of predicates that are currently true. However, this approach is not efficient because, firstly, in reality, the world state can be obtained by aggregating information from different modular sources represented through multiple knowledge representation techniques and secondly, the performance of a planner can be affected when the size of state is enormously large. To overcome the stated limitations, we redefine the notion of a plan state and represent it as the combination of state predicates as well as the references to non-predicate data sources like databases, ontologies etc. The main challenge of our work is to handle plan state representation using distributed heterogeneous data sources, without altering planning algorithm in concern. Though we have based our idea around HTN planning, the approach is applicable to other planners, without any additional overhead.

1. Introduction

In the field of Artificial Intelligence, planning is the process of selecting and organizing actions by considering their expected outcomes or goals. In planning, the plan state denotes the state of the world and is represented as a set of predicates. During the planning process, the state of the system undergoes changes as determined by post conditions of the actions included in the plan. In existing approach, the plan state S is represented as Equation 1.

$$S -> (la)^* \tag{1}$$

where la represents a grounded atom (predicate) that constitutes of a predicate head and a list of arguments. For example, a fact – "*amount of available cash is 10,000*" can be represented in the plan state as *(avail-cash 10000)*. Here, *avail-cash* is the predicate head and 10000 is an *argument*. A planner has to select a set of actions (*operators*) and execute them in order to reach the goal state. The representation of current plan state suffers from shortcomings like practicability, modularity.

- *Practicability:* In real world scenario, the world state is a collection of information available from different sources like databases, ontologies (e.g. travel ontology) [2], API calls (e.g. google maps API) [1] and web services etc. However, the available planners do not consider the heterogeneity of state information.
- *Modularity:* Data sources follow modularity approach in general. E.g. weather information, road information, transport information can be maintained in a data sources of similar type (e.g. database system), but separate tables are used to keep those data. In this approach, occurrence of any change in one module, does not affect the other modules. The current plan state representation does not follow modularity approach as all information are presented together, using predicate-based schema.

Scalability is also another issue in existing plan state representation. In practical cases, the state may be enormously large which in turn affects the performance of a planner. As an example, the HTN planner JSHOP2 [3] suffers from scalability problem for a plan state having large number of state predicates represented in a monolithic predicate-based scheme.

Copyright © 2013 Future Technology Press and the authors

During planning the total plan state is loaded into memory, whereas only a subset is required at different points of execution. It leads to a failure in generation of a valid plan. So, in the purview of real life application requirement, there is a need to represent plan state as the combination of predicate based and non-predicate based state facts.
In this work, our objective is to incorporate heterogeneous data sources in plan state representation. So, a planner has to handle the non-predicate based data along with the predicate based data and access state information on demand basis. Now, the challenge is to handle the overheads caused by non-predicate based data without changing the main planning process. Therefore, we propose an adaptation layer to handle the overheads.
The specific contributions of this work are –

- *Modification in plan state representation:* According to our architecture, a plan state is represented as the combination of logical predicates and references to non-predicate based data sources.
- *Modification in precondition:* The precondition is represented as a logical expression of predicate based state facts as well as non-predicate based state facts.
- *Modification in operator's effect:* An operator's effect part now refers to the modifications in plan state predicate(s) as well as modifications in non-predicate data source(s), if required.
- *Handling non-predicate data:* The overhead (connection establishment, non-predicate based data precondition verification etc.) caused by non-predicate based data source is handled by adaptation layer without modifying the main planning process.

Though the stated problem pertains to any planner, we base our realization on HTN planning paradigm. The objective of HTN planning procedure is to complete a set of tasks known as *task network*. HTN planner uses a set of operators to accomplish the primitive tasks and a set of *methods* to decompose a non-primitive task into smaller subtasks. The process of decomposition is repeated until it reaches to a set of primitive tasks which can be directly accomplished by the operators. The task network consists of task nodes and a set of precedence relations (execution ordering of two tasks). The HTN planning domain refers to a set of operators and a set of methods. For a given planning problem, a plan is a solution, if there exists a primitive decomposition of initial task network and the plan is a solution of that primitive decomposition [5].
SHOP2 is a domain-independent planning system based on HTN planning [4]. Our work is towards extension of JSHOP2, the Java implementation of SHOP2. JSHOP2 has a grammar (written in ANTLR language) that defines JSHOP2 language. ANTLR tool takes this grammar as input and generates JSHOP2 specific lexer and parser. Domain description and problem description are the inputs to JSHOP2. With the help of the lexer and parser, JSHOP2 parses domain description and problem description and obtains a domain specific planner. According to the problem the planner generates a plan in the given domain.
Planning with external data sources are introduced through build-in predicates [6, 7, 8, 9]. The concept of information gathering is proposed in the modified version of UCPOP planner [8]. The primary difficulty of this proposal is that due to lack of clarity in the semantics of build-in predicates, soundness and completeness is not guaranteed. A modified version of SHOP HTN planning system used IMPACT's multi agent environment in order to obtain the facilities to interact with external agents and different data sources [10]. However, the process requires modification in main planning algorithm.

3. Solution Architecture

In the introductory section, we have discussed the limitations regarding existing plan state representation. To overcome the limitations, we modify the architecture of planner as shown in Fig. 1. The proposed architecture accepts predicate based and non-predicate based data sources. An action results in modification of plan state during planning process. To

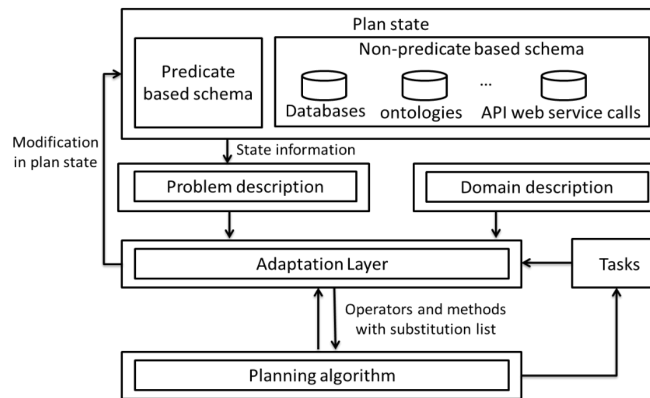

Fig.1. Architecture of proposed approach

avoid frequent non-predicate based data source modification overhead, we prefer to represent frequently changeable data (dynamic data) using predicate schema and slowly changeable or unchangeable data (static data) in non-predicate based data sources. For example, to represent available cash of a person, we prefer to use predicates, as an application of different operators may change its value frequently. On the other hand, if we want to represent an agent's information like agent name, address, contact number, charge etc. or the transport network of a city, we prefer to use database. However, selection of data source depends on the user. The data from predicate or non-predicate based data sources are supplied to planner through adaptation layer. Problem description and domain description are two inputs to the adaptation layer. Problem description contains the initial state information and the task network. Domain description contains different actions. The working methodology of the proposed architecture is explained below.

1. To accomplish a task in task network, a planner unifies the chosen task with methods' head (for non-primitive task) and operators' head (for primitive task). Unified method(s) or operator(s) are passed to the adaptation layer.
2. The adaptation layer then verifies the precondition of the method(s) and operator(s), with respect to the current plan state. A precondition is a logical expression of predicate-based facts and non-predicate based facts. So, its satisfaction depends on whether there are facts in plan state matching the constituents of the logical expression. If there are no such fact in plan state, the truth value of precondition becomes false, otherwise true.
3. The set of predicates in plan state verifies logical precondition. For non-predicate based precondition, adaptation layer connects the specified data source and checks that the data present in the data source can satisfy the precondition or not. For successful verification, the adaptation layer passes all possible substitutions (grounded value) that satisfy the precondition to the planner.
4. A planner assigns the grounded value to the uninstantiated argument list.
5. Next, in case of method, the planner adds the decomposed task list of that method to the remaining subtask network. For operator, the planner passes the control again to adaptation layer.
6. Operators' effect part contains *deletelist* and *addlist*. If the deletelist/addlist refers to a modification in plan state predicate(s), then the adaptation layer modifies the predicate(s) accordingly. Otherwise, a planner checks the permission of modification for non-predicate data sources (in the case of dynamic data maintained in non-predicate based data sources). If it is permitted and required, the adaptation layer connects to the non-predicate data sources and modifies (add or delete) the data accordingly.
7. Steps 1-6 are repeated until the goal state is reached or all tasks of task network are over. The process also exits if no action can accomplish the remaining tasks.

4. Implementation Details

Incorporation of heterogeneous data sources in planning system requires significant modification in grammar rule as well as in domain and problem description. The modifications are suggested considering database as the non-predicate information source.

Modification in Grammar Rule: All the keywords (e.g. call, imply) and grammar terminals (e.g. :method, :sort-by) used by problem definition and domain definition are mentioned in JSHOP2 grammar [3]. We add a new grammar terminal ':import-db' to differentiate database information from the state information represented by predicate based schema. Any logical expression that starts with :import-db refers to a database.

Modification in Problem Description: With the modification the plan state S becomes,

$$S -> (la|dba)^* \quad (2)$$

where la represents logical atom and dba represents database atom. In order to access data from database we need to mention server name, database name, table name, driver settings, user name and password in database atom. User name and password are required for user authentication purpose. Otherwise an unauthenticated user can make change in provider's database. Syntactically database atom is represented as,
(:import-db server_name db_name driver_settings username password)
When the adaptation layer gets a predicate while reading the problem description, it checks whether it is used by the domain or not. If it is used, the predicate is directly added to the plan state, else discarded. If the adaptation layer finds a reference to database, it establishes a connection with that database with the help of the details provided in database atom. Adaptation layer adds the connection to the plan state. This connection is used in a later stage of the planning process. The complete plan state is passed to the planner at the initial stage of planning.

Modification in Domain Description: The domain description also requires some syntactic, operational modifications.

Modification in Precondition: Both method and operator have a precondition part (P) represented as,

$$P \rightarrow lp = le \mid FIRST\ le \mid SORT\ VARID\ (fid)?\ le \quad (3)$$

$$\begin{aligned} le \rightarrow &(\ AND\ )?\ (\ le\ )^* \mid OR\ (\ le\ )^+ \mid NOT\ le \mid IMPLY\ le\ le \mid la \mid \\ &FORALL\ (((\ VARID\ )^*) \mid NIL\ )\ le\ le \mid EXISTS\ (((\ VARID\ )^*) \mid NIL\ )\ le\ le \mid \\ &ASSIGN\ VARID\ term \mid CALL\ fid\ terml \mid NIL \end{aligned} \quad (4)$$

Here, lp represents the logical precondition which is actually a logical expression (le) or first satisfier precondition or sorted precondition. Precondition $(FIRST\ le)$ compels planner to consider only the first set of bindings with respect to plan state that satisfies le [3]. The logical expression can be a logical atom or any complex expression of conjunctions, disjunctions, negations, implications, universal quantifications, assignments, or call expressions [3]. We represent the modified logical expression as Equation 5.

$$\begin{aligned} le \rightarrow &(\ AND\ )?\ (\ le\ )^* \mid OR\ (\ le\ )^+ \mid NOT\ le \mid IMPLY\ le\ le \mid la \mid da \mid \\ &FORALL\ (((\ VARID\ )^*) \mid NIL\ )\ le\ le \mid EXISTS\ (((\ VARID\ )^*) \mid NIL\ )\ le\ le \mid \\ &ASSIGN\ VARID\ term \mid CALL\ fid\ terml \mid NIL \end{aligned} \quad (5)$$

Here, a database atom consists of a da head and an argument list. The da head contains server name, database name, driver setting and the table name. The argument list of da refers to the uninstantiated variables. This uninstantiated argument list is instantiated with the values obtained from database tables. If we want to fetch data from database table with

restriction (some specific value for a particular column), we have to mention the column name. Otherwise, mentioning column names is optional. The variables of da are substituted after precondition checking. Syntactically the basic database precondition is represented as,

(server name:db name:driver settings:table name arg1[/col1] … argm[/colm]).
When the adaptation layer reads this basic database precondition it uses the already established connections and obtains data from databases by executing a sql query –
SELECT * FROM table name WHERE colk = argk.
The plan state may contain more than one predicate that satisfies the precondition. The logical expression can filter those predicates by applying restrictions (call, sort-by etc.). For example, the logical expression starts with sort-by, sorts the predicates on some arguments mentioned in argument list and passes the top most predicate to planner. In case of database precondition, these restrictions are handled by formulating equivalent sql query. We give an example of such precondition and equivalent sql query.
sort-by database precondition:
(:sort-by ?col < (server name:db name:driver settings:table name arg1[/col1] … argm[/colm])).
Equivalent sql query:
SELECT * FROM table name ORDER BY col ASC;

Modification in Effect: Other than precondition part, syntactic modification is also required in operator's effect part that is in addlist and deletelist. It can change database entry, although, we prefer to keep static data in database table and therefore no modifications are required (insert and delete). Addlist can refer to an update and addition of value in database table and deletelist may refer to a deletion of data in database table. Syntactically, they are represented as similar database precondition expression.
(server name:db name:driver settings:table name arg1[/col1] … argm[/colm]).
Here, server_name, db_name, driver_settings, table_name follows the same notation as mentioned earlier. If the adaptation layer encounters a predicate in addlist or deletelist, it adds or deletes corresponding predicate in plan state. If it refers to a database record, adaptation layer first checks the write permission of that database. If permitted, for addlist the adaption layer executes equivalent sql query,
INSERT INTO table name VALUES (arg1, …, argm);
And for delete list following query is executed.
DELETE FROM table name WHERE colk = argk;

6. Discussion

The aim of our work is to use the heterogeneous data sources in a plan state, fit for real life application. To substantiate the usability of the proposed approach, we have conducted an experiment on city tour and travel domain using JSHOP2 [3] planner. The domain consists of various methods and operators for selecting agent or online booking, selecting flight, booking hotel, visiting point of interest, watching movie, shopping and withdrawing cash. The problem description consists of plan state and a task network specifying consecutive tasks of booking ticket, accommodation, visiting point of interest, watching movie, shopping and withdrawing cash. As a plan state, nine different types of information about agent, ATM location, flight, hotel, online booking, point of interest, restaurant, entertainment, shopping and some general information e.g. start time of planning, initial location, available cash etc. (total 748 facts) are presented. In practical scenario, this information are available in different sources. Therefore, we use nine different tables (agent, ATM location, flight, hotel, online booking, point of interest, restaurant, entertainment, shopping) to maintain different types of information. MYSQL database is used for this purpose. We represent the general state information using predicate based schema. This type of plan state representation supports practicability as well as modularity. We observed

that the planner generates a valid plan using this type of plan state representation. In real life scenario a plan state can be enormously large. However, most of the programming languages and tools have a bytecode limit, which means for successful compilation, the size of a single function or method must be less that limit. Particularly, in our test case, when we used only predicate-based schema for plan state representation, the planner failed to create any plan and threw an error – “code too large” as a single java method cannot handle more than 64KB of code. Therefore, we can say our proposed approach can also resolve the scalability issue for a plan state having large number of state predicates.

6. Conclusion

In this work we point out that the existing planner suffers from practicability, modularity and scalability issues with respect to plan state representation. As the solution of this problem, we propose to represent plan state as the combination of predicate based and non-predicate based heterogeneous data sources. Our solution overcomes the limitations but it also has some disadvantages. As the planner depends also on non-predicate based data, each step of planning, data need to be fetched, which may fail sometime due to the unavailability of connection. If we consider ontology, we have to take care of schema mapping techniques. Moreover, inclusion of non-predicate based data will demand extra time for planning due to data fetching overhead. But, in contrary, the solution offers a modular and scalable approach for plan state representation. Moreover, it is a useful solution for many real life applications where data gets collected from different and distributed sources. Till now we have only considered the MYSQL database. In future, we plan to incorporate the feature of database portability, so that we can support various database vendors. Hibernation can be used for this purpose. We also plan to integrate ontology, API and web service information in the planner as future enhancement.

7. References

1. Google Maps API, Google Developers. Accessed on: March, 2013. https://developers. google.com/maps.
2. Semantic Web Authoring Tool/Ontologies. Accessed on: March, 2013. www.swat project.org/ travelOntology.asp.
3. Ilghami, O. Documentation for JSHOP2. Technical report. Department of Computer Science, University of Maryland (2006).
4. Nau, D. et al. SHOP2: An HTN Planning System. Journal of Artificial Intelligence Research, Vol 20, pp. 379–404 (2003).
5. Sohrabi, S et al. HTN Planning with Quantitative Preferences via Heuristic Search. In Eighteenth International Conference on Automated Planning and Scheduling (ICAPS) Workshop on Oversubscribed Planning and Scheduling, Sydney, Australia, (2008).
6. Etzioni, O. et al. An Approach to Planning with Incomplete Information. In Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning. Vol. 20, pp. 379–404 (2003).
7. Golden, K. et al. Omnipotence Without Omniscience: Efficient Sensor Management for Planning. In Proc. of AAAI (1994).
8. Knoblock, C. A. Building a Planner for Information Gathering: A Report from the Trenches. In Proc. of AIPS (1996).
9. Friedman, M., and Weld, D. S. Efficiently Executing Information-Gathering Plans. In Proc. of the Int. Joint Conf. of AI (IJCAI), pp. 785-791 (1997).
10. Munoz-Avila, H. et al. IMPACTing SHOP: Foundations for Integrating HTN Planning and Multi-Agency. Technical report. UM Computer Science Department (2000).