In: 5th IFAC Workshop on Intelligent Manufacturing Systems (IMS´98) – Gramado -RS – Brazil - November 9-11 1998. Elsevier Science 1999

CASE-BASED PLANNING IN TRANSACTION LOGIC FRAMEWORK

Flavio Tonidandel and Márcio Rillo

Universidade de São Paulo
Divisão de Automação e Inteligência Artificial - DAIA
Av. Luciano Gualberto, 158, trav3 - São Paulo -SP - Brazil
05508-900 e-mail: {flavio,rillo}@lsi.usp.br

Abstract: This work stands out the use of Transaction Logic (TR) in case-based planning. The TR provides a correct and complete logical theory based planner that can be computationally implemented keeping the formal theory semantic. The TR is efficient on states treatment besides to create a retriever and cases adaptation easier than others formalized systems. Transaction Logic provides a clean fashion in knowledge representation and its semantic based on path of states is next to planning necessities, making possible the formalization of whole case-based planning system without the existence of the semantic gap between theory and implementation.

Keywords: Artificial intelligence, computational methods, formal specification, planning, problem solvers.

1. INTRODUCTION

In manufacturing areas, the importance of planning systems is clear for many researches, because many processes are made under some sequence of defined actions, which characterize a plan.

Many works in planning area are not made under strict formalism, what makes it difficult to understand its fundamental procedures, limitations and theoretical complexity. It is hard to know how it works with other situations besides the previously tested ones. The formalization is essential for the construction of a sound and complete system that works well in any situation. However, the systems based on formalism make harder their computational applicability, because many formalization theories are hard to translate in any computational language. Therefore, the use of a kind of logic that has sound and complete inference rules is essential to turn the formal theory into an implemented system.

So that, many logical theories were proposed, but they always lack a treatment of updates or a computational way. Some logical theories have a clean fashion as first-order logic, but it is not so good to work with nonmonotonic reasoning. Others have a powerful theory, but have a difficult computational implementation (Koehler, 1996). In order to avoid these problems, this paper will introduce the use of Transaction Logic (Bonner and Kifer, 1995), abbreviated TR, as the formal theory that provides a sound and complete implementation through its inference rules and its update theory. It has a clean syntax to represent knowledge, actions and the world state.

The use of Case-Based planning (CBP) (Hammond, 1989) is justified by the relative efficiency improved in planning by reuse and modification of complete existing plans (Koehler, 1996). In planning, if a planner receives the same planning problem, it will repeat the same operations and will give the same answer, and Case-Based Planning systems try to overcome this waste of time to do the same process through the use of previously generated plans.

The Case-Based planning formalization approach has very few studies. One of them is the MRL system (Koehler, 1996) that has the same difficulty of others formalization systems about its computational implementation, and uses a kind of modal logic that doesn't have a clean syntax. Differently, a CBP system formalized with TR, besides introducing a

clean fashion logical theory for knowledge representation, it still makes possible the theoretical control of the whole planning system, without the existence of a semantic gap between theory and computational implementation.

Therefore, this work stands out the use of the TR in case-based planning as a tool to provide a correct and complete logical theory based planner. In addition, the TR provides efficiency in state treatment besides to create a retriever and adaptation of cases easier than others formalized systems.

2. OVERVIEW OF TRANSACTION LOGIC

The Transaction Logic (TR) is a logic proposed by Bonner and Kifer (1995) that specifies formalism about database update phenomena, path of execution and sequential procedure. In planning, these features are very common, firstly observed by Rillo (1994), because it can be seen as a process that, after executing a plan of action, it updates the world model. In TR can be described actions and formally executed through the inference system with database updating.

The TR is an extension of first-order logic, with the introduction of a new operator called serial conjunction (⊗). This operator represents a procedural activity, a transaction, where $\alpha \otimes \beta$ means "first execute α, and then execute β". With this new operator, can be specified sequence of actions easier than other logics.

To describe a transaction, consider this notation: $P,D_0,...,D_n \models \phi$, where P is called transaction base, that is a set of TR formulas, like is ϕ. Although, each D_i is a set of first-order formula, where each formula is called literal, that represents the database state. Intuitively, P is a set of transactions definitions, ϕ is an invocation of these transactions, and $D_0,...,D_n$ is the sequence of database, representing all the states of transaction execution. For example, calling the transaction ϕ, the database goes from initial state D_0 to final state D_n. However, if ϕ is a query, the database will not change, and it can be represented by $P,D \models \phi$.

In TR, a database state is defined by data oracle O^d. For each state identifier, *i*, $O^d(i)$ is a mapping of *i* and a set of first-order formulas that represent the truth of database state. Depending on the application domain, different kinds of data oracles can be specified. The more common is the generalized-Horn data oracle, where $O^d(i)$ is a set of generalized-Horn formula[1], and it will be used in this paper. A query is controlled by O^d. For example, consider a literal called *onfloor* that is true in state D_k. So, $onfloor \in O^d(D_k)$ and $P,D_k \models onfloor$.

The TR also works with updates, and the specification of transition is done through transition oracle, O^t, that is a mapping function between states pairs and a set of atomic formulas. In this paper, transition will be defined by ins and del predicates. For example: ins(c) and del(d), where, formally: $ins(c) \in O^t(D,D+\{c\})$ and $del(d) \in O^t(D,D-\{d\})$, and $P,D,D+\{c\}, D+\{c\}-\{d\} \models ins(c) \otimes del(d)$.

2.1. Proof Theory

Unlike others logics, the TR semantic is based on paths, and not on arcs between states like modal logics. In TR, a state sequence is verified by states path executed by a transaction, i.e.:$P,D_1,...,D_n \models \phi$; where ϕ is some transaction formula, where $<D_1,...,D_n>$ is a path of states.

The TR has a more restrictive theory that has sound and complete proof theory with the definition of axiom and inference rules. Called serial-Horn version, this restrictive theory uses only the serial conjunction operator ⊗, and has the same characteristics of Horn formula in first-order logic. It consists in transaction base P, which is a set of serial-Horn formula, and of database D, which is a set of first-order formula.

With this restrictive version, an inference system can be defined in order to turn the TR into computational language. The Serial-Horn version has three inference rules, that defines deduction for queries, updates and rules in transaction base P (Bonner and Kifer, 1995). These inference rules are necessary to implement the TR deduction and the executional path.

The implementation of inference rules can be made in PROLOG language. The work of Santos (Santos, 1997) proves the equivalence of the TR serial-horn version semantic, with the use of predicates ins(_) and del(_) by transition oracle, and its implementation in PROLOG. It provided an additional contribution with a demonstration that a representation of the TR serial-horn version can be translated in one, and only one, pure PROLOG program.

3. CORRECT AND COMPLETE PLANNERS

Based on (Santos and Rillo, 1997), consider ***L*** an serial-Horn version language of TR, an instance of

[1] Generalized-Horn formulas are first-order formulas with negation.

an action system $\sum = <\boldsymbol{W,K,A}>$ is a triple, with the world state $\boldsymbol{W}$, the domain rules $\boldsymbol{K}$ and the set of action $\boldsymbol{A}$. Where $\boldsymbol{W} \subseteq \boldsymbol{L}$ is a set of literal described in first order logic, $\boldsymbol{K} \subseteq \boldsymbol{L}$ are classical Horn rules in first order logic and each α, $\alpha \in \boldsymbol{A}$, is represented by a serial-Horn formula.

Each serial-horn formula, that represents an action $\alpha \in \boldsymbol{A}$, needs to be in compliance with transition oracle O^t. This is necessary because an action is responsible for state change, and so, it will be in its composition some predicates ins(_) and del(_). The state change caused by an action is called *action effects*. Some works on the formalization of actions using TR can be founded in (Bonner and Kifer, 1995; Santos, *et al*., 1996; Santos, 1996; Santos and Rillo, 1997). Therefore, with the language structure $\boldsymbol{L}$, some definitions in TR framework can be established:

Definition 1: *The database D in TR represents the truth about world state in Closed-World Assumption and where* $\boldsymbol{W}$ and $\boldsymbol{K} \subseteq D$.

As the database is in Closed-World Assumption, all predicates that are not present in database are supposed to be false.An example of a database D is the set of predicates that indicates the state of each thing in the car construction domain:

painting_machine(red).
car_setup(sedan).
Robot1_at(room1).
...

With the formal specification of an action instance in TR, a plan, a case (stored plan) and a goal can be defined in logical limits. The TR serial-conjunction operator may the existence of sequence of actions, which characterize a plan in planning.

Definition 2: (Plan) *A plan* $\delta = \alpha_1 \otimes .... \otimes \alpha_n$ *is a serial-Horn formula, where* $\alpha_i \in \boldsymbol{A}$*;* $1 \leq i \leq n$.

A plan can change the world (database) as:
$P,D_0...D_n \models \delta$, where $\delta = \alpha_1 \otimes \alpha_2 \otimes \alpha_3$ and $P,D_0...D_i \models \alpha_1$; $P,D_i...D_j \models \alpha_2$ and $P,D_j...D_n \models \alpha_3$. An example of a plan to construct a blue-colored car with 8cl engine is: ...⊗ weld(roof) ⊗ paint(car_structure, blue) ⊗ put(engine,8cl) ⊗ ... Each action will change the state in order to make a path of car state in each situation.

Definition 3: *An empty plan* δ_0 *is defined as a plan without any action* $\alpha, \alpha \in \boldsymbol{A}$**.**

Definition 4: *An instance* ***pln*** *can be a plan* δ *or an empty plan* δ_0.

Definition 5: (Goal) *A goal is a serial-Horn formula and has the following structure:*

Goal: $\boldsymbol{pln} \otimes \mathbf{D_f}$;

where D_f *is a queries set in TR that represents desirable literal in the world after the plan execution and* ***pln*** *is an instance that can be a plan* δ *or an empty plan* δ_0.

After the planner finds a desirable instance of ***pln***, the plan, after some changes, becomes a case to be stored for future uses. For all, the plan is completed by sets of literal, that characterize the initial and final world of the plan. So, let be a case as plan modification:

Definition 6: (Case) *A case* η *is a serial-Horn rule and has the following structure:*

$$\boldsymbol{\eta} \leftarrow \boldsymbol{W_i} \otimes \alpha_1 \otimes .... \otimes \alpha_n \otimes \boldsymbol{W_f}, \quad ; \text{where:}$$

- $\boldsymbol{\eta}$ *is a serial-Horn rule that represents the stored case.*
- $\alpha_i \in \boldsymbol{A}$*;* $1 \leq i \leq n$*, a sequence of actions defined by planner to satisfy a proposed goal.*
- $\boldsymbol{W_i}$ *is a set of queries in TR that represents the literal at initial state and that it will be not true after the execution of the sequence of actions*
- $\boldsymbol{W_f}$ *is a set of queries in TR that represents the literal that doesn't exist before the execution of the sequence of actions and that it will be true at the final state*

An example of a case is: $\eta \leftarrow$ robot1_at(room5) ⊗ .... ⊗ paint(car_structure,black) ⊗ put(engine,4cl) ⊗ ... ⊗ painting_machine(black) ⊗ car_setup(hatch).
Where: W_i = robot1_at(room5); and
W_f = painting_machine(black) ⊗ car_setup(hatch).

Therefore, the TR transaction base P can be defined:

Definition 7: *A transaction base P in TR is a set of serial-Horn rules that represents actions and stored cases.*

A planning problem is characterized by $\psi = <\boldsymbol{D_0, K, A, G}>$, where $\boldsymbol{D_0}$ is the initial state, $\boldsymbol{K}$ is the causal domain knowledge, $\boldsymbol{A}$ is actions set and $\boldsymbol{G}$ is the goal.

Consider a planning problem ψ and a complete and sound knowledge base, the soundness and the completeness of a plan with relation to (w.r.t) proposed planning problem ψ can be defined. In (Hertzberg and Thiébaux, 1994), the intuition to say that a plan is correct is to know if a plan says that something is or can be true, then it really is or should be. In fact, this is only guaranteed if the planning problem ψ is sound and complete w.r.t application domain. So that, can be defined the correctness of a plan, w.r.t ψ, following the steps of (Hertzberg and Thiébaux, 1994):

Definition 8: *A plan* Π *is an directed graph composed of nodes and branches, where:*

1. *All branches represent the invocation of actions from **A** set.*
2. *The graph root is the state D_0. All nodes represent states of the database and they stay on graph only once.*
3. *Each branch has only nodes as successor.*
4. *Excepting the root, all nodes have a branch as predecessor*
5. *All leafs are nodes*
6. *From each node leaves, at least, one path that achieve the leaf node.*
7. *Only leafs don't have any branch that go to others nodes.*

But only with the definitions above is still possible to create plans without any sense and that not achieve the goal G. So, a plan δ can only be correct w.r.t ψ, if it has the following conditions:

Definition 9: (Soundness and Completeness of a plan) *Consider a plan Π, and a planning problem ψ:*

***Soundness of roots**: $D_0 \in \Pi$ is sound w.r.t ψ iff for all $\alpha \in \mathbf{A}$ of the D_0, α is executable from D_0, or formally, $P,D_0 \ldots \models \alpha$.*

***Soundness of branches**:Let α a branch of the Π and let be D_i its predecessor node, α is sound w.r.t ψ iff for all α successor node D_j in Π, $P,D_i \ldots D_j \models \alpha$.*

***Soundness of leafs**: A leaf D_n is sound w.r.t ψ iff $P,D_n \models D_f$*

***Soundness of plans**: Π is sound w. r.t. ψ iff:*
1. *$D_0 \in \Pi$ is sound w.r.t ψ;*
2. *All branches of Π are sound w.r.t ψ.*
3. *All leafs of Π are sound w.r.t ψ.*

***Completeness of roots**: $D_0 \in \Pi$ is complete w.r.t ψ iff for all $\alpha \in \mathbf{A}$, there is a branch that leaves D_0, such that $P,D_0 \ldots \models \alpha$.*

***Completeness of branches**: Consider a branch of the Π and let be D_i its predecessor node, α is complete w.r.t ψ iff for all successor node of α there is a D_j node such that $P,D_i \ldots D_j \models \alpha$.*

***Completeness of plans**: Π is sound w.r.t. ψ iff:*
1. $D_0 \in \Pi$ is complete w.r.t ψ;
2. All branches of Π are complete w.r.t ψ.

The plan completeness conception, also by (Hertzberg and Thiébaux, 1994), is related to the soundness. The idea is to say that if something can happen by execution of a plan in accordance with ψ, then the plan will respect it. This means that a plan must allow that all possible worlds can be initial situations, and that it must have all the actions results which are directed to apply.

Due to the notion of soundness and completeness of defined plan, the soundness and completeness of a planner P can be defined, that it is nothing more than a plan generator.

Definition 10 (Soundness of P) *A planner P is sound iff the plan Π generated by $P(\psi)$ is sound and complete w.r.t for all planning problem ψ.*

Definition 11 (Completeness of P) *A planner P is complete iff for all planning problem ψ and for all soundness and completeness plan Π w.r.t ψ, there is $P(\psi) = \Pi$.*

With the use of the TR in case-based planning, the definition of a planner in theoretical framework and its practical implementation is possible, through proof theory of TR. The planner P, that is a case-based planner, has the cases retriever phase, adaptation phase and storing phase. Each phase is added in order to find, for each ***pln*** instance, a sound and complete plan.

4. RETRIEVER IN TR

In a case-based planning system, the retriever of cases previously stored and similar to goal is one of principal points of system's efficiency. Depending on the method used in the system, it can be faster or slower, i.e., the time to find a plan that satisfy the goal depends on the cases retriever time.

The case retriever has a property to restore features of all states with the use of TR. Each case has in its formula composition some features from initial state, represented by W_i, and from goal responsible for its case generation, W_f. With the TR semantic based on states path is possible to obtain sub-cases, with an analysis from executional path of the case.

However, a CBP system needs a method of cases retriever that can find, besides similar cases, sub-cases conjunctions that, when gathered, provide more similarity then isolated. This kind of retriever process has being made in some CBP systems, like PRODIGY (Veloso, 1994), CAPER (Kettler,1995) and MPA (Ram e Francis, 1994).

A CBP system formalized with TR needs to contemplate all situations of possible cases and sub-cases. Therefore, a method of two Cases-Sets (CS) is proposed, where each CS is a set of TR formulas, and where each formula represents a case or a sub-case. The Cases-Sets are filled by similar cases restricted by some kind of limits in ***pln*** instance. The retriever uses CS to satisfy the constraints for each ***pln*** instance founded in an incomplete plan.

There are two types of possible constraints, which are before-constraint and after-constraint. The before-constraint is a constraint before the ***pln*** instance, and it can be:
Initial state constraint: ***pln*** ⊗ ....
Action Constraint: ... ⊗ α_i ⊗ ***pln*** ⊗...

So, a CS_1 will be responsible for the existing restriction, and it will have in its composition similar cases that should satisfy in some percentage the constraint before the ***pln*** instance. For example, a subset of W_i, called W_{is}, can be satisfied in D_0, $P,D_0 \models W_{is}$; or there will be a case with the before action, responsible by constraint, in its composition.

Restrictions after ***pln***, called after-constraint, can be:
Final state constraint: ...⊗ ***pln*** ⊗ D_f
Action Constraint: ... ⊗ ***pln*** ⊗ α_i ⊗ ...

In addition, for this constraint, another CS, called CS_2, will be responsible to get cases that satisfy, in some percentage, the after-constraint. For example, cases that has a subset of W_f , called W_{fs}, where $W_{fs} \subseteq D_f$, or cases that have in its composition the action responsible for the existence of the after-constraint. An example of the power of the action constraint, suppose the following after-constraint: ... ⊗ ***pln*** ⊗ paint(car,blue) ⊗ ... So, the planner must find a proper plan to substitute ***pln*** that needs change the color of the painting_machine by blue, for example.

The ***pln*** instance will use two sets, one to satisfy the before-constraint and another to satisfy after-constraint. In order to turn a retriever phase an efficient process, is necessary a crossing between the both cases-set to find cases that have similar intermediary states among them, and then, they can be jointed to create a new case to satisfy the both constraint type. The use of the TR may the crossing between cases in both sets easier, because it has a semantic structure based on paths of states.

If a case is presented in both Cases-Set, it can be used directly, because it satisfies as the before-constraint as after-constraint. When a case has not total match, or total satisfaction, with the before-constraint or after-constraint, a new ***pln*** instance is included at that point. Each ***pln*** instance shows that a case must be adapted or a new case must be retrieved to satisfy it.
As retriever process adds ***pln*** instances in not-total matching points, the retriever keeps the soundness and completeness of the system, because its permits the possibility to find the place that needs a plan generation, by the adaptation process. The system only keeps the soundness and completeness if the chosen adaptation process keeps it too.

5. ADAPTATION IN TR

The adaptation of a CBP is responsible for the soundness and completeness of the system, because it guaranties the construction of a plan from scratch if the retriever doesn't find any similar case. The majority of sound and complete CBP systems have generative planners, that planning from scratch, in its composition, as MRL (Koehler, 1996) and PRODIGY (Veloso, 1994).

The adaptation needs to satisfy the existing constraint and the final goal. For each existence ***pln*** instance, the adaptation process must find a plan that satisfies both constraints, before and after-constrain. Therefore, the adaptation must be capable of finding a plan to refit the case retrieved or find a plan with from scratch planning, through a search method, to achieve the proposed goal in ψ.

The used searching method with the TR formalization is the state-space based, because the TR semantic is based on path of states, and shows, by deduction, all the states that plan went through to achieve the goal.

With the adaptation process defined by a soundness and completeness algorithm, the construction of a sound and complete planner $P(\psi)$ can be guaranteed. In the plan creation process is necessary, for each ***pln*** instance founded in goal, to decide if the retriever is better than the adaptation. To define this question is necessary a complete study of computational costs of the retriever and the adaptation for each ***pln*** instance, and so, decide for the lower computational cost. It can be made by a kind of heuristic process depending on the application domain.

6. STORING THE CASES

In TR, the plan execution makes a path of states in accordance with executional deduction of the TR theory. This path is a plan, or in others words, it shows the changing states behavior. Therefore, the storing happens into the logical structure itself.

The storing is defined by the reconstruction of a plan by the executional deduction of TR. For example, consider a plan δ satisfied from initial state D_0, and consider D_n the final state after the plan execution, the executional deduction provides a states path: $P, D_0.... D_n \models \delta$; where $\pi = <D_0,...,D_n>$ is the execution path.

The construction of the case is possible with the given sequence of states, defining the W_i and the W_f. Due to the storing process, a plan become a new case in memory, and it can be used in others goals latter.

7. CONCLUSION

The Transaction Logic, as described in this paper, due to its semantic based on path of states, for working with sequence of formulas and a theory that makes possible the use of a database controlled by oracles, it meets the needs of planning systems.

The Transaction Logic, for its inference rules, makes possible the computational implementation of any theory described in serial-Horn version of TR, and, by the work of Santos (1997) , any conclusion extracted from TR execution is also extracted from PROLOG program resulted by application of translation rules presented in (Santos, 1997). So that, for being a logical theory that has a computable version, the TR provides a better direct implementation of formalized systems, avoiding the semantic gap between theory and implementation that exists in others logics used by some planning system.

The description of plans, cases, goals and actions is still clean and precise. This is possible because the TR has a clean syntax as first-order logic. Others logics, like temporal logic and modal logic, have confuse syntax when used in a complex formula, that makes difficult the definition of actions and plans and its comprehension.

A CBP system that has a complete formalization in a logical theory is the MRL system (Koehler, 1996), that uses the LLP logic in a generative planner called PHI and a descriptive logic to formalize the case retriever. The MRL system presents some problems. Firstly, it does not return cases and sub-cases composition, and second the LLP logic has syntax based on modal logic and so, its syntax is confuse. Besides that, it has many inference rules, what makes difficult its computational implementation.

The system presented by this paper, by the use of transaction logic as a formalization theory, improves the extraction of sub-cases from cases, through the analysis about executional path produced by the case, allowing a composition of cases and sub-cases by means of finding a better and possible solution for a new planning problem. For all these features, the TR becomes one of the power logics to formalize planning systems, making possible a clean representation of cases and plans, and a perfect definition of a sound and complete CBP system, as a retriever, as cases adaptation, as well as its capacity to be implemented in a computational way.

ACKNOWLEDGMENTS

This work was supported by a grant from the Brazilian Research Council - CNPq. Thanks to Marcus V. Tolentino dos Santos for the discussions on Transaction Logic in planning.

REFERENCES

Bonner, A. J. and M. Kifer (1995). *Transaction logic programming*. Technical Report CSRI-323. University of Toronto, Toronto.

Hammond, K. (1989) *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press Inc.

Hertzberg J. and S. Thiébaux (1994). Turning an action formalism into a planner - a case study. *Journal of Logic and Computation*, **4**, 617-654.

Kettler, B. P. (1995). *Case-Based Planning with a High-Performance Parallel Memory*. Ph.D. Thesis, University of Maryland Park, Maryland.

Koehler, J. (1996). Planning from Second Principles. *Artificial Intelligence*, **87.**

Ram,A and A.G.J. Francis (1996) Multi-Plan Retrieval and Adaptation in an Experience-Based Agent. In, *Case-Based Reasoning: Experiences, Lessons and Future Directions* (Leake,D.B. Ed.), pp. 167-184, AAAI Press, Massachusets.

Rillo, M. (1994) *A Robotized cell with high-level autonomy*. (*Uma célula robotizada com alto grau de autonomia).* Technical Report University of São Paulo, São Paulo.

Santos, M.V.T.; P.E. Santos; F.C.S. Silva and M. Rillo (1996). Actions as prolog programs. In: *IEEE Proceedings of the Joint Symposia on Intelligence and System*, pp. 178-183 Washington.

Santos, M. V. T. (1996). On the formalization of actions using transaction logic. In*: Proceedings of 12th ECAI - Worshop on Cross-fertilization in planning*, Budapeste.

Santos, M.V.T. and M. Rillo (1997). Approaching the Plans are Program paradigm using transaction logic. In: *Proceedings of Forth European Conference on Planning* - ECP '97. Toulouse.

Santos, P. E. (1997). *Equivalence between transaction logic semantic and its implementation semantic in PROLOG* (Equivalência entre a semântica da lógica de transações e a semântica de sua implementação Prolog). Master's Dissertation, University of São Paulo, São Paulo.

Veloso, M. (1994). prodigy / analogy: Analogical Reasoning in General Problem Solving. In: *Topics on Case-Based Reasoning*, S.Wess, K.D.Althoff and M. Richter Ed. pp. 33-50, Springer Verlag.