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

ON THE TREATMENT OF THE RAMIFICATION PROBLEM WITH TRANSACTION LOGIC

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**:** In the planning area, the reasoning about action is important, because the plans are made by a sequence of actions. Each action has its effects that need to be treated and formalized in some logical framework.The use of Transaction Logic (TR) makes possible a perfect definition of fluents to treat the effects of ramification problems, because the TR works with database controlled by transition and data oracles. The TR has a powerful set of inference rules that makes possible its implementation in PROLOG, like SLD-resolution, allowing the application of effects in a computational way. *Copyright © 1998 IFAC*

Keywords: Artificial intelligence, computational methods, databases, formal languages, reasoning

1. INTRODUCTION

In the planning area, the reasoning about action is important, because the plans are made by a sequence of actions. So, a deep study of how to treat the ramification problem in actions is essential, in order to provide for planning systems correct effects caused by each action presented in a plan.

In Ramification Problem, an action can cause extended effects that need to be detected. As used in (Giunchiglia and Lifschitz, 1995), there are two kinds of fluent that simulate the problem: the inertial and the dependent. The first is characterized by its continuos existence since the fluent that caused it comes to the end. The second is different, it means that a fluent depends on values of others fluents.

In the Ramification Problem, translating all the consequences of actions into formal framework is difficult, in order to control the reasoning about change caused by actions. For example, if a box will be moved, all the things inside will move along with it. The papers on top will move also, but the lamp on top will roll along the box's surface and will fall down. These kind of effects are natural in human intelligence for reasoning about change, but they are hard to be expressed in another language, as in logical framework.

Many logical theories were proposed, like the situation calculus (SITCAL) (Lin, 1996) based on first-order logic, the temporal and the modal logics, to work with reasoning about action and its effects. However, they are difficult to work well with fluents, because they don't work with databases, what make difficult the definitions of dependent fluents with persistence feature. In (Giunchiglia and Lifschitz, 1995), another action language was proposed, called ARD, that has the characteristic to work with the dependent fluents, but difficult to use in planning.

Therefore, to formalize the ramification problem in some logical framework that allow its use in the planning area, a kind of logical theory that works with update in databases, that makes possible the treatment of fluent and that has a powerful tool to work with the relation of actions to fluent is essential.

The use of Transaction Logic (TR) (Bonner and Kifer, 1995) can solve all the necessities described above, because it works with database update controlled by transition oracle, it allows the treatment of fluent behavior by the use of data oracle, and it has a powerful set of inference rules that make possible the relation between TR formulas and the database.

Some papers, like (Bonner and Kifer, 1998; Santos, 1996; Santos and Rillo, 1997), work with formalization of actions in TR without any concern with dependent effects of them. So, this paper will provide a logical treatment of the ramification problem in the Transaction Logic Framework, and will show how to implement inertial and dependent fluent in PROLOG.

2. OVERVIEW OF TRANSACTION LOGIC

The Transaction Logic is an extension of first-order logic, with the introduction of a new operator called serial conjunction ($\otimes$). This operator represents a procedural activity, where $\alpha \otimes \beta$ means "first execute α, and then execute β". To describe a transaction, consider this notation: $P,D_0,...,D_n \models \Psi$.

Where P is called transaction base, that is a set of TR formulas, like is Ψ. Although, each D_i is a set of first-order formula, that represents the database state. Intuitively, P is a set of transactions definitions, Ψ is a 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 Ψ, the database can go from initial state D_0 to final state D_n. But, if Ψ is a query, the database will not change, and will be represented by $P,D \models \Psi$.

2.1. Query and Update Operations

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 is the kind of data oracle that will be used in this paper.
The TR works with updates through transition oracle, O^t, that is a mapping function between states pairs and a set of atomic formulas. For example, a first-order formula λ can be inserted with λ.ins, that can change a database D_0 to $D_1=D_0+\{\lambda\}$. So, $P,D_0,D_1 \models \lambda.ins$ because $\lambda.ins \in Ot(D_0,D_1)$. The same can be made by λ.del, as $P,D_0,D_1 \models \lambda.del$ iff $\lambda.del \in Ot(D_0,D_1)$ where $D_1=D_0 - \{\lambda\}$. This paper will use the predicates ins(λ) and del(λ) to represent λ.ins and λ.del.

It can be observed that, depending on the application domain, there can be many kinds of transition oracle. E.g., it can be specified that some predicates can't be removed from database, or that others can't be inserted. This shows that the TR may have a independent domain application without changing the TR formal structure, through specification of the transition oracle.

2.3. Models 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.: $D_1,...,D_n \models \Psi$, where Ψ is some transaction formula.

The TR has its model restricted by the both oracles, data oracle and transition oracle. So, the TR needs the compliance with both oracles such that:

1. If $O^t(D_0,D_1) \models \alpha$ then $P,D_0,D_1 \models \alpha$; where α is a TR formula.
2. If $O^d(D_0) \models \alpha$ then $P,D_0 \models \alpha$; where α is a first order formula.

2.4. Proof Theory

The use of a more restrictive theory of the TR, called serial-Horn version, is necessary to provide TR with a sound and complete proof theory. The serial-Horn formula uses only the serial conjunction operator $\otimes$, 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, where a transaction can insert or remove formula. With this restrictive version, a inference system as SLD-resolution in PROLOG can be defined:

Definition 1 (Inference System) *If P is a transaction base, then the inference system is the following scheme of axiom and inference rules, where D and Di are any database state identifier.*

AXIOM : P,D ... |- ().
INFERENCE RULES: In rules 1-3 below, σ is a substitution, a and b are atomic formulas, and Φ and rest are transaction formulas.

1. Defining transaction: if $a \leftarrow \Phi$ is a rule in P and if a and b unify with mgu σ, then

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

$$\frac{P,D...\vdash (\exists)(\Phi \otimes rest)\sigma}{P,D...\vdash (\exists)(b \otimes rest)}$$

2. Querying the database: if $b\sigma$ and $rest\sigma$ share no variables, and $O^d(D) \models (\exists)\, b\sigma$, then

$$\frac{P,D...\vdash (\exists)rest\sigma}{P,D...\vdash (\exists)(b \otimes rest)}$$

3. Elementary updates: if $b\sigma$ and $rest\sigma$ share no variables, and $O^i(D_1,D_2) \models (\exists)b\sigma$, then

$$\frac{P,D_2...\vdash (\exists)rest\sigma}{P,D_1...\vdash (\exists)(b \otimes rest)}$$

2.5. Implementation of the TR in PROLOG

The serial-Horn version implementation was studied by Hung (1996), where was proposed an algorithm of insertions and removals of predicates in PROLOG that works with *backtracking*, defining for this some predicates like *ins* and *del*.

The Hung's implementation defines a set of rules that specifies the transition oracle, establishing a logical treatment for assert and retract predicates of PROLOG through onbktk(_) predicate.

The work of 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, and an additional contribution with a demonstration that a representation of the TR serial-horn version can be translated in one, and only one, PROLOG program. The equivalence between the TR semantic and the PROLOG program was made by definition of a translation function to PROLOG from TR with Hung's algorithm. The translation function defined in (Santos, 1997) is the following:

Definition 2 (Translation Function τ) *Given a TR formula ϕ, its translation function, described as $\tau(\phi)$ to PROLOG is defined as:*

1. $\tau(q) = qry(q)$ *for all q = query on database*
2. $\tau(a \otimes b) = \tau(a)\,,\ \tau(b)$
3. $\tau(a1 \otimes a2 \otimes a3 \otimes ... \otimes an) = \tau(a1)\,,\ \tau(\beta)$ *for* $\beta = a2 \otimes a3 \otimes ... \otimes an$
4. $\tau(p \leftarrow a) = p{:-}\ \tau(a)$
5. $\tau(p \leftarrow a) = db(p) :-\ qry(a)$ *if $p \leftarrow a$ is a rule in database.*
6. $\tau(\neg a) = \backslash +\ \tau(a)$
7. $\tau(ins(X)) = ins(X)$
8. $\tau(del(X)) = del(X)$
9. $\tau(f) = f$ *where f is a function*

```
onbktk(X).
onbktk(X) :- call(X), !, fail.
insp(X) :- assert(X), onbktk(retract(X)).
delp(X) :- retract(X), onbktk(assert(X)).
delp(X).
ins(X) :- not (db(ip(X))), insp(db(ip(X))).
ins(X) :- db(ip(X)), delp(db(dp(X))).
del(X) :- db(ip(X)), ins(db(dp(X))).
```

Fig. 1. The Hung's algorithm.

Santos defines, based on Hung's algorithm showed in figure 1, a system called Ξ that change the assert and retract predicates of the PROLOG language by pos(_) and neg(_) predicates, as matter to use only a pure PROLOG program. This system Ξ, that implements the transition oracle and database of the TR in PROLOG, is used as a fixed algorithm that must be jointed with the resulted algorithm by translation function.

Therefore, Santos proved the equivalence between the serial-horn version of TR with the PROLOG program made from translation function and system Ξ. For this proof, Santos used the generalized immediate consequence operator $T_p^j(I)$ (Apt, 1986) that turns the minimal model of Herbrand of a logical program P, where J is a pre-interpretation of an first-order language, and I is an interpretation based on J, which can be a model of P iff $T_p^j(I) \subseteq I$ (Apt, 1986). So, with the use of $T_p^j(I)$, Santos defined the following corollary and theorem:

Theorem 3 (TR equivalence) *Consider a serial Horn program P and its respective translation in pure PROLOG P', where $P' = \tau(P)$ and τ is the translation algorithm. So, P and P' are equivalent.*

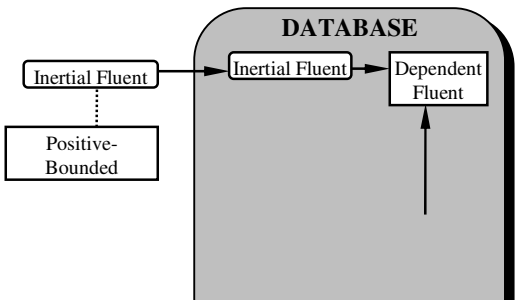
Proof: in (Santos, 1997).

In addition, the translation function expressed in theorem above uses the Ξ system.

Corollary 4 *if $P,D_0...D_n \vdash \phi$ then $\phi' \in \{T_p^j(I) \uparrow k\}$, where $k=n-1$.*

Proof: in (Santos, 1997).

The corollary states that if a formula ϕ, wrote in TR, can be proved with the use of P in the path $\langle D_0...D_n \rangle$, then the translation of ϕ, $\phi' = \tau(\phi)$, is presented after k interaction of the T_p^j operator, where k=n-1. With (Santos, 1997), the serial-Horn version of the TR can be implemented in pure PROLOG without changes in semantic structure, what makes possible the implementation of formalized systems with the TR serial-horn version in a computational language



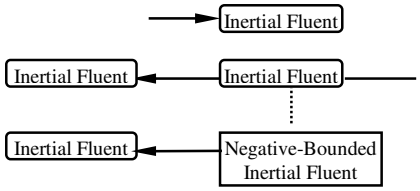


Fig. 2. The fluent schema in database and it's update. The dotted line represents the bounding property and arrows the flow.

3. RAMIFICATION PROBLEM

On the ramification problem, the difficulty is to determine the indirect effects of the all actions in knowledge base. The Transaction Logic become a powerful logic to declare indirect effects because it has a inference mechanism in accordance with the both oracles that states all the truth about database.

Considering each first-order formula in the TR database as a fluent, by the work of Giunchiglia and Lifschitz (1995), can be defined the indirect effects as two kind of fluent:

- Inertial Fluent
- Dependent Fluent

The inertial fluents (f_I) are the fluents that don't depend on others fluents to be true in any world state. The dependent fluents (f_D), differently, only become true in some world state if some inertial fluents is true or false at that state. On this basis, inertial and dependent fluents in TR are defined:

Definition 5: *The inertial fluents f_I are divided into three fluents:*

f_I = *not-bounded inertial fluent*
f_{IN} = *negative-bounded inertial fluent*
f_{IP} = *positive-bounded inertial fluent*

The division of inertial fluent is necessary to determine the persistent dependent fluent, what means that some fluents are dependent only in some situations. For example, f_{IP} is inserted in database if some f_I is inserted too, but if this f_I is removed from database, the f_{IP} doesn't get out with. The figure 2 can show the fluent behavior from the definition above.The fluents are defined by Transition Oracle:

Definition 6 (Not-bounded inertial fluent) *A Not-bounded inertial fluent is defined as:*
$ins(f_I) \in O^t(D_1,D_2)$ *iff* $D_1=D_2+\{f_I\}$
$del(f_I) \in O^t(D_1,D_2)$ *iff* $D_1=D_2-\{f_I\}$
The f_{IP} is defined as the fluent that comes into database bounded with some not-bounded inertial fluent f_I:

Definition 7 (Positive-bounded inertial fluent) *An Positive-bounded inertial fluent is defined as:*
$ins(f_{IP}) \in O^t(D_1,D_2)$ *iff* $D_1=D_2+\{f_I,f_{IP}\}$

The f_{IN} is defined as the fluent that comes out database bounded with some not-bounded inertial fluent f_I:

Definition 8 (Negative-bounded inertial fluent) *A Negative-bounded inertial fluent is defined as:*
$del(f_{IN}) \in O^t(D_1,D_2)$ *iff* $D_1=D_2-\{f_I, f_{IN}\}$

An example, as in (Giunchiglia and Lifschitz, 1995), can be showed as: If some product is in the water, then it gets wet, and it will still be wet after comes out the water. So:

f_{IP} = InWater(product)
f_I = Wet(product)

Inserting into database the InWater(product) fluent, the Wet(product) fluent will be inserted too. It can be notice that Wet(product) fluent is not dependent of InWater(product), because Wet(product) is still true independent on the existence or not of InWater(product) in database.

Definition 9 (Dependent Fluents) *A f_D is defined as a TR database rule and has the follow structure:*

$f_D \leftarrow f_{I1} \wedge f_{I2} \wedge \ldots \wedge f_{In}$; $n>0$
Where each f_{Ii} can be negative or positive.

In order to guarantee that the set of rules in database doesn't allow the looping definition, the definition of dependent fluents was made as the classical conjunction of inertial fluents instead of $f_D \leftarrow f_D$.

The existence of a positive-bounded and negative-bounded fluents is necessary for the same reason that dependent fluent is dependent of inertial fluent only, that is to avoid the looping rules.

An example of dependent fluent, as in (Giunchiglia and Lifschitz, 1995): Consider a key that while being in downward, the human that passes in front of that machine is safety. So,

f_D = safety (human)
f_I = position (key, down)
safety (human) ← position (key, down).

On this way, safety(human) will only be true if the key is downward, and if not, the safety of the human that is passing in front of that machine will be not guaranteed. This is different of the persistence feature, because the fluent safety(human) depends on the existence of the other to be true.
The definitions need to be verified if they don't change the TR model, and consequently the inference rules and the TR soundness and completeness.

The TR model is based on paths, and these paths, as model definition, must be in accordance with O^d e O^t as follows:

1. If $O^t(D_0,D_1)|= \alpha$ then $P,D_0,D_1|= \alpha$;where α is a TR formula.
2. If $O^d(D_0)|= \alpha$ then $P,D_0|=\alpha$; where α is a first order formula.

All theories of TR are made with a model that has the compliance above. So, the TR model will not change with the definitions above because it needs to be in accordance with them.

4. NEW IMPLEMENTATION IN PROLOG

The work of Santos (1997), in the equivalence proof of the semantic between pure PROLOG and the TR, used an algorithm called system Ξ, that is the implementation (representation) of the Transition Oracle in pure PROLOG. In addition, as Ξ extension, the system Ξ^R can be defined. However, to use the new system in PROLOG, the semantic equivalence analysis is necessary. So, following the Santos' steps, the new system Ξ^R can be defined.

Definition 10 (System ΞR) *A Ξ^R system is defined as being the following algorithm:*

```
onbktk(X).
onbktk(X) :- X,
pos(_):-
neg(_):-
insp(X) :- pos(X), onbktk(neg(X)).
delp(X) :- neg(X), onbktk(pos(X)).
delp(X).

insb(X) :- not (db(ip(X))), insp(db(ip(X))).
insb(X) :- db(ip(X)), delp(db(dp(X))).

ins(X):-¬(db(ip(X))),bounded_ins(X),
insp(db(ip(X))).
ins(X) :- db(ip(X)), bounded_ins(X), delp(db(dp(X))).
ins(X) :- ¬bounded_ins(X), insb(X).

delb(X) :- db(ip(X)), insp(db(dp(X))).

del(X) :- db(ip(X)), bounded_del(X), insp(db(dp(X))).
del(X) :- ¬bounded_del(X), delb(X).
```

To use the definitions of inertial and dependent fluents, the new rules in Ξ^R system are necessary:

Definition 11: *Given a transition oracle, its translation function will complete the Ξ^R as follows:*

1. For each definition: $f_{IP}.ins \in O^t(D_1,D_2)$ iff $D_1=D_2 + \{f_I f_{IP}\}$, the rule will be included in Ξ^R system:
$bounded\_ins(f_{IP})$:- $insb(f_I)$ in PROLOG.

2. For each definition: $f_{IN}.del \in O^t(D_1,D_2)$ iff $D_1=D_2 - \{f_I f_{IN}\}$ the rule will be included in Ξ^R system:
$bounded\_del(f_{IN})$:- $delb(f_I)$ in PROLOG.

So, with the definitions above, the Ξ^R system will be given by the algorithm described in definition 10 and the bounded_ins and bounded_del rules introduced by definition 11 following the transition oracle conditions.

4.1. Equivalence Proof between the TR and PROLOG for the Ξ^R system.

Santos proved the equivalence between TR and PROLOG through the proof of the following three items:

1. $P \Leftrightarrow MP^*$
2. $P \Leftrightarrow P'$
3. $P' \Leftrightarrow \{ T_p^j (I) \uparrow k \}$

The proof of the first item is presented in (Bonner and Kifer, 1995). The second item keeps the same proof of Santos (1997), because the translation function is the same, and the Ξ^R system still does not allow the double definition from the same transition oracle.

In the third item, the proof is the following: Consider a formula ϕ such that P,D --- |= ϕ, it can be proved if after the translation of the P program in P' and ϕ in ϕ', $\phi' \in \{ T_p^j (I) \uparrow k \}$. As the new system Ξ^R will keep the same proof of Santos (1997), it happens because the new pre-interpretation J will consider the new predicates bounded_ins and bounded_del, and keeping the pos(_) and neg(_) predicates, the tree of possibilities from them will be such that follows the path of states. Therefore, the conclusion obtained is similar to Santos' one, a path of the TR execution of length n will be equivalent to the interpretation given after k interactions of $T_p^j(I)$, such as k=n-1.

Keeping the same equivalence proof, the system Ξ^R keeps the equivalence of the TR semantic and pure PROLOG semantic, and it also keeps the corollary 4 and theorem 3.

5. CONCLUSION

Due to the increase of systems in formal description, the TR stands out its sub-theory called serial-Horn that has inference rules which makes possible the computational application. Santos (1997) proved the semantic equivalence between serial-Horn version of the TR and the semantic of a pure PROLOG program acquired by translation function, making possible that any system formalized in serial-Horn version can be implemented in PROLOG.

So that, many works in reasoning about actions area with a formalism based in TR were developed, showing the power of the TR in definition of actions (Bonner and Kifer, 1994; Bonner and Kifer, 1998; Santos, *et al*., 1996; Santos, 1996; Santos and Rillo, 1997). Others logical theories were also used in this area, but they don't have a database treatment and inference rules to be implemented, as SITCAL and Temporal logic.

The SITCAL (Levesque, *et al*., 1994; Lin, 1996) has some limitation to describe knowledge inherited from limitations of the first order logic. It is difficult to work with non-monotonic actions and still doesn't have a database treatment. The implementation of SITCAL in GOLOG (Levesque, *et al*., 1994) was not based in logical inference rules, what makes difficult the implementation of theories described in SITCAL in GOLOG language. Temporal logic have the same problem, it lacks in inference rules and database treatment. In addition, it still has a powerful theory to describe temporal knowledge, however it doesn't have a clean syntax, making difficult a good and easy interpretation of its formulas.

The knowledge representation in TR, therefore, is objective and efficient, overcoming the first order logic and the temporal logic by the existence of a database and a pair of oracles that controls updates and what is true in database. In reasoning about actions, the TR allows the use of inertial and dependent fluent as the recently theories from the area (Giunchiglia and Lifschitz, 1995), that can be applied in production planning in manufacturing areas for a perfect definition of actions effects. It allows its implementation in PROLOG without losing the semantic equivalence with formal theory.

These features make possible a clear and effective manifestation of inertial and dependent fluents, which are very important in the construction of the knowledge base for real applications.

ACKNOWLEDGMENTS

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

REFERENCES

Apt, K.R. (1986) Introduction to Logic Programming. In: *Handbook of Theoretical Computer Science*, p. 495-574. J. Van Leeuwen Ed., North Holland.

Bonner, A.J. and M. Kifer (1994). Applications of Transaction Logic to Knowledge Representation. In: *Proceedings of the International Conference on Temporal Logic* (ICTL), Lecture Notes in Artificial Intelligence, vol. 827, p. 67-81,Springer Verlag, Bonn, Germany.

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

Bonner, A.J. and M. Kifer (1998). Results on Reasoning about Updates in Transaction Logic. In: *Lecture Notes in Computer Science* (B. Freitag, H. Decker, M. Kifer and A. Voronkov Ed.), Vol. 1472, Springer-Verlag, Berlin.

Giunchiglia, E. and V. Lifschitz (1995). Dependent fluents. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence - IJCAI'95*, pp. 1964-1969, Montreal.

Hung, S.K (1996). *Implementation and performance of transaction logic in prolog*. Master's dissertation, University of Toronto, Toronto.

Levesque, H.L.; R. Reiter; Y. Lespérance; F. Lin and R.B. Scherl (1994). GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, **31**, 59-84.

Lin, F. (1996). Embracing Causality in Specifying the Indeterminate Effects of Actions. In: *Proceedings of AAAI-96*, pp. 670-676, AAAI Press, Portland.

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.