Transaction Logic with Defaults and Argumentation Theories[⋆]

Paul Fodor and Michael Kifer

State University of New York at Stony Brook, USA

Abstract. Transaction Logic (abbr., $\mathcal{TR}$) [9,12] is a logic that gracefully integrates both declarative and procedural knowledge. It has proved itself as a powerful logic language for many advanced applications, including actions specification and planning in AI. In a parallel development, much work has been devoted to various theories of *defeasible reasoning* [19]. In this paper, we unify these two streams of research and develop $\mathcal{TR}^{DA}$—*Transaction Logic with Defaults and Argumentation Theories*, an extension of both $\mathcal{TR}$ and the recently proposed unifying framework for defeasible reasoning called *logic programs with defaults and argumentation theories* (LPDA) [53]. We show that this combination has a number of interesting applications, including specification of defaults in action theories and heuristics for directed search in AI problems such as planning. We also demonstrate the usefulness of the approach by experimenting with a prototype of $\mathcal{TR}^{DA}$ and showing how heuristics expressed as defeasible actions can significantly reduce the search space as well as execution time and space requirements.

Keywords: *Action theory, defeasible reasoning, default negation, Transaction Logic, argumentation theory, planning.*

1 Introduction

Transaction logic (abbr., $\mathcal{TR}$) [9,11,12,6] is a general logic for representing knowledge base dynamics. Its model and proof theories cleanly integrate declarative and procedural knowledge and the logic has been employed in domains ranging from reasoning about actions ([13,8]), to knowledge representation ([10]), to event processing ([1]), to workflow management and Semantic Web services ([17,18,43,44]), AI planning ([7,28]), security policy frameworks [5], and general knowledge base programming ([14]).

Defeasible reasoning is another important paradigm, which has been extensively studied in knowledge representation, including in fields such as policies, regulations, law, learning, and others [2,15,16,20,25,26,32,36,40,41,45,54,55].

In this paper we propose to combine $\mathcal{TR}$ with defeasible reasoning and show that the resulting logic language has many important applications. This new logic is called *Transaction Logic with Defaults and Argumentation Theories* (or $\mathcal{TR}^{DA}$) because it extends $\mathcal{TR}$ in the direction of the recently proposed unifying framework for defeasible reasoning called *logic programming with defaults and argumentation theories* (LPDA)

[⋆] This work is part of the SILK (Semantic Inference on Large Knowledge) effort within Project Halo, sponsored by Vulcan Inc. It was also partially supported by the NSF grant 0964196.

[53]. Along the way we define a well-founded semantics [52] for $\mathcal{TR}$, which, to the best of our knowledge, has never been done before.

$\mathcal{TR}^{DA}$ extends traditional logic programming, Transaction Logic, LPDA, and their application domains. Moreover, we show that the combined logic enables a number of interesting applications, such as specification of defaults in action theories and heuristics for pruning search in search-intensive applications such as planning. We also demonstrate the usefulness of the approach by experimenting with a prototype of $\mathcal{TR}^{DA}$ and showing that heuristics expressed as defeasible actions can drastically prune the search space together with the execution time and space requirements.

This paper is organized as follows. Section 2 motivates reasoning with defaults in $\mathcal{TR}$ with an example. Section 3 provides background on Transaction Logic to make the paper self-contained. Section 4 extends $\mathcal{TR}$ by incorporating defeasible reasoning. Section 5 specializes the logic developed in Section 4 by defining a useful argumentation theory that extends Generalized Courteous Logic Programs (GCLP) [36]. In Section 6 we discuss related work and Section 7 provides an empirical evaluation of $\mathcal{TR}^{DA}$ based on a prototype, which we developed. Section 8 summarizes the paper and outlines future work.

2 Motivating examples

In this section, we give two examples in order to illustrate the advantages of extending Transaction Logic with defeasible reasoning.

The syntax of $\mathcal{TR}^{DA}$ is similar to that of standard logic programming except for the fact that literals in the rule bodies are connected via the *serial conjunction*, $\otimes$, which specifies an order of action execution. For instance, $pickup(block1) \otimes puton(block1, block2)$ says that the action $pickup(block1)$ is to be executed first and the action $puton(block1, block2)$ second. The set of predicate symbols of the program is partitioned into:

– a set of *fluents*, which are facts stored in database states or derived propositions that do not change the state of the database; and
– a set of *actions*, which represent actions that change those states.

In addition to the user defined predicate symbols, there are built-in actions called *elementary actions* for basic manipulation of states. These include *delete(f)* and *insert(f)* for every ground fluent f. Examples of such elementary actions include $delete(on(block1, block0)$ and $insert(clear(block0)$.

As usual in defeasible reasoning, rules in $\mathcal{TR}^{DA}$ can be *tagged* with terms. For instance, the rules in the example below are tagged with the constants buy_action and $sell\_action$. The predicate `!`**`opposes`** is used to specify that some rules are incompatible with others (for example, buying and selling the same stock). The predicate `!`**`overrides`** specifies that some actions have higher priority than other actions.

Following the standard convention in Logic Programming, we will be using alphanumeric symbols that begin with an uppercase letter to denote variables. Alphanumeric symbols that begin with lowercase letters will denote constant, function, and predicate symbols.

Example 1 (Stock market actions). Consider a broker who trades stock on the market. He uses a computerized system, which makes various decisions about buying and selling stocks. The system weighs recommendations, which sometimes might conflict with each other, and performs appropriate actions. For simplicity, we ignore issues such as the amount of funds available for purchase and so on.

```
@buy_act  buy(Stock,Amount) :-
         recommendation(buy,Stock) ⊗ owns(Stock,Qty) ⊗
         delete(owns(Stock,Qty)) ⊗ insert(owns(Stock,Qty+Amount)).
@sell_act sell(Stock,Amount) :-
         recommendation(sell,Stock) ⊗ owns(Stock,Qty) ⊗
         delete(owns(Stock,Qty)) ⊗ insert(owns(Stock,Qty-Amount)).
!opposes(sell(Stock),buy(Stock)).
!overrides(sell_action,buy_action).
recommendation(buy,C) :- services(X).
recommendation(sell,C) :- media(X).
services(acme).
media(acme).
owns(acme,100).
trade(Stock,Amount) :- buy(Stock,Amount).
trade(Stock,Amount) :- sell(Stock,Amount).
```

The above rules specify that selling and buying the same stock as part of the same decision is contradictory, so these rules are declared to be in conflict. To be on the safe side, the second rule (sell) is said to override the first (buy). Lets consider an existential goal $(\exists)trade(acme, 100)$. Without the `!`**`opposes`** and `!`**`overrides`** information this goal would have two non-deterministic possible executions: one in which the trader buys an additional 100 stocks in the company acme, and another one in which the trader sells his 100 stocks because he got recommendations both to buy stocks for services companies and to sell the stocks for media companies. However, the second execution is preferred because, in such a contradictory state it's advisable to sell the stocks. □

Example 2 (Block world planning). This example illustrates the use of defeasible reasoning for heuristic optimization of planning in the blocks world. The $\mathcal{TR}^{DA}$ program below is designed to build pyramids of blocks that are stacked on top of each other so that smaller blocks are piled up on top of the bigger ones. The construction process is non-deterministic and several different blocks can be chosen as candidates to be stacked on top of the current partial pyramid. The heuristic uses defeasibility to give priority to larger blocks so that higher pyramids would tend to be constructed.[1]

In this example, we represent the blocks world using the fluents $on(x, y)$, which say that block x is on top of block y; $isclear(x)$, which says that nothing is on top of block x; and $larger(x, y)$, which says that the size of x is larger than the size of y. The action $pickup(X, Y)$ lifts the block X from the top of block Y and the action $putdown(X, Y)$ puts it down on top of block Y. These actions are specified by the second and third rules, respectively. The action $move(X, From, To)$, specified by the first rule, moves block X from its current position on top of block $From$ to a new position on top of block To.

[1] For more information on planning with $\mathcal{TR}$ see [7].

This action is defined by combining the aforementioned actions $pickup$ and $putdown$, if certain pre-conditions are satisfied. The stacking action (not included in the program) then uses the *move* action to construct pyramids.

The key observation here is that at any given point several different instances of the rule tagged with *move_action* might be applicable and several different moves might be performed. The predicate `!`**`opposes`** stipulates that two different move-actions for different block are considered to be in conflict (because only one action at a time is allowed).

```
@mv_rule(Block,To) move(Block,From,To) :-
         (on(Block,From) ∧ larger(To,Block)) ⊗
         pickup(Block,From) ⊗ putdown(Block,To).
pickup(X,Y) :- (clear(X) ∧ on(X,Y)) ⊗
         delete(on(X,Y)) ⊗ insert(clear(Y)).
putdown(X,table) :- (clear(X) ∧ not on(X,Z))
         ⊗ insert(on(X,table)).
putdown(X,Y) :- (clear(X) ∧ clear(Y) ∧ not on(X,Z))
         ⊗ delete(clear(Y)) ⊗ insert(on(X,Y)).
!opposes(move(B1,F1,T1),move(B2,F2,T2)) :- B1 ≠ B2.
```

Note that the first rule is tagged with a term, $mv\_rule(Block)$ and, according to our conventions, such a rule is defeasible. Various heuristics can be used to improve construction of plans for building pyramid of blocks. In particular, we can use preferences among the rules to cut down on the number of plans that need to be looked at. For instance, the following rule says that move-actions that move bigger blocks are preferred to move-action that move smaller blocks—unless the blocks are moved down to the table surface.

```
!overrides(mv_rule(B2,To), mv_rule(B1,To)) :-
         larger(B2,B1) ∧ To ≠ table.
```

Consider the following configuration of blocks:

```
on(blk1,blk4). on(blk2,blk5).
on(blk3,table). on(blk4,table). on(blk5,table).
clear(blk1). clear(blk2). clear(blk3).
larger(blk2,blk1). larger(blk3,blk1). larger(blk3,blk2).
larger(blk4,blk1). larger(blk5,blk2). larger(blk2,blk4).
```

Although, both $blk1$ and $blk2$ can be moved on top of $blk3$, moving $blk2$ has higher priority because it is larger.

For moving blocks to the table surface, we use the opposite heuristic, one which prefers unstacking smaller blocks:

```
!overrides(mv_rule(B2,table), mv_rule(B1,table)) :- larger(B1,B2).
```

In the above example, this would make unstacking `blk1` and moving it to the table surface preferable to unstacking `blk2`, since the former is a smaller block. This would present an opportunity to then move `blk4` on top of `blk2` and subsequently put `blk1` on top of `blk4`.

We can now use the above heuristic preference rules in order to improve the performance of a pyramid building program such as this:

```
stack(0,Block).
stack(N,X) :- N>0 ⊗ move(Y,_,X) ⊗ stack(N-1,Y) ⊗ on(Y,X).
stack(N,X) :- (N>0 ∧ on(Y,X)) ⊗ unstack(Y) ⊗ stack(N,X).

unstack(X) :- on(Y,X) ⊗ unstack(Y) ⊗ unstack(X).
unstack(X) :- isclear(X) ∧ on(X,table).
unstack(X) :- (isclear(X) ∧ on(X,Y) ∧ Y≠ table) ⊗ move(X,_,table).
unstack(X) :- on(Y,X) ⊗ unstack(Y) ⊗ unstack(X).
```

Running this program on the interpreter described in [29] shows that the above preferences can significantly reduce the number of plans that need to be considered—sometimes to just one plan. □

3 Serial-Horn Transaction Logic

In this section we describe a subset of Transaction Logic called serial-Horn $\mathcal{TR}$. This subset has been shown to be sufficiently expressive for many applications, including planning, workflow management, and action languages [7,13,8,10,17,18,43,44].

The syntax of $\mathcal{TR}$ is derived from that of standard logic programming. The alphabet of the language $\mathcal{L}_{\mathcal{TR}}$ of $\mathcal{TR}$ contains an infinite number of constants, function symbols, predicate symbols, and variables. The *atomic formulas* have the form $p(t_1, ..., t_n)$, where p is a predicate symbol, and t_i are terms (variables, constants, function terms). However, unlike standard logic programming, predicate symbols are partitioned into *fluents* and *actions*. Fluents are predicates whose execution does not change the state of the database, while actions are predicates that can change the state of the database. Fluents are further partitioned into *base fluents* and *derived fluents*. Base fluents correspond to the classical base predicates in relational databases; they represent stored data and may be inserted or deleted. Derived fluents correspond to derived predicates, which represent database views. An atomic formula $p(t_1, ..., t_n)$ will be also called a *fluent* or an *action atomic formula* depending on whether p is a fluent or an action symbol. Furthermore, if p is a derived or base fluent symbol then $p(t_1, ..., t_n)$ is said to be a derived or base fluent atomic formula. An expression is *ground* if it does not contain any variables.

The symbol `neg` will be used to represent the explicit negation (also called "strong" negation) and `not` will be used for default negation, that is, negation as failure. A *fluent literal* is either an atomic fluent or has one of the following negated forms:

– `neg` α, `not` α, `not neg` α,

where α is a fluent atomic formula. An *action literal* is an action atomic formula or has the form `not` α, where α is a action atomic formula. Literals of the form `neg` α, where α is an action, are not allowed. Atoms of the form `neg not` $alpha$ are also not allowed.

A *database state* is a set of ground base fluents. All database states are assumed to be *consistent*, meaning that they cannot have both f and `neg` f, for any base fluent f.

Transaction Logic distinguishes a special sort of actions, called *elementary transitions* or *elementary updates*. Intuitively, an elementary transition is a "builtin" action that transforms a database from one state into another. All other actions are defined via rules using elementary transitions and fluents. In this paper, elementary transitions are deletions and insertions of base fluents. Formally, an *elementary state transition* is an action atomic formula of the form $insert(f)$ or $delete(f)$, where f is a ground base fluent or has the form $\texttt{neg}\, g$, where g is a ground base fluent. For any given database state $\mathbf{D}$,

– $insert(f)$ causes a transition from $\mathbf{D}$ to the state $\mathbf{D} \cup \{f\} \setminus \{\texttt{neg}\, f\}$; and
– $delete(f)$ causes a transition from $\mathbf{D}$ to $\mathbf{D} \setminus \{f\} \cup \{\texttt{neg}\, f\}$.

In addition to the classical connectives $\wedge$, $\vee$, and quantifiers, $\mathcal{TR}$ has new logical connectives:

– $\otimes$ - the sequential conjunction
– $\Diamond$ - the modal operator of hypothetical execution

The formula $\phi \otimes \psi$ represents an action composed of an execution of ϕ followed by an execution of ψ, while the formula $\Diamond\phi$ is an action of *hypothetically* testing whether ϕ can be executed at the current state, but no actual state changes takes place. In procedural terms, executing $delete(on(blk1, table)) \otimes insert(on(blk1, blk2))$ means "first delete $on(blk1, table)$ from the database, and then insert $on(blk1, blk2)$." The current database state changes as a result. In contrast, $\Diamond move(blk1)$ is only a "hypothetical" execution: it checks whether $move(blk1)$ can be executed in the current state, but regardless of whether it can or not the current state does not change.

The semantics of Transaction Logic is such that when f_1 and f_2 are fluents, $f_1 \otimes f_2$ is equivalent to $f_1 \wedge f_2$ and $\Diamond f$ to f. Therefore, when no actions are present, $\mathcal{TR}$ reduces to classical logic. This also explains our use of $\wedge$ in Example 2 where it could have been replaced with $\otimes$ without changing the meaning (but, the uses of $\otimes$ in the Examples 1 and 2 *cannot* be replaced with $\wedge$ without changing the meaning).

Definition 1 (Serial goal). *Serial goals are defined recursively as follows:*

– *If P is a fluent or an action literal then P is a serial goal. Note that fluent literals can contain both* `not` *and* `neg`*, and action literals can contain* `not`*.*
– *If P is a serial goal, then so are* `not` *P and $\Diamond P$.*
– *If P_1 and P_2 are serial goals then so are $P_1 \otimes P_2$ and $P_1 \wedge P_2$.* □

Definition 2 (Serial rules). *A **serial rule** is an expression of the form:*

$$H \ :- \ B. \tag{1}$$

where H is a `not`*-free literal and B is a serial goal. We will be dealing with two classes of serial rules:*

– ***Fluent rules:** In this case, H is a derived fluent of the form f or a fluent literal of the form* `neg` *f and $B = f_1 \otimes ... \otimes f_n$, where each f_i is a fluent literal (and thus $\otimes$ could be replaced with $\wedge$).*
– ***Action rules:** In this case, H must be an atomic action formula, while the body of the rule, B, is a* serial goal.

*A **transaction base** is a finite set of serial rules.* □

An existential serial goal is a statement of the form $\exists \bar{X} \psi$ where ψ is a serial goal and $\bar{X}$ is a list of all free variables in ψ. For instance, $\exists X move(X, blk2)$ is an existential serial goal. Informally, the truth value of an existential goal in $\mathcal{TR}$ is determined over sequences of states, called *execution paths*, which makes it possible to view truth assignments in $\mathcal{TR}$'s models as executions. If an existential serial goal, ψ, defined by a program $\mathbf{P}$, evaluates to true over a sequence of states $\mathbf{D_0}, \ldots \mathbf{D_n}$, we say that it can *execute* at state $\mathbf{D_0}$ by passing through the states $\mathbf{D_1}$, ..., $\mathbf{D_{n-1}}$, and ending in the final state $\mathbf{D_n}$. Formally, this is captured by the notion of *executional entailment*, which is written as follows:

$$\mathbf{P}, \mathbf{D_0}, \ldots \mathbf{D_n} \models \psi$$

Further details on $\mathcal{TR}$ can be found in [7,6].

4 Defeasibility in Transaction Logic

In this section we define a form of defeasible Transaction Logic, which we call *Transaction logic with defaults and argumentation theories* ($\mathcal{TR}^{DA}$). The development was inspired by our earlier work on logic programming with argumentation theories, which did not support actions [53]. Language-wise, the only difference between $\mathcal{TR}^{DA}$ and serial $\mathcal{TR}$ is that the rules in $\mathcal{TR}^{DA}$ are tagged.

Definition 3 (Tagged rules). *A **tagged rule** in the language $\mathcal{TR}^{DA}$ is an expression of the form*

$$@r\ H\ :-\ B. \qquad (2)$$

*where the **tag** r of a rule is a term. The head literal, H, and the body of the rule, B, have the same restrictions as in Definition 2.*
*A serial $\mathcal{TR}^{DA}$ **transaction base** $\mathbf{P}$ is a set of rules, which can be **strict** or **defeasible**.*
$\square$

Definition 4 (Transaction formula). *A **transaction formula** in the language $\mathcal{TR}^{DA}$ is a literal, a serial goal, a tagged or an untagged serial rule.* $\square$

We note that the rule tag in the above definition is not a rule identifier: several rules can have the same tag, which can be useful for specifying priorities among sets of rules.

Strict rules are used as *definite* statements about the world. In contrast, defeasible rules represent *defaults* whose instances can be "defeated" by other rules. Inferences produced by the defeated rules are "overridden." We assume that the distinction between strict and defeasible rules is specified in some way: either syntactically or by means of a predicate (in this paper, we consider strict rules to be non-tagged rules, as in Definition 2).

Definition 5 (Rule handle). *Given a rule of the form (2), the term*

$$\mathtt{handle}(r, H) \qquad (3)$$

*is called the **handle** of that rule.* $\square$

$\mathcal{TR}^{DA}$ transaction bases are used in conjunction with *argumentation theories*, which are sets of rules that define conditions under which some rule instances in the transaction base may be defeated by other rules. The argumentation theory and the transaction base share the same set of fluent and action symbols.

Definition 6 (Argumentation theory). *An **argumentation theory**, AT, is a set of strict serial rules. We also assume that the language of $\mathcal{TR}^{DA}$ includes a unary predicate, `$defeated`$_{AT}$, which may appear in the heads of some rules in AT but not in the transaction base. A $\mathcal{TR}^{DA}$ $\mathbf{P}$ is said to be **compatible** with AT if `$defeated`$_{AT}$ does not appear in any of the rule heads in $\mathbf{P}$,* □

The rules *AT* are used to specify how the rules in $\mathbf{P}$ get defeated. This is usually done using special predicates defined in $\mathcal{TR}^{DA}$, such as `!`**`opposes`** and `!`**`overrides`** used in our example. For the purpose of defining the semantics, we assume that the argumentation theories *AT* are grounded. This grounding can be done by appropriately instantiating the variables and meta-predicates in *AT*.

Although Definition 6 imposes almost no restrictions on the predicate `$defeated`$_{AT}$, practical argumentation theories are likely to require that it is executed hypothetically, i.e., that its execution does not change the current state. This is certainly true of the argumentation theories used in this paper.

Definition 7 (Herbrand universe and base). *The **Herbrand universe** of $\mathcal{TR}^{DA}$, denoted $\mathcal{U}$, is the set of all ground terms built using the constants and function symbols of the language of $\mathcal{TR}^{DA}$.*

*The **Herbrand base**, denoted $\mathcal{B}$, is the set of all ground `not`-free literals that can be constructed using the language of $\mathcal{TR}^{DA}$.*

The key concept underlying the semantics of $\mathcal{TR}$ and $\mathcal{TR}^{DA}$ is the concept of *execution paths*, which are sequences of database states. The truth assignment in $\mathcal{TR}$ is done using *path structures*, which are mappings from paths of states.

Definition 8 (Path and Split). *A **path** of length k, or a* k-path, *is a finite sequence of states, $\pi = \langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $k \geq 1$. A **split** of π is any pair of subpaths, π_1 and π_2, such that $\pi_1 = \langle \mathbf{D}_1 \ ... \ \mathbf{D}_i \rangle$ and $\pi_2 = \langle \mathbf{D}_i \ ... \ \mathbf{D}_k \rangle$ for some **I** ($1 \leq i \leq k$). If π has a split π_1, π_2 then we write $\pi = \pi_1 \circ \pi_2$.*

We extend the well-founded semantics for logic programing [52] to $\mathcal{TR}^{DA}$ using a definition in the style of [42]. In the following, we use the usual three truth values **t**, **f**, and **u**, which stand for *true*, *false*, and *undefined*, respectively. We also assume the existence of the following total order on these values: **f** $<$ **u** $<$ **t**.

Definition 9 (Partial Herbrand interpretation). *A **partial Herbrand interpretation** is a mapping $\mathcal{H}$ that assigns* **f**, **u**, *or* **t** *to every formula L in $\mathcal{B}$.*
*A partial Herbrand interpretation $\mathcal{H}$ is **consistent relative to an atomic formula** L if it is not the case that $\mathcal{H}(L) = \mathcal{H}(\texttt{neg}\, L) =$* **t**. *$\mathcal{H}$ is* **consistent** *if it is consistent relative to every formula. $\mathcal{H}$ is* **total** *if, for every ground `not`-free formula L (other than* **u***), either $\mathcal{H}(L) =$* **t** *and $\mathcal{H}(\texttt{neg}\, L) =$* **f** *or $\mathcal{H}(L) =$* **f** *and $\mathcal{H}(\texttt{neg}\, L) =$* **t**. □

Partial Herbrand interpretations are used to define *path structures*, which tell which ground atoms (fluents or actions) are true on what paths. Path structures play the same role in $\mathcal{TR}^{DA}$ as that played by the classical semantic structures in classical logic. The semantic structures of $\mathcal{TR}^{DA}$ are *mappings* from paths to partial Herbrand interpretations.

Definition 10 (Herbrand Path Structure). *A **partial Herbrand Path Structure** is a mapping **I** that assigns a partial Herbrand interpretation to every **path** subject to the following restrictions:*

1. *For every ground base fluent literal d and every database state $\mathbf{D}$:*
   $\boldsymbol{I}(\langle \mathbf{D} \rangle)(d) = \mathbf{t}$*, if* $d \in \mathbf{D}$*;*
   $\boldsymbol{I}(\langle \mathbf{D} \rangle)(d) = \mathbf{f}$*, if* $d \notin \mathbf{D}$*;*
   $\boldsymbol{I}(\langle \mathbf{D} \rangle)(d) = \mathbf{u}$*, otherwise*
2. $\boldsymbol{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(insert(p)) = \mathbf{t}$ *if* $\mathbf{D}_2 = \mathbf{D}_1 \cup \{p\} \setminus \{\mathtt{neg}\, p\}$ *and p is a ground fluent-literal;*
   $\boldsymbol{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(insert(p)) = \mathbf{f}$*, otherwise.*
3. $\boldsymbol{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(delete(p)) = \mathbf{t}$ *if* $\mathbf{D}_2 = \mathbf{D}_1 \setminus \{p\} \cup \{\mathtt{neg}\, p\}$ *and $\mathbf{P}$ is a ground fluent-literal;*
   $\boldsymbol{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(delete(p)) = \mathbf{f}$*, otherwise.* $\Box$

Without loss of generality, while defining the semantics of $\mathcal{TR}^{DA}$ we will consider ground rules only. This is possible because all variables in a rule are considered to be universally quantified, so such rules can be replaced with the set of all of their ground instantiations.

We assume that the language includes the special propositional constants: $\mathbf{u}^{\pi}$ and $\mathbf{t}^{\pi}$, for each path π. Informally, $\mathbf{t}^{\pi}$ is a propositional transaction that is true precisely over the path π and false on all other paths; $\mathbf{u}^{\pi}$ is a propositional transaction that has the value $\mathbf{u}$ over π and is false on all other paths.

Definition 11 (Truth valuation in path structures). *Let **I** be a path structure, π a path, L a ground* `not`*-free literal, and let F, G be ground serial goals. We define **truth valuations** with respect to the path structure **I** as follows:*

- *If $\mathbf{P}$ is a* `not`*-free literal then $\boldsymbol{I}(\pi)(p)$ is already defined because $\boldsymbol{I}(\pi)$ is a Herbrand interpretation, by definition of **I**.*
- *If ϕ and ψ are serial goals and $\pi = \pi_1 \circ \pi_2$ then* $\boldsymbol{I}(\pi)(\phi \otimes \psi) = \mathtt{min}(\boldsymbol{I}(\pi_1)(p), \boldsymbol{I}(\pi_2)(q))$.
- *If ϕ and ψ are serial goals then* $\boldsymbol{I}(\pi)(\phi \wedge \psi) = \mathtt{min}(\boldsymbol{I}(\pi)(p), \boldsymbol{I}(\pi)(q))$.
- *If ϕ is a serial goal then* $\boldsymbol{I}(\pi)(\mathtt{not}\ \phi) = \sim \boldsymbol{I}(\pi)(\phi)$*, where* $\sim \mathbf{t} = \mathbf{f}$*,* $\sim \mathbf{f} = \mathbf{t}$*, and* $\sim \mathbf{u} = \mathbf{u}$.
- *If ϕ is a serial goal and $\pi = \langle \mathbf{D} \rangle$, where $\mathbf{D}$ is a database state, then*
  $\boldsymbol{I}(\pi)(\Diamond\phi) = \mathtt{max}\{\boldsymbol{I}(\pi')(\phi) \mid \pi'$ is a path that starts at $\mathbf{D}\}$
  $\boldsymbol{I}(\pi)(\Diamond\phi) = \mathbf{f}$*, otherwise.*
- *For a strict serial rule* F `:-` G*,*
  $\boldsymbol{I}(\pi)(F$ `:-` $G) = \mathbf{t}$ *iff* $\boldsymbol{I}(\pi)(F) \geq \boldsymbol{I}(\pi)(G)$.
- *For a defeasible rule* $@r\ F$ `:-` G*,*
  $\boldsymbol{I}(\pi)(@\mathtt{r}\ \mathtt{F}$ `:-` $\mathtt{G}) = \mathbf{t}$ *iff*
  $$\boldsymbol{I}(\pi)(F) \geq \mathtt{min}\,(\boldsymbol{I}(\pi)(G),\ \boldsymbol{I}(\pi)(\mathtt{not}\ \Diamond\ \mathtt{\$defeated}(\mathtt{handle}(r, F)))).$$

– *For any path π:*
  $\boldsymbol{I}(\pi)(\mathbf{t}^{\pi}) = \mathbf{t}$ *and* $\boldsymbol{I}(\pi')(\mathbf{t}^{\pi}) = \mathbf{f}$, *if* $\pi' \neq \pi$;
  $\boldsymbol{I}(\pi)(\mathbf{u}^{\pi}) = \mathbf{u}$ *and* $\boldsymbol{I}(\pi')(\mathbf{u}^{\pi}) = \mathbf{f}$, *if* $\pi' \neq \pi$.

We will write $\boldsymbol{I}, \pi \models \phi$ *and say that* ϕ *is* **satisfied** *on path* π *in the path structure* $\boldsymbol{I}$ *if* $\boldsymbol{I}(\pi)(\phi)$=**t**. *A path structure* $\boldsymbol{I}$ *is said to be* ***total*** *if, for every path* π *and every serial goal* ϕ, $\boldsymbol{I}(\pi)(L)$ *is either* **t** *or* **f**. □

Definition 12 (Model of a transactional formula). *A path structure,* $\boldsymbol{I}$*, is a* model *of a transaction formula* ϕ *if* $\boldsymbol{I}, \pi \models \phi$ *for every path* π*. In this case, we write* $\boldsymbol{I} \models \phi$ *and say that* $\boldsymbol{I}$ *is a* ***model*** *of* ϕ *or that* ϕ *is* ***satisfied*** *in* $\boldsymbol{I}$*. A path structure* $\boldsymbol{I}$ *is a model of a set of formulas if it is a model of every formula in the set.* □

Definition 13 (Model of $\mathcal{TR}^{DA}$). *A path structure* $\boldsymbol{I}$ *is a model of a* $\mathcal{TR}^{DA}$ *transaction base* $\mathbf{P}$ *if all rules in* $\mathbf{P}$ *are satisfied in* $\boldsymbol{I}$ *(i.e.,* $\boldsymbol{I} \models R$ *for every* $R \in \mathbf{P}$*). Given a* $\mathcal{TR}^{DA}$ *transaction base* $\mathbf{P}$*, an argumentation theory AT, and a path structure* $\boldsymbol{M}$*, we say that* $\boldsymbol{M}$ *is a model of* $\mathbf{P}$ *with respect to the argumentation theory AT, written as* $\boldsymbol{M} \models (\mathbf{P}, AT)$*, if* $\boldsymbol{M} \models \mathbf{P}$ *and* $\boldsymbol{M} \models AT$*.* □

Like classical logic programs, the Herbrand semantics of serial-Horn $\mathcal{TR}$ can be formulated as a fixpoint theory [13]. In classical logic programming, given two Herbrand partial interpretations σ_1 and σ_2, we write $\sigma_1 \preceq \sigma_2$ if all `not`-free literals that are true in σ_1 are also true in σ_2 and all `not`-literals that are true in σ_2 are also true in σ_1. Similarly, for partial interpretations, $\sigma_1 \leq \sigma_2$ if all `not`-free literals that are true in σ_1 are also true in σ_2 and all `not`-literals that are true in σ_1 are also true in σ_2.

Definition 14 (Order on Path Structures). *If* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *are partial Herbrand path structures, then* $\mathbf{M}_1 \preceq \mathbf{M}_2$ *if* $\mathbf{M}_1(\pi) \preceq \mathbf{M}_2(\pi)$ *for every path,* π*. Similarly, we write* $\mathbf{M}_1 \leq \mathbf{M}_2$ *if* $\mathbf{M}_1(\pi) \leq \mathbf{M}_2(\pi)$ *for every path,* π*.*
A model $\mathbf{M}$ *of* $\mathbf{P}$ *is* **minimal** *with respect to* $\preceq$ *iff for any other model,* $\mathbf{N}$*, of* $\mathbf{P}$*, we have that* $\mathbf{N} \preceq \mathbf{M}$ *implies* $\mathbf{N} = \mathbf{M}$*. The* **least** *model of* $\mathbf{P}$ *is a minimal model that is unique (if it exists).*

It is well-known that in ordinary logic programming any set of Horn rules always has a least model. In [7], it is shown that every positive serial-Horn $\mathcal{TR}$ program has a unique least total model. Theorem 1, below, shows that this property is preserved by serial `not`-free $\mathcal{TR}$ programs, but in this case the model might be a partial path structure. Serial `not`-free programs are more general than the positive $\mathcal{TR}$ programs because the undefined propositional symbol $\mathbf{u}^{\pi}$ for some path π may occur in the bodies of the program rules.

Theorem 1 (Unique Least Partial Model for serial `not`-free $\mathcal{TR}$ programs). *If* $\mathbf{P}$ *is a* `not`*-free* $\mathcal{TR}^{DA}$ *program, then* $\mathbf{P}$ *has a least Herbrand model, denoted* $LPM(\mathbf{P})$*.*

Proof. Let $\mathbf{P}^{+}$ denote the positive program obtained from $\mathbf{P}$ by replacing all body literals of the form $\mathbf{u}^{\pi}$, where π is a path, with $\mathbf{t}^{\pi}$. (We will call such literals **u**-literals and **t**-literals, respectively.) Similarly, let denote $\mathbf{P}^{-}$ the positive program obtained by deleting the rules whose body includes **u**-literals. (This is equivalent to replacing all **u**-literals with a propositional constant **f** that is false on any paths). Note that both $\mathbf{P}^+$ and $\mathbf{P}^-$ have unique minimal Herbrand models, since they do not have the special literals $\mathbf{u}^\pi$ and thus are simply serial-Horn clauses; these minimal models are two-valued, as shown in [7].

Let $\mathbf{M}^+$ be the least model of $\mathbf{P}^+$ and $\mathbf{M}^-$ be the least model of $\mathbf{P}^-$. As noted above, both of these models are two-valued. Clearly, $\mathbf{P}^-$ is a subprogram of $\mathbf{P}^+$, so $\mathbf{M}^+$ is also a model of $\mathbf{P}^-$. Since $\mathbf{M}^-$ is the least model of $\mathbf{P}^-$, it follows that $\mathbf{M}^- \preceq \mathbf{M}^+$. Thus, for any path π and any `not`-free literal L

$$\mathbf{M}^-(\pi)(L) \leq \mathbf{M}^+(\pi)(L) \quad \text{and} \quad \mathbf{M}^+(\pi)(\texttt{not}\, L) \leq \mathbf{M}^-(\pi)(\texttt{not}\, L) \tag{4}$$

This means that all `not`-free literals that are true in $\mathbf{M}^-(\pi)$ are also true in $\mathbf{M}^+(\pi)$ and all `not`-literals that are true in $\mathbf{M}^+(\pi)$ are also true in $\mathbf{M}^-(\pi)$.

We construct the least model $\mathbf{M}$ of $\mathbf{P}$ as a path structure such that, for any path π, $\mathbf{M}(\pi)$ is the classical Herbrand structure where

$$\begin{array}{lll}
- \ \mathbf{M}(\pi)(L) = \mathbf{t} & \text{iff} & \mathbf{M}^-(\pi)(L) = \mathbf{t} \\
\ \mathbf{M}(\pi)(\texttt{not}\, L) = \mathbf{f} & \text{iff} & \mathbf{M}^-(\pi)(\texttt{not}\, L) = \mathbf{f} \\
- \ \mathbf{M}(\pi)(L) = \mathbf{f} & \text{iff} & \mathbf{M}^+(\pi)(L) = \mathbf{f} \\
\ \mathbf{M}(\pi)(\texttt{not}\, L) = \mathbf{t} & \text{iff} & \mathbf{M}^+(\pi)(\texttt{not}\, L) = \mathbf{t} \\
- \ \text{otherwise,} \quad \mathbf{M}(\pi)(L) = \mathbf{M}(\pi)(\texttt{not}\, L) = \mathbf{u} & &
\end{array} \tag{5}$$

for any ground `not`-free literal L. We will now prove that $\mathbf{M}$ is $LPM(\mathbf{P})$, the unique minimal model of $\mathbf{P}$.

By (4), $\mathbf{M}$ is well-defined, since it is not possible that $\mathbf{M}^-(\pi)(L) = \mathbf{t}$ and $\mathbf{M}^+(\pi)(L) = \mathbf{f}$ or that $\mathbf{M}^-(\pi)(\texttt{not}\, L) = \mathbf{f}$ and $\mathbf{M}^+(\pi)(\texttt{not}\, L) = \mathbf{t}$.

Next, we show that $\mathbf{M}$ is a partial model of $\mathbf{P}$. Suppose C is a rule in $\mathbf{P}$ of the form $H \ :- B_1 \otimes ... \otimes B_n$, such that none of the B_is is a **u**-literal. By definition, C belongs *both* to $\mathbf{P}^-$ and $\mathbf{P}^+$. If, for some path π, $\mathbf{M}(\pi)(B_1 \otimes ... \otimes B_n) = \mathbf{t}$, then $\mathbf{M}^-(\pi)(B_1 \otimes ... \otimes B_n) = \mathbf{t}$, by the construction of $\mathbf{M}$ in (5). Since $\mathbf{M}^-$ is a model of C, the head H of C must be true in $\mathbf{M}^-(\pi)$ hence also in $\mathbf{M}(\pi)$. Thus, $\mathbf{M}(\pi)$ makes C true. If $\mathbf{M}(\pi)(B_1 \otimes ... \otimes B_n) = \mathbf{f}$ then $\mathbf{M}(\pi)$ satisfies C trivially. If $\mathbf{M}(\pi)(B_1 \otimes ... \otimes B_n) = \mathbf{u}$, it means that, for some split $\pi = \pi_1 \circ ... \circ \pi_n$, $\mathbf{M}(\pi_i)(B_i)$ is either **u** or **t**. By (5), this implies $\mathbf{M}^+(\pi_i)(L) = \mathbf{t}$, and since $\mathbf{M}^+$ is also a model of C it follows that H must be true in $\mathbf{M}^+(\pi)$. The definition of $\mathbf{M}$ then implies that H must have the truth value **u** or **t** in $\mathbf{M}(\pi)$, so $\mathbf{M}(\pi)$ satisfies C once again.

Next, suppose that C is a clause $H \ :- B_1 \otimes ... \otimes B_n$ in $\mathbf{P} \setminus \mathbf{P}^-$, and suppose π is a path with a split $\pi = \pi_1 \circ ... \circ \pi_n$ such that none of the $\mathbf{M}(\pi_i)(B_i) = \mathbf{f}$. Note that since C is not in $\mathbf{P}^-$, at least one of the B_is must be $\mathbf{u}^{\pi_i}$ for some subpath π_i. So, it must be the case that $\mathbf{M}(B_1 \otimes ... \otimes B_n) = \mathbf{u}$ (it cannot be **t** because of $\mathbf{u}^{\pi_i}$ and it cannot be **f** because of the assumption that none of the $\mathbf{M}(\pi_i)(B_i)$s is **f**). This implies (again by (5)) that none of the $\mathbf{M}^+(\pi_i)(B_i)$s equals **f**. Therefore $\mathbf{M}(\pi_i)(B_i) = \mathbf{t}$ for all body literals in the corresponding clause C^+ in $\mathbf{P}^+$ (one that is obtained from C by changing each $\mathbf{u}^{\pi_i}$ to $\mathbf{t}^{\pi_i}$). Therefore, H (which is the head of both C and C^+) must be true in $\mathbf{M}^+(\pi)$. So $\mathbf{M}(\pi)(H)$ is either **t** or **u**. Thus, $\mathbf{M}(\pi)$ models every rule in $\mathbf{P} \setminus \mathbf{P}^-$ either and, therefore, $\mathbf{M}$ is a model of $\mathbf{P}$.

To prove minimality and uniqueness of $\mathbf{M}$, let $\mathbf{N}$ be a model of $\mathbf{P}$. We will show that $\mathbf{M} \preceq \mathbf{N}$, which would imply that $\mathbf{M}$ is the least model. We need to establish the following properties:

Property 1: $\mathbf{M}(\pi)(L) \leq \mathbf{N}(\pi)(L)$
Property 2: $\mathbf{N}(\pi)(\mathtt{not}\,L) \leq \mathbf{M}(\pi)(\mathtt{not}\,L)$

The proof of these relies on the following claims, which will be proved at the end:

Claim 1: If $\mathbf{M}^{-}(\pi)(L) = \mathbf{t}$ then $\mathbf{N}(\pi)(L) = \mathbf{t}$.
Claim 2: If $\mathbf{M}^{-}(\pi)(\mathtt{not}\,L) = \mathbf{f}$ then $\mathbf{N}(\pi)(\mathtt{not}\,L) = \mathbf{f}$.
Claim 3: If $\mathbf{M}^{+}(\pi)(L) = \mathbf{t}$ then $\mathbf{N}(\pi)(L) \geq \mathbf{u}$.
Claim 4: If $\mathbf{M}^{+}(\pi)(\mathtt{not}\,L) = \mathbf{f}$ then $\mathbf{N}(\pi)(\mathtt{not}\,L) \leq \mathbf{u}$.

To establish Property 1, suppose that $\mathbf{M}(\pi)(L) = \mathbf{t}$. By (5), this means that $\mathbf{M}^{-}(\pi)(L) = \mathbf{t}$ and, by Claim 1, $\mathbf{N}(\pi)(L) = \mathbf{t}$. If $\mathbf{M}(\pi)(L) = \mathbf{u}$ then, by (5), this implies $\mathbf{M}^{+}(\pi)(L) = \mathbf{t}$ and, by Claim 3, $\mathbf{N}(\pi)(L) \geq \mathbf{u} = \mathbf{M}(\pi)(L)$. This proved Property 1.

For Property 2, suppose $\mathbf{M}(\pi)(\mathtt{not}\,L) = \mathbf{u}$. By the definition of $\mathbf{M}$, this implies $\mathbf{M}^{+}(\pi)(\mathtt{not}\,L) = \mathbf{f}$ and, by Claim 4, $\mathbf{N}(\pi)(\mathtt{not}\,L) \leq \mathbf{u} = \mathbf{M}(\pi)(\mathtt{not}\,L)$. Similarly, if $\mathbf{M}(\pi)(\mathtt{not}\,L) = \mathbf{f}$, then $\mathbf{M}^{-}(\pi)(\mathtt{not}\,L) = \mathbf{f}$ and, by Claim 2, $\mathbf{N}(\pi)(\mathtt{not}\,L) = \mathbf{f}$.

It remains to prove claims 1–4. Claims 1 and 2 follow directly from the fact that $\mathbf{N}$ is be a model of $\mathbf{P}^{-}$ for which $\mathbf{M}^{-}$ is the least model, so $\mathbf{M}^{-} \preceq \mathbf{N}$. Claim 3 can be easily proved by induction on the number of inference rules that need to be used in order to prove that L is true on π with respect to the program $\mathbf{P}^{+}$. Claim 4 follows from Claim 3: If $\mathbf{M}^{+}(\pi)(\mathtt{not}\,L) = \mathbf{f}$ then $\mathbf{M}^{+}(\pi)(L) = \mathbf{t}$. By Claim 3, $\mathbf{N}(\pi)(L) \geq \mathbf{u}$, which implies that $\mathbf{N}(\pi)(\mathtt{not}\,L) \leq \mathbf{u}$. □

Before we show an example of a serial `not` -free $\mathcal{TR}$ program containing **u**-literals, we add to our language a new predefined propositional constant called `state`. The propositional constant `state` is true only on paths of length 1, that is, for any path structure $\mathbf{M}$ and path π, it is the case that $\mathbf{M}(\pi)(\mathtt{state}) = \mathbf{t}$ iff π is a path of length 1, that is, a database state, and $\mathbf{M}(\pi)(\mathtt{state}) = \mathbf{f}$ if the path π's length is different than 1.

Example 3. Let the $\mathcal{TR}$ program $\mathbf{P}$ be:

$$\begin{array}{l} a\ :-\ \mathtt{state}. \\ b\ :-\ a \otimes \mathbf{u}^{\langle \mathbf{D}_\emptyset \rangle}. \\ c\ :-\ c \otimes \mathbf{u}^{\langle \mathbf{D}_\emptyset \rangle}. \end{array}$$

where a, b, and c are action symbols and $\mathbf{D}_\emptyset$ is the empty database state. The least partial model of $\mathbf{P}$ is a path structure that maps any 1-path to a classical Herbrand partial model where a is true, c is false, and b is undefined. All other paths are mapped to the classical Herbrand partial model where all formulas are mapped to **u**. Note that b is *not* false in $LPM(\mathbf{P})$ because the truth value of the sequential conjunction of premises in the second rule is **u**, so the truth value of b must be at least **u**. □

For `not`-free $\mathcal{TR}$ programs, the least partial model $LPM(P)$ can be obtained as the least fixed point of the immediate consequence operator, which is applied to all paths. However, we will not pursue this line here.

Next we define well-founded models for $\mathcal{TR}^{DA}$ by adapting the definition from [42]. First, we define the quotient operator, which takes a $\mathcal{TR}^{DA}$ program $\mathbf{P}$ and a path structure $\boldsymbol{I}$ and yields a serial-Horn $\mathcal{TR}$ program $\dfrac{\mathbf{P}}{\boldsymbol{I}}$. Despite what one might have been expecting, this adaptation is rather subtle.

Definition 15 (Quotient). *Let* $\mathbf{P}$ *be a set of* $\mathcal{TR}^{DA}$ *rules,* $\boldsymbol{I}$ *a path structure for* $\mathbf{P}$*, and* π *a path. The* $\mathcal{TR}^{DA}$ ***quotient of*** $\mathbf{P}$ ***by*** $\boldsymbol{I}$*, written as* $\dfrac{\mathbf{P}}{\boldsymbol{I}}$*, is defined through the following sequence of steps:*

1. *First, each occurrence of every* `not`*-literal of the form* `not` L *in* $\mathbf{P}$ *is replaced by* $\mathbf{t}^{\pi}$*, for every path* π *such that* $\boldsymbol{I}(\pi)(\texttt{not}\ L) = \mathbf{t}$*, and with* $\mathbf{u}^{\pi}$ *for every path* π *such that* $\boldsymbol{I}(\pi)(\texttt{not}\ L) = \mathbf{u}$*.*
   Note: each occurrence of `not` L *is replaced with different* $\mathbf{t}^{\pi}$ *and* $\mathbf{u}^{\pi}$ *for different* π*'s, so every rule in* $\mathbf{P}$ *may be replaced with several (possibly infinite number of)* `not`*-free rules. All combinations of replacements for the* `not`*-literals in the body of the rules have to be used. Only the* π*'s where* $\boldsymbol{I}(\pi)(\texttt{not}\ L) = \mathbf{f}$ *are not used, which effectively means that the rule instances that correspond to those cases are removed from consideration.*
2. *For each labeled rule of the form* $@r\ L$ **:-** $Body$ *obtained in the previous step, replace it with the rules of the form:*

$$L \texttt{ :- } \mathbf{t}^{\langle \mathbf{D}_t \rangle} \otimes Body$$
$$L \texttt{ :- } \mathbf{u}^{\langle \mathbf{D}_u \rangle} \otimes Body$$

   for each database state $\mathbf{D}_t$ *such that* $\boldsymbol{I}(\langle \mathbf{D}_t \rangle)(\texttt{not}\ (\triangledown\ \texttt{\$defeated}(\texttt{handle}(r, L)))) = \mathbf{t}$
   and each database state $\mathbf{D}_u$ *such that* $\boldsymbol{I}(\langle \mathbf{D}_u \rangle)(\texttt{not}\ (\triangledown\ \texttt{\$defeated}(\texttt{handle}(r, L)))) = \mathbf{u}$
3. *Remove tags from the remaining rules.*

The resulting set of rules is the quotient $\dfrac{\mathbf{P}}{\boldsymbol{I}}$*.* □

Definition 16 (Well-founded model). *The* ***well-founded model*** *of a* $\mathcal{TR}^{DA}$ *transaction base* $\mathbf{P}$ *with respect to the argumentation theory AT, written as* $WFM(\mathbf{P}, AT)$*, is defined as the limit of the following transfinite induction:*
Start with $\boldsymbol{I}_0$ *— the path structure that maps each path* π *to the* empty *Herbrand interpretation, i.e., one in which all propositions are undefined. Suppose* $\boldsymbol{I}_n$ *has already been defined for every* $n < m$*, where* m *is an ordinal. Then:*

- *If* m *is a non-limit ordinal:* $\boldsymbol{I}_m = LPM\left(\dfrac{\mathbf{P} \cup AT}{\boldsymbol{I}_{m-1}}\right)$
- *If* m *is a limit ordinal:*

$$\boldsymbol{I}_m(\pi)(L) = \begin{cases} \mathbf{t} & \text{if } \boldsymbol{I}_n(\pi)(L) = \mathbf{t} \text{ for some } n < m \\ \mathbf{f} & \text{if } \boldsymbol{I}_n(\pi)(L) = \mathbf{f} \text{ for some } n < m \\ \mathbf{u} & \text{otherwise (if } \boldsymbol{I}_n(\pi)(L) = \mathbf{u} \text{ for all } n < m) \end{cases}$$

  for every path π *and every literal* L*.* □

The following theorem states that the above recursively defined sequence of path structures is always monotonically increasing the knowledge by adding new information to each path structure $\boldsymbol{I}_n$ (that is, mappings for literals on paths to undefined are being resolved to true or false), and that stating from $\boldsymbol{I}_0$ (the path structure that maps each path π to the *empty* classical Herbrand interpretation which maps all propositions to undefined), one always reaches a fixed point $\boldsymbol{I}_m$ with the property that $\boldsymbol{I}_{m+1} = \boldsymbol{I}_m$. In other words, it is not possible that at some stage, n_1, $\boldsymbol{I}_{n_1}(\pi)(L) = \mathbf{t}$, while at another stage, n_2, $\boldsymbol{I}_{n_2}(\pi)(L) = \mathbf{f}$. The theorem also states that $\mathcal{TR}^{DA}$ programs under the well-founded semantics reduce to ordinary $\mathcal{TR}$ programs under the same well-founded semantics. This means that $\mathcal{TR}^{DA}$ can be implemented using an implementation of ordinary transaction logic that supports the well-founded semantics.

Theorem 2 (Fixpoint and Reduction). *The transfinite sequence of Herbrand path structures $\langle \boldsymbol{I}_0, \boldsymbol{I}_1, \ldots \rangle$ in Definition 16 is always increasing (with respect to the order relation $\leq$) and has a unique limit. The limit is reached for some (possibly transfinite) ordinal, α, such that $\mathbf{I}_\alpha = \mathbf{I}_{\alpha+1}$. This limit is the least model of $(\mathbf{P}, AT)$ and therefore $WFM(\mathbf{P}, AT)$ is well-defined.*
$WFM(\mathbf{P}, AT)$ coincides with the well-founded model of the (ordinary) $\mathcal{TR}$ program $\mathbf{P}' \cup AT$, where $\mathbf{P}'$ is obtained from $\mathbf{P}$ by changing every defeasible rule $(@\mathtt{r}\ \mathtt{L} :\!-\ \mathtt{Body}) \in \mathbf{P}$ *to the strict rule* `L :- not (`$\lozenge$ `$defeated(handle(r, L)) )` $\otimes$ `Body` *and removing all the remaining tags.*

Executional entailment in $\mathcal{TR}$ defines the notion of *execution* of $\mathcal{TR}$ programs and provides the theoretical foundation for an SLD-style proof procedure for the serial $\mathcal{TR}$. We now extend this notion to $\mathcal{TR}^{DA}$.

Definition 17. (Executional Entailment) *Let $\mathbf{P}$ be a serial $\mathcal{TR}^{DA}$ transaction base, AT be argumentation theory, ϕ be a transaction formula, and let $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n$ be a sequence of database states. Then, the statement:*

$$(\mathbf{P}, AT), \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \models \phi \tag{6}$$

is true if $\mathbf{M}, \langle \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \rangle \models \phi$ for every model $\mathbf{M}$ of $(\mathbf{P}, AT)$.

Executional entailment enables procedural interpretation of $\mathcal{TR}^{DA}$ in terms of program execution. Normally, one knows only the initial database state $\mathbf{D}_0$. Proving an executional entailment (6) can then be interpreted as finding an execution path for ϕ that starts in $\mathbf{D}_0$.

5 The $GCLP^{\mathcal{TR}}$ Argumentation Theory

We present here a particularly interesting argumentation theory which extends GCLP—*generalized courteous logic programs* [36]—to $\mathcal{TR}$ under the $\mathcal{TR}^{DA}$ framework. The inferences claimed in the discussion of the planning example in Section 2 assumed that particular argumentation theory. We will call this argumentation theory $GCLP^{\mathcal{TR}}$. As any argumentation theory in our framework, $GCLP^{\mathcal{TR}}$ defines a version of the predicate `$defeated` using various auxiliary concepts. We define these concepts first.

The user-defined predicates `!`**`opposes`** and `!`**`overrides`** are relations specified over rule handles. They tell the system what rule instances are in conflict with each and which rule instances are preferred over other rules. For instance, in Example 1, the predicate instance `!`**`opposes`**$(\texttt{handle}(\_, sell(Stock)), \texttt{handle}(\_, buy(Stock)))$ is used to specify that any rule whose head is an instance of the $sell/1$ relation is incompatible with any rule whose head is an instance of the $buy/1$ relation with the same argument $Stock$. That is, selling and buying the same stock in the same state is contradictory. The predicate `!`**`overrides`** specifies that some actions have higher priority than other actions. For instance, in the same Example 1, the statement `!`**`overrides`**$(\texttt{handle}(sell\_action, \_), \texttt{handle}(buy\_action, \_))$ says that any rule tagged with $sell\_action$ has higher priority than any rule tagged with buy_action.

The predicate `$defeated` is defined indirectly in terms of the predicates `!`**`opposes`** and `!`**`overrides`**. In the following definitions the variables R and S are assumed to range over rule handles, while the implicit current state identifier D is assumed to range over the possible database states. A rule is *defeated* if it is *refuted* or *rebutted* by some other rule, assuming that the first rule is defeasible and the second rule is not *compromised* or *disqualified*. We will define these notions shortly, but first we explain them informally. A rule is *refuted* if a higher-priority rule implies a conclusion that is incompatible with the conclusion implied by the refuted rule, A rule *rebuts* another rule if the two rules assert conflicting conclusions and there is no way to resolve the conflict. A rule is $compromised$ if it is defeated by some other rule, and a rule *disqualified* if that rule defeats itself.

$$\begin{array}{l} \texttt{\$defeated}(R) :- \texttt{\$refutes}(S,R) \wedge \texttt{not}\ \texttt{\$compromised}(S). \\ \texttt{\$defeated}(R) :- \texttt{\$rebuts}(S,R) \wedge \texttt{not}\ \texttt{\$compromised}(S). \\ \texttt{\$defeated}(R) :- \texttt{\$disqualified}(R). \end{array} \tag{7}$$

In this paper we define a single argumentation theory, so we will use the most common interpretation of the aforementioned predicates. However, the reader should keep in mind that the argumentation theory is an input in our theory and can be changed as needed.

A rule R `$refutes` another rule S if R has higher-priority than S and R's conclusion is incompatible with the conclusion of S. Two rule handles are in conflict if they are both candidates and (their handles) are in opposition to each other.

It is defined as follows:

$$\begin{array}{l} \texttt{\$refutes}(R,S)\ \texttt{:-}\ \texttt{\$conflict}(R,S) \wedge \texttt{!}\mathbf{overrides}(R,S). \\ \texttt{\$conflict}(R,S)\ \texttt{:-} \\ \qquad \texttt{\$candidate}(R), \texttt{\$candidate}(S), \texttt{!}\mathbf{opposes}(R,S). \end{array} \tag{8}$$

A rule R *rebuts* another rule S if the two rules assert conflicting conclusions, but neither rule is “more important” than the other, *i.e.*, no preference can be infered between the two rules. This intuition can be expressed in several different ways, but we selected the one below, which mimics the definition in [53].

$$\texttt{\$rebuts}(R,S)\ \texttt{:-}\ \texttt{\$candidate}(R) \wedge \texttt{\$candidate}(S) \wedge \quad (9)$$
$$\texttt{!}\mathbf{opposes}(R,S) \wedge \texttt{not}\,\texttt{\$compromised}(R) \wedge$$
$$\texttt{not}\,\texttt{\$refutes}(\_,R) \wedge \texttt{not}\,\texttt{\$refutes}(\_,S).$$

The important difference here compared to [53] is that we are dealing with state-changing actions and so all tests for refutation, rebuttal, and the like, must be hypothetical. This is reflected in the definition of a rule candidate. We say that a rule instance is a *candidate* if its body is *hypothetically* true in the current database state. The other two rules in the group below specify the symmetry of `!`**`opposes`** and the fact that literals H and `neg` H are in conflict with each other.

$$\texttt{\$candidate}(R)\ \texttt{:-}\ \textbf{body}(R,B) \otimes \Diamond\textbf{call}(B). \quad (10)$$

$$\texttt{!}\mathbf{opposes}(X,Y)\ \texttt{:-}\ \texttt{!}\mathbf{opposes}(Y,X). \quad (11)$$

$$\texttt{!}\mathbf{opposes}(\texttt{handle}(\_,H),\ \texttt{handle}(\_,\ \texttt{neg}\ H)). \quad (12)$$

A rule is compromised if it is defeated, and it is disqualified if it transitively defeats itself. Here the predicate $\texttt{\$defeats}_{tc}$ denotes the transitive closure of `$defeats`.

$$\begin{array}{l}
\texttt{\$compromised}(R)\ \texttt{:-}\ \texttt{\$refuted}(R) \wedge \texttt{\$defeated}(R).\\
\texttt{\$disqualified}(X)\ \texttt{:-}\ \texttt{\$defeats}_{tc}(X,X).\\
\texttt{\$defeats}_{tc}(X,Y)\ \texttt{:-}\ \texttt{\$defeats}(X,Y).\\
\texttt{\$defeats}_{tc}(X,Y)\ \texttt{:-}\ \texttt{\$defeats}_{tc}(X,Z) \wedge \texttt{\$defeats}(Z,Y).
\end{array} \quad (13)$$

As in [53], one can define other versions of the above argumentation theory, which differ from the above in various edge cases. However, defining such variations is tangential to the main focus of the present paper.

6 Discussion and related work

Although a great number of works deal with defeasibility in logic programming, few have a goal similar to ours, namely to lift defeasible reasoning from static logic programming to a dynamic logic that allows the program to change the internal database (such as, transaction logic). Such a lifting would allow applications of such dynamic languages (that is, state changing actions, AI planning, and modeling workflows) to make use of preferences between rules. Due to the large literature on defeasible reasoning, we will not compare the results of this paper to defeasible reasoning applied to static logic programming, but instead, we refer the reader to the comparisons in [53] and [37], that compare static versions of the origin of our logic to other defeasible formalisms in logic programming based on the well-founded semantics and , respectively, the stable model semantics for disjunctive logic programs. In particular, these two papers compare

LPDA based approaches to the frameworks presented by Gelfond and Son in [32] (i.e., the logic of prioritized defaults), by Delgrande, Schaub, and Tompits in [20] (i.e., the ordered logic programs), and by Eiter et al. in [26] (i.e., the meta-interpretation approach to handling preferences) because they allow adaptive behaviours in using the preference information similar to our argumentation various theories. However, the main contribution of this paper relies not in the use of different argumentation theories, but in the lifting of LPDA to a dynamic logic, such as $\mathcal{TR}$. To our knowledge, none of the works surveyed by [19] has a similar goal as ours, but some defeasible logic formalisms match various applications of $\mathcal{TR}$, and such, we will compare our work to these works aimed to apply defeasible reasoning to various dynamic domains. In particular, we compare our work with the most representative works aiming to apply defeasible reasoning in planning represented in ASP, namely, the approach by Son and Pontelli in [46,47,49,48] and the approach by Delgrande, Schaub and Tompits in [21,23,22]. On another hand, the approaches adopted in [35,33,34] aim to apply defeasible reasoning for another application of the results presented in this paper, namely, in modeling, execution and verification of workflows.

Most of the works in applying preferences in AI planning work at the level of actual planning solutions (called trajectories in the works of Son and Pontelli or histories in the works of Delgrande, Schaub and Tompits). Such a planning solution is a sequence $s_0 a_1 s_1 ... a_n s_n$ where s_i's are states and a_i's are actions. On the contrary, our work applies preferences at the choice of action level and not at the planning solution level, this making impossible for us to represent certain application of preferences at the solution level, for instance, proffering the shortest plan possible (initially presented in [27]), desiring solutions that satisfy (at some point in time) a formula ϕ_2 over those that satisfy another formula ϕ_1 (called *choice order* in [22]), or preferring trajectories that satisfy one formula first and another formula later in the solution (a formula called *temporal order* in [22]). It is also worth mentioning that our work paper adopts the well-founded semantics [52] as opposed to most of the other works mentioned above adopting variants of the stable model semantics.

[46,47,49,48] develop a high-level language for the specification of preferences over trajectories (those are, planning histories), and then provide a logic programming encoding of the language based on answer set planning [50,24,38] (AI planning implemented in answer set programming [31,39]). Son and Pontelli's work combines the action language $\mathcal{B}$ [30] with the prioritized default theory developed by Gelfond and Son in [32]. Although, the original approach of [32] was close to that of [53] which is the precursor to our work here, the language being a meta-theory and allowing the various theories of defaults to differ from one domain to another), passing this initial origin, $\mathcal{TR}^{DA}$ is quite different from [48]. $\mathcal{TR}^{DA}$ is a full procedural logic with priorities between actions, where [48] specifies preferences between trajectories. In our logic the user is able to write rules that define preferences between actions for sets of starting states which satisfy certain formulas, while in the language of [48] one needs to list all such trajectory preferences making this task very cumbersome. This is also mentioned by the fact that certain additional arguments are needed by the user, such as, the length of the longest acceptable planning solution. The implementation of the various argumentation theories and of $\mathcal{TR}^{DA}$ is quite straightforward keeping in mind that the semantics reduces to the $\mathcal{TR}$ well-founded semantics of the unified program. However, the encoding of the $\mathcal{PP}$ language in [48] (see section 4.2) is specific to a single behaviour of preferences and includes a complex transformation to answer set programming. Finally, while we consider infinite domains for planning, we allow function symbols and non-deterministic actions, the approach in [48] only considers planning with complete information on finite domains and deterministic actions.

The approach by Delgrande, Schaub, and Tompits in [21,23,22] is even more different than our approach than the approach by Son and Pontelli. Instead of using preferences over actions, their language uses entirely preferences over fluents, namely, the two types of preferences mentioned above, choice and temporal order of achievement of fluent goals. However, this is totally under the capabilities of our system, since this can be, in fact, part of the query. For example, the preference of achieving a goal of $hasBanana$ followed by a goal of $hasChocolate$ can easily be represented at the level of actions by stating that `!`**`overrides`**$(getBanana, getChocolate)$ where the preconditions of these actions are that the monkey doesn't have these items, while the effects are the update of the internal database with these fluents $hasBanana$, respectively, $hasChocolate$. On the other hand, while the original work by Delgrande, Schaub, and Tompits in [20] was a framework of ordered logic programming that could use a variety of preference handling strategies, its application to planning resumes to a single behaviour of dealing with preferences.

Other systems have also adopted various kinds of preferences in planning, for instance, quality of planning in [3], solving multiple prioritized goals [4], but these works do not study a unified context for an active deductive database such as ours, but special cases of using defeasible logic programming formalisms to implement certain problems or translations of certain dynamic languages (for instance, action languages) in LP formalisms supporting certain kinds of defeasibility. Eiter et al. in [27] introduced a framework for planning with action costs using logic programming, where each action is assigned an integer cost, and plans with the minimal cost are considered to be optimal. This is again different from our approach since we don't deal with plans, but only with actions. Our language cannot express the preferences of the shortest plans discussed in [27], but it is more high-level since it expresses preferences between actions in various states. Similar to us, [27]'s work is the only other work that deals with planning in the presence of non-deterministic actions.

Finally, regarding dealing with preferences in modeling, execution and verification of workflows, we mention the work of Governatori et al. on modelling notions like delegation of tasks in the execution of a workflow. Another work by the same group, [33], deals compliance of workflows to a given regulation formulated in a variant of deontic logic, allowing expressions similar to what we have in transaction logic (with sequences of task/actions and or branching of actions), but not dealing with defeasible reasoning. Recently, [34] extended the work of [33] to model control flow patterns in workflows. However, in the last two papers defeasible reasoning is not studied at all, while in the case of the first paper is just tangential to our goals, being applied in the special case of delegation from one agent to another (more important) agent.

7 Implementation and evaluation

To demonstrate the benefits of defeasible reasoning in Transaction Logic, we implemented an interpreter for $\mathcal{TR}^{DA}$ in XSB [51] and tested it on a number of examples, including Example 2. The goal of these tests was to demonstrate how preferential heuristics can be expressed in $\mathcal{TR}^{DA}$ and to evaluate their effects on the efficiency of planning.

Table 1 shows how the preferential heuristic of Example 2 helps reduce the number of plans for pyramid construction (pruning away the plans for uninteresting pyramids), space, and time requirements. It shows that the number of plans and space requirements are reduced by one or two orders of magnitude and time is reduced by a factor of about 5. The discrepancy between improvements in the runtime and the reduction in the number of plans can be explained by the fact that, even without the optimizing heuristics, out implementation of $\mathcal{TR}^{DA}$ takes advantage of sharing of partially constructed plans among the different searches. Therefore, the reduction in the runtime is not as dramatic compared to the reduction and space and the number of plans.

World size		No heuristics	With preferential heuristics
10 blocks	Plans	120	8
	Time(sec.)	0.078	0.016
	Space(kBs)	155	26
20 blocks	Plans	1140	18
	Time(sec.)	0.563	0.109
	Space(kBs)	1162	60
30 blocks	Plans	4060	28
	Time(sec.)	2.390	0.438
	Space(kBs)	3730	90
40 blocks	Plans	9880	38
	Time(sec.)	7.000	1.219
	Space(kBs)	8562	120
50 blocks	Plans	19600	48
	Time(sec.)	17.109	2.938
	Space(kBs)	16347	150

Table 1. Planing in blocks world with and without preferential heuristics

We conclude the evaluation section with the extreme case where we have a world of 10,000 blocks blk_1, blk_2, ..., $blk_{10,000}$ being on the $table$ with blk_2 being $larger$ than the block blk_1 and blk_3 being $larger$ than both blocks blk_1 and blk_2, and so on, and an existential goal, $(\exists)stack(10,000, blk_{10,000})$ for stacking a pyramid of 9,999 blocks on the block $blk_{10,000}$ as a base. The original tabling algorithm presented in [29] would try to try plan 9,999 different pyramids where one block blk_i, $1 \leq i \leq 9,999$, would sit separately on the $table$ and easily fail because this requires a very large memory to store all reachable states. With the heuristic rules in Section 2, the new algorithm will return a single pyramid containing the blocks blk_2 to $blk_{10,000}$ with the block blk_1 sitting separately on the table, the rule being that on top of each clear block blk_i is preferred to stack the block blk_{i-1} since it's the largest clear block on the table. This will succeed in a short time because it requires only 1,000 steps and only 1,000 intermediate states to store in tables.

8 Conclusions

This paper developed a theory of defeasible reasoning in Transaction Logic, an extension of classical logic for representing both declarative and procedural knowledge. This new logic, called $\mathcal{TR}^{DA}$, extends our prior work on defeasible reasoning with argumentation theories from static logic programming to a logic that captures the dynamics in knowledge representation. We also extend the Courteous style of defeasible reasoning [36] to incorporate actions, planning, and other dynamic aspects of knowledge representation. We believe that $\mathcal{TR}^{DA}$ can become a rich platform for expressing heuristics about actions. We also made a contribution to the development of Transaction Logic itself by defining the well-founded semantics for it and for its $\mathcal{TR}^{DA}$ extension—a non-trivial adaptation of the classical well-founded semantics of [52].

References

1. Darko Anicic, Paul Fodor, Roland Stühmer, and Nenad Stojanovic. An approach for data-driven logic-based complex event processing. In *The 3rd ACM International Conference on Distributed Event-Based Systems (DEBS)*, 2009.
2. F. Baader and B. Hollunder. Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *Journal of Automated Reasoning*, 15(1):41–68, 1995.
3. Marcello Balduccini. Usa-smart: Improving the quality of plans in answer set planning. In *PADL*, 2004.
4. Marcello Balduccini. Solving the wise mountain man riddle with answer set programming. In *Ninth International Symposium on Logical Formalizations of Commonsense Reasoning*, 2009.
5. Moritz Y. Becker and Sebastian Nanz. A logic for state-modifying authorization policies. In *ESORICS*, 2007.
6. A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.
7. A.J. Bonner and Michael Kifer. Transaction logic programming (or a logic of declarative and procedural knowledge). Technical Report CSRI-323, University of Toronto, November 1995. `http://www.cs.toronto.edu/~bonner/transaction-logic.html`.
8. Anthony J. Bonner. Modular composition of transaction programs with deductive databases. In *DBPL*, pages 373–395, 1997.
9. Anthony J. Bonner and Michael Kifer. Transaction logic programming. In *ICLP*, pages 257–279, 1993.

10. Anthony J. Bonner and Michael Kifer. Applications of transaction logic to knowledge representation. In *ICTL*, pages 67–81, 1994.
11. Anthony J. Bonner and Michael Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133 (2):205–265, 1994.
12. Anthony J. Bonner and Michael Kifer. A logic for programming database transactions. In *Logics for Databases and Information Systems*, pages 117–166, 1998.
13. Anthony J. Bonner and Michael Kifer. Results on reasoning about updates in transaction logic. In *Transactions and Change in Logic Databases*, pages 166–196, 1998.
14. Anthony J. Bonner and Michael Kifer. The state of change: A survey. In *Transactions and Change in Logic Databases*, pages 1–36, 1998.
15. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
16. G. Brewka and T. Eiter. Prioritizing default logic. In *Intellectics and Computational Logic – Papers in Honour of Wolfgang Bibel*, pages 27–45. Kluwer Academic Publishers, 2000.
17. Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan. Logic based modeling and analysis of workflows. In *PODS*, pages 25–33, 1998.
18. Hasan Davulcu, Michael Kifer, and I. V. Ramakrishnan. Ctr-s: a logic for specifying contracts in semantic web services. In *WWW*, pages 144–153, 2004.
19. J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(12):308–334, 2004.
20. James P. Delgrande, Torsten Schaub, and Hans Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 2:129–187, 2003.
21. James P. Delgrande, Torsten Schaub, and Hans Tompits. Domain-specific preferences for causal reasoning and planning. In *ICAPS*, 2004.
22. James P. Delgrande, Torsten Schaub, and Hans Tompits. A general framework for expressing preferences in causal reasoning and planning. In *J. Log. Comput. 17(5)*, 2007.
23. James P. Delgrande, Torsten Schaub, and Hans Tompits. A preference-based framework for updating logic programs. In *LPNMR*, 2007.
24. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proceedings of European Conference on Planning*, pages 169–181, 1997.
25. P.M. Dung and Tran Cao Son. An argument-based approach to reasoning with specificity. *Artificial Intelligence*, 133(1-2):35–85, 2001.
26. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *Theory and Practice of Logic Programming*, 3(4):463–498, 2003.
27. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Answer set planning under action cost. In *Journal of Artificial Intelligence Research, 19*, pages 25–71, 2002.
28. Paul Fodor. Initial results on justification for the tabled transaction logic. In *AAAI Spring Symposium*, 2009.
29. Paul Fodor and Michael Kifer. Tabling transaction logic. In *PPDP*, 2010.
30. Michael Gelfond and Vladimir Lifschitz. Action languages. In *Electronic Transactions on AI, 3(16)*, 1998.
31. Michael Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between cwa, minimal model, and minimal herbrand model semantics. In *International Journal of Intelligent Systems 5 (5)*, pages 549–565, 1990.
32. Michael Gelfond and Tran Cao Son. Reasoning with prioritized defaults. In *Third International Workshop on Logic Programming and Knowledge Representation*, volume 1471 of *Lecture Notes in Computer Science*, pages 164–223. Springer, 1997.

33. Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business processes and business contracts. In *International Enterprise Distributed Object Computing Conference (EDOC)*, pages 221–232, 2006.
34. Guido Governatori and Antonino Rotolo. Norm compliance in business process modeling. In *International Web Rule Symposium (RuleML)*, pages 194–209, 2010.
35. Guido Governatori, Antonino Rotolo, and Shazia Sadiq. A model of dynamic resource allocation in workflow systems. In *Klaus-Dieter Schewe and Hugh E. Williams, editors, Database Technology 2004, Dunedin, New Zealand. Conference Research and Practice of Information Technology*, pages 197–206, 2004.
36. B.N. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report Supplementary Update Follow-On to RC 21472, IBM, July 1999.
37. Michael Kifer Hui Wan and Benjamin Grosof. Defeasibility in answer set programs via argumentation theories. In *Web Reasoning and Rule Systems (RR)*, 2010.
38. Vladimir Lifschitz. Answer set programming and plan generation. In *Artificial Intelligence 138 (1,2)*, pages 39–54, 2002.
39. I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. In *Annals of Mathematics and Artificial Intelligence 25 (3,4)*, pages 241–273, 1999.
40. D. Nute. Defeasible logic. In *Handbook of logic in artificial intelligence and logic programming*, pages 353–395. Oxford University Press, 1994.
41. H. Prakken. An argumentation framework in default logic. *Annals of Mathematics and Artificial Intelligence*, 9(1-2):93–132, 1993.
42. T.C. Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.
43. Dumitru Roman and Michael Kifer. Reasoning about the behavior of semantic web services with concurrent transaction logic. In *VLDB*, pages 627–638, 2007.
44. Dumitru Roman and Michael Kifer. Semantic web service choreography: Contracting and enactment. In *International Semantic Web Conference*, pages 550–566, 2008.
45. C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.
46. Tran Cao Son and Enrico Pontelli. Reasoning about actions in prioritized default theory. In *Logics in Artificial Intelligence*, pages 369–381, 2002.
47. Tran Cao Son and Enrico Pontelli. Adding preferences to answer set planning. In *ICLP*, 2003.
48. Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. In *LPNMR*, 2004.
49. Tran Cao Son and Enrico Pontelli. Reasoning about actions and planning with preferences using prioritized default theory. In *Computational Intelligence 20(1)*, 2004.
50. V. Subrahmanian and C. Zaniolo. Relating stable models and AI planning domains. In *Proceedings of the International Conference on Logic Programming*, pages 233–247, 1995.
51. The XSB Team. Xsb: A logic programming and deductive database system, 2009.
52. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.
53. H. Wan, B. Grosof, Michael Kifer, P. Fodor, and S. Liang. Logic programming with defaults and argumentation theories. In *Proceedings of 25th International Conference on Logic Programming (ICLP)*, pages 432–448, 2009.
54. K. Wang, L. Zhou, and F. Lin. Alternating fixpoint theory for logic programs with priority. In *First Int'l Conference on Computational Logic (CL'00)*, number 1861 in Lecture Notes in Computer Science, pages 164–178. Springer, 2000.

55. Y. Zhang, C.M. Wu, and Y. Bai. Implementing prioritized logic programming. *AI Communications*, 14(4):183–196, 2001.