Abstract. Transaction Logic (or $\mathcal{T}R$) is an extension of classical logic that gracefully integrates both declarative and procedural knowledge and has proved itself as a powerful formalism for many advanced applications, including modeling robot movements, actions specification, and planning in AI. In a parallel development, much work has been devoted to various theories of defeasible reasoning. In this paper, we unify these two streams of research and develop Transaction Logic with Defaults and Argumentation Theories (abbr., $\mathcal{T}R^{DA}$), an extension of both Transaction Logic and the recently proposed unifying framework for defeasible reasoning called Logic Programs with Defaults and Argumentation Theories (LPDA). 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 planning problems. We also demonstrate the usefulness of the approach by experimenting with a prototype of $\mathcal{T}R^{DA}$ 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, Transaction Logic, planning.

1 Introduction

Transaction logic (abbr., $\mathcal{T}R$) [9, 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 [7], to knowledge representation [5], AI planning [9], workflow management and Web services [32], and general knowledge base programming [8]. Defeasible reasoning is another important paradigm, which has been extensively studied in knowledge representation, policy specification, regulations, law, learning, and more [14, 22, 26].

In this paper we propose to combine $\mathcal{T}R$ 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 $\mathcal{T}R^{DA}$) because it extends $\mathcal{T}R$ in the direction of the recently proposed unifying framework for defeasible reasoning called logic programming with defaults and argumentation theories (LPDA) [38]. Along the way we define a well-founded semantics [37] for $\mathcal{T}R$, which, to the best of our knowledge, has never been done before.
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 $TR^{DA}$ 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 7.1 motivates reasoning with defaults in $TR$ with an example. Section 3 provides background on Transaction Logic to make the paper self-contained. Section 4 extends $TR$ by incorporating defeasible reasoning. Section 5.1 specializes the logic developed in Section 4 by defining a useful argumentation theory that extends Generalized Courteous Logic Programs (GCLP) [26]. In Section 6 we discuss related work and Section 8 summarizes the paper and outlines future work.

2 Motivating Examples

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

The syntax of $TR^{DA}$ 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, $\text{pickup}(\text{block}1) \otimes \text{puton}(\text{block}1, \text{block}2)$ says that the action $\text{pickup}(\text{block}1)$ is to be executed first and the action $\text{puton}(\text{block}1, \text{block}2)$ 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 transitions for basic manipulation of states. These include $\text{delete}(f)$ and $\text{insert}(f)$ for every ground fluent $f$. Examples of such elementary transitions include $\text{delete(on(\text{block}1, \text{block}0))}$ and $\text{insert(clear(\text{block}0))}$.

As usual in defeasible reasoning, rules in $TR^{DA}$ can be tagged with terms. For instance, the rules in the example below are tagged with the constants $\text{buy_action}$ and $\text{sell_action}$. The predicate $\text{opposes}$ is used to specify that some rules are incompatible with others (for example, buying and selling the same stock). The predicate $\text{overrides}$ 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 21 (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. 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.

\[
\begin{align*}
@\text{buy} & \text{act } \text{buy}(\text{Stock}, \text{Amount}) :- \\
& \text{recommendation}(\text{buy}, \text{Stock}) \otimes \text{owns}(\text{Stock}, \text{Qty}) \otimes \\
& \text{delete}(\text{owns}(\text{Stock}, \text{Qty})) \otimes \text{insert}(\text{owns}(\text{Stock}, \text{Qty}+\text{Amount})). \\
@\text{sell} & \text{act } \text{sell}(\text{Stock}, \text{Amount}) :- \\
& \text{recommendation}(\text{sell}, \text{Stock}) \otimes \text{owns}(\text{Stock}, \text{Qty}) \otimes \\
& \text{delete}(\text{owns}(\text{Stock}, \text{Qty})) \otimes \text{insert}(\text{owns}(\text{Stock}, \text{Qty}-\text{Amount})).
\end{align*}
\]

\[
!\text{opposes}(\text{sell}(\text{Stock}), \text{buy}(\text{Stock})). \\
!\text{overrides}(\text{sell} \text{act}, \text{buy} \text{act}). \\
\text{recommendation}(\text{buy}, \text{C}) :- \text{services}(\text{X}). \\
\text{recommendation}(\text{sell}, \text{C}) :- \text{media}(\text{X}). \\
\text{services}(\text{acme}). \\
\text{media}(\text{acme}). \\
\text{owns}(\text{acme}, 100). \\
\text{trade}(\text{Stock}, \text{Amount}) :- \text{buy}(\text{Stock}, \text{Amount}). \\
\text{trade}(\text{Stock}, \text{Amount}) :- \text{sell}(\text{Stock}, \text{Amount}).
\]

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). Let us consider a transaction \text{trade}(\text{acme}, 100). Without the \text{!opposes} and \text{!overrides} information this goal would have two non-deterministic possible executions: one in which the trader buys an additional 100 stocks in the company \text{acme}, and another one in which the trader sells 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 an uncertain situation it is advisable to sell the stocks.

Example 22 (Block world planning) This example illustrates the use of defeasible reasoning for heuristic optimization of planning in the blocks world. The \text{TR}^\text{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.\footnote{For more information on planning with \text{TR} see \cite{9}.}

In this example, we represent the blocks world using the fluents \text{on}(x,y), which say that block \text{x} is on top of block \text{y}; \text{isclear}(x), which says that nothing is on top of block \text{x}; and \text{larger}(x,y), which says that the size of \text{x} is larger than the size of \text{y}. The action \text{pickup}(X,Y) lifts the block \text{X} from the top of block \text{Y} and the action \text{putdown}(X,Y) puts it down on top of block \text{Y}. These actions are specified by the second and third rules, respectively. The action \text{move}(X, \text{From}, \text{To}), specified by the first rule, moves block \text{X} from its current position on top of block
From to a new position on top of block To. This action is defined by combining the aforementioned actions pickup and putdown, if certain preconditions 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) :- (isclear(X) ∧ on(X,Y)) ⊗
  delete(on(X,Y)) ⊗ insert(isclear(Y)).
putdown(X,table) :- (isclear(X) ∧ not on(X,Z))
  ⊗ insert(on(X,table)).
putdown(X,Y) :- (isclear(X) ∧ isclear(Y) ∧ not on(X,Z))
  ⊗ delete(isclear(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).
isclear(blk1). isclear(blk2). isclear(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 our example, this makes unstacking blk1 and moving it to the table surface preferable to unstacking blk2, since the former is a smaller block. This blocks...
the opportunity to then move blk4 on top of blk2 and subsequently put blk1 on top of blk4. These preference rules can be applied to a pyramid-building program like this:

```prolog
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 by the interpreter described in [20] shows that the above preferences drastically reduce the number of plans that need to be considered—sometimes to just one plan. These experiments are described in Section 7.

## 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 [9].

The syntax of $\mathcal{TR}$ is derived from that of standard logic programming. The alphabet of the language $L_{\mathcal{TR}}$ 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 $\text{not}$ 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:

1. $\neg \alpha$, $\text{not} \alpha$, $\text{not neg} \alpha$,

where $\alpha$ is a fluent atomic formula. An action literal is an action atomic formula or has the form $\text{not} \alpha$, where $\alpha$ is a action atomic formula. Literals of the form...
neg $\alpha$, where $\alpha$ is an action, are not allowed. Atoms of the form $\text{neg not } \alpha$ 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 $\text{neg } f$, 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 $\text{insert}(f)$ or $\text{delete}(f)$, where $f$ is a ground base fluent or has the form $\text{neg } g$, where $g$ is a ground base fluent. For any given database state $D$,

- $\text{insert}(f)$ causes a transition from $D$ to the state $D \cup \{f\} \setminus \{\text{neg } f\}$; and
- $\text{delete}(f)$ causes a transition from $D$ to $D \setminus \{f\} \cup \{\text{neg } f\}$.

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

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

The formula $\phi \otimes \psi$ represents an action composed of an execution of $\phi$ followed by an execution of $\psi$, while the formula $\lozenge \phi$ is an action of hypothetically testing whether $\phi$ can be executed at the current state, but no actual state changes take place. In procedural terms, executing $\text{delete(on(blk1,table))} \otimes \text{insert(on(blk1,blk2))}$ means “first delete $\text{on(blk1,table)}$ from the database, and then insert $\text{on(blk1,blk2)}$.” The current database state changes as a result. In contrast, $\lozenge \text{move(blk1)}$ is only a “hypothetical” execution: it checks whether $\text{move(blk1)}$ 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 $f_1 \land f_2$ and $\lozenge f$ to $f$. Therefore, when no actions are present, $\mathcal{TR}$ reduces to classical logic. This also explains our use of $\land$ in Example 22 where it could have been replaced with $\otimes$ without changing the meaning (but, the uses of $\otimes$ in the Example 22 cannot be replaced with $\land$ 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 $\text{not}$ and $\text{neg}$, and action literals can contain $\text{not}$.
- If $P$ is a serial goal, then so are $\text{not } P$ and $\lozenge P$.
- If $P_1$ and $P_2$ are serial goals then so are $P_1 \otimes P_2$ and $P_1 \land P_2$. \hfill $\square$
Definition 2 (Serial rules). A serial rule is an expression of the form:

\[ H : \neg B. \]  

(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 \( B = f_1 \otimes \ldots \otimes f_n \), where each \( f_i \) is a fluent literal (and thus \( \otimes \) could be replaced with \( \land \)).

- **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.

A existential serial goal is a statement of the form \( \exists \bar{X} \psi \) where \( \psi \) is a serial goal and \( \bar{X} \) is a list of all free variables in \( \psi \). For instance, \( \exists \text{move}(X, \text{blk}2) \) is an existential serial goal. Informally, the truth value of an existential goal in \( \mathcal{T} \mathcal{R} \) is determined over sequences of states, called execution paths, which makes it possible to view truth assignments in \( \mathcal{T} \mathcal{R} \)'s models as executions. If an existential serial goal, \( \psi \), defined by a program \( P \), evaluates to true over a sequence of states \( D_0, \ldots, D_n \), we say that it can execute at state \( D_0 \) by passing through the states \( D_1, \ldots, D_{n-1} \), and ending in the final state \( D_n \). Formally, this is captured by the notion of executional entailment, which is written as follows:

\[ P, D_0, \ldots, D_n \models \psi \]

Further details on \( \mathcal{T} \mathcal{R} \) can be found in [9, 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 (\( \mathcal{T} \mathcal{R}^{DA} \)). The development was inspired by our earlier work on logic programming with argumentation theories, which did not support actions [38]. Language-wise, the only difference between \( \mathcal{T} \mathcal{R}^{DA} \) and serial \( \mathcal{T} \mathcal{R} \) is that the rules in \( \mathcal{T} \mathcal{R}^{DA} \) are tagged.

4.1 \( \mathcal{T} \mathcal{R}^{DA} \) Syntax

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

\[ @r H : - B. \]  

(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 \( \mathcal{T} \mathcal{R}^{DA} \) transaction base \( P \) is a set of rules, which can be strict or defeasible.
Definition 4 ($\mathcal{TR}^{DA}$ Transaction formula). A $\mathcal{TR}^{DA}$ transaction formula in the language $\mathcal{TR}^{DA}$ is a literal, a serial goal, a tagged or an untagged serial rule.

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 defeasible 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 (note that in this report, we consider strict rules to be unlabeled rules as in Definition 2).

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

$$\text{handle}(r, H)$$

is called the handle of that rule.

$\mathcal{TR}^{DA}$ 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, $\text{defeated}_{AT}$, which may appear in the heads of some rules in $AT$ but not in the transaction base. A $\mathcal{TR}^{DA}$ $P$ is said to be compatible with $AT$ if $\text{defeated}_{AT}$ does not appear in any of the rule heads in $P$.

The rules $AT$ are used to specify how the rules in $P$ get defeated. This can be accomplished using special predicates defined in $\mathcal{TR}^{DA}$, such as, the $\text{opposes}$ and $\text{overrides}$ predicates in the courteous argumentation theories. 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 $\text{defeated}_{AT}$, 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 report.

4.2 $\mathcal{TR}^{DA}$ Well-founded Semantics

We extend the well-founded semantics for logic programming [37] to $\mathcal{TR}^{DA}$ using the Przymusinski-style definition [31]. In the following definition, 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 7 (3-valued Partial Herbrand interpretation). A partial Herbrand interpretation is a mapping $\mathcal{H}$ that assigns $f$, $u$ or $t$ to every formula $L$ in $B$.

A partial Herbrand interpretation $\mathcal{H}$ is consistent relative to an atomic formula $L$ if it is not the case that $\mathcal{H}(L) = \mathcal{H}(\neg L) = t$. $\mathcal{H}$ is consistent if, for every ground not-free formula $L$ (other than $u$), either $\mathcal{H}(L) = t$ and $\mathcal{H}(\neg L) = f$ or $\mathcal{H}(L) = f$ and $\mathcal{H}(\neg L) = t$.

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

Definition 8 (3-valued 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. $I(\langle D \rangle)(d) = t$, if $d \in D$;
   $I(\langle D \rangle)(d) = f$, if $\neg d \in D$;
   $I(\langle D \rangle)(d) = u$, otherwise, for every ground base fluent literal $d$ and every database state $D$.

2. $I(\langle D_1, D_2 \rangle)(\text{insert}(p)) = t$ if $D_2 = D_1 \cup \{p\} \setminus \{\neg p\}$ and $p$ is a ground fluent literal;
   $I(\langle D_1, D_2 \rangle)(\text{insert}(p)) = f$, otherwise.

3. $I(\langle D_1, D_2 \rangle)(\text{delete}(p)) = t$ if $D_2 = D_1 \setminus \{p\} \cup \{\neg p\}$ and $p$ is a ground fluent literal;
   $I(\langle D_1, D_2 \rangle)(\text{delete}(p)) = f$, otherwise.

Without loss of generality, in defining the semantics of $TR^{DA}$ 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 a set of all of their ground instantiations.

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

Definition 9 ($TR^DA$ 3-valued Truth valuation in path structures). Let $I$ be a path structure, $\pi$ a path, $L$ a ground not-free literal, and let $F$, $G$ ground serial goals We define truth valuations with respect to the path structure $I$ as follows:

- If $p$ is a not-free literal then $I(\pi)(p)$ is already defined because $I(\pi)$ is a Herbrand interpretation, by definition of $I$. 

For any path $\pi$:
\[
I(\mathcal{P})(\pi^t) = t \quad \text{and} \quad I(\mathcal{P})(\pi^u) = f, \quad \text{if} \quad \pi' \neq \pi;
\]
\[
I(\mathcal{P})(\pi^w) = u \quad \text{and} \quad I(\mathcal{P})(\pi^w) = f, \quad \text{if} \quad \pi' \neq \pi.
\]

If $\phi$ and $\psi$ are serial goals, then
\[
I(\mathcal{P})(\phi \otimes \psi) = \max\{\min(I(\mathcal{P})(\phi), I(\mathcal{P})(\psi)) | I = \pi \lor \pi\}.
\]

If $\phi$ and $\psi$ are serial goals then $I(\mathcal{P})(\phi \land \psi) = \min(I(\mathcal{P})(\phi), I(\mathcal{P})(\psi))$.

If $\phi$ is a serial goal then $I(\mathcal{P})(\text{not } \phi) = \neg I(\mathcal{P})(\phi)$, where $\neg t = f$, $\neg f = t$, and $\neg u = u$.

If $\phi$ is a serial goal and $\pi = (D)$, where $D$ is a database state, then
\[
I(\mathcal{P})(\phi \circ \phi) = \max\{I(\mathcal{P})(\phi) | \pi' \text{ is a path that starts at } D\}
\]
$I(\mathcal{P})(\phi \circ \phi) = f$, otherwise.

For a strict serial rule $F : G$,
\[
I(\mathcal{P})(F : G) = t \iff I(\mathcal{P})(F) \geq I(\mathcal{P})(G).
\]

For a defeasible rule $F : F$,
\[
I(\mathcal{P})(F : F) = t \iff I(\mathcal{P})(F) \geq I(\mathcal{P})(G).
\]

We will write $I, \pi \models \phi$ and say that $\phi$ is satisfied on path $\pi$ in the path structure $I$ if $I(\mathcal{P})(\phi) = t$.

We will say that a path structure $I$ is total if, for every path $\pi$ and every serial goal $\phi$, $I(\mathcal{P})(\phi) = t$ or $f$.

**Definition 10. (TRA $^D$A 3-valued Model of a Transactional Formula) A path structure, $I$, is a model of a transaction formula $\phi$ if $I, \pi \models \phi$ for every path $\pi$. In this case, we write $I \models \phi$ and say that $I$ is a model of $\phi$ or that $\phi$ is satisfied in $I$. A path structure $I$ is a model of a set of formulas if it is a model of every formula in the set.**

**Definition 11. (Model of TRA $^D$A). A path structure $I$ is a model of a serial TRA $^D$A transaction base $P$ if all the rules in $P$ are satisfied in $I$ (that is, if $I \models R$ for every $R \in P$). Given a TRA $^D$A transaction base $P$, an argumentation theory $AT$, and a path structure $M$, we say that $M$ is a model of $P$ with respect to the argumentation theory $AT$, written as $M \models (P, AT)$, if $M \models P$ and $M \models AT$.**

Like classical logic programs, the Herbrand semantics of serial TRA can be formulated as a fixpoint theory [7]. In classical logic programming, given two Herbrand partial interpretations $\sigma_1$ and $\sigma_2$, $\sigma_1 \leq \sigma_2$ if all not-free literals that are true in $\sigma_1$ are also true in $\sigma_2$ and all not- literals that are true in $\sigma_2$ are also true in $\sigma_1$. Similarly, given two Herbrand partial interpretations $\sigma_1$ and $\sigma_2$, $\sigma_1 \leq \sigma_2$ if all not-free literals that are true in $\sigma_1$ are also true in $\sigma_2$ and all not- literals that are true in $\sigma_1$ are also true in $\sigma_2$.

**Definition 12. (Order on Path Structures) If $M_1$ and $M_2$ are two Herbrand partial path structures, then $M_1 \preceq M_2$ if $M_1(\pi) \preceq M_2(\pi)$ for every path, $\pi$ (truth ordering). Similarly, we have $M_1 \preceq M_2$ if $M_1(\pi) \preceq M_2(\pi)$ for every path, $\pi$ (information ordering).**
A model $M$ of $P$ is minimal with respect to $\preceq$ iff for any other model, $N$, of $P$, $N \preceq M$ implies $N = M$. The least model of $P$ is a minimal model that is unique.

It is well-known that in ordinary logic programming any set of Horn rules always has a least model. In [9], it is shown that every definite Horn TR program has a unique least total model. Theorem 1, below, shows that this property is preserved by serial not-free TR programs, but in this case the model might be a partial path structure. Serial not-free programs are more general than the positive TR programs because the undefined propositional symbol $u^\pi$ for some path $\pi$ may occur in the bodies of the program clauses.

**Theorem 1 (Unique Least Partial Model for serial not-free TR programs).** If $P$ is a not-free TR program, then $P$ has a least Herbrand model, denoted $LPM(P)$.

**Proof.** See Appendix A. \hfill \square

**Example 1.** Let the TR program $P$ be:

$$
\begin{align*}
a & : - \text{state}, \\
b & : - a \otimes u^{(D_\emptyset)}, \\
c & : - c \otimes u^{(D_\emptyset)}.
\end{align*}
$$

where $a$, $b$, and $c$ are action symbols and $D_\emptyset$ is the empty database state. The least partial model of $P$ is a path structure that maps the 1-path $\langle D_\emptyset \rangle$ 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 $LPM(P)$ 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 $u$. \hfill \square

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

Next we define well-founded models for TRDA by adapting the definition from [31]. First, we define the quotient operator, which takes a $TR^{DA}$ program $P$ and a path structure $I$ and yields a serial-Horn TR program $\frac{P}{I}$

**Definition 13 (TRDA Quotient).** Let $P$ be a set of $TR^{DA}$ rules and $I$ a path structure for $P$. The $TR^{DA}$ quotient of $P$ by $I$, written as $\frac{P}{I}$, is defined through the following sequence of steps:

1. First, each occurrence of every not-literal of the form $\lnot L$ in $P$ is replaced by $t^\pi$ for every path $\pi$ such that $I(\pi)(\lnot L) = t$ and with $u^\pi$ for every path $\pi$ such that $I(\pi)(\lnot L) = u$. 

2. For each labeled rule of the form $\Box r L : - Body$ obtained in the previous step, replace it with the rules of the form:

\[
L : - t^{(D_t)} \otimes Body \\
L : - u^{(D_u)} \otimes Body
\]

for each database state $D_t$ such that

\[
I(\langle D_t \rangle)(\text{not} \Diamond \text{defeated(handle(r,L)))) = t
\]

and each database state $D_u$ such that

\[
I(\langle D_u \rangle)(\text{not} \Diamond \text{defeated(handle(r,L)))) = u
\]

3. Remove the labels from the remaining rules.

The resulting set of rules is the quotient $\frac{P}{I}$. \hfill \square

Note that in Step 1 of the above definition of the quotient each occurrence of $\text{not} L$ is replaced with different $t^\pi$ and $u^\pi$ for different $\pi$’s, so every rule in $P$ may be replaced with several (possibly infinite number of) $\text{not}$-free rules. All combinations of replacements for the $\text{not}$- literals in the body of the rules have to be used. Only the $\pi$’s where $I(\pi)(\text{not} L) = f$ are not used, which effectively means that the rule instances that correspond to those cases are removed from consideration. Also note that, the $\mathcal{TR}^{DA}$ quotient of a $\mathcal{TR}^{DA}$ transaction base $P$ with respect to an argumentation theory $AT$ takes as input a path structure $I$ and generates a new path structure as follows:

$$\Gamma(I) = \text{LPM}(\frac{P \cup AT}{I})$$

Suppose $I_0$ is the path structure that maps each path $\pi$ to the empty Herbrand interpretation in which all propositions are undefined (i.e., for every path $\pi$ and every literal $L$, we have $I_0(\pi)(L) = u$.

The ordinal powers of the immediate consequence operator $\Gamma$ are defined inductively as follows:

- $\Gamma^0(I_0) = I_0$;
\[ \Gamma^{\alpha}(I_0) = \Gamma(\Gamma^{\alpha-1}(I_0)), \text{ for } \alpha \text{ a successor ordinal}; \]
\[ \Gamma^{\alpha}(I_0)(\pi) = \bigcup_{\beta<\alpha} \Gamma^{\beta}(I_0)(\pi), \text{ for every path } \pi \text{ and } \alpha \text{ a limit ordinal}. \]

The operator \( \Gamma \) is monotonic with respect to the \( \leq \) order relation when \( P \) and \( AT \) are fixed (see Appendix B). Because \( \Gamma \) is monotonic, the sequence \( \{\Gamma^{\alpha}(I_0)\} \) has a least fixed point and is computable via transfinite induction (see Appendix B).

**Definition 15 (Well-founded model).** The well-founded model of a TRDA transaction base \( P \) with respect to the argumentation theory \( AT \), written as \( WFM(P, AT) \), is defined as the limit of the sequence \( \{\Gamma^{\alpha}(I_0)\} \).

The next theorem states that our constructive computation of the least model of the program \( (P, AT) \) is correct.

**Theorem 2 (Correctness of the Constructive TRDA Least Model).**

\( WFM(P, AT) \) is the least model of \( (P, AT) \).

**Proof.** See Appendix B.

The next theorem shows that TRDA programs under the well-founded semantics reduce to ordinary TR programs under the same well-founded semantics. In conclusion, TRDA can be implemented using ordinary transaction logic programming systems that support the well-founded semantics.

**Theorem 3 (TRDA Reduction).**

\( WFM(P, AT) \) coincides with the well-founded model of the TR program \( P' \cup AT \), where \( P' \) is obtained from \( P \) by changing every defeasible rule \( (@r L :- Body) \in P \) to the plain rule \( L :- \neg \text{defeated(handle(r,L))} \) \( \otimes \) Body and removing all the remaining tags.

**Proof.** See Appendix C.

## 5 Argumentation theory representatives

Various argumentation theories can be defined to abstract the multiple intuitions about defeasibility. These argumentation theories are a set of definitions for concepts that a reasoner might use to argue why certain conclusions are to be defeated or to win over other conclusions. In the following sections we define two such argumentation theories: one for the *generalized courteous logic programs* (GCLP) ([26] being the only commercially available defeasible reasoning formalism, i.e., IBM’s CommonRules\(^2\)), and one for defeasible logic, a popular formalism that attracts a lot of attention in the field.

5.1 The \( GCLP^{TR} \) courteous argumentation theory

As our first example of an argumentation theory, we present here a particularly interesting argumentation theory which extends \textit{generalized courteous logic programs} (GCLP) \cite{26} to \( TR \) under the \( TR^{DA} \) framework. This argumentation theory was used in the trade 2 and planning 22 examples in Section 7.1. We will call this argumentation theory \( GCLP^{TR} \). As any argumentation theory in this framework, \( GCLP^{TR} \) defines a version of the \$defeated predicate using various auxiliary concepts. We define these concepts first. The user - defined predicates \$opposes and \$overrides are relations specified over rule handles telling the system what rules are in opposition, respectively, what rules are preferred over the application of other rules. For instance, in the example 2, the predicate instance \$opposes(handle(_, sell(Stock)), 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 with the same argument \textit{Stock} (that is, selling and buying the same stock in the same state is contradictory). In a parallel manner, the predicate \$overrides specifies that some actions have higher priority than other actions. For instance, in the same trade example 2, the predicate instance \$overrides(handle(sell\_action, _), handle(buy\_action, _)) is used to specify that the rule \textit{sell\_action} has higher priority than the rule \textit{buy\_action}, regardless to their rule heads if an opposition situation arises.

The predicate \$defeated is defined indirectly in terms of the predicates \$opposes and \$overrides. In the following definitions the variables \( R \) and \( S \) are expected to range over rule handles, while the implicit current state identifier \( D \) is expected to range over the possible database states. A rule is \$defeated if it is \$refuted or \$rebuts by some other rule, where the former rule itself is defeasible (in our case, tagged) and the winning rule is not \$compromised, or the rule is \$disqualified. We will define these relations shortly, for the moment we just mention the most common meanings of these predicates: \$refutes means that a higher-priority rule implies a conclusion that is incompatible with the conclusion implied by the another rule, \$rebuts means that a pair of rules assert conflicting conclusions without being able to select a conclusion “more important” than the other conclusion, \$compromised means that it’s argument rule handle is defeated by some other rule handle, while \$disqualified is a special situation when a rule refutes itself (for instance, such a situation is actually possible in the block world when the action of moving an unique block requires this action to beat all other move actions, but not itself).

\[
\begin{align*}
    \text{defeated}(R) & : = \text{refutes}(S, R) \land \neg \text{compromised}(S), \\
    \text{defeated}(R) & : = \text{rebuts}(S, R) \land \neg \text{compromised}(S), \\
    \text{defeated}(R) & : = \neg \text{disqualified}(R),
\end{align*}
\]

In this report we define a single GCLP-style 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 $\text{R}$ $\text{refutes}$ another rule $\text{S}$ if $\text{R}$ has higher - priority than $\text{S}$ and $\text{R}$’s conclusion is incompatible with the conclusion of $\text{S}$. Two rule handles are in conflict if they are both candidates and (their handles) are in opposition to each other. These are defined as follows:

\[
\text{$\text{refutes}(\text{R}, \text{S}) :\text{= $\text{conflict}(\text{R}, \text{S}) \land \neg \text{overrides}(\text{R}, \text{S}).$}}
\]

\[
\text{candidate}(\text{R}), \text{candidate}(\text{S}), \neg \text{opposes}(\text{R}, \text{S}).
\]

A rule $\text{R}$ $\text{rebuts}$ another rule $\text{S}$ if the two rules assert conflicting conclusions, but neither rule is “more important” than the other, that is, there is preference relation can be inferred between the two rules. This intuition can also be expressed in several different ways, but we have selected the following definition 7 as the most intuitive definition matching the GCLP theory. We define a candidate rule handler as a rule instance whose body is hypothetically true in the current database state (that is, it can be executed hypothetically in the current state) in the rule 8, and the symmetric $\neg \text{opposes}$ relation in the rule 9 with the addition that every literal must oppose its explicit negation ($\neg$) in the rule 10. We use two meta-predicates, $\text{body}$ and $\text{call}$, where the $\text{body}$ meta - predicate in $\text{candidate}$ binds $\text{B}$ to the body of a rule with handle $\text{R}$, and the $\text{call}$ meta - predicate takes a serial goal and executes it. We emphasize that the key aspect of the candidacy predicate is the fact that the bodies are executed hypothetically, so they do not modify the current state of the database.

\[
\text{$\text{rebuts}(\text{R}, \text{S}) :\text{= $\text{candidate}(\text{R}) \land \text{candidate}(\text{S}) \land$}}
\]

\[
\neg \text{opposes}(\text{R}, \text{S}) \land \neg \text{compromised}(\text{R}) \land
\]

\[
\neg \text{$\text{refutes}(\text{R}, \text{S}) \land \neg \text{refutes}(\text{R}, \text{S}).$}
\]

\[
\text{candidate}(\text{R}) :\text{:= $\text{body}(\text{R}, \text{B}) \odot \text{call}(\text{B}).$}
\]

\[
\neg \text{opposes}(\text{X}, \text{Y}) :\text{:= $\neg \text{opposes}(\text{Y}, \text{X}).$}
\]

\[
\neg \text{opposes($\text{handle}(\text{R}, \text{H}), \text{handle}(\text{neg} \text{H}))}. \quad (10)
\]

A rule is compromised if it is defeated, and it is disqualified if it transitively refutes itself. The predicate $\text{refutes}_c$ denotes the transitive closure of the predicate $\text{refutes}$.

\[
\text{$\text{compromised}(\text{R}) :\text{= $\text{refutes}_c(\text{R}) \land \text{defeated}(\text{R}).$}}
\]

\[
\text{$\text{disqualified}(\text{X}) :\text{= $\text{refutes}_c(\text{X}, \text{X}).$}}
\]

\[
\text{$\text{refutes}_c(\text{X}, \text{Y}) :\text{= $\text{refutes}_c(\text{X}, \text{Y}).$}}
\]

\[
\text{$\text{refutes}_c(\text{X}, \text{Y}) :\text{= $\text{refutes}_c(\text{X}, \text{Z}) \land \text{refutes}_c(\text{Z}, \text{Y}).$}}
\]

As in [38], 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 our main focus here.
5.2 An argumentation theory for defeasible logic

We develop here an argumentation theory that captures the reasoning in the Defeasible Logic family of logics [30, 1, 2, 28]. This form of defeasible reasoning is particularly interesting because Governatori, Rotolo and Sadiq used it to execute workflows in [25]. Formally, Defeasible Logic is a triple \((R, >, K)\), where \(K\) is a finite set of literals, \(R\) is a set of rules, such that if \(q\) is any ground literal then the rules whose head is \(q\), \(R[q]\) is finite, and “\(>\)” is a superiority relation on \(R\). Defeasible Logic partitions the rules \(R\) into strict, defeasible, and defeater rules, where:

(a) the strict rules are rules which cannot be defeated and need to be satisfied even if the database is inconsistent,
(b) the defeasible rules are rules that can be defeated either by facts inferred by the strict rules or by the defeaters, and
(c) the defeater rules are used only to defeat other rules, but they do not produce any inferences. The purpose of the defeater rules is to block inferences produced by other rules.

The opposition among literals is limited to \(p\) and \(\text{neg } p\), for each fluent \(p\), while the use of the default negation is not allowed, so all literals are not-free, and rule tags are unique identifiers of the rules. The theory of Defeasible Logic easily translates into the computation of the fixed point of four sets: ground literals that are strictly true, ground literals that are strictly false, ground literals that are defeasible true, and ground literals that are defeasible false.

We need a few special predicates provided by the interpreter: a meta-predicate \(\text{head}/2\) that binds the first argument to the head of a rule with the identifier the second argument, a meta-predicate \(\text{body}/2\) that binds the first argument to the body of a rule with the identifier the second argument, a meta-predicate \(\text{call}/1\) that takes a serial goal and executes it on an execution path and and a predicate \(\text{break} \otimes /3\) that takes a serial conjunction \(B_1 \otimes B_2 \otimes \ldots \otimes B_n\) and returns the first element of the conjunction as the second argument and the rest of the conjunction as the third argument or \(\text{state}\) if the conjunction was a single element \(B_1\).

The program the following extra predicates: \(!\text{strict}/1\) for rules that cannot be defeated, \(\$\text{defeater}/1\) for rules used only to defeat other rules, but do not produce any inferences, and \(/>/2\) as a superiority relation between rules.

A rule is defeated if any of the following conditions hold: another conclusion in conflict with the current conclusion is definitely proved, the conclusion is detected by a defeater (because defeaters make no inferences), or the conclusion is refuted.

\[
\text{defeated}(\text{handle}(T, H)) \leftarrow \text{conflict}(\text{handle}(T, H), \text{handle}(S, H)) \\
\land \text{head}(S, H) \land \text{definitely}(H).
\]

\[
\text{defeated}(\text{handle}(T, H)) \leftarrow \text{defeater}(\text{handle}(T, H)).
\]

\[
\text{defeated}(\text{handle}(T, H)) \leftarrow \text{refutes}(\_, \text{handle}(T, H)).
\]

Two rules are in conflict if they are both candidates and their literals are incompatible (i.e., a literal \(L\) and its explicit negation \(\text{neg } L\)).
$conflict(\text{handle}(T_1, H_1), \text{handle}(T_2, H_2)) :-$

$\text{candidate}(\text{handle}(T_1, H_1)) \land \text{candidate}(\text{handle}(T_2, H_2)) \land \neg \text{opposes}(H_1, H_2)$.

$\text{candidate}(R) :- \text{body}(R, B) \otimes \text{call}(B)$.

$\neg \text{opposes}(L_1, \neg L_1)$.

$\neg \text{opposes}(\neg L_1, L_1)$.

A literal is definitely proved if it is the head of a strict rule whose body is proved only by strict clauses. We prove the body by proving all the literals in the body using the meta predicate break_⊗ and recursion. The effects of the $\text{definitely}$ call are not visible to the rest of the computation.

$\text{definitely}(L) :- \circ \text{strictly proved}(L)$.

strictly_proved(state).

strictly_proved(L) :- !\text{strict}(R) \land \text{head}(R, L) \land \text{body}(R, B) \otimes \text{strictly proved conj}(B)$.

strictly_proved conj(state).

strictly_proved conj(B) :- break_\otimes (B, B_1, B_2) \otimes \text{strictly proved conj}(B_1) \otimes \text{strictly proved conj}(B_2)$.

An additional rule is added to the instances of the predicate > /2 to state that any strict rule has priority to any non-strict rule:

$>(R_1, R_2) :- !\text{strict}(R_1) \land \neg !\text{strict}(R_1)$.

Finally, the $\text{refutes}/2$ relation is defined using the notions of $\text{candidate}$ and $\text{conflict}$.

$\text{refutes}(S, T) :- \text{conflict}(S, T) \land \text{candidate}(S) \land \text{candidate}(T) \land \neg \text{refutes}(\_, S)$.

$\text{refutes}(\_, S) :- \text{conflict}(S, T) \land \text{candidate}(T) \land > (T, S) \land \neg \text{defeater}(T)$.

6 **TR\textsuperscript{DA} discussion and related work**

Although a great number of works deal with defeasibility in logic programming, few have goals similar to ours: to lift defeasible reasoning from static logic programming to a logic for expressing knowledge base dynamics, such as TR. Such lifting opens up new applications for Transaction Logic by allowing it to take advantage of preferences among rules and defeasibility. As far as the actual chosen approach to defeasible reasoning is concerned, this work is based on [38], and extensive in-depth comparison with other works on defeasible reasoning can be found there. There, we compare LPDA based approaches to the frameworks presented by Gelfond and Son in [22] (i.e., the logic of prioritized defaults),
by Delgrande, Schaub, and Tompits in [14] (i.e., the ordered logic programs), and by Eiter et al. in [18] (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. The main difference from [22] is that our approach distills all the differences between the different default theories to the notion of an argumentation theory with a simple interface to the user-provided domain description, the predicate $\text{defeated}$. In the case of [14], the framework does not come with a unifying model-theoretic semantics, but comes as a transformation of normal logic programs under the stable model semantics. The variable part is the transformation, which encodes a fairly low-level mechanism: the order of rule applications required to generate the preferred answer set. In the following paragraphs we will focus on comparing our work with prior research on defeasibility of actions.

The main contribution here 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 [13] has a similar goal as ours, but some defeasible logic formalisms match various applications of $\mathcal{TR}$, and such, we will compare our work with 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 [33–36] and the approach by Delgrande, Schaub and Tompits in [15, 17, 16]. On another hand, the approaches adopted in [25, 23, 24] aim to apply defeasible reasoning for another application of the results presented in this report, namely, in modeling, execution and verification of workflows.

The work [36] develops a high-level language for the specification of preferences over trajectories and provides a logic programming encoding of the language based on answer set planning. They combine the action language $\mathcal{B}$ [21] with the prioritized default theory developed in [22]. $\mathcal{TR}^{DA}$ is quite different from [36] in that it is a full-fledged logic that combines both declarative and procedural elements, while [36] specifically is geared towards specifying preferences over trajectories in planning. Whereas $\mathcal{TR}^{DA}$ deals with infinite domains and allows function symbols and non-deterministic actions, the approach in [36] considers only planning with complete information on finite domains and deterministic actions. Thus, although the two approaches have common applications in the area of planning, they target different knowledge representation scenarios.

The approach in [15, 16] uses two types of preferences over plans for achieving goals in a plan. The choice order specifies when a plan satisfying a goal, $\phi_1$, is preferred over another plan satisfying another goal $\phi_2$. The temporal order specifies when the planning heuristic has a preference concerning the order in which subgoals are to be achieved. That is, when subgoals must become true in a specific order. The set of solution histories is ordered according to these partial order relations, $\leq_c$ (choice) and $\leq_t$ (temporal), and the maximal elements are chosen as the most preferred solutions. Both of these types of preference can be expressed in the $\mathcal{TR}^{DA}$ framework, although due to the difference in the
semantics the exact relationship needs further study. In the choice ordering, the application of rule definitions for actions whose effect is to update fluents in the fluent serial goal $\phi_2$ are defeated when actions that update fluents in the fluent serial goal $\phi_1$ can be executed. In the temporal ordering, application of rule definitions for actions whose effect is to update fluents in the fluent serial goal $\phi_2$ are defeated if some fluents in the fluent serial goal $\phi_1$ were not satisfied. This encoding mirrors the dualism between fluents and corresponding actions that update these fluents signaled in [16]. Moreover, while the original work by Delgrande, Schaub, and Tompits in [14] 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. In [19], a framework for planning with cost preferences is introduced. Each action is assigned a numeric cost, and plans with the minimal cost are considered to be optimal. Clearly, this work uses a completely different type of preferences and tackles a different and very specific problem in planning, which we do not address. Similar to our work, [19]’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, [23], 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, [24] extended the work of [23] 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 Applications, implementation and evaluation

We implemented an interpreter for $TR^{DA}$ in XSB ³ and tested it on a number of examples, including Example 22. The goal of these tests was to demonstrate how preferential heuristics can be expressed in $TR^{DA}$ and to evaluate their effects on the efficiency of planning (see Example 3).

³ http://xsb.sourceforge.net/
7.1 $\mathcal{T}\mathcal{R}^{DA}$ Applications in action priorities, planning and workflows

In this section, we employ several applications in order to illustrate the advantages of extending Transaction Logic with the well founded semantics and defeasible reasoning. Using the $\mathcal{GCLP}^{\mathcal{T}\mathcal{R}}$ courteous argumentation theory, the rules in these examples are more powerful than simple $\mathcal{T}\mathcal{R}$ rules since they use a relative independence in writing the rule bodies, but retain the concept of defeasible reasoning. The meaning of the $!\text{opposes}$ and $!\text{overrides}$ predicates is the same as in the Section 5.1.

**Example 2 (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.

\begin{align*}
@\text{buy}_{\text{action}}(\text{Stock}, \text{Amount}) & : - \text{recommendation}(\text{buy}, \text{Stock}) \otimes \text{owns}(\text{Stock}, \text{Qty}) \otimes \\
& \text{delete}(\text{owns}(\text{Stock}, \text{Qty})) \otimes \text{insert}(\text{owns}(\text{Stock}, \text{Qty} + \text{Amount})). \\
@\text{sell}_{\text{action}}(\text{Stock}, \text{Amount}) & : - \\
& \text{recommendation}(\text{sell}, \text{Stock}) \otimes \text{owns}(\text{Stock}, \text{Qty}) \otimes \\
& \text{delete}(\text{owns}(\text{Stock}, \text{Qty})) \otimes \text{insert}(\text{owns}(\text{Stock}, \text{Qty} - \text{Amount})). \\
!\text{opposes}(\text{sell}_{\text{action}}, \text{buy}_{\text{action}}). \\
!\text{overrides}(\text{sell}_{\text{action}}, \text{buy}_{\text{action}}). \\
\text{recommendation}(\text{buy}, \text{C}) & : - \text{services}(\text{X}). \\
\text{recommendation}(\text{sell}, \text{C}) & : - \text{media}(\text{X}). \\
\text{services}(\text{acme}). \\
\text{media}(\text{acme}). \\
\text{owns}(\text{acme}, 100). \\
\text{trade}(\text{Stock}, \text{Amount}) & : - \text{buy}(\text{Stock}, \text{Amount}). \\
\text{trade}(\text{Stock}, \text{Amount}) & : - \text{sell}(\text{Stock}, \text{Amount}).
\end{align*}

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).

Let's consider an existential goal ($\exists \text{trade}(\text{acme}, 100)$. Without the $!\text{opposes}$ and $!\text{overrides}$ 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 3 (Blocks world planning).** This example illustrates the use of defeasible reasoning for heuristic optimization of planning in the blocks world. The example is similar to the one used in Section 22, but here the rules are labeled
and additional information about the opposition and the priority between actions is used in the defeasible reasoning. The TRDA 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 can be constructed.\(^4\)

In this example, we represent the blocks world using the familiar fluents \(on(x, y)\) and \(isclear(x)\) (see Section 22), but also the new fluent \(larger(x, y)\), which says that the size of \(x\) is larger than the size of \(y\). The action \(pickup(X)\) picks up block \(X\) 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 \(move(Block, From, To)\), specified by the first rule, moves \(Block\) from its current position on top of block \(From\) to a new position on top of block \(To\), where the block \(Block\) is smaller then the block \(To\). This action is defined by combining the afore mentioned actions \(pickup\) and \(putdown\) if certain pre-conditions are satisfied. The stacking action (included later in this section) 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 \(mv\_rule\) might be applicable and several different moves might be performed. The predicate \(\text{opposes}\) stipulates that two different move actions for different block are considered to be in conflict (because only one action at a time is allowed).

\[
\begin{align*}
@mv\_rule(Block, To) & \quad move(Block, From, To) : - \\
& \quad \text{(on(Block, From) \land larger(To, Block))} \odot \\
& \quad pickup(Block, From) \odot putdown(Block, To). \\
pickup(X, Y) & \quad : - (isclear(X) \land on(X, Y)) \odot \\
& \quad \text{delete(on(X, Y))} \odot \text{insert(isclear(Y))}. \\
putdown(X, \text{table}) & \quad : - (isclear(X) \land \text{not on}(X, Z)) \\
& \quad \odot insert(on(X, \text{table})). \\
putdown(X, Y) & \quad : - (isclear(X) \land isclear(Y) \land \text{not on}(X, Z)) \\
& \quad \odot delete(isclear(Y)) \odot insert(on(X, Y)). \\
\text{opposes}(move(B1, F1, T1), move(B2, F2, T2)) & \quad : - B1 \neq B2.
\end{align*}
\]

Note that the first rule is tagged with a term, \(mv\_rule(Block, To)\) 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 on the table surface).

\[
\begin{align*}
\text{overrides}(mv\_rule(B2, To), mv\_rule(B1, To)) & \quad : - \\
& \quad larger(B2, B1) \land To \neq \text{table}.
\end{align*}
\]

\(^{4}\) For more information about planning with TR the reader is referred to [9].
Consider the configuration of blocks in (15).

\[
\begin{align*}
on(blk1, blk4). & \quad on(blk2, blk5). \\
on(blk3, table). & \quad on(blk4, table). \quad on(blk5, table). \\
isclear(blk1). & \quad isclear(blk2). \quad isclear(blk3). \\
larger(blk2, blk1). & \quad larger(blk3, blk1). \quad larger(blk3, blk2). \\
larger(blk4, blk1). & \quad larger(blk5, blk2). \quad larger(blk2, blk4).
\end{align*}
\]  

(15)

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:

\[
\text{overrides}(mv\_rule(B2, table), mv\_rule(B1, table)) : - larger(B1, B2).
\]  

(16)

In our example, this makes unstacking blk1 and moving it to the table surface preferable to unstacking blk2, since the former is a smaller block. This blocks the opportunity to then move blk4 on top of blk2 and subsequently put blk1 on top of blk4. These preference rules can be applied to a pyramid-building program like this:

\[
\begin{align*}
stack(0, Block). \\
stack(N, X) & : - N > 0 \otimes move(Y, X) \otimes stack(N - 1, Y) \otimes on(Y, X). \\
stack(N, X) & : - (N > 0 \land on(Y, X)) \otimes unstack(Y) \otimes stack(N, X). \\
unstack(X) & : - on(Y, X) \otimes unstack(Y) \otimes unstack(X). \\
unstack(X) & : - isclear(X) \land on(X, table). \\
unstack(X) & : - (isclear(X) \land on(X, Y) \land Y \neq table) \otimes move(X, table). \\
unstack(X) & : - on(Y, X) \otimes unstack(Y) \otimes unstack(X).
\end{align*}
\]  

(17)

Testing the above program on the tabled interpreter shows that the aforesaid rule preferences can significantly reduce the number of plans that need to be considered — sometimes to just one plan.

Example 4 (Workflow modeling and execution example). This example illustrates the use of defeasible reasoning for modeling business workflows. Transaction Logic have been used before for modeling concurrent workflows in [11, 10, 12]. Although, these works address a multitude of issues, including model checking for verifying workflows, integration of the data flow into the control flow by using transition conditions, sub - workflows, loops and iteration, and so on, priorities between different execution paths in the workflow haven’t been considered before, leaving the task of implementing opposition and preferences between execution branches to the programmer. Although we don’t talk in the TRDA defeasible reasoning about various aspects of transaction logic used in modeling workflows, like concurrency and constraints on the interleaved execution, in this case of non - recursive workflows, this program can be systematically transformed into a purely sequential TR program.
Let’s consider the following example of a workflow where various branches in the workflow execution oppose other branches. In this scenario depicted in Figure 1, a buy transaction is designed to make a financial transaction and a delivery of a product.

![Workflow Diagram](image)

**Fig. 1.** A transaction workflow example for defeasible reasoning in $\mathcal{T}R$

The following $\mathcal{TR}^{DA}$ program implements this simple workflow using sub-workflow actions defined in $\mathcal{T}R$. The execution process is non-deterministic and several different OR-branches of the workflow can be chosen to be executed. In this example, we represent the transaction buy as an interleaving of transactions, namely pay|delivery, where these transactions can non-deterministically choose various options: pay with credit card or with wire transfer from a bank account and deliver using express or ground mail. However, the policy of the store is that if the customer is a gold member, then the delivery is done using express mail, otherwise using ground mail, or, that a wire transfer from a bank account is preferred to a credit card payment since the payment does not require a filling period. These actions are specified below by the opposes and overrides rules, respectively. The action delivery, defined by the second and the third rules, and the action pay, defined by the forth and the fifth rules, combine sub-workflows (actions) determined by what internal conditions are satisfied.

The key observation here is that at any given point certain workflow branches are preferred over other workflow branches. For instance, a successful branch @b4 pay_cheque is preferred instead of the branch @b3 pay_credit_card although both might be applicable and several different combinations of the concurrent branches in the workflow might be performed.
buy :- pay|delivery.
@b1delivery :- gold_member ⊗ express_mail.
@b2delivery :- ground_mail.
@b3pay :- pay_credit_card.
@b4pay :- pay_cheque.
!opposes(b1, b2).
!overrides(b1, b2).
!opposes(b4, b3).
!overrides(b4, b3).
gold_member.
express_mail :- insert(delivered_express_mail).
ground_mail :- insert(delivered_ground_mail).
pay_credit_card :- credit_card_credentials ⊗ insert(credit_card_payment).
pay_cheque :- bank_account ⊗ insert(bank_payment).
credit_card_credentials.
bank_account.

7.2 \( T^{DA} \) Evaluation

Table 1 shows how the preferential heuristic of Example 3 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 an order 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, our implementation of \( T^{DA} \) 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.

We conclude the evaluation section with the extreme case where we have a world of 10,000 blocks \( \text{blk}_1, \text{blk}_2, \ldots, \text{blk}_{10,000} \) being on the table with \( \text{blk}_2 \) being larger than the block \( \text{blk}_1 \) and \( \text{blk}_3 \) being larger than both blocks \( \text{blk}_1 \) and \( \text{blk}_2 \), and so on, and an existential goal, \( \exists \text{stack}(10,000, \text{blk}_{10,000}) \) for stacking a pyramid of 9,999 blocks on the block \( \text{blk}_{10,000} \) as a base. The original tabling algorithm presented in [20] would try to try plan 9,999 different pyramids where one block \( \text{blk}_i, 1 \leq i \leq 9,999 \), 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 7.1, the new algorithm will return a single pyramid containing the blocks \( \text{blk}_2 \) to \( \text{blk}_{10,000} \) with the block \( \text{blk}_1 \) sitting separately on the table, the rule being that on top of each clear block \( \text{blk}_i \) is preferred to stack the block \( \text{blk}_{i-1} \) 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 proposes 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 [26] 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. The paper also makes a contribution directly 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 [37].

References


A Unique Least Model for not-free \( T\mathcal{R} \) Programs

In this appendix, we prove that any not-free \( T\mathcal{R} \) program has a unique least partial model (see Section 4).

Theorem 1 (Unique Least Partial Model for serial not-free \( T\mathcal{R} \) programs) If \( P \) is a not-free \( T\mathcal{R} \) program, then \( P \) has a least Herbrand model, denoted \( LPM(P) \).

Proof. Let \( P^+ \) denote the positive program obtained from \( P \) by replacing all body literals of the form \( u^\pi \), where \( \pi \) is a path, with \( t^\pi \). (We will call such literals \( u \) - literals and \( t \) - literals, respectively.) Similarly, let denote \( P^- \) 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 \( P^+ \) and \( P^- \) have unique minimal Herbrand models, since they do not have the special literals \( u^\pi \) and thus are simply serial-Horn clauses; these minimal models are 2 - valued, as shown in [9].

Let \( M^+ \) be the least model of \( P^+ \) and \( M^- \) be the least model of \( P^- \). As noted above, both of these models are 2 - valued. Clearly, \( P^- \) is a subprogram of \( P^+ \), so \( M^- \) is also a model of \( P^- \). Since \( M^- \) is the least model of \( P^- \), it follows that \( M^- \preceq M^+ \). Thus, for any path \( \pi \) and any not-free literal \( L \)

\[
M^- (\pi)(L) \leq M^+ (\pi)(L) \quad \text{and} \quad M^+ (\pi)(\text{not } L) \leq M^- (\pi)(\text{not } L)
\] (19)
Note that since $\text{Claim D.1: If}$

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

Therefore, $f$ because of $u$ subpath $\pi$

Property D.2:

Next, suppose that $C$ is a path with a split $\pi = \pi_1 \ldots \pi_n$ such that none of the $M(\pi_i)(B_i) = f$. Note that since $C$ is not in $P^-$, at least one of the $B_i$ must be $u^*$ for some subpath $\pi_i$. So, it must be the case that $M(B_1 \ldots \pi_n) = u$ (it cannot be $t$ because of $u^*$ and it cannot be $f$ because of the assumption that none of the $M(\pi_i)(B_i)$s is $f$). This implies (again by (20)) that none of the $M^+(\pi_i)(B_i)$s equals $f$. Therefore $M^+(\pi_i)(B_i) = t$ for all body literals in the corresponding clause $C^+$ in $P^+$ (one that is obtained from $C$ by changing each $u^*$ to $t^*$). Therefore, $H$ (which is the head of both $C$ and $C^+$) must be true in $M^+(\pi)$. So $M(\pi))(H)$ is either $t$ or $u$. Thus, $M(\pi)$ models every rule in $P \setminus P^-$ either and, therefore, $M$ is a model of $P$.

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

*Property D.1: $M(\pi)(L) \leq N(\pi)(L)$*

*Property D.2: $N(\pi)(\text{not } L) \leq M(\pi)(\text{not } L)$*

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

**Claim D.1:** If $M^-(\pi)(L) = t$ then $N(\pi)(L) = t$. 

### Proof of Claim D.1:

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

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

\[
\begin{align*}
-M(\pi)(L) = t & \quad \text{iff} \quad M^-(\pi)(L) = t \\
M(\pi)(\text{not } L) = f & \quad \text{iff} \quad M^-(\pi)(\text{not } L) = f \\
-M(\pi)(L) = f & \quad \text{iff} \quad M^+(\pi)(L) = f \\
M(\pi)(\text{not } L) = t & \quad \text{iff} \quad M^+(\pi)(\text{not } L) = t \\
\end{align*}
\]

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

By (19), $M$ is well-defined, since it is not possible that $M^-(\pi)(L) = t$ and $M^+(\pi)(L) = f$ or that $M^-(\pi)(\text{not } L) = f$ and $M^+(\pi)(\text{not } L) = t$.

Next, we show that $M$ is a partial model of $P$. Suppose $C$ is a rule in $P$ of the form $H : -B_1 \ldots \ldots B_n$, such that none of the $B_i$'s is a $u$-literal. By definition, $C$ belongs both to $P^-$ and $P^+$. If, for some path $\pi$, $M(\pi)(B_1 \ldots \ldots B_n) = t$, then $M^-(\pi)(B_1 \ldots \ldots B_n) = t$, by the construction of $M$ in (20). Since $M^-$ is a model of $C$, the head $H$ of $C$ must be true in $M^-(\pi)$ hence also in $M(\pi)$. Thus, $M(\pi)$ makes $C$ true. If $M(\pi)(B_1 \ldots \ldots B_n) = f$ then $M(\pi)$ satisfies $C$ trivially. If $M(\pi)(B_1 \ldots \ldots B_n) = u$, it means that, for some split $\pi = \pi_1 \ldots \ldots \pi_n$, $M(\pi_i)(B_i)$ is either $u$ or $t$. By (20), this implies $M^+(\pi_i)(L) = t$, and since $M^+$ is also a model of $C$ it follows that $H$ must be true in $M^+(\pi)$. The definition of $M$ then implies that $H$ must have the truth value $u$ or $t$ in $M(\pi)$, so $M(\pi)$ satisfies $C$ once again.

Next, suppose that $C$ is a clause $H : -B_1 \ldots \ldots B_n$ in $P \setminus P^-$, and suppose $\pi$ is a path with a split $\pi = \pi_1 \ldots \ldots \pi_n$ such that none of the $M(\pi_i)(B_i) = f$. Note that since $C$ is not in $P^-$, at least one of the $B_i$ must be $u^*$ for some subpath $\pi_i$. So, it must be the case that $M(B_1 \ldots \ldots B_n) = u$ (it cannot be $t$ because of $u^*$ and it cannot be $f$ because of the assumption that none of the $M(\pi_i)(B_i)$s is $f$). This implies (again by (20)) that none of the $M^+(\pi_i)(B_i)$s equals $f$. Therefore $M^+(\pi_i)(B_i) = t$ for all body literals in the corresponding clause $C^+$ in $P^+$ (one that is obtained from $C$ by changing each $u^*$ to $t^*$). Therefore, $H$ (which is the head of both $C$ and $C^+$) must be true in $M^+(\pi)$. So $M(\pi)(H)$ is either $t$ or $u$. Thus, $M(\pi)$ models every rule in $P \setminus P^-$ either and, therefore, $M$ is a model of $P$.
Claim D.2: If $M^{-}(\pi)(\mathit{not } L) = f$ then $N(\pi)(\mathit{not } L) = f$.

Claim D.3: If $M^{+}(\pi)(L) = t$ then $N(\pi)(L) \geq u$.

Claim D.4: If $M^{+}(\pi)(\mathit{not } L) = f$ then $N(\pi)(\mathit{not } L) \leq u$.

To establish Property D.1, suppose that $M(\pi)(L) = t$. By (20), this means that $M^{-}(\pi)(L) = t$ and, by Claim D.1, $N(\pi)(L) = t$. If $M(\pi)(L) = u$ then, by (20), this implies $M^{+}(\pi)(L) = t$ and, by Claim D.3, $N(\pi)(L) \geq u = M(\pi)(L)$. This proved Property D.1.

For Property D.2, suppose $M(\pi)(\mathit{not } L) = u$. By the definition of M, this implies $M^{+}(\pi)(\mathit{not } L) = f$ and, by Claim D.4, $N(\pi)(\mathit{not } L) \leq u = M(\pi)(\mathit{not } L)$. Similarly, if $M(\pi)(\mathit{not } L) = f$, then $M^{-}(\pi)(\mathit{not } L) = t$. By Claim D.3, $N(\pi)(\mathit{not } L) \geq u$, which implies that $N(\pi)(\mathit{not } L) \leq u$.

\[ \Box \]

B \( \mathcal{T} \mathcal{R}^{DA} \) Fixpoint and Well-founded Model

In this appendix, we prove that any \( \mathcal{T} \mathcal{R}^{DA} \) program has a unique well-founded model. For convenient reference, we reproduce the Definition 13 from Section 4 below.

**Definition 13 (\( \mathcal{T} \mathcal{R}^{DA} \) Quotient):** Let P be a set of \( \mathcal{T} \mathcal{R}^{DA} \) rules and I a path structure for P. The \( \mathcal{T} \mathcal{R}^{DA} \) quotient of P by I, written as \( \frac{P}{I} \), is defined through the following sequence of steps:

1. First, each occurrence of every \( \mathit{not} \)-literal of the form \( \mathit{not } L \) in P is replaced by \( t^{\pi} \) for every path \( \pi \) such that \( I(\pi)(\mathit{not } L) = t \) and with \( u^{\pi} \) for every path \( \pi \) such that \( I(\pi)(\mathit{not } L) = u \).
2. For each labeled rule of the form \( r : \mathit{not } L \) :- Body obtained in the previous step, replace it with the rules of the form:

\[
L :: t^{(D_t)} \otimes Body \\
L :: u^{(D_u)} \otimes Body
\]

for each database state \( D_t \) such that

\[
I((D_t))(\mathit{not } \Diamond \mathit{defeated}(\mathit{handle}(r, L))) = t
\]

and each database state \( D_u \) such that

\[
I((D_u))(\mathit{not } \Diamond \mathit{defeated}(\mathit{handle}(r, L))) = u
\]
3. Remove the labels from the remaining rules.

The resulting set of rules is the quotient $\frac{P}{I}$.

Note that, the $TR^{DA}$ quotient of a $TR^{DA}$ transaction base $P$ with respect to an argumentation theory $AT$ (denoted $(P, AT)$) for any path structure $I$, $\frac{P \cup AT}{I}$, is a negation-free $TR$ program, so, by Theorem 1, it has a unique least Herbrand model, $LPM(\frac{P \cup AT}{I})$.

We will now give the definition for the immediate consequence operator $\Gamma$. We will use the set representation of Herbrand models:

$I^+ = \{L | L \in I \text{ is a not-free literal}\}$,

$I^- = \{L | L \in I \text{ is a not-literal}\}$

and $I = I^+ \cup I^-$.

**Definition 14 ($TR^{DA}$ immediate consequence operator):**

The incremental consequence operator, $\Gamma$, for a $TR^{DA}$ transaction base $P$ with respect to the argumentation theory $AT$ takes as input a path structure $I$ and generates a new path structure as follows:

$$\Gamma(I) = def \ LPM(\frac{P \cup AT}{I})$$

Suppose $I_\emptyset$ is the path structure that maps each path $\pi$ to the empty Herbrand interpretation in which all propositions are undefined (i.e., for every path $\pi$ and every literal $L$, we have $I_\emptyset(\pi)(L) = u$).

The ordinal powers of the immediate consequence operator $\Gamma$ are defined inductively as follows:

- $\Gamma^0(I_\emptyset) = I_\emptyset$;
- $\Gamma^\alpha(I_\emptyset) = \Gamma(\Gamma^{\alpha-1}(I_\emptyset))$, for $\alpha$ a successor ordinal;
- $\Gamma^\alpha(I_\emptyset)(\pi) = \cup_{\beta < \alpha} \Gamma^{\beta}(I_\emptyset)(\pi)$, for every path $\pi$ and $\alpha$ a limit ordinal.

The following lemma states a basic result about the immediate consequence operator $\Gamma$.

**Lemma 1 ($\Gamma$ is monotonic).** The operator $\Gamma$ is monotonic with respect to the information order relation $\leq$ when $P$ and $AT$ are fixed, i.e.

$$\Gamma(I) \leq \Gamma(I') \text{ if } I \leq I'.$$

**Proof.** The proof relies on the following claim.

**Claim E.1:** if $I$ and $I'$ are two Herbrand path structures, $I \leq I'$, $D$ is a database state and $\Gamma^{|n}(I)(D))(not(\diamond defeated(handle(r, B)))) = v$ with $v \in \{f, t\}$, then $\Gamma^{|n}(I')(D))(not(\diamond defeated(handle(r, B)))) = v$.

Proof of Claim E.1:

By hypothesis, $\Gamma^{|n}(I)(D))(\diamond defeated(handle(r, B))) = v$. If $v = t$, then $\Gamma^{|n}(I)(\rho)(\diamond defeated(handle(r, B))) = f = v$ for every path $\rho$ that starts at $D$,
and, by induction hypothesis, \( \Gamma'^n(I)(\rho)(\text{defeated}(\text{handle}(r, B))) = f = \sim v \).

If \( v = f \), then \( \Gamma'^n(I)(\rho)(\text{defeated}(\text{handle}(r, B))) = t \) for some path \( \rho \) that starts at \( D \), and, by induction hypothesis, \( \Gamma'^n(I)(\rho)(\text{defeated}(\text{handle}(r, B))) = t = \sim v \). In both cases, by Definition 9, \( \Gamma'^n(I)(\rho)(\text{defeated}(\text{handle}(r, B))) = \sim v \). Hence, \( \Gamma'^n(I)(\rho)(\text{defeated}(\text{handle}(r, B))) = v \). Q.E.D.

Continuing with the proof of Lemma 1, let \( I \) and \( I' \) be two Herbrand path structures, where \( I \leq I' \). Thus, for any path \( \pi \), \( I(\pi)^+ \subseteq I'(\pi)^+ \) and \( I(\pi)^- \subseteq I'(\pi)^- \). In order to show that \( \Gamma(I) \leq \Gamma(I') \), we will prove that \( \Gamma'^n(I) \leq \Gamma'^n(I') \), for all \( n \). This is true for \( n = 0 \) (since \( I \leq I' \)).

Suppose \( \Gamma'^n(I) \leq \Gamma'^n(I') \) holds true for some \( n \), and \( \Gamma'^{n+1}(I)(\pi)(A) = t \) for some literal \( A \). There must be a clause \( @r B : -L_1 \otimes \ldots \otimes L_m \in P \) and a ground substitution \( \pi \) such that \( A = B \theta, \Gamma'^n(I)(\pi)(L_1 \otimes \ldots \otimes L_m) = t \) and \( \Gamma'^n(I)(\text{defeated}(\text{handle}(r, B \theta))) = t \), where \( D_0 \) is the initial database of \( \pi \). By Definition 9, there exists a split \( \pi = (D_0) \circ \pi_1 \circ \ldots \circ \pi_m \) such that \( \Gamma'^n(I)(\pi_i)(L_i \theta) = t \) for each \( 1 \leq i \leq m \). In \( \frac{I}{P \cup AT} \), we have rules of the form \( B : \neg t(D_1) \otimes L_1' \otimes \ldots \otimes L_m' \) and \( B : \neg u(D_n) \otimes L_1' \otimes \ldots \otimes L_m' \), where the literals \( L_i' (1 \leq i \leq m) \) denote the results of Step 1 transformation, i.e., \( L_i' \) is either \( L_i \), if \( L_i \) is a \( \neg \)-literal, or \( t^\theta \) or \( u^\theta \), for some path \( \rho \) where the Step 1 conditions in Definition 13 are satisfied. By the induction hypothesis, if \( \Gamma'^n(I)(\pi_i)(L_i \theta) = t \), then \( \Gamma'^n(I')(\pi_i)(L_i \theta) = t \). For every \( L_i \theta \) \( \neg \)-literal, we have \( L_i \theta = L_i \theta, \) so \( \Gamma'^n(I')(\pi_i)(L_i \theta) = t \). If \( L_i \theta \) is a \( \neg \)-literal, then from \( \Gamma'^n(I)(\pi_i)(L_i \theta) = t \) follows that \( L_i \theta = t \pi_i \) and, by Definition 9, \( \Gamma'^n(I')(\pi_i)(L_i \theta) = \Gamma'^n(I'(\pi_i))(t \pi_i) = t \). Since \( \pi \) can be split into \( \pi_1 \circ \ldots \circ \pi_m \), we have that \( \Gamma'^n(I)(\pi)(L_i \theta) = \Gamma'^n(I')(\pi)(L_i \theta) = t \). By Claim E.1, \( \Gamma'^n(I')(\text{defeated}(D_0)))(\text{defeated}(\text{handle}(r, B \theta))) = t \). It follows that \( \Gamma'^{n+1}(I')(\pi)(B \theta) = \Gamma'^{n+1}(I')(\pi)(A) = t \).

Suppose \( \Gamma'^{n+1}(I)(\pi)(A) = f \) for some literal \( A \). For any clause \( @r B : -L_1 \otimes \ldots \otimes L_m \in P \) such that \( A = B \theta \) for some substitution \( \theta \), \( \Gamma'^{n+1}(I)(\pi)(L_i \theta \otimes \ldots \otimes L_m \theta) = 1 \) or \( \Gamma'^{n+1}(I)(\text{defeated}(D_0))(\text{defeated}(\text{handle}(r, B \theta))) = f \) (and the rule is not present in the quotient \( \frac{I}{P \cup AT} \)). By Definition 9, for any split \( \pi = (D_0) \circ \pi_1 \circ \ldots \circ \pi_m \), we have \( \Gamma'^{n+1}(I)(\pi_i)(L_i \theta) = f \) for some \( 1 \leq i \leq m \). By induction hypothesis, if \( \Gamma'^{n+1}(I)(\pi_i)(L_i \theta) = f \), then \( \Gamma'^{n+1}(I')(\pi_i)(L_i \theta) = f \). If \( L_i \theta \) is a \( \neg \)-literal, then \( L_i' = L_i \) and \( \Gamma'^{n+1}(I')(\pi_i)(L_i \theta) = f \). Hence, \( \Gamma'^{n+1}(I')(\pi)(L_1' \otimes \ldots \otimes L_m') = f \). If \( L_i \theta \) is a \( \neg \)-literal, then the corresponding rule is not present in the quotient. On the other hand, by Claim E.1, \( \Gamma'^{n+1}(I')(\text{defeated}(D_0))(\text{defeated}(\text{handle}(r, B \theta))) = f \) and the rule is not present in the quotient corresponding to \( I' \). In all cases, it follows that \( \Gamma'^{n+1}(I')(\pi)(A) = f \). Q.E.D.

Since \( \Gamma \) is monotonic, the sequence \( \{\Gamma'^n(I_0)\} \) has a limit which is the unique least fixed point of \( \Gamma \). It is computable via transfinite induction [29, 27].
Definition 15 (Well-founded model): The well-founded model of a transaction base $P$ with respect to the argumentation theory $AT$, written as $WFM(\mathcal{P}, AT)$, is defined as the limit of the sequence $\{F^{\uparrow n}(\mathcal{I}_0)\}$. \hfill \Box

We will show that $WFM(\mathcal{P}, AT)$ is a model of $(\mathcal{P}, AT)$ by using the following lemma. This lemma states that the application of the immediate consequence operator $\Gamma$ on models of the program $(\mathcal{P}, AT)$ results in smaller models with respect to the truth order $\preceq$. The reciprocal direction is also true.

Lemma 2. : $N$ is a model of $(\mathcal{P}, AT)$ iff $\Gamma(N) \preceq N$.

Proof. The proof relies on the following claim.

Claim E.2: If a rule of the form:

$$B : - L_0 \otimes L_1 \otimes \ldots \otimes L_m,$$

in $\frac{P \cup AT}{N}$ was obtained using the quotient Definition 13 from a rule of the form:

$$@r B : - L_1 \otimes \ldots \otimes L_m$$

in $(P, AT)$, then $\min(N(\pi)(L'_1), \ldots, N(\pi)(L'_m)) = \min(N(\pi)(L_1), \ldots, N(\pi)(L_m))$ and $N((D_0)(L'_0) = N((D_0)(\text{not}(\odot defeated(handle(r, L))))$, for any path $\pi = \langle D_0 \circ \pi_1 \circ \ldots \circ \pi_m$.

Proof of Claim E.2:

If $L_i$ is a not-free literal, then $L'_i = L_i$, so $N(\rho)(L'_i) = N(\rho)(L_i)$ for every path $\rho$. If $L_i$ is a not-literal not $C_i$, then:

- if $N(\rho)(\text{not} C_i) = t$, then $L'_i = t^\rho$. Hence, $N(\rho)(L'_i) = N(\rho)(t^\rho) = t$,
- if $N(\rho)(\text{not} C_i) = u$, then $L'_i = u^\rho$. Hence, $N(\rho)(L'_i) = N(\rho)(u^\rho) = u$,
- if $N(\rho)(\text{not} C_i) = f$, then the rule in $\frac{P \cup AT}{N}$ is not created,

for any path $\rho$. Hence, in all the above cases, when the rule (21) exists, we have that $N(\rho)(L'_0) = N(\rho)(L_0)$. Therefore, $\min(N(\pi)(L'_1), \ldots, N(\pi)(L'_m)) = \min(N(\pi)(L_1), \ldots, N(\pi)(L_m))$.

By the quotient Definition 13, the literal $L'_0$ in the Rule (21) must be either the propositional constant $t$, if $N((D_0))(\text{not}(\odot defeated(handle(r, L)))) = t$, or the propositional constant $u$, if $N((D_0))(\text{not}(\odot defeated(handle(r, L)))) = u$. However, the propositional constant $t$ is true only on the path $\langle D_0 \rangle$, otherwise it is false, while the propositional constant $u$ is undefined only on the path $\langle D_0 \rangle$, otherwise it is false. Hence, $N((D_0)(L'_0) = N((D_0))(\text{not}(\odot defeated(handle(r, L))))$. Q.E.D.

We continue with the proof of Lemma 2.

$(\Rightarrow)$:
In order to show that $\Gamma(N) \subseteq N$, we have to prove that $\Gamma(N)(\pi)(A) \leq N(\pi)(A)$ for all paths $\pi$ and all literals $A$.

Since the path structure $\Gamma(N)$ is the least partial model of the program $P \cup AT \not\subseteq N$, it follows that:

- $\Gamma(N)$ satisfies every rule in $P \cup AT \not\subseteq N$, i.e., for every clause in $P \cup AT \not\subseteq N$ of the form (21) we have:
  \[ \Gamma(N)(\pi)(B) \geq \Gamma(N)(\pi)(L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m) \] (23)
  for every path $\pi$;
- $\Gamma(N)$ is minimum model, i.e., for any other model $M'$ of $P \cup AT \not\subseteq N$, we have:
  \[ \Gamma(N) \preceq M' \] (24)

By Definition 13, each rule of $P \cup AT \not\subseteq N$ must correspond to a rule in $(P, AT)$ of the form (22) where the $L'_i$ literals ($1 \leq i \leq m$) are the results of Step 1 transformation, i.e., $L'_i$ is either $L_i$ if $L_i$ is a not-free literal, or $t^\rho$ or $u^\rho$ for some path $\rho$, if $L_i$ is a not-literal. By hypothesis, $N$ models every rule in $(P, AT)$:

$$N(\pi)(B) \geq \min(N(\pi)(L_1 \otimes \ldots \otimes L_m), N((D_0))(\text{not} \land \$defeated(handle(r, B))))$$ (25)

where $D_0$ is the initial database in the path $\pi$.

Consider a split $\pi = (D_0) \circ \pi_1 \circ \ldots \circ \pi_m$. By Definition 9, $N(\pi)(L_1 \otimes \ldots \otimes L_m) = \min(N(\pi_1)(L_1), \ldots, N(\pi_m)(L_m))$. Hence, $N(\pi)(B) \geq \min(N(\pi_1)(L_1), \ldots, N(\pi_m)(L_m))$. We also have $N(\pi)(L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m) = \min(N((D_0))(L'_0), N(\pi_1)(L'_1), \ldots, N(\pi_m)(L'_m))$. Hence, by Claim E.2, $N(\pi)(L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m) = \min(N((D_0))(\text{not} \land \$defeated(handle(r, B))))$, $N(\pi_1)(L'_1), \ldots, N(\pi_m)(L'_m))$. Hence, $N(\pi)(L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m) = \min(N((D_0))(\text{not} \land \$defeated(handle(r, B))))$, $N(\pi_1)(L'_1), \ldots, N(\pi_m)(L'_m))$. Therefore, $N$ models every rule (21). It follows that $N$ is a model of $P \cup AT \not\subseteq N$, and, by (24), that $\Gamma(N) \subseteq N$.

($\Leftarrow$):

In order to show that $N$ is a model of $(P, AT)$ we have to prove that $N$ models every rule in $(P, AT)$.

By hypothesis, $N \geq LPM(P \cup AT \not\subseteq N)$. Hence, $N$ is a model of the program $P \cup AT \not\subseteq N$. Therefore, for every rule $B : - L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m$ and path $\pi$, we have $N(\pi)(B) \geq N(\pi)(L'_0 \otimes L'_1 \otimes \ldots \otimes L'_m)$. Consider a split $\pi = (D_0) \circ \pi_1 \circ \ldots \circ \pi_m$. By Definition 9, $N(\pi)(L_1 \otimes L'_1 \otimes \ldots \otimes L'_m) = \min(N((D_0))(L'_0), N(\pi_1)(L'_1), \ldots, N(\pi_m)(L'_m))$.

By Claim E.2, $N((D_0))(L'_0) = N((D_0))(\text{not} \land \$defeated(handle(r, L'_0)))$ and $\min(N(\pi)(L_1), \ldots, N(\pi)(L'_m)) = \min(N(\pi)(L'_1), \ldots, N(\pi)(L'_m))$. Hence,
\[ N(\pi)(L_1 \otimes \ldots \otimes L_m) = \min(N(\pi_1)(L'_1), \ldots, N(\pi_m)(L'_m)). \] Therefore, \[ N(\pi)(B) \geq \min(N(\pi)(L_1 \otimes \ldots \otimes L_m), N(D_0)(\text{not} \Diamond \text{defeated}(\text{handle}(r, B)))) \]. It follows that \( N \) models every rule in \( (P, AT) \). Hence \( N \) is a model of \( (P, AT) \). \qed

The following corollary states that \( \text{WFM}(P, AT) \) is a model of the program \( P \) with respect to the argumentation theory \( AT \).

**Corollary 1.** \( \text{WFM}(P, AT) \) is a model of \( (P, AT) \).

**Proof.** By Definition 15, \( \Gamma(\text{WFM}(P, AT)) = \text{WFM}(P, AT) \). Hence, by Lemma 2, it follows that \( \text{WFM}(P, AT) \) is a model. \qed

The next theorem states that our constructive computation of the least model of the program \( (P, AT) \) is correct.

**Theorem 2 (Correctness of the Constructive TR\textsuperscript{DA} Least Model)**

\( \text{WFM}(P, AT) \) is the least model of \( (P, AT) \).

**Proof.** By Corollary 1, \( \text{WFM}(P, AT) \) is a model of \( (P, AT) \). We will show by contradiction that \( \text{WFM}(P, AT) \) is the least model. Suppose that \( N \) is a model of \( (P, AT) \) such that \( N \preceq \text{WFM}(P, AT) \). Hence, \( N(\pi)(A) \preceq \text{WFM}(P, AT)(\pi)(A) \) for every path \( \pi \) and literal \( A \). We will show that we must have that \( \text{WFM}(P, AT) \preceq N \), i.e., \( \text{WFM}(P, AT)(\pi)^+ \subseteq N(\pi)^+ \) and \( N(\pi)^- \subseteq \text{WFM}(P, AT)(\pi)^- \).

The set inclusion for positive literals can be shown using the monotonicity of the immediate consequence operator \( \Gamma \) since \( I_0(\pi)^+ \subseteq N(\pi)^+ \): \( \Gamma^{i+1}(I_0)(\pi)^+ \subseteq \Gamma(N)(\pi)^+ \) after one step, and, \( \Gamma^{i+\alpha}(I_0)(\pi)^+ \subseteq \Gamma^{i+\alpha}(N)(\pi)^+ \), after \( \alpha \) steps (where \( \alpha \) is a limit ordinal), for any path \( \pi \). By Lemma 2, we also have: \( \Gamma(N) \preceq N \), \( \Gamma^{i+2}(N) \preceq \Gamma(N) \), and \( \Gamma^{i+\alpha}(N) \preceq \Gamma^{i+\alpha}(N) \) after applying \( \Gamma \) a number of \( \alpha \) times. Therefore, \( \Gamma^{i+\alpha}(I_0)(\pi)^+ \subseteq N(\pi)^+ \), for any path \( \pi \). Hence, \( \text{WFM}(P, AT)(\pi)^+ \subseteq N(\pi)^+ \).

We will now prove by contradiction that \( N(\pi)^- \subseteq \text{WFM}(P, AT)(\pi)^- \). Suppose \( N(\pi)(A) = f \) and \( \text{WFM}(P, AT)(\pi)(A) > f \), for some path \( \pi \) and a literal \( A \). For any clause \( @r B : - L_1 \otimes \ldots \otimes L_m \in (P, AT) \), ground substitution \( \theta \) such that \( A = B\theta \), and any split \( \pi = \langle D_0 \rangle \circ \pi_1 \circ \ldots \circ \pi_m \), either \( N(\pi_i)(L_i\theta) = f \) or \( N(D_0)(\text{not} \Diamond \text{defeated}(\text{handle}(r, B \theta))) = f \) for some \( i \) (where \( i \leq m \)). If \( L_i\theta \) is a not-literal not \( C \), then \( N(\pi_i)(C) = t \), and thus, \( \text{WFM}(P, AT)(\pi_i)(C) = t \), by the first part of this proof. Therefore, \( \text{WFM}(P, AT)(\pi_i)(L_i\theta) = f \). If \( L_i\theta \) is an atom, then \( N(\pi_i)(L_i\theta) = f \). We will use the following property to show that \( \text{WFM}(P, AT)(\pi_i)(L_i\theta) \) must also be false.

**Property E.1:** a set \( S \) of atom/path pairs is \( N \)-unsupported (analogous to unfounded sets in [37]) if for every pair \( L/\pi \) in \( S \), \( N(\pi)(L) = f \) and for every...
rule in \((P, AT)\) that has \(L\) in the head, \(\forall r' L : - \bigotimes L_i'\), has a split of \(\pi = \langle D_0 \rangle \circ \pi_1 \circ \ldots \circ \pi_k\), so that for some body atom \(L_i'\), the corresponding pair \(L_i'/\pi_i'\) also belongs to \(S\). It can be shown by induction that in every iteration of \(\Gamma\) all the pairs \(L/\pi\) in \(S\) are such that \(\Gamma^{\uparrow n}(\pi)(L) = f\).

By Property E.1, if \(L_i\theta\) is an atom, then \(WFM(P, AT)(\pi_i)(L_i\theta) = f\). Therefore, \(WFM(P, AT)(\pi)(L_1\theta \otimes \ldots \otimes L_m\theta) = f\). Hence, \(WFM(P, AT)(\pi)(B\theta) = f\). However, this is impossible since we started the proof by assuming \(WFM(P, AT)(\pi)(A) > f\). □

C \(\mathcal{TR}^{DA}\) Reduction to Transaction Logic

In this appendix, we prove that the well-founded model of any \(\mathcal{TR}^{DA}\) program is identical with the well-founded model of a \(\mathcal{TR}\) program obtained via a transformation of the original \(\mathcal{TR}^{DA}\) program. Suppose a \(\mathcal{TR}^{DA}\) program \((P, AT)\) where \(P\) is a set of labeled \(\mathcal{TR}^{DA}\) rules and \(AT\) is an argumentation theory.

**Theorem 3 (\(\mathcal{TR}^{DA}\) Reduction)**

\(WFM(P, AT)\) coincides with the well-founded model of the \(\mathcal{TR}\) program \(P' \cup AT\), where \(P'\) is obtained from \(P\) by changing every defeasible rule

\[
\forall r L : - \text{Body}
\]

in \(P\) to the plain rule

\[
L : - \text{not} (\Diamond \text{defeated(handle}(r, L))) \otimes \text{Body}
\]

and removing all the remaining tags.

**Proof.** We will prove that the programs resulted after each quotient operation in the transfinite sequence during the computation of the well-founded model are the same for both the original \(\mathcal{TR}^{DA}\) program \(P\) and the transformed program \(P'\).

We split the first step of the Definition 13 into two steps:

**Step 1a.** Each occurrence of every \textbf{not}-literal of the form \textbf{not} \(L\) in the bodies of the rules of \(P\), except the \textbf{defeated}/1 literals, is replaced by \(t^\pi\) for every path \(\pi\) such that \(I(\pi)(\text{not} L) = t\) and with \(u^\pi\) for every path \(\pi\) such that \(I(\pi)(\text{not} L) = u\). This step applies identically to both 26 and to 27.

**Step 1b.** The literals in 27 of the form \textbf{not} (\(\Diamond \text{defeated(handle}(r, L))\)) are replaced by \(t^\pi\) for every path \(\pi\) such that \(I(\pi)(\text{not} \Diamond \text{defeated(handle}(r, L))) = t\) and with \(u^\pi\) for every path \(\pi\) such that \(I(\pi)(\text{not} \Diamond \text{defeated(handle}(r, L))) = u\). Step 1b is precisely what Step 2 does to 26 except that instead of replacing the literals \textbf{not} (\(\Diamond \text{defeated(handle}(r, L))\)) (which 26 does not have) Step 2 simply adds the appropriate \(t^\pi\) and \(u^\pi\). □
<table>
<thead>
<tr>
<th>World size</th>
<th>No heuristics</th>
<th>With preferential heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>120</td>
<td>8</td>
</tr>
<tr>
<td>Time (sec.)</td>
<td>0.078</td>
<td>0.016</td>
</tr>
<tr>
<td>Space (kBs)</td>
<td>155</td>
<td>26</td>
</tr>
<tr>
<td>Tabled states</td>
<td>296</td>
<td>36</td>
</tr>
<tr>
<td>Transient states</td>
<td>165</td>
<td>17</td>
</tr>
<tr>
<td>State comps.</td>
<td>605</td>
<td>89</td>
</tr>
<tr>
<td>20 blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>1140</td>
<td>18</td>
</tr>
<tr>
<td>Time (sec.)</td>
<td>0.563</td>
<td>0.109</td>
</tr>
<tr>
<td>Space (kBs)</td>
<td>1162</td>
<td>60</td>
</tr>
<tr>
<td>Tabled states</td>
<td>2491</td>
<td>76</td>
</tr>
<tr>
<td>Transient states</td>
<td>1330</td>
<td>37</td>
</tr>
<tr>
<td>State comps.</td>
<td>4410</td>
<td>189</td>
</tr>
<tr>
<td>30 blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>4060</td>
<td>28</td>
</tr>
<tr>
<td>Time (sec.)</td>
<td>2.390</td>
<td>0.438</td>
</tr>
<tr>
<td>Space (kBs)</td>
<td>3730</td>
<td>90</td>
</tr>
<tr>
<td>Tabled states</td>
<td>8586</td>
<td>116</td>
</tr>
<tr>
<td>Transient states</td>
<td>4495</td>
<td>57</td>
</tr>
<tr>
<td>State comps.</td>
<td>14415</td>
<td>289</td>
</tr>
<tr>
<td>40 blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>9880</td>
<td>38</td>
</tr>
<tr>
<td>Time (sec.)</td>
<td>7.000</td>
<td>1.219</td>
</tr>
<tr>
<td>Space (kBs)</td>
<td>8562</td>
<td>120</td>
</tr>
<tr>
<td>Tabled states</td>
<td>20581</td>
<td>156</td>
</tr>
<tr>
<td>Transient states</td>
<td>10660</td>
<td>77</td>
</tr>
<tr>
<td>State comps.</td>
<td>33620</td>
<td>389</td>
</tr>
<tr>
<td>50 blocks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans</td>
<td>19600</td>
<td>48</td>
</tr>
<tr>
<td>Time (sec.)</td>
<td>17.109</td>
<td>2.938</td>
</tr>
<tr>
<td>Space (kBs)</td>
<td>16347</td>
<td>150</td>
</tr>
<tr>
<td>Tabled states</td>
<td>40476</td>
<td>196</td>
</tr>
<tr>
<td>Transient states</td>
<td>20825</td>
<td>97</td>
</tr>
<tr>
<td>State comps.</td>
<td>65025</td>
<td>489</td>
</tr>
</tbody>
</table>

Table 1. Time, space, tabled states and state comparisons for planning in the blocks world with and without preferential heuristics