Planning with Regression Analysis in Transaction Logic

Reza Basseda(✉) and Michael Kifer

Stony Brook University, Stony Brook, NY 11794, USA
{rbasseda,kifer}@cs.stonybrook.edu

Abstract. Heuristic search is arguably the most successful paradigm in Automated Planning, which greatly improves the performance of planning strategies. However, adding heuristics usually leads to very complicated planning algorithms. In order to study different properties (e.g. completeness) of those complicated planning algorithms, it is important to use an appropriate formal language and framework. In this paper, we argue that Transaction Logic is just such a specification language, which lets one formally specify both the heuristics, the planning algorithm, and also facilitates the discovery of more general and more efficient algorithms. To illustrate, we take the well-known regression analysis mechanism and show that Transaction Logic lets one specify the concept of regression analysis easily and thus express *RSTRIPS*, a classical and very complicated planning algorithm based on regression analysis. Moreover, we show that extensions to that algorithm that allow indirect effects and action ramification are obtained almost for free. Finally, a compact and clear logical formulation of the algorithm lets us prove the completeness of *RSTRIPS*—a result that, to the best of our knowledge, has not been known before.

1 Introduction

Heuristic planning is an application of heuristic search to the domain of planning. To find a plan, a heuristic planner relies on the information about the regions of the search space in which a successful plan solution is likely to be found. Planning algorithms can use such information to guide their search and thus reduce the search space that will likely be explored in order to find a solution.

Most planners apply various sophisticated domain-independent heuristics [3,15,21,25]. Other approaches [1,11,22,26,29] use declarative formalisms (e.g. situation calculus or linear temporal logic) to express heuristic information. Declarative heuristic information can be used to prune the search space [1,26] or steer the search in promising directions [13,18]. Although the declarative representation of heuristic information provides multiple advantages for specifying and generalizing heuristics, many of the above approaches do not get as much attention as the heuristic methods. There are several reasons for this:

– Declarative approaches typically propose a framework where a user can specify some heuristics but they are not flexible enough for representing

This work was supported, in part, by the NSF grant 0964196.

B. ten Cate and A. Mileo (Eds.): RR 2015, LNCS 9209, pp. 45–60, 2015.
DOI: 10.1007/978-3-319-22002-4_5

domain-independent heuristics. In contrast, non-declarative heuristic approaches are very versatile [4,21,25] and provide ways to automatically generate heuristic planners from descriptions of a planning problem.
– The declarative approaches are typically still too complex, and this often defeats their stated goal of simplifying proofs of the different properties of the planning algorithms, such as completeness and termination.

In this paper, we will argue that *Transaction Logic* (or $\mathcal{TR}$) [8–10] provides multiple advantages for specifying and generalizing planning heuristics. Specifically, we will show that sophisticated planning heuristics, such as regression analysis, can be naturally represented in $\mathcal{TR}$ and that such representation can be used to express complex planning strategies such as *RSTRIPS*.

Transaction Logic is an extension of classical logic with dedicated support for specifying and reasoning about actions, including sequential and parallel execution, atomicity of transactions, and more. To illustrate the point, we take the regression analysis heuristic and a related planning algorithm *RSTRIPS* [23,28], and show that both the heuristics and the planning algorithm naturally lend themselves to compact representation in Transaction Logic. The resulting representation opens up new possibilities and, in particular, lets us prove the completeness of *RSTRIPS*. Clearly, existing formalizations of regression analysis and goal regression [24] are too complicated to be used as formal frameworks of such proofs. In contrast, the clear and compact form used to represent *RSTRIPS* in $\mathcal{TR}$ enables us to both implement and analyze *RSTRIPS* in a declarative framework, which is also much less complicated than its implementations even in Prolog [28] (which is not declarative).

The present paper continues the line of work in [2], where we (plain) *STRIPS* was represented in $\mathcal{TR}$ declaratively, extended, and made into a complete strategy. One can similarly apply $\mathcal{TR}$ to other heuristics proposed for planning algorithms [3]. As in [2], because *RSTRIPS* is cast here as a purely logical problem in a suitable general logic, a number of otherwise non-trivial extensions become easily achievable, and we get them almost for free. In particular, *RSTRIPS* planning can be naturally extended with derived predicates defined by rules. This endows the framework with the ability to express indirect effects of actions, but the resulting planning algorithm is still complete.

This paper is organized as follows. Section 2 reviews the *STRIPS* planning framework and extends it with the concept of regression of actions. Section 3 briefly overviews Transaction Logic in order to make this paper self-contained. Section 4 shows how regression of literals through actions can be computed. Section 5 shows how $\mathcal{TR}$ can represent *RSTRIPS* planning algorithm. Section 6 concludes the paper.

2 Extended *STRIPS*-Style Planning

In this section we first remind some standard concepts in logic and then introduce the *STRIPS* planning problem extended with the concept of regression.

We assume denumerable pairwise disjoint sets of variables $\mathcal{V}$, constants $\mathcal{C}$, extensional predicate symbols $\mathcal{P}_{ext}$, and intensional predicate symbols $\mathcal{P}_{int}$. As usual, atoms are formed by applying predicate symbols to ordered lists of constants or variables. Extending the logical signature with function symbols is straightforward in this framework, but we will not do it, as this is tangential to our aims. An atom is ***extensional*** if $p \in \mathcal{P}_{ext}$ and ***intensional*** if $p \in \mathcal{P}_{int}$. A ***literal*** is either an atom P or a negated extensional atom $\neg P$. Negated intensional atoms are not allowed (but such an extension is possible). In the original *STRIPS*, all predicates were extensional, and the addition of intentional predicates to *STRIPS* is a major enhancement, which allows us to deal with the so-called *ramification problem* [14], i.e., with indirect consequences of actions. Table 1 shows the syntax of our language.

Table 1. The syntax of the language for representing *STRIPS* planning problems.

Term	$t := V \mid c$	where $V \in \mathcal{V}, c \in \mathcal{C}$
Atom	$P_\tau := p(t_1, \ldots, t_k)$	where $p \in \mathcal{P}_\tau, \tau \in \{ext, int\}$
Literal	$L := P_{int} \mid P_{ext} \mid \neg P_{ext}$	
Rule	$R := P_{int} \leftarrow L_1 \wedge \cdots \wedge L_m$	where $m \geq 0$

Extensional predicates represent database facts: they can be directly manipulated (inserted or deleted) by actions. Intensional predicate symbols are used for atomic statements defined by *rules*—they are *not* affected by actions directly. Instead, actions make extensional facts true or false and this indirectly affects the dependent intensional atoms. These indirect effects are known as action *ramifications* in the literature.

A ***fact*** is a ***ground*** (i.e., variable-free) extensional atom. A set $\mathbf{S}$ of literals is ***consistent*** if there is no atom, *atm*, such that both *atm* and $\neg atm$ are in $\mathbf{S}$.

A ***rule*** is a statement of the form $head \leftarrow body$ where *head* is an intensional atom and body is a conjunction of literals. A ***ground instance*** of a rule, R, is any rule obtained from R by a substitution of variables with constants from $\mathcal{C}$ such that different occurrences of the same variable are always substituted with the same constant. Given a set $\mathbf{S}$ of literals and a ground rule of the form $atm \leftarrow \ell_1 \wedge \cdots \wedge \ell_m$, the rule is *true* in $\mathbf{S}$ if either $atm \in \mathbf{S}$ or $\{\ell_1, \ldots, \ell_m\} \not\subseteq \mathbf{S}$. A (possibly non-ground) rule is *true* in $\mathbf{S}$ if all of its ground instances are true in $\mathbf{S}$.

Definition 1 (State). *Given a set $\mathbb{R}$ of rules, a* ***state*** *is a consistent set $\mathbf{S} = \mathbf{S}_{ext} \cup \mathbf{S}_{int}$ of literals such that*

1. *For each fact atm, either $atm \in \mathbf{S}_{ext}$ or $\neg atm \in \mathbf{S}_{ext}$.*
2. *Every rule in $\mathbb{R}$ is true in $\mathbf{S}$.* □

Definition 2 (*STRIPS* Action). *A STRIPS* ***action*** *is a triple of the form* $\alpha = \langle p_\alpha(X_1, ..., X_n), Pre_\alpha, E_\alpha \rangle$*, where*

- $p_\alpha(X_1, ..., X_n)$ *is an intensional atom in which* $X_1, ..., X_n$ *are variables and* $p_\alpha \in \mathcal{P}_{int}$ *is a predicate that is reserved to represent the action* α *and can be used for no other purpose;*
- Pre_α*, called the* ***precondition*** *of* α*, is a set that may include extensional as well as intensional literals;*
- E_α*, the* ***effect*** *of* α*, is a consistent set that may contain extensional literals only;*
- *The variables in* Pre_α *and* E_α *must occur in* $\{X_1, ..., X_n\}$*.*[1] □

Note that the literals in Pre_α can be both extensional and intensional, while the literals in E_α can be extensional only.

Definition 3 (Execution of a *STRIPS* Action). *A STRIPS action* α *is* ***executable*** *in a state* $\mathbf{S}$ *if there is a substitution* $\theta : \mathcal{V} \longrightarrow \mathcal{C}$ *such that* $\theta(Pre_\alpha) \subseteq \mathbf{S}$*. A* ***result of the execution*** *of* α *with respect to* θ *is the state, denoted* $\theta(\alpha)(\mathbf{S})$*, defined as* $(\mathbf{S} \setminus \neg\theta(E_\alpha)) \cup \theta(E_\alpha)$*, where* $\neg E = \{\neg\ell \mid \ell \in E\}$*. In other words,* $\theta(\alpha)(\mathbf{S})$ *is* $\mathbf{S}$ *with all the effects of* $\theta(\alpha)$ *applied. When* α *is ground, we simply write* $\alpha(\mathbf{S})$*.* □

Note that $\mathbf{S}'$ is well-defined since $\theta(E_\alpha)$ is unique and consistent. Observe also that, if α has variables, the result of an execution, $\mathbf{S}'$, depends on the chosen substitution θ.

The following simple example illustrates the above definition. We follow the standard logic programming convention whereby lowercase symbols represent constants and predicate symbols. The uppercase symbols denote variables that are implicitly universally quantified outside of the rules.

Example 1. Consider a world consisting of just two blocks and the action $pickup = \langle pickup(X, Y), \{clear(X)\}, \{\neg on(X, Y), clear(Y)\}\rangle$. Consider also the state $\mathbf{S} = \{clear(a), \neg clear(b), on(a, b), \neg on(b, a)\}$. Then the result of the execution of *pickup* at state $\mathbf{S}$ with respect to the substitution $\{X \rightarrow a, Y \rightarrow b\}$ is $\mathbf{S}' = \{clear(a), clear(b), \neg on(a, b), \neg on(b, a)\}$. It is also easy to see that *pickup* cannot be executed at $\mathbf{S}$ with respect to any substitution of the form $\{X \rightarrow b, Y \rightarrow ...\}$. □

Definition 4 (Planning Problem). *A* ***planning problem*** $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *consists of a set of rules* $\mathbb{R}$*, a set of STRIPS actions* $\mathbb{A}$*, a set of literals* G*, called the* ***goal*** *of the planning problem, and an* ***initial state*** $\mathbf{S}$*. A sequence of actions* $\sigma = \alpha_1, \ldots, \alpha_n$ *is a* ***planning solution*** *(or simply a* ***plan****) for the planning problem if:*

- $\alpha_1, \ldots, \alpha_n \in \mathbb{A}$*; and*

[1] Requiring all variables in Pre_α to occur in $\{X_1, ..., X_n\}$ is not essential: we can easily extend our framework and consider the extra variables to be existentially quantified.

– *there is a sequence of states* $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_n$ *such that*
  - $\mathbf{S} = \mathbf{S}_0$ *and* $G \subseteq \mathbf{S}_n$ *(i.e., G is satisfied in the final state);*
  - *for each* $0 < i \leq n$, α_i *is executable in state* $\mathbf{S}_{i-1}$ *and the result of that execution (for some substitution) is the state* $\mathbf{S}_i$.

In this case we will also say that $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_n$ *is an execution of* σ. □

Definition 5 (Non-redundant Plan). *Given a planing problem* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *and a sequence of actions* $\sigma = \alpha_1, \ldots, \alpha_n$, *we call* σ *a* ***non-redundant*** *plan for* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *if and only if:*

– σ *is a planning solution for* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$*;*
– *None of* σ*'s sub-sequences is a planning solution for the given planning problem.*

In other words, removing any action from σ *either makes the sequence non-executable at* $\mathbf{S}$ *or* G *is not satisfied after the execution.* □

In this section, we give a formal definition of the regression of literals through *STRIPS* actions. Section 4 shows how one can compute the regression of a literal through an action.

Definition 6 (Regression of a *STRIPS* Action). *Consider a STRIPS action* $\alpha = \langle p(\overline{X}), Pre, E \rangle$ *and a consistent set of fluents* L. *The* ***regression of*** L ***through*** α, *denoted* $\mathfrak{R}(\alpha, L)$,[2] *is a set of actions such that, for every* $\beta \in \mathfrak{R}(\alpha, L)$, $\beta = \langle p(\overline{X}), Pre_\beta, E \rangle$, *where* $Pre_\beta \supseteq Pre$ *is a minimal (with respect to* $\subsetneq$*) set of fluents satisfying the following condition: For every state* $\mathbf{S}$ *and substitution* θ *such that* $\theta(\alpha)(\mathbf{S})$ *exists, if* $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$, *then* $\theta(\alpha)(\mathbf{S}) \models \theta(L)$. *In other words,* β *has the same effects as* α, *but its precondition is more restrictive and it preserves (does not destroy) the set of literals* L.

Each action in $\mathfrak{R}(\alpha, L)$ *will also be called a regression of* L *via* α. □

The *minimal set of fluents* in this definition is, as noted, with respect to subset, i.e., there is no action $\beta' = \langle p(\overline{X}), Pre_{\beta'}, E \rangle$ such that $Pre_{\beta'} \subsetneq Pre_\beta$ and β' satisfies the conditions of Definition 6.

Consider $\beta \in \mathfrak{R}(\alpha, L)$ and let $\check{\beta} = \langle p(\overline{X}), Pre_\beta \cup L, E \rangle$. We will call $\check{\beta}$ a **restricted** regression of L through α and denote the set of such actions by $\check{\mathfrak{R}}(\alpha, L)$. We will mostly use the restricted regressions of actions in the representation of *RSTRIPS* planning algorithm.

3 Overview of Transaction Logic

To make this paper self-contained, we provide a brief introduction to the relevant *subset* of $\mathcal{TR}$ [5,7–10] needed for the understanding of this paper.

[2] We simply write $\mathfrak{R}(\alpha, \ell)$ whenever L just contains a single literal ℓ.

As an extension of first-order predicate calculus, $\mathcal{TR}$ shares much of its syntax with that calculus. One of the new connectives that $\mathcal{TR}$ adds to the calculus is the ***serial conjunction***, denoted $\otimes$. It is binary associative, and non-commutative. The formula $\phi \otimes \psi$ represents a composite action of *execution* of ϕ followed by an execution of ψ. When ϕ and ψ are regular first-order formulas, $\phi \otimes \psi$ reduces to the usual first-order conjunction, $\phi \wedge \psi$. The logic also introduces other connectives to support hypothetical reasoning, concurrent execution, etc., but these are not going to be used here.

To take the *frame problem* out of many considerations in $\mathcal{TR}$, it has an extensible mechanism of ***elementary updates*** (see [6,7,9,10]). Due to the definition of *STRIPS* actions, we just need the following two types of elementary updates (actions): $+p(t_1, \ldots, t_n)$ and $-p(t_1, \ldots, t_n)$, where $p(t_1, \ldots, t_n)$ denotes an *extensional* atom. Given a state $\mathbf{S}$ and a *ground* elementary action $+p(a_1, \ldots, a_n)$, an execution of $+p(a_1, \ldots, a_n)$ at state $\mathbf{S}$ deletes the literal $\neg p(a_1, \ldots, a_n)$ and adds the literal $p(a_1, \ldots, a_n)$. Similarly, executing $-p(a_1, \ldots, a_n)$ results in a state that is exactly like $\mathbf{S}$, but $p(a_1, \ldots, a_n)$ is deleted and $\neg p(a_1, \ldots, a_n)$ is added. If $p(a_1, \ldots, a_n) \in \mathbf{S}$, the action $+p(a_1, \ldots, a_n)$ has no effect, and similarly for $-p(a_1, \ldots, a_n)$.

We define ***complex actions*** using ***serial rules***, which are statements of the form

$$h \leftarrow b_1 \otimes b_2 \otimes \ldots \otimes b_n. \tag{1}$$

where h is an atomic formula denoting the complex action and b_1, ..., b_n are literals or elementary actions. This means that h is a complex action and one way to execute h is to execute b_1 then b_2, etc., and finally to execute b_n. Note that we have regular first-order as well as serial-Horn rules. For simplicity, we assume that the sets of intentional predicates that can appear in the heads of regular rules and those in the heads of serial rules are disjoint. *Extensional atoms* and *intentional atoms* that can appear in the states (see Definition 1) will be called ***fluents***. Note that a serial rule all of whose body literals are fluents is essentially a regular rule, since all the $\otimes$-connectives can be replaced with $\wedge$. Therefore, one can view the regular rules as a special case of serial rules.

The following example illustrates the above concepts where we continue to use the standard logic programming convention regarding capitalization of variables, which are assumed to be universally quantified outside of the rules. It is common practice to omit quantifiers.

$$\begin{aligned}
move(X,Y) \;&\leftarrow (on(X,Z) \wedge clear(X) \\
&\qquad \wedge clear(Y) \wedge \neg tooHeavy(X)) \otimes \\
&\quad -on(X,Z) \otimes +on(X,Y) \otimes \\
&\quad -clear(Y). \\
tooHeavy(X) \;&\leftarrow weight(X,W) \wedge limit(L) \wedge \\
&\qquad W < L. \\
?- \;\; &move(blk1, blk15) \otimes move(SomeBlk, blk1).
\end{aligned}$$

Here *on*, *clear*, *tooHeavy*, *weight*, etc., are fluents and *move* is an action. The predicate *tooHeavy* is an intentional fluent, while *on*, *clear*, and *weight* are extensional

fluents. The actions $+on(...)$, $-clear(...)$, and $-on(...)$ are elementary and the intentional predicate *move* is a complex action. This example illustrates several features of Transaction Logic. The first rule is a serial rule defining a complex action of moving a block from one place to another. The second rule defines the intensional fluent *tooHeavy*, which is used in the definition of *move* (under the scope of default negation). As the second rule does not include any action, it is a regular rule.

The last statement above is a *request to execute* a composite action, which is analogous to a query in logic programming. The request is to move block *blk1* from its current position to the top of *blk15* and then find some other block and move it on top of *blk1*. Traditional logic programming offers no logical semantics for updates, so if after placing *blk1* on top of *blk15* the second operation $(move(SomeBlk, blk1))$ fails (say, all available blocks are too heavy), the effects of the first operation will persist and the underlying database becomes corrupted. In contrast, Transaction Logic gives update operators a logical semantics of an *atomic database transaction*. This means that if any part of the transaction fails, the effect is as if nothing was done at all. For example, if the second action in our example fails, all actions are "backtracked over" and the underlying database state remains unchanged.

$\mathcal{TR}$'s semantics is given in purely model-theoretic terms and here we will just give an informal overview. The truth of any action in $\mathcal{TR}$ is determined over sequences of states—***execution paths***—which makes it possible to think of truth assignments in $\mathcal{TR}$'s models as executions. If an action, ψ, defined by a set of serial rules, $\mathbb{P}$, evaluates to true over a sequence of states $\mathbf{D}_0, \ldots, \mathbf{D}_n$, we say that it can *execute* at state $\mathbf{D}_0$ by passing through the states $\mathbf{D}_1$, ..., $\mathbf{D}_{n-1}$, ending in the final state $\mathbf{D}_n$. This is captured by the notion of ***executional entailment***, which is written as follows:

$$\mathbb{P}, \mathbf{D}_0 \ldots \mathbf{D}_n \models \psi \tag{2}$$

Due to lack of space, we put more examples about $\mathcal{TR}$ in the full report.[3]

Various inference systems for serial-Horn $\mathcal{TR}$ [7] are similar to the well-known SLD resolution proof strategy for Horn clauses plus some $\mathcal{TR}$-specific inference rules and axioms. Given a set of serial rules, $\mathbb{P}$, and a *serial goal*, ψ (i.e., a formula that has the form of a body of a serial rule such as (1), these inference systems prove statements of the form $\mathbb{P}, \mathbf{D} \cdots \vdash \psi$, called ***sequents***. A proof of a sequent of this form is interpreted as a proof that action ψ defined by the rules in $\mathbb{P}$ can be successfully executed starting at state $\mathbf{D}$.

An inference succeeds iff it finds an execution for the transaction ψ. The execution is a sequence of database states $\mathbf{D}_1, \ldots, \mathbf{D}_n$ such that $\mathbb{P}, \mathbf{D}\, \mathbf{D}_1 \ldots \mathbf{D}_n \vdash \psi$. We will use the following inference system in our planning application. For simplicity, we present only a version for ground facts and rules. The inference rules can be read either top-to-bottom (if *top* is proved then *bottom* is proved) or bottom-to-top (to prove *bottom* one needs to prove *top*).

[3] http://ewl.cewit.stonybrook.edu/planning/RSTRIPS-TR-full.pdf.

Definition 7 ($\mathcal{TR}$ Inference System). *Let $\mathbb{P}$ be a set of rules (serial or regular) and $\mathbf{D}$, $\mathbf{D}_1$, $\mathbf{D}_2$ denote states.*

- *Axiom: $\mathbb{P}, \mathbf{D} \cdots \vdash ()$, where $()$ is an empty clause (which is true at every state).*
- *Inference Rules*
  1. *Applying transaction definition: Suppose $t \leftarrow body$ is a rule in $\mathbb{P}$.*

$$\frac{\mathbb{P}, \mathbf{D} \cdots \vdash body \otimes rest}{\mathbb{P}, \mathbf{D} \cdots \vdash t \otimes rest} \tag{3}$$

  2. *Querying the database: If $\mathbf{D} \models t$ then*

$$\frac{\mathbb{P}, \mathbf{D} \cdots \vdash rest}{\mathbb{P}, \mathbf{D} \cdots \vdash t \otimes rest} \tag{4}$$

  3. *Performing elementary updates: If the elementary update t changes the state $\mathbf{D}_1$ into the state $\mathbf{D}_2$ then*

$$\frac{\mathbb{P}, \mathbf{D}_2 \cdots \vdash rest}{\mathbb{P}, \mathbf{D}_1 \cdots \vdash t \otimes rest} \tag{5}$$

A ***proof*** of a sequent, seq_n, is a series of sequents, seq_1, seq_2, ... , seq_{n-1}, seq_n, where each seq_i is either an axiom-sequent or is derived from earlier sequents by one of the above inference rules. This inference system has been proven sound and complete with respect to the model theory of $\mathcal{TR}$ [7]. This means that if ϕ is a serial goal, the executional entailment $\mathbb{P}, \mathbf{D}_0 \mathbf{D}_1 \ldots \mathbf{D}_n \models \phi$ holds if and only if there is a proof of $\mathbb{P}, \mathbf{D}_0 \cdots \vdash \phi$ over the execution path $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n$. In this case, we will also say that such a proof derives $\mathbb{P}, \mathbf{D}_0 \mathbf{D}_1 \ldots \mathbf{D}_n \vdash \phi$.

4 Computation of Regression

In this section, we briefly explain how the previously introduced regression of actions (Definition 6) can be computed. This computation is a key component of the *RSTRIPS* planning algorithm.

In the following we will be using an *identity* operator, $=$, which will be treated as an immutable extensional predicate, i.e., a predicate defined by a non-changeable set of facts: For any state $\mathbf{S}$ and a pair of constants or ground fluents ℓ and ℓ', $(\ell = \ell') \in \mathbf{S}$ if and only if ℓ and ℓ are identical. Similarly, *non-identity* is defined as follows: $\ell \neq \ell' \in \mathbf{S}$ if and only if ℓ, ℓ' are distinct.

To illustrate regression, consider a *STRIPS* action $copy = \langle copy(Src, Dest, V), \{value(Src, V)\}, \{\neg value(Dest, V'), value(Dest, V)\}\rangle$ from the *Register Exchange* example in [2]. For convenience, this example is also found in the full report[4] along with *RSTRIPS* planning rules. Here, $copy \in$

[4] http://ewl.cewit.stonybrook.edu/planning/RSTRIPS-TR-full.pdf.

$\mathfrak{R}(copy, value(Src, V))$ since for every state $\mathbf{S}$ and substitution θ such that $\theta(copy)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(value(Src, V))$, then $\theta(\alpha)(\mathbf{S}) \models \theta(value(Src, V))$. This example is a special case of the following property of regression, which directly follows from the definitions: if ℓ is an extensional literal and $\alpha = \langle p(\overline{X}), Pre, E\rangle$ is a *STRIPS* action,

- $\mathfrak{R}(\alpha, \ell) = \emptyset$ if and only if, for every ground substitution, $\neg\theta(\ell) \in \theta(E)$.
- $\mathfrak{R}(\alpha, \ell) = \{\alpha\}$ if and only if, for every ground substitution $\neg\theta(\ell) \notin \theta(E)$.

The following proposition and lemmas present a method to compute regression. The method is complete for extensional literals; for intentional literals, it yields some, but not always all, regressions.

Proposition 1 (Regression of Sets of Literals). *Given a set of literals $L = L_1 \cup L_2$ and a STRIPS action $\alpha = \langle p(\overline{X}), Pre_\alpha, E_\alpha\rangle$, let $\beta_1 \in \mathfrak{R}(\alpha, L_1)$ and $\beta_2 \in \mathfrak{R}(\alpha, L_2)$, where $\beta_1 = \langle p(\overline{X}), Pre_{\beta_1}, E_\alpha\rangle$ and $\beta_2 = \langle p(\overline{X}), Pre_{\beta_2}, E_\alpha\rangle$. There is some $\beta = \langle p(\overline{X}), Pre_\beta, E_\alpha\rangle$ such that $Pre_\beta \subseteq Pre_{\beta_1} \cup Pre_{\beta_2}$ and $\beta \in \mathfrak{R}(\alpha, L)$.*

Proof. From the assumptions, it follows that for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_{\beta_1} \cup Pre_{\beta_2}) \wedge \theta(L)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(L)$.

To find a minimal subset of $Pre_{\beta_1} \cup Pre_{\beta_2}$ satisfying the regression property, one can repeatedly remove elements from $Pre_{\beta_1} \cup Pre_{\beta_2}$ and check if the regression property still holds. When no removable elements remain, we get a desired set Pre_β. □

Lemma 1. (Regression of Extensional Literals). *Consider an extensional literal ℓ and a STRIPS action $\alpha = \langle p(\overline{X}), Pre, E\rangle$ where α and ℓ do not share variables. Let $Pre_\beta = Pre \cup \{\ \ell \neq e \mid \neg e \in E \ \wedge\ \exists\theta \text{ s.t. } \theta(e) = \theta(\ell)\ \}$. Then $\beta \in \mathfrak{R}(\alpha, L)$, where $\beta = \langle p(\overline{X}), Pre_\beta, E\rangle$.*

Proof. Let $\mathbf{S}$ be a state and there is θ such that $\theta(\alpha)(\mathbf{S})$ exists. Clearly, if $\mathbf{S} \models \theta(Pre_\beta)$, there is no $\neg e \in E$ such that $\theta(e) = \theta(\ell)$. Therefore, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \ell$.

We need to show that Pre_β is a minimal set of literals satisfying the above property. Assume, to the contrary, that there is some $Pre_{\beta'}$, $Pre \subseteq Pre_{\beta'} \subsetneqq Pre_\beta$, such that for every state $\mathbf{S}$ and substitution θ, if $\theta(\alpha)(\mathbf{S})$ exists and $\mathbf{S} \models \theta(Pre_{\beta'}) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Since $Pre_{\beta'} \subset Pre_\beta$, there must be $(\ell \neq e) \in Pre_\beta \setminus Pre_{\beta'}$. Let θ_1 be a substitution such that $\theta_1(e) = \theta_1(\ell)$. In that case for every $\mathbf{S}$ such that $\theta_1(\alpha)(\mathbf{S})$ exists, $\mathbf{S} \models \theta_1(Pre_{\beta'}) \wedge \theta_1(\ell)$ but $\theta_1(\alpha)(\mathbf{S}) \not\models \theta(\ell)$, since $\theta_1(\neg\ell) = \theta(\neg e) \in \theta_1(E)$. This contradicts the assumption that $\theta_1(\ell) \in \mathbf{S}$. Thus Pre_β is a minimal set of fluents satisfying the regression condition for ℓ, so $\beta \in \mathfrak{R}(\alpha, \ell)$. □

To illustrate the lemma, consider an extensional literal $value(R, V'')$ and the *STRIPS* action $copy = \langle copy(S, D, V), Pre_{copy}, E_{cppy}\rangle$, where $Pre_{copy} = \{value(S, V)\}$ and $E_{copy} = \{\neg value(D, V'), value(D, V)\}$. Then $\beta \in \mathfrak{R}(copy, value(R, V''))$, where $\beta = \langle copy_\beta(S, D, V), Pre_\beta, E_{copy}\rangle$ and $Pre_\beta = Pre_{copy} \cup \{value(R, V'') \neq value(D, V')\}$.

Lemma 2 (Regression of Intensional Literals). *Consider a set of rules* $\mathbb{R}$*, an intensional literal* ℓ*, and a STRIPS action* $\alpha = \langle p(\overline{X}), Pre, E\rangle$*, where* α *and* ℓ *do not share variables. Let* L *be a minimal set of extensional literals such that* $\mathbb{R} \cup L \cup \{\leftarrow \ell\}$ *has an SLD-refutation [19]. Then for every* $\beta \in \mathfrak{R}(\alpha, L)$ *of the form* $\beta = \langle p(\overline{X}), Pre_\beta, E\rangle$*, there is* $L_\beta \subseteq Pre_\beta \cup L$ *such that* $\langle p(\overline{X}), L_\beta, E\rangle \in \mathfrak{R}(\alpha, \ell)$*.*

Proof. By Definition 6, for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(L)$. Due to the soundness of SLD-refutation [19], if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$ then $\mathbf{S} \models \theta(\ell)$; and if $\theta(\alpha)(\mathbf{S}) \models \theta(L)$ then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Therefore, for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Therefore, $Pre_\beta \cup L$ satisfies the conditions for regressing ℓ through α except, possibly, minimality. To get the minimality, we can start removing elements from this set, as in Proposition 1, until a minimal set is reached. □

Definition 8. (Regression Deterministic Action). *A STRIPS action* $\alpha = \langle p(\overline{X}), Pre_\alpha, E\rangle$ *is called* ***regression-deterministic*** *if for every set of literals* L*, one of the following holds:*

- *There exists* $\beta \in \mathfrak{R}(\alpha, L)$ *such that* $Pre_\beta \setminus Pre_\alpha$ *is a set of literals of the form* $\ell = e$ *or* $\ell \neq e$*.*
- $\mathfrak{R}(\alpha, L) = \emptyset$*.*

Similarly, a set of actions $\mathbb{A}$ *is* ***regression-deterministic*** *if all of its actions are regression-deterministic.* □

Clearly, if a set of actions $\mathbb{A}$ is regression-deterministic, one can find an action $\beta \in \mathfrak{R}(\alpha, L)$ for every $\alpha \in \mathbb{A}$ and set of literals L using Lemmas 1 and 2. From now on, we assume that the set of actions $\mathbb{A}$ is regression-deterministic and is closed under regression and restricted regression.

5 The *RSTRIPS* Planner

The idea of using $\mathcal{TR}$ as a planning formalism and an encoding of *STRIPS* as a set of $\mathcal{TR}$ rules first appeared informally in the unpublished report [7]. The encoding did not include ramification and intensional predicates. Based on that encoding, we proposed a *non-linear* and *complete* planning algorithm in [2]. In this paper, we extend the original method with regression analysis and use $\mathcal{TR}$ to represent the *RSTRIPS* planning algorithm. This also generalizes the original *RSTRIPS* with intentional predicates and we prove the completeness of the resulting planner. To the best of our knowledge, completeness of *RSTRIPS* has not been proven before.

Regression analysis of literals, as a search heuristic for planners, can be used to improve the performance of planning strategies. The idea behind planning

with regression is that the already achieved goals should be protected so that subsequent actions of the planner would not "unachieve" those goals [27].

To keep our encoding simple, we assume a built-in 3-ary predicate *regress* such that, for any state $\mathbf{S}$, $regress(L, p_\alpha(\overline{X}), p_{\alpha'}(\overline{X}))$ is true (on any path) if and only if $\alpha' \in (\mathfrak{R}(\alpha, L) \cup \check{\mathfrak{R}}(\alpha, L) \cup \{\alpha\})$. During plan construction, *RSTRIPS* may consider a subgoal that cannot be achieved without unachieving an already achieved goal [28]. Instead of checking for unachieved goals *after* performing actions (and undoing these actions when an unachieved goal is found), *RSTRIPS* verifies that no unachieving will take place *before* performing each action by modifying action preconditions using regression.

Definition 9 (Enforcement Operator). *Let G be a set of extensional literals. We define $Enf(G) = \{+p \mid p \in G\} \cup \{-p \mid \neg p \in G\}$. In other words, $Enf(G)$ is the set of elementary updates that makes G true.* □

Next we introduce a natural correspondence between *STRIPS* actions and $\mathcal{TR}$ rules.

Definition 10 (Actions as $\mathcal{TR}$ Rules). *Let $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ be a STRIPS action. We define its* ***corresponding $\mathcal{TR}$ rule****, $tr(\alpha)$, to be a rule of the form*

$$p_\alpha(\overline{X}) \leftarrow (\wedge_{\ell \in Pre_\alpha} \ell) \otimes (\otimes_{u \in Enf(E_\alpha)} u). \tag{6}$$

Note that in (6) the actual order of action execution in the last component, $\otimes_{u \in Enf(E_\alpha)} u$, is immaterial, since all such executions happen to lead to the same state.

We now define a set of $\mathcal{TR}$ clauses that simulate the well-known *RSTRIPS* planning algorithm and extend this algorithm to handle intentional predicates and rules. The reader familiar with the *RSTRIPS* planner may notice that these rules are essentially a natural, more concise, and more general verbalization of the classical *RSTRIPS* algorithm [12]. These rules constitute a *complete* planner when evaluated with the $\mathcal{TR}$ proof theory.

Definition 11 ($\mathcal{TR}$ Planning Rules with Regression Analysis). *Let $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ be a STRIPS planning problem (see Definition 4). We define a set of $\mathcal{TR}$ rules, $\mathbb{P}^r(\Pi)$, which simulates the RSTRIPS planning algorithm. $\mathbb{P}^r(\Pi)$ has three disjoint parts: $\mathbb{P}^r_{\mathbb{R}}$, $\mathbb{P}^r_{\mathbb{A}}$, and $\mathbb{P}^r_G$, as described below.*

– *The $\mathbb{P}^r_{\mathbb{R}}$ part: for each rule $p(\overline{X}) \leftarrow p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$ in $\mathbb{R}$, $\mathbb{P}^r_{\mathbb{R}}$ is a set of rules of the following form—one rule per permutation $\langle i_1, \ldots, i_n \rangle$:*

$$\begin{aligned} achieve^r_p(\overline{X}, L) \leftarrow\ & achieve^r_{p_{i_1}}(\overline{X}_{i_1}, L) \otimes \\ & achieve^r_{p_{i_2}}(\overline{X}_{i_2}, L \cup \{p_{i_1}(\overline{X}_{i_1})\}) \otimes \ldots \otimes \\ & achieve^r_{p_{i_n}}(\overline{X}_{i_n}, L \cup \{p_{i_1}(\overline{X}_{i_1}), \ldots, p_{i_{n-1}}(\overline{X}_{i_{n-1}})\}) \end{aligned} \tag{P1}$$

Rule (P1) extends the classical RSTRIPS setting with intentional predicates and ramification of actions.

– *The part* $\mathbb{P}^r_{\mathbb{A}} = \mathbb{P}^r_{actions} \cup \mathbb{P}^r_{atoms} \cup \mathbb{P}^r_{achieves}$ *is constructed out of the actions in* $\mathbb{A}$ *as follows:*

- $\mathbb{P}^r_{actions}$*: for each* $\alpha \in \mathbb{A}$, $\mathbb{P}^r_{actions}$ *has a rule of the form*

$$p_\alpha(\overline{X}) \leftarrow (\wedge_{\ell \in Pre_\alpha} \ell) \ \otimes \ (\otimes_{u \in Enf(E_\alpha)} u). \tag{P2}$$

This is the $\mathcal{TR}$ *rule that corresponds to the action* α*, introduced in Definition 10.*

- $\mathbb{P}^r_{atoms} = \mathbb{P}^r_{achieved} \cup \mathbb{P}^r_{enforced}$ *has two disjoint parts as follows:*
  – $\mathbb{P}^r_{achieved}$*: for each extensional predicate* $p \in \mathcal{P}_{ext}$, $\mathbb{P}^r_{achieved}$ *has the rules*

$$\begin{array}{l} achieve^r_p(\overline{X}, L) \leftarrow p(\overline{X}). \\ achieve^r_{not\_p}(\overline{X}, L) \leftarrow \neg p(\overline{X}). \end{array} \tag{P3}$$

These rules say that if an extensional literal is true in a state then that literal has already been achieved as a goal.
– $\mathbb{P}^r_{enforced}$*: for each action* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ *in* $\mathbb{A}$ *and each* $e(\overline{Y}) \in E_\alpha$, $\mathbb{P}^r_{enforced}$ *has the following rule:*

$$\begin{array}{l} achieve^r_e(\overline{Y}, L) \leftarrow regress(L, p_\alpha(\overline{X}), p_{\alpha'}(\overline{X})) \\ \qquad\qquad\qquad\qquad \otimes \ execute^r_{p_{\alpha'}}(\overline{X}, L). \end{array} \tag{P4}$$

This rule says that one way to achieve a goal that occurs in the effects of an action is to execute that action after regressing the "protected" literals L *through that action.*

- $\mathbb{P}^r_{achieves}$*: for each action* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ *in* $\mathbb{A}$ *where* $Pre_\alpha = \{p_1(\overline{X}_1), \ldots, p_n(\overline{X}_n)\}$, $\mathbb{P}^r_{achieves}$ *is a set of rules of the following form, one per permutation* $\langle i_1, \ldots, i_n \rangle$*:*

$$\begin{array}{l} execute^r_{p_\alpha}(\overline{X}, L) \leftarrow achieve^r_{p_{i_1}}(\overline{X}_{i_1}, L) \ \otimes s \\ \qquad\qquad achieve^r_{p_{i_2}}(\overline{X}_{i_2}, L \cup \{p_{i_1}(\overline{X}_{i_1})\}) \otimes \ldots \ \otimes \\ \qquad\qquad achieve^r_{p_{i_n}}(\overline{X}_{i_n}, L \cup \{p_{i_1}(\overline{X}_{i_1}), \ldots, p_{i_{n-1}}(\overline{X}_{i_{n-1}})\}) \\ \qquad\qquad \otimes \ p_\alpha(\overline{X}). \end{array} \tag{P5}$$

This means that to execute an action, one must first achieve the precondition of the action while making sure to not unachieve the already achieved parts of the precondition.

– $\mathbb{P}^r_G$*: Let* $G = \{g_1, ..., g_k\}$*. Then* $\mathbb{P}^r_G$ *is a set of the following rules, one per permutation* $\langle i_1, \ldots, i_n \rangle$*:*

$$\begin{array}{l} achieve_G \leftarrow achieve^r_{g_{i_1}}(L) \\ \qquad\qquad \otimes \ achieve^r_{g_{i_2}}(L \cup \{g_{i_1}\}) \otimes \ldots \\ \qquad\qquad \otimes \ achieve^r_{g_{i_n}}(L \cup \{g_{i_1}, \ldots, g_{i_{n-1}}\}). \end{array} \tag{P6}$$

Due to space limitation, we cannot include an example of $\mathcal{TR}$-based *RSTRIPS* planning here. Instead, we refer the reader to Example 3 in the full report.[5]

Given a set $\mathbb{R}$ of rules, a set $\mathbb{A}$ of *STRIPS* actions, an initial state $\mathbf{S}$, and a goal G, Definition 11 gives a set of $\mathcal{TR}$ rules that specify a planning strategy for that problem. To find a solution for that planning problem, one simply needs to place the request

$$? - achieve_G\,. \tag{7}$$

at a desired initial state and use the $\mathcal{TR}$'s inference system of Sect. 3 to find a proof. The inference system in question is sound and complete for *serial clauses* [5,7,9], and the rules in Definition 11 satisfy that requirement.

As mentioned before, a solution plan for a *STRIPS* planning problem is a sequence of actions leading to a state that satisfies the planning goal. Such a sequence can be extracted by picking out the atoms of the form p_α from a successful derivation branch generated by the $\mathcal{TR}$ inference system. Since each p_α uniquely corresponds to a *STRIPS* action, this provides us with the requisite sequence of actions that forms a plan.

Suppose $seq_0, \ldots, seq_m$ is a deduction by the $\mathcal{TR}$ inference system. Let $i_1, \ldots, i_n$ be exactly those indexes in that deduction where the inference rule (3) was applied to some sequent using a rule of the form $tr(\alpha_{i_r})$ introduced in Definition 10. We will call $\alpha_{i_1}, \ldots, \alpha_{i_n}$ the ***pivoting sequence of actions***. The corresponding ***pivoting sequence of states*** $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$ is a sequence where each $\mathbf{D}_{i_r}$, $1 \le r \le n$, is the state at which α_{i_r} is applied. We will prove that the pivoting sequence of actions is a solution to the planning problem. The proofs are found in the full report.

Theorem 1 (Soundness of $\mathcal{TR}$ Planning Solutions). *Consider a STRIPS planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ and let $\mathbb{P}^r(\Pi)$ be the corresponding set of $\mathcal{TR}$ rules, as in Definition 11. Then any pivoting sequence of actions in the derivation of the sequent $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$ is a solution plan.*

Completeness of a planning strategy means that, for any *STRIPS* planning problem, if there is a solution, the planner will find *at least one* plan.

Theorem 2 (Completeness of $\mathcal{TR}$ Planning). *Given a STRIPS planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$, let $\mathbb{P}^r(\Pi)$ be the corresponding set of $\mathcal{TR}$ rules as in Definition 11. If there is a plan for Π then $\mathcal{TR}$ inference system will fins some plan using the rules $\mathbb{P}^r(\Pi)$, as described in Definition 11.*

Theorem 3 (Decidability). *$\mathcal{TR}$-based planners for RSTRIPS always terminates.*

6 Conclusion

This paper has demonstrated that the use of Transaction Logic accrues significant benefits in the area of planning. As an illustration, we have shown that

[5] http://ewl.cewit.stonybrook.edu/planning/RSTRIPS-TR-full.pdf.

sophisticated planning heuristics and algorithms, such as regression analysis and *RSTRIPS*, can be naturally represented in $\mathcal{TR}$ and that the use of this powerful logic opens up new possibilities for generalizations and proving properties of planning algorithms. For instance, just by using this logic, we were able to extend *RSTRIPS* with action ramification almost for free. Furthermore, benefiting from the proof theory, we were able to establish the completeness and termination of the resulting strategy. In the full report,[6] we also present our experiments that show that *RSTRIPS* can be orders of magnitude better than *STRIPS* both in time and space. These non-trivial insights were acquired merely due to the use of $\mathcal{TR}$ and not much else. The same technique can be used to cast even more advanced strategies such as GraphPlan, *ABSTRIPS* [23], and HTN [20] as $\mathcal{TR}$ rules.

There are several promising directions to continue this work. One is to investigate other planning strategies and, hopefully, accrue similar benefits. Other possible directions include non-linear plans and plans with loops [16,17,30]. For instance non-linear plans could be represented using Concurrent Transaction Logic [8], while loops are easily representable using recursive actions in $\mathcal{TR}$.

Acknowledgments. We are thankful to the anonymous referees for their thorough reviews and suggestions.

References

1. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artif. Intell. **116**(12), 123–191 (2000). http://www.sciencedirect.com/science/article/pii/S0004370299000715
2. Basseda, R., Kifer, M., Bonner, A.J.: Planning with transaction logic. In: Kontchakov, R., Mugnier, M.-L. (eds.) RR 2014. LNCS, vol. 8741, pp. 29–44. Springer, Heidelberg (2014)
3. Bonet, B., van den Briel, M.: Flow-based heuristics for optimal planning: landmarks and merges. In: Chien, S., Do, M.B., Fern, A., Ruml, W. (eds.) Proceedings of the Twenty Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014. AAAI, Portsmouth (2014). http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7933
4. Bonet, B., Geffner, H.: Planning as heuristic search. Artif. Intell. **129**(1–2), 5–33 (2001). http://dx.doi.org/10.1016/S0004-3702(01)00108-4
5. Bonner, A., Kifer, M.: Transaction logic programming. In: International Conference on Logic Programming, pp. 257–282. MIT Press, Budaspest (1993)
6. Bonner, A., Kifer, M.: Applications of transaction logic to knowledgerepresentation. In: Gabbay, D.M., Ohlbach, H.J. (eds.) ICTL 1994. LNCS, vol. 827, pp. 67–81. Springer, Heidelberg (1994)
7. Bonner, A., Kifer, M.: Transaction logic programming (or a logic of declarative and procedural knowledge). Technical report CSRI-323, University of Toronto (November 1995). http://www.cs.toronto.edu/~bonner/transaction-logic.html

[6] http://ewl.cewit.stonybrook.edu/planning/RSTRIPS-TR-full.pdf.

8. Bonner, A., Kifer, M.: Concurrency and communication in transaction logic. In: Joint International Conference and Symposium on Logic Programming, pp. 142–156. MIT Press, Bonn, September 1996
9. Bonner, A., Kifer, M.: A logic for programming database transactions. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 117–166. Kluwer Academic Publishers, Norwell (1998). Chap. 5
10. Bonner, A.J., Kifer, M.: An overview of transaction logic. Theo. Comput. Sci. **133**, 205–265 (1994)
11. Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Auton. Agent. Multi-Agent Syst. **19**(3), 332–377 (2009). http://dx.doi.org/10.1007/s10458-009-9079-8
12. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(34), 189–208 (1971)
13. Gerevini, A., Schubert, L.: Accelerating partial-order planners: some techniques for effective search control and pruning. J. Artif. Intell. Res. (JAIR) **5**, 95–137 (1996)
14. Giunchiglia, E., Lifschitz, V.: Dependent fluents. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pp. 1964–1969 (1995)
15. Joslin, D., Pollack, M.E.: Least-cost flaw repair: A plan refinement strategy for partial-order planning. In: Proceedings of the Twelth National Conference on Artificial Intelligence, vol. 2, pp. 1004–1009. American Association for Artificial Intelligence, AAAI 1994, Menlo Park (1994). http://dl.acm.org/citation.cfm?id=199480.199515
16. Kahramanogullari, O.: Towards planning as concurrency. In: Hamza, M.H. (ed.) Artificial Intelligence and Applications, pp. 387–393. IASTED/ACTA Press, Orlando (2005)
17. Kahramanoğulları, O.: On linear logic planning and concurrency. In: Martín-Vide, C., Otto, F., Fernau, H. (eds.) LATA 2008. LNCS, vol. 5196, pp. 250–262. Springer, Heidelberg (2008)
18. Lin, F.: Applications of the situation calculus to formalizing control and strategic information: the prolog cut operator. Artif. Intell. **103**(1–2), 273–294 (1998). http://dx.doi.org/10.1016/S0004-3702(98)00054-X
19. Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag, New York (1984)
20. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco (2004)
21. Nguyen, T.A., Kambhampati, S.: A heuristic approach to planning with incomplete STRIPS action models. In: Chien, S., Do, M.B., Fern, A., Ruml, W. (eds.) ICAPS 2014. AAAI, Portsmouth (2014). http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7919
22. de Nijs, F., Klos, T.: A novel priority rule heuristic: learning from justification. In: Chien, S., Do, M.B., Fern, A., Ruml, W. (eds.) ICAPS 2014. AAAI, Portsmouth (2014). http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7935
23. Nilsson, N.: Principles of Artificial Intelligence. Tioga Publication Co., Paolo Alto (1980)
24. Pollock, J.L.: The logical foundations of goal-regression planning in autonomous agents. Artif. Intell. **106**(2), 267–334 (1998). http://dx.doi.org/10.1016/S0004-3702(98)00100-3
25. Pommerening, F., Röger, G., Helmert, M., Bonet, B.: Lp-based heuristics for cost-optimal planning. In: Chien, S., Do, M.B., Fern, A., Ruml, W. (eds.) ICAPS 2014. AAAI, Portsmouth (2014). http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7892

26. Reiter, R.: Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
27. Rossi, F. (ed.) Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, August 3–9, 2013. IJCAI/AAAI (2013)
28. Shoham, Y.: Artificial Intelligence Techniques in Prolog. Morgan Kaufmann, New York (2014)
29. Sierra-Santibáñez, J.: Declarative formalization of strategies for action selection: applications to planning. In: Brewka, G., Moniz Pereira, L., Ojeda-Aciego, M., de Guzmán, I.P. (eds.) JELIA 2000. LNCS (LNAI), vol. 1919, p. 133. Springer, Heidelberg (2000)
30. Srivastava, S., Immerman, N., Zilberstein, S., Zhang, T.: Directed search for generalized plans using classical planners. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011). AAAI, June 2011