Journal of Intelligent & Robotic Systems (2019) 96:179–191
https://doi.org/10.1007/s10846-018-0971-9

Robotic Task Planning Using a Backchaining Theorem Prover for Multiplicative Exponential First-Order Linear Logic

Sitar Kortik[1] · **Uluc Saranli**[2]

Received: 13 February 2017 / Accepted: 4 December 2018 / Published online: 7 January 2019

Abstract

In this paper, we propose an exponential multiplicative fragment of linear logic to encode and solve planning problems efficiently in STRIPS domain, that we call the Linear Planning Logic (LPL). Linear logic is a resource aware logic treating resources as single use assumptions, therefore enabling encoding and reasoning of domains with dynamic state. One of the most important examples of dynamic state domains is robotic task planning, since informational or physical states of a robot include non-monotonic characteristics. Our novel theorem prover is using the backchaining method which is suitable for logic languages like Lolli and Prolog. Additionally, we extend LPL to be able to encode non-atomic conclusions in program formulae. Following the introduction of the language, our theorem prover and its implementation, we present associated algorithmic properties through small but informative examples. Subsequently, we also present a navigation domain using the hexapod robot RHex to show LPL's operation on a real robotic planning problem. Finally, we provide comparisons of LPL with two existing linear logic theorem provers, llprover and linTAP. We show that LPL outperforms these theorem provers for planning domains.

Keywords Robotic task planning · Linear logic · Automated theorem proving · Visual navigation · Backchaining · RHex hexapod

1 Introduction

Task planning refers to the problem of choosing and ordering *actions* from an action list, transforming given set of *initial states*, into a possibly different set of *goal states* [28]. In order to enable this discretized view of actions, the physical behavior of a robot related with a particular action is represented by a list of prerequisite state components, called the *preconditions* and a list of state components that arise from successful realization of the action, called the *effects*. An action can only be used when all of its preconditions are satisfied. Once an applicable action is performed, its preconditions are deleted and its effects are added to the environment. According to this definition, a valid plan, which may take the form of either a total or partial order of chosen actions [39], is expected to manipulate the environment to reach the desired final state from the initial state. Automated generation of such plans is an important problem in Artificial Intelligence, with numerous application areas such as scheduling, automated programming, robotics and resource planning [30, 43].

An important application area for automated planning systems is motion planning. Traditionally, research in the field of robotics and control considers this to be a continuous problem, wherein the trajectories of a continuous dynamical system are to be manipulated as desired [24]. On the contrary, this approach becomes insufficient for many problems, including hybrid robotic systems with different contact configurations (e.g. legged robots) as well as higher level planning tasks with the need to sequence multiple different behaviors [2]. Recent research efforts focusing on this latter set of problems seeks to combine of methods in continuous trajectory planning and control with methods in discrete task planning [1, 8]. In this paper, we focus on

✉ Sitar Kortik
sitar@cs.bilkent.edu.tr

Uluc Saranli
saranli@ceng.metu.edu.tr

[1] Department of Computer Engineering, Bilkent University, Ankara, Turkey

[2] Department of Computer Engineering, Middle East Technical University, Ankara, Turkey

two main concepts. One of them is the problem of task planning and the other one is visual navigation for a hybrid robotic platform, the RHex hexapod robot [40], using it to demonstrate automated task planning through proof search in a sufficiently expressive logical formalism to achieve autonomous behavior.

In the context of task planning, substantial research effort has been devoted in the last few decades into the development of automated planners [34]. Earlier attempts at using logical representations and plan construction based on automated theorem proving using, for example, situational calculus [33], were subsequently found to be infeasible, due to both the additional semantic complexity associated with classical first-order logical representations of the time, as well as the difficulty of constructing efficient theorem provers for them. In light of these issues, alternative methods that were semantically less complete but practically much more feasible were developed. Among these, partial-order planning (POP) searches through the space of plans rather than states [32, 35] to achieve efficiency. In contrast, GraphPlan still performs plan search in state space, but relies on the construction of an approximate reachability graph augmented with mutual exclusion constraints [4]. A different, yet powerful idea that still finds use in some of the modern automated planners is using Constraint Satisfaction Problems (CSP) [12, 27] for the formulation of planning problems. More recently, further improvements in the efficiency and performance of automated plan construction was achieved through the use of increasingly sophisticated heuristics. Most modern automated planners benefit some form of heuristics to guide plan search [34].

In the meantime, logical approaches to automated plan generation have not received as much attention. In logical representations of planning problems, a valid proof can be associated with a valid plan, wherein semantic rules of the logical system and assumptions provided as part of the domain definition together ensure plan soundness. Unfortunately, the choice of a particular logical language and associated proof theory may introduce complexity that is far beyond the planning problem itself, through irrelevant permutations of otherwise equivalent proof alternatives [20]. It has been very difficult to find logical formalisms that minimize this additional complexity to make plan search through proof construction a practical alternative to other methods for automated planning.

A notable exception in this context is linear logic, which was introduced by Girard [14] to unify elegant symmetries of classical logic with the representational strengths of intuitionistic logic. More recently, it was observed that linear logic, and its ability to interpret assumptions as single-use, consumable resources, was uniquely suitable to encode planning problems [7]. Various intuitionistic fragments of linear logic were hence found to be effective languages in which dynamic state could be natively and effectively represented, avoiding issues such as the frame problem [11, 17, 17, 23, 38]. The utility of non-monotonic features were also observed and used by other planners, including those that were based on logic programming and answer sets [9, 13, 25, 26].

Despite these promising properties in representing planning problems, linear logic remains a difficult language to construct automated theorem provers. Even though in some aspects, it has better computational properties (e.g. expressive fragments of first-order linear logic are decidable [3]), it introduces its own set of problems in automated theorem proving such as resource management [6]. Consequently, current implementations of efficient and practical theorem provers for linear logic are still restricted to limited subsets of the language such as the Linear Hereditary-Harrop Formulas adopted by the Lolli logic programming language [16].

One of our primary contributions in this paper is a backchaining, efficient theorem prover for an exponential multiplicative fragment of linear logic, that we call the Linear Planning Logic (LPL). LPL is an extension of Linear Hereditary Harrop formulas (LHHF) to allow program formulas with non-atomic conclusions meanwhile unchanging the resulting nondeterminism. Thus, in contrast to existing works such as Lolli, we can express actions that can create multiple effects in LPL. We achieve this by transferring unused resources from the left branch to the right branch in the proof tree, within a sound and complete proof theory. Preserving the backchaining structure of proof search while allowing non-atomic conclusions in this fashion requires careful resource management and bookkeeping during proof construction. The resulting language is sufficiently expressive to easily encode task planning problems, which we demonstrate in the context of visual mobile robot navigation with the hexapod robot RHex in a planning domain including markers.

We proceed by reviewing background and related literature in Section 2, followed by the language and proof theory for LPL in Section 3. We then introduce our application domain of visual mobile robot navigation in Section 4 and present our experimental results.

2 Background

2.1 Task Planning with Intuitionistic Linear Logic

The utility of intuitionistic linear logic (ILL) for task planning has previously been observed in the literature [18–22, 31]. In this part, we overview basic ideas on the use of ILL for planning problems, illustrating on a simple blocks

world domain. Our treatment of this topic is similar to the framework presented in [10].

Formalizations of planning problems in linear logic, as we also do in this paper, often use *intuitionistic* fragments since *proofs* correspond to executable *programs* in intuitionistic logic which is known as the Curry-Howard isomorphism [42]. This enables to treat an automatically generated proof as a valid plan, assuming that an adequate encoding of the domain is provided.

In ILL, a *linear implication* $A \multimap B$ can be used to encode an action within task planning problems, which consumes its precondition A, and creates the effect B. Both of these formulae can represent multiple resources themselves through the use of *simultaneous conjunction* as $A \otimes B$, which shows that A and B should be separately consumed or generated simultaneously. Sequent calculus rules associated with these connectives are given as

$$\frac{\Delta_1 \Rightarrow A \quad \Delta_2 \Rightarrow B}{\Delta_1, \Delta_2 \Rightarrow A \otimes B}\otimes R \qquad \frac{\Delta, A, B \Rightarrow C}{\Delta, A \otimes B \Rightarrow C}\otimes L$$

$$\frac{\Delta, A \Rightarrow B}{\Delta \Rightarrow A \multimap B}\multimap R \qquad \frac{\Delta_1 \Rightarrow A \quad \Delta_2, B \Rightarrow C}{\Delta_1, \Delta_2, A \multimap B \Rightarrow C}\multimap L.$$

The most critical detail in these inference rules is the splitting of the multiset of available assumptions (allowing duplicate occurrences of the same proposition) into Δ_1 and Δ_2 in the rules $\otimes R$ and $\multimap L$, which ensures that an assumption (and hence a physical resource for planning) cannot be used twice. Combined with the initial sequent

$$\frac{}{A \Rightarrow A}\ init$$

this formulation disallows contraction and weakening within the proof theory, requiring that every assumption is used exactly once. Inference rules for other connectives in linear logic, including quantifiers for first-order linear logic, follow similar principles [37].

Considering the well-known blocks world example [15], the four available actions of a block for picking up, putting down, picking up from table and putting down to table can be encoded as shown in Fig. 1, with individual predicates modeling various components of the physical state as consumable resources. Universal quantification is used to generalize these actions to all available objects in the environment, with left rules for the quantifier instantiating the action with specific objects.

Unlike physical resources, however, there should be no constraints on the number of times these action formulas can be used since they represent persistent facts about the domain. These "classical" semantics for persistent components in a domain are recovered in linear logic through the use of an "unrestricted context", leading to the sequent definition

$$\Gamma; \Delta \Rightarrow G\ , \tag{1}$$

where the goal formula, G, can be obtained using every ephemeral resources in the multiset Δ exactly once, and unrestricted resources in the set Γ as many times as needed or not at all. These semantics are implemented through the inference rules

$$\frac{(\Gamma, A); (\Delta, A) \Rightarrow B}{(\Gamma, A); \Delta \Rightarrow B}\ copy \qquad \frac{}{\Gamma; A \Rightarrow A}\ init$$

which allows duplication of resources in Γ, as well as the possibility of leaving them unused. An example problem in the domain of Blocks World is encoded as the goal sequent $\Gamma_{bw}; \Delta_{bw} \Rightarrow G_{bw}$, with

$$\begin{aligned}
\Gamma_{bw} &:= \{\text{pickUp}(x, y), \text{putOn}(x, y),\\
&\quad\ \text{pickUpFromTable}(x), \text{PutOnTable}(x)\}\\
\Delta_{bw} &:= [clear(c), empty, on(b, a), on(c, b), ontable(a)]\\
G_{bw} &:= ontable(c) \otimes clear(c) \otimes empty \otimes clear(b)\\
&\quad \otimes on(b, a) \otimes ontable(a)\ ,
\end{aligned}$$

whose solution has the robot grab c and subsequently put it on the table. Thereby, the extracted plan of the given example is the action PickUp(c,b) followed by the action PutOnTable(c).

2.2 Proof Search in Intuitionistic Linear Logic

In different ways, we can construct the proof of a valid sequent consisting of valid applications of inference rules in a proof theory [36]. For the sequent calculus formulation we adopt in this paper, automated methods for proof construction use either a *forward proof search* strategy, attempting to reach the goal formula through repeated forward application of inference rules starting from initial assumptions, or a *backward proof search* strategy wherein applicable inference rules are considered in reverse as candidates to generate specific goal formulas. In this paper,

PickUp(x,y)	$: \forall x.\forall y. empty \otimes on(x, y) \otimes clear(x) \multimap holds(x) \otimes clear(y)$
PutOn(x,y)	$: \forall x.\forall y. holds(x) \otimes clear(y) \multimap empty \otimes on(x, y) \otimes clear(x)$
PickUpFromTable(x)	$: \forall x. empty \otimes ontable(x) \otimes clear(x) \multimap holds(x)$
PutOnTable(x)	$: \forall x. holds(x) \multimap empty \otimes ontable(x) \otimes clear(x)$

Fig. 1 Linear logic encodings of available actions in the Blocks World

we focus on a backwards proof search strategy since, for suitably restricted languages such as horn clauses, it allows backchaining as an efficient and well-defined basis for goal-directed search and logic programming languages.

In general, the difficulty of backward proof search comes from the nondeterminism associated with the presence of multiple inference rules whose conclusions match a particular goal sequent. Ensuring completeness of proof search requires that all applicable alternatives are attempted for all levels in the proof tree, often resulting in prohibitive complexity. One way in which this complexity can be reduced is to consider smaller fragments of the logic, reducing expressivity but decreasing the number of nondeterministic choices during proof construction. A known example for this way is the use of Horn Clauses for logic programming with Prolog, which enables goal-oriented backchaining by restricting atomic conclusions on implicative formulas that appear as assumptions (program formulas) [5], taking the form

$$(p \ \wedge \ q \wedge ... \wedge t) \rightarrow u \ . \tag{2}$$

During proof search, this allows *backchaining*, which first decomposes a given goal formula into its atomic conjuncts, identifies which program formulae have the resulting atomic goals as their conclusions and proceeds to attempt the subgoals that are given as its premises. This results in a substantial reduction in available alternatives to be considered during proof construction.

A similar approach has also been proposed for linear logic through the use of Linear Hereditary Harrop formulas (LHHF), constituting the foundation for the Lolli logic programming language [16]. Despite gains in efficiency, however, the limitation of using atomic conclusions in implicative formulas makes it impossible to directly encode planning problems in the style shown in Fig. 1, wherein the effects of an action need to include multiple resources. In this paper, one of main contributions is constructing a new proof theory that relaxes this restriction to allow conjunctive conclusions in implicative formulas, while preserving the backchaining structure of proof search for linear logic.

A different but equally challenging difficulty faced by backward theorem provers for Linear Logic arises from the need to manage the splitting of consumable resources for inference rules such as the $\otimes R$ and $\multimap L$ in Section 2.1. Naive implementations that consider all combinatorial options introduce unnecessary complexity. Fortunately, recent research in this area proposes an input/output model of resource management [6], wherein the proof theory keeps track of which resources are required by one of the proof branches to restrict available alternatives for the second branch for these rules. The backchaining proof theory we present in this paper for the LPL language also uses this approach to eliminate this additional complexity associated with resource management.

3 Linear Planning Logic (LPL)

3.1 LPL: the Language

In this section, we introduce and formally define the Linear Planning Logic (LPL) language as an extension of the LHHF. LPL is described through the following grammar:

Program formulas:
$D \ ::= a \mid D_1 \otimes D_2 \mid G \ \multimap \ D \mid \forall x.\, D$
LPL Goal formulas:
$G \ ::= a \mid G_1 \otimes G_2 \mid D \ \multimap \ G \mid D \supset G \mid \exists x.\, G \ ,$

where a denotes first order atomic propositions that are suitably defined for a particular application domain. As we mentioned before, LPL is extending the LHHF grammar by incorporating simultaneous conjunction ($\otimes$) into D (program formulae). This extension allows the encoding of task planning actions that include more than one resource as part of their effects such as those shown in Fig. 1. The goal formula $D \supset G$ denotes classical implication, meaning that the resource D can be used as many times as necessary.

Note, also, that even though certain additional features and connectives could be included in this grammar, including disjunctive and universally quantified goal formulas as well as additive connectives of linear logic, we have kept the language as simple as possible to ensure a clear presentation of the main novel ideas introduced in the paper. Other features, such as disjunctive and existentially quantified program formulas, are much more challenging and are not covered in the scope of LPL and this paper.

3.2 LPL: Backchaining Proof Theory

In this section, we briefly describe a sequent calculus formulation of our backchaining proof theory for LPL, which refer to as the BL system. In doing so, we focus on our main contributions and only highlight important features that underlie our experimental implementation and application.

3.2.1 Resource Management

As observed in Section 2.1, backwards proof search in linear logic suffers from a problem called *resource management*, which is related to deciding on an accurate splitting of resources while decomposing conjunct goals. Recently, an *input/output* model (I/O model in short) was introduced by Hodas and Miller [16] to solve resource management problem, which we adopt and extend in LPL.

We begin by defining the sequent

$$\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} G \ , \tag{3}$$

as an extension of the I/O model described in [6]. The given sequent states that we can achieve the goal G by using resources from Γ (unrestricted resources) and Δ_I (consumable resources) while leftover resources are put into Δ_O. We should also note that, F is the set of *forbidden* labels including labels of assumptions from Δ_I. None of subproofs derived from the given sequent can use these assumptions in F.

We should note that, a program formula is stored as $(u : {}^L D)$ in Γ, Δ_I and Δ_O. Each resource D has an unique label u and a depended list L which is storing labels of depended resources. As we will show in subsequent sections, we store these dependencies since we want to limit scopes of resources in Δ_O.

Based on this definition, an LPL goal formula G is provable if we can find a proof for the sequent

$$\cdot; \cdot \setminus \cdot \overset{\emptyset}{\Longrightarrow} G \ .$$

The main idea behind the *input/output model* of resource management is to prove the first subgoal, determining which resources are used from within initially available ones, and then achieving the remaining subgoal with leftover resources. For example, the right rule for simultaneous conjunction takes the form

$$\frac{\Gamma; \Delta_I \setminus \Delta_M \overset{F}{\Longrightarrow} G_1 \qquad \Gamma; \Delta_M \setminus \Delta_O \overset{F}{\Longrightarrow} G_2}{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} G_1 \otimes G_2}\textbf{bl-}\otimes,$$

where the resources in Δ_M that are unused by the proof of the right subgoal are supplied into the left subgoal which then produces the leftover resources in Δ_O.

3.2.2 Backchaining and Residuation

The backchaining proof theory for LHHF presented in [6] is based on eager decomposition (also called *reduction*) of the goal formula until an atomic goal is obtained, followed by applying resolution rules on either a linear or unrestricted resource that has this atomic goal in its conclusion. Resolution rules defined in this work rely on a *residuation* judgement, which focuses on a single program formula, continually decomposing it to extract its subgoals until its conclusion is found to match the desired atomic goal. Proof search then continues by attempting to prove the generated subgoals.

This strategy is unfortunately not directly applicable for LPL since its program formulas are not constrained to have atomic conclusions. This means that when a single program formula is used to prove an atomic goal, additional resources would remain after the application of the resolution rule. Our new proof theory addresses this issue by allowing leftover resources from the decomposition of a program formula, carefully maintaining restrictions on their usage. To this end, we give an extended version of the residuation judgement in [6] as

$$(u : {}^L D) \gg a \setminus \Delta_G \triangleright \Delta_O \ , \tag{4}$$

where a is the atomic goal, u is the unique label of the program formula D that we focus and L is the list of dependencies. This judgment means that we can achieve the atomic goal a by using the resource D, where we can create a proof for each goal in Δ_G, yielding the output context Δ_O which is a list of leftover resources following decomposition of D.

Reduction rules for our backchaining proof theory BL implementing eager decomposition of the goal formula are detailed in Appendix A. When all applicable right rules in our prover are applied and completely used, the goal on the right part of the sequent will be an atomic goal, a. Then, our prover picks up a resource D from either Γ or Δ, using resolution rules given in Appendix B. Subsequently, associated subgoals are collected in Δ_G and the system checks whether any of the conclusions in the program formula match the atomic goal. Each goal in Δ_G is represented as $(u_g : G^F)$, where u_g is an unique label of the goal formula G, and F is a set of forbidden resources which we can not use in the proof of the goal G. This last annotation is needed to ensure that formulas leftover from the decomposition of a program formula are not used in proving subgoals generated by the same decomposition.

To illustrate, consider the linear resolution rule

$$\frac{u : {}^L D \gg a \setminus (g_1 : G_1^{F_1}, ..., g_n : G_n^{F_n}) \triangleright \Delta_{O,0} \ ... \ \Gamma; \Delta_{I,k} \setminus \Delta_{O,k} \overset{F_k \cup F}{\Longrightarrow} G_k ...}{\Gamma; \Delta_I, (u : {}^L D) \setminus \Delta_{O,n} \overset{F}{\Longrightarrow} a}\textbf{bl-lin},$$

which also has a side condition $u \notin F$ to prevent focusing on forbidden resources. First, we remove the selected program formula D from the linear context (i.e. it does not reappear in the output context) to prevent its reuse. Backwards application of this resolution rule first decomposes the program formula to obtain its subgoals G_i and the output context $\Delta_{O,0}$. This decomposition is performed using residuation rules presented in Appendix C. We then attempt to prove all of these subgoals, starting from the initially available set of resources $\Delta_{I,1} = \Delta_I \cup \Delta_{O,0}$, and using the input/output model for resource management to transfer unused formulas from one sub-goal to the following one, until the proof of the last subgoal generates the overall output context $\Delta_{O,n}$. Note, also, that the proof of each subgoal G_i is prevented from using the forbidden resources in the associated label set F_i through the use of the forbidden resource annotation above the arrow in our sequent definition.

Similar as other efficient backchaining methods for LHHF, the residuation judgement reveals subgoals needed to prove an atomic goal from a focused resource. On the other hand, the main difference of our prover is that we can handle left simultaneous conjunction while this is missing in LHHF. To illustrate, consider one of the residuation rules for a conjunctive program formula, defined as

$$\frac{v_1 : {}^{L\cup\{v_1\}}D_1 \gg a \setminus \Delta_G \rhd \Delta_O}{u : {}^{L}(D_1 \otimes D_2) \gg a \setminus \Delta_G \rhd \Delta_O, (v_2 : {}^{L\cup\{v_2\}}D_2)}\ \textbf{d-}\otimes_1 .$$

This rule attempts to match the left conjunct D_1 with the atomic goal, and if successful, records the unused right conjunct D_2 in the *output* section of the sequent, thereby allowing its use by other branches of the proof. The main contribution of our system is that it defines certain boundaries for when and where to use the extracted formulae during proof search while keeping soundness. in making sure that proper boundaries are established for where and when these subformula outputs can be used within the proof without compromising soundness. Apart from these rules for simultaneous conjunction, all other residuation rules for LPL mirror their counterparts for LHHF detailed in [6] and are given in Appendix C.

3.2.3 Illustrative Examples

For the given LPL formula $a, a \multimap (c \otimes b) \Rightarrow (b \otimes c)$, a backchaining approach based on standard sequent rules for intuitionistic linear logic (e.g. as described in [37]) would first attempt to decompose the goal $b \otimes c$ into its conjuncts. For example, a possible partially constructed proof takes the form

$$\frac{\dfrac{?}{a, a \multimap (c \otimes b) \Rightarrow b}\ \multimap L \qquad \dfrac{?}{\cdot \Rightarrow c}}{a, a \multimap (c \otimes b) \Rightarrow (b \otimes c)}\ \otimes R .$$

Unfortunately, none of the possible splits for the available resources a and $a \multimap (c \otimes b)$ achieve the given goal. This shows that when program formulas are allowed to have multiple conclusions, eager decomposition of the goal is not directly applicable. In contrast, the BL system we described above and detailed in the appendices results in the successful proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b \gg b \setminus \cdot \rhd \cdot}\ \textbf{d-atm}}{(c \otimes b) \gg b \setminus \cdot \rhd c}\ \textbf{d-}\otimes_R}{a \multimap (c \otimes b) \gg b \setminus a \rhd c}\ \textbf{d-}\multimap \qquad \dfrac{\dfrac{}{a \gg a \setminus \cdot \rhd \cdot}\ \textbf{d-atm}}{\cdot; a, c \setminus c \Longrightarrow a}\ \textbf{bl-lin}}{\cdot; a, a \multimap (c \otimes b) \setminus c \Longrightarrow b}\ \textbf{bl-lin} \qquad \dfrac{\dfrac{}{c \gg c \setminus \cdot \rhd \cdot}\ \textbf{d-atm}}{\cdot; c \setminus \cdot \Longrightarrow c}\ \textbf{bl-lin}}{\cdot; a, a \multimap (c \otimes b) \setminus \cdot \Longrightarrow (b \otimes c)}\ \textbf{bl-}\otimes .$$

Note that we have left proof annotations such as forbidden resources and labels for propositions out of this illustration for clarity. This structure allows eager decomposition of the goal formula, thereby decreasing nondeterminism in proof search while maintaining the completeness of the theory.

We now give another example to show the need for the sequent annotation for forbidden resources F. Consider the example sequent

$$\cdot; a \multimap (a \otimes b) \setminus \cdot \overset{\emptyset}{\Longrightarrow} b ,$$

which can not be provable in a sound proof theory for LPL. However, when the resolution step in the BL system focuses on the single program formula $a \multimap (a \otimes b)$ to prove the atomic goal b, the remaining conjunct a in the conclusion of this formula will be leftover as a result of the residuation judgement. This leftover resource should not be usable in proving the subgoal a of the same program formula. In other words, the following proof should be disallowed:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{u_4 : b \gg b \setminus \cdot \rhd \cdot}\ \textbf{d-atm}}{u_2 : (a \otimes b) \gg b \setminus \cdot \rhd u_3 : a}\ \textbf{d-}\otimes_2}{u_1 : (a \multimap (a \otimes b)) \gg b \setminus a^{[u_3]} \rhd u_3 : a}\ \textbf{d-}\multimap \qquad \dfrac{\dfrac{}{u_3 : a \gg a \setminus \cdot \rhd \cdot}\ \textbf{d-atm}}{\cdot; u_3 : a \setminus \cdot \overset{\{u_3\}}{\Longrightarrow} a}\ \textbf{bl-lin}}{\cdot; u_1 : (a \multimap (a \otimes b)) \setminus \cdot \overset{\emptyset}{\Longrightarrow} b}\ \textbf{bl-lin} ,$$

where we have omitted dependency labels for resources for clarity. The BL system for LPL we have introduced disallows this proof because the application of the resolution rule **bl-lin** on the right part of the proof is disallowed because of the failing side condition $u_3 \notin \{u_3\}$.

The backchaining proof theory BL we introduced in this section with the detailed sequent calculus rules given in Appendx A, B and C is sound and also complete according to a standard proof system of the LPL language. We give a proof outline for this claim, but avoid a detailed treatment in this paper because of space considerations.

3.2.4 Implementation of the LPL Theorem Prover

Our implementation of the LPL language and the BL system for proof construction is done in SWI-Prolog, relying on Prolog's backtracking facilities to perform proof search. First-order variables in the LPL language are implemented through Prolog variables, using unification throughout proof search for eliminating universal quantifiers and instantiating variables with specific objects. Other than these details, our implementation is a straightforward instantiation of the inference rules for the BL system presented in the appendices.

An important feature of our implementation, which we left out of the scope of the present paper for space considerations, is our use of proof terms to annotate proof trees, yielding a final expression that captures the "program" that is associated with the proof itself. Our

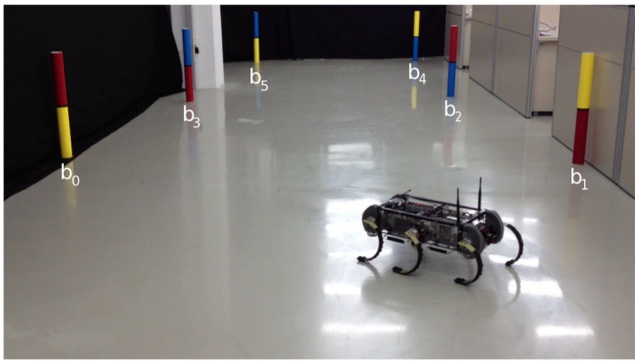

Fig. 2 The experimental setup for the visual robot navigation domain including six uniquely colored markers and the robot RHex with a camera on it

restriction of the LPL language to intuitionistic connectives of linear logic allows this correspondence between proof and programs, which we then transform into a valid action list corresponding to a valid plan for robot behaviors. The transformation of proofs for LPL goal propositions in the BL system into valid proof terms for linear logic also corresponds to the soundness of the BL system according to the standard proof theoretic semantics for linear logic, but is left outside the scope of the present paper for space considerations. In summary, this transformation uses proof rewriting to commute eager applications of right and left rules for simultaneous conjunction, yielding a final proof term wherein ordered applications of the linear resolution rule correspond to sequencing of successive actions within the planning problem.

4 Application: Visual Navigation with the RHex Hexapod

In this section, we illustrate the use of the LPL language and proof theory for task planning problems in the context of autonomous visual navigation for the hexapod robot RHex [40]. RHex is a power-autonomous hexapod robot platform that is capable of a variety of locomotory behaviors. However, many existing capabilities of this robot require remote user control for navigation as well as choosing a particular sequence of behaviors to accomplish a task. Progress towards fully autonomous operation with this platform requires the ability to decide on both the sequence and timing of different behaviors that are appropriate for different environmental conditions, while ensuring that the desired high-level goal is accomplished.

In order to illustrate how the LPL language and proof theory can be used in this context, we focus on an example domain wherein the RHex hexapod is expected to navigate in an environment having six unique markers. Each marker is assumed to be connected with a path possessing surface properties (e.g. mud, rock field, sand, asphalt etc.) effecting traversability as well as particular locomotory behaviors that are feasible. In particular, we consider two behaviors, walking and running, which are feasible on different surfaces with the RHex robot. The platform can also tag markers, modeling specific actions that might be carried out by the robot at a particular landmark. Figure 2 illustrates a snapshot of our experimental setup for this domain with six uniquely colored landmarks and the RHex hexapod. Our experiments were performed in an indoor laboratory environment to avoid difficulties with visual processing in outdoor conditions.

4.1 LPL Encoding of the Domain

The first step in the encoding of this domain within the LPL language is the determination of atomic predicates. These predicates must be sufficient to capture relevant components of the physical state for both the robot and the environment. Based on the description above, descriptions for the atomic predicates to be used in our LPL encoding of the visual robot navigation domain are given in Fig. 3.

The next step in the encoding is the formalization of the actions available to the robot. First, two actions are defined for the locomotory behaviors of the RHex robot. Slow walking behavior towards a landmark X is appropriate for rough surfaces and is represented by the Walk(X). In contrast, fast running towards a landmark X is feasible on smooth surfaces and is denoted by Run(X). Both of these behaviors are assumed to actively seek the targeted landmark x, realized through visual servoing principles as described in Section 4.3. Complementing these behaviors is a visual searching behavior for a particular landmark X, denoted by Seek(X), which forces the robot to stand in place and perform a complete 360^o visual scan of its environment by using a turning behavior until the landmark X is found. Finally, the robot uses the action Tag(X) for marking the landmark X, which is required to be both visible and near the robot. In Fig. 4, we give descriptions and LPL definitions for available actions to encode the robotic domain.

Fig. 3 Descriptions of the atomic predicates for the visual robot navigation domain

at(X)	:	Robot is at the landmark X.
tagged(X)	:	Landmark X is tagged.
untagged(X)	:	Landmark X is not tagged.
see(X)	:	Robot can see the landmark X.
surface(X, Y, Z)	:	Path between X and Y has type Z.

Fig. 4 Descriptions and LPL definitions for available actions in the visual robot navigation domain

$\text{Walk}(X)$	:	Robot walks to the landmark X. $\forall y.\ \text{at}(y) \otimes \text{see}(X) \otimes \text{surface}(y, X, \text{rough}) \multimap \text{at}(X) \otimes \text{see}(X)$
$\text{Run}(X)$	:	Robot runs to the landmark X. $\forall y.\ \text{at}(y) \otimes \text{see}(X) \otimes \text{surface}(y, X, \text{smooth}) \multimap \text{at}(X) \otimes \text{see}(X)$
$\text{Seek}(X)$	:	Robot searches for the landmark X. $\forall y.\ \text{see}(y) \multimap \text{see}(X)$
$\text{Tag}(X)$	:	Robot tags the landmark X. $\text{at}(X) \otimes \text{untagged}(X) \multimap \text{at}(X) \otimes \text{tagged}(X)$

Similar to the blocks world example in Section 2.1, encodings of actions in LPL take the form of linear implications, consuming conjoined preconditions to generate a conjunction of effects. The definitions in Fig. 4 incorporate universal quantifiers, which are instantiated with appropriate objects during proof search through unification, corresponding to the application of a specific action for specific objects. Note, also, that surface properties for paths connecting landmarks defined through the $surface(...)$ predicate will be incorporated into the unrestricted context Γ and hence are not reintroduced by the actions Walk and Run.

4.2 A Planning Problem Example

In order to illustrate the application of the introduced planner and theorem prover LPL on the visual robot navigation problem, we will focus on a specific problem instance corresponding to the scenario given in Fig. 2, having the robot RHex and six unique markers.

First, we encode the static properties of the environment within the unrestricted context, taking the form

$$\Gamma_0 := \{surface(Start, b_1, rough), surface(b_1, b_0, rough), surface(b_0, b_3, rough), surface(b_3, b_4, smooth), surface(b_4, b_2, smooth), surface(b_2, b_5, smooth), \forall x.\ \text{Walk}(x), \forall x.\ \text{Run}(x), \forall x.\ \text{Seek}(x), \forall x.\ \text{Tag}(x)\}\ ,$$

describing traversability properties of paths between suitable pairs of landmarks, and providing definitions for all available actions.

Next, the initial state is provided in the linear context, taking the form

$$\Delta_0 := [at(Start), untagged(b_3), untagged(b_5), see(b_0)]\ .$$

This definition encodes that the robot is initially at the *Start* position, where the marker b_0 is assumed to be seen with landmarks b_3 and b_5 currently untagged. Finally, the desired goal state for this example is chosen as

$$G_f := [at(b_5), tagged(b_3), tagged(b_5), see(b_5)]\ ,$$

encoding that the robot will finally be at the marker b_5, having previously tagged landmarks b_3 and b_5. The theorem prover we described for the LPL language is then invoked with the sequent

$$\Gamma_0; \Delta_0 \setminus . \stackrel{\emptyset}{\Longrightarrow} G_f\ ,$$

whose proof is expected to yield a valid plan for the planning problem described above.

We have used our SWI-Prolog implementation of the backchaining BL system for LPL to generate a feasible navigational plan for this example problem. The resulting proof yields the following sequence of actions:

$$[Seek(b_1), Walk(b_1), Seek(b_0), Walk(b_0), Seek(b_3), Walk(b_3), Tag(b_3), Seek(b_4), Run(b_4), Seek(b_2), Run(b_2), Seek(b_5), Run(b_5), Tag(b_5)]\ ,$$

which was extracted from the proof by identifying ordered applications of the resolution rule on unrestricted resources that encode actions.

4.3 Visual Tracking of Colored Landmarks

All of RHex's behaviors underlying the domain described in this section rely on skills of the robot to visually identify and locate uniquely colored landmarks in the environment. For this purpose, we have equipped the autonomous RHex robot with a front-facing RGB camera (Point Grey Flea2), connected to an onboard single-board computer unit (RTD CME137LX) through a firewire interface. This setup allows real-time processing of images acquired from the scene at 5Hz, identifying landmarks and determining their distance and bearing relative the robot itself. This information is then used by walking and running behaviors to keep a landmark in the field of view, as well as the seeking behavior to determine when a desired landmark is located in the

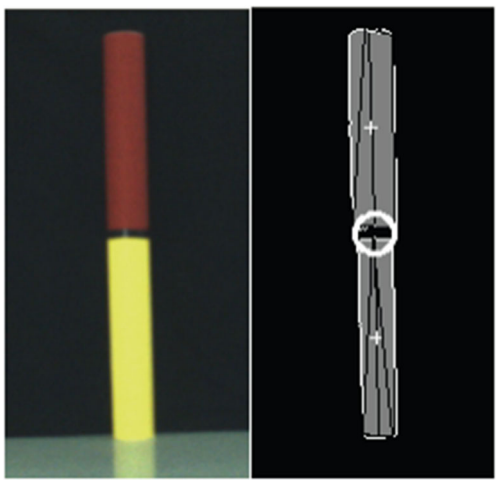

Fig. 5 Left hand side is a snapshot from a colored landmark as the robot sees. Right hand side shows extracted blobs from the left image. Centers of blobs are shown with "+" sign. The circle shows midpoint of two blobs which is the location of the landmark

Table 1 Color assignments for the six landmarks used in our example problem

Landmark ID	b_0	b_1	b_2	b_3	b_4	b_5
Top color	red	yellow	red	blue	yellow	blue
Bottom color	yellow	red	blue	red	blue	yellow

field of view. In this section, we describe visual processing algorithms we have developed for this purpose.

Our example domain in this paper is introduced primarily to illustrate the application of LPL and its proof theory. Consequently, we have chosen to simplify the visual processing problem by using colored landmarks having a radius of $8cm$ and a height of $1.5m$. A landmark is painted with two of the tree colors blue, yellow and red as illustrated in Fig. 2. Both the choice of colors, and their vertical ordering (bottom or top) on a landmark separates it from other landmarks, allowing up to six different landmarks. An example is shown on the left of Fig. 5, and our landmark color assignments for the example problem are given in Table 1.

The algorithm we use to process images starts with a color filter that uses Mahalanobis distance based on RGB covariance matrices that are trained offline for each of the three colors to transform the given color image into a tagged bitmap. We extract large linked components within tagged bitmap with the cvBlob library, subsequently we filter these components through domain based properties such as orientation, aspect ratio and blob size. Then we associate each blob with another according to their relative positions, providing the last centroid positions for each landmark as displayed in the right hand side of Fig. 5. This centroid yields the landmark bearing, while blob sizes allow measuring distance to the landmark. Pose information associated with all of visible landmarks is then communicated to the control computer that is responsible for applying the planned list of actions.

4.4 Plan Execution on RHex

Separate from the vision computer, an additional computer onboard the RHex platform is responsible from controlling the actions of the robot, coordinating individual motors on each leg and implementing specific behaviors such as walking and running. Based on the plan generated by the LPL theorem prover, and data provided by the visual tracking software described in Section 4.3, this control computer autonomously controls the operation of the RHex platform. This brings all of the components in our system together, executing the plan towards the desired goal using low level behaviors of the robot.

Figure 6 illustrates the execution of the sequence of actions described in Section 4.2 (14 total, pre-computed by the planner in 90 seconds on a Intel Pentium Dual-Core CPU E6500 Processor 2.93GHz PC with 2 GB of RAM with an implementation of SWI-Prolog). We show that the robot successfully follows the plan in an autonomous fashion, going through the desired sequence of landmark locations and tagging specific landmarks to reach the final goal.

5 Experimental Results

We analyze the performance of LPL on the well-known blocks world planning domain and show that it outperforms two other linear logic theorem provers. In particular, we focus

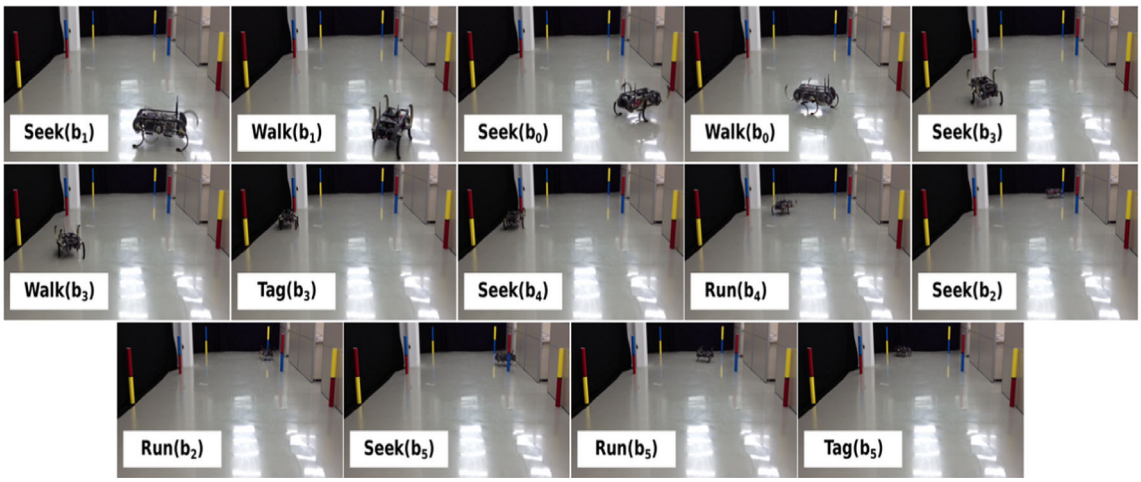

Fig. 6 Snapshots of the application of the extracted plan for the RHex robot using the LPL planner

Table 2 Six planning examples in the blocks world domain. Predicates empty, clear(X), on(X,Y), holds(X) and onTable(X) have been shortened as e, c(X), o(X,Y), h(X) and ot(X), respectively. Actions PickUp(X,Y), PutOn(X,Y), PickUpFromTable(X) and PutOnTable(X) have been shortened as U(X,Y), O(X,Y), FT(X) and OT(X)

n	Δ	G	P
1	$e \otimes c(a) \otimes ot(a)$	$h(a)$	[FT(a)]
2	$e \otimes c(a) \otimes ot(a) \otimes c(b) \otimes ot(b)$	$e \otimes c(a) \otimes o(a,b) \otimes ot(b)$	[FT(a), O(a,b)]
3	$h(a) \otimes c(b) \otimes ot(b)$	$e \otimes c(b) \otimes o(b,a) \otimes ot(a)$	[OT(a), FT(b), O(b,a)]
4	$e \otimes c(a) \otimes o(a,b) \otimes ot(b)$	$e \otimes c(b) \otimes o(b,a) \otimes ot(a)$	[U(a,b), OT(a), FT(b), O(b,a)]
5	$e \otimes c(a) \otimes o(a,b) \otimes o(b,c) \otimes ot(c)$	$h(c) \otimes c(b) \otimes o(b,a) \otimes ot(a)$	[U(a,b), OT(a), U(b,c), O(b,a), FT(c)]
6	$e \otimes c(a) \otimes o(a,b) \otimes o(b,c) \otimes ot(c)$	$e \otimes c(c) \otimes o(c,b) \otimes o(b,a) \otimes ot(a)$	[U(a,b), OT(a), U(b,c), O(b,a), FT(c), O(c,b)]

on comparison of LPL's execution time with these theorem provers for different problems with gradually increasing number of actions within the blocks world domain.

First, we consider llprover [41], which is a linear logic theorem prover finding a proof of a given sequent. Our comparisons also include linTAP [29], which is a tableau theorem prover for linear logic. Basically, it takes a linear logic sequent and checks its validity by building an analytic tableau while ensuring the validity with prefix unification.

We implemented the LPL planner in SWI-Prolog. In the blocks world domain, all experiments were performed using 1,4 GHz Intel Core i5 Processor and 4 GB of RAM.

We investigate six planning problems (number of actions are gradually increasing) of the blocks world domain described in Section 2.1, having four actions given in Fig. 1 and several blocks. Assume the robot arm moves these available blocks on a table. The linear logic encodings of initial states, goal states and plans (list of actions) for these problems are shown in Table 2. In order to satisfy a goal G_n with a list of resources Δ_n, a list of actions, P_n, should be applied.

Table 3 shows execution times for llprover, linTap and LPL for the number of applied actions n to achieve given six problems. llprover and linTap can not handle increasing non-determinism after a certain point when the problem size increases regularly. In contrast, as a result of efficiently preserving backchaining, LPL can solve all problems less then one second. We also provide graphical comparison of LPL with other linear logic theorem provers llprove and linTap in Fig. 7.

Table 3 Execution times for the blocks world domain with the number of actions n

n	llprover	linTAP	LPL
1	4ms	1ms	1ms
2	$>$ 30min	50ms	2ms
3	-	$>$ 30min	2ms
4	-	-	3ms
5	-	-	6ms
6	-	-	0.7sec

6 Conclusion

We proposed both the theory and application of Linear Planning Logic (LPL), a multiplicative fragment of linear logic to encode and solve planning problems efficiently. Our work extends our previous presentation in [22], providing details on the backchaining BL proof theory that we have developed for this language. Our main contribution is building a novel theorem prover for finding plans of robotics planning problems, capable of expressing actions that can create multiple effects.

Our other contribution is demonstrating the efficiency and expressivity of this novel planner on an application domain which includes a hexapod robot capable of moving with a camera that can detect and tag colorful landmarks. We have encoded the novel robotic domain in LPL and also presented a planning scenario and some experiments of this domain on the real robot RHex.

Finally, we present comparison of LPL's execution time with two existing linear logic theorem provers, llprover

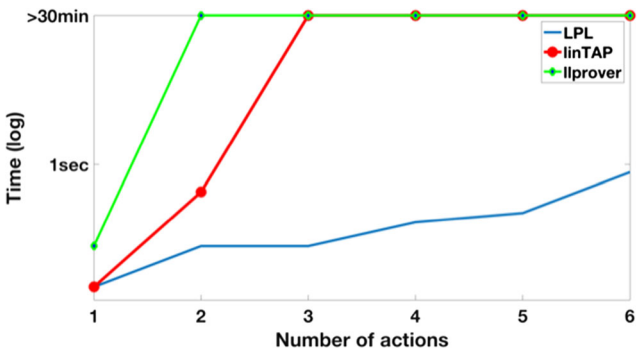

Fig. 7 Execution times (in logarithmic scale) for LPL, llprover and linTAP on the blocks world planning example

and linTAP. We show that LPL outperforms these theorem provers for planning domains, as a result of efficient backchaining proof search method that LPL has, which allows LPL to solve larger instances of planning problems.

We should note that, given order of subgoals for an instance of a planning problem is effecting efficiency of the LPL performance. As a future work, LPL can be parallelized by running parallel LPL planners for different orders of subgoals on the same instance of a planning problem.

At the end of this section, we list some future lines of our work. One possible direction is more efficiently implementing introduced theorem prover and planner. One future direction is allowing encoding of nondeterministic actions in planning problems by using additive and disjunctive connections. Another direction is integrating expressions of continuous constraints to be able to encode properties of robotic behaviors and dynamic environments.

Acknowledgments This work was supported by Tubitak project 114E277.

Appendix A: Reduction (Right) Rules for the BL Proof Theory

Right rules for both simultaneous conjunction, linear implication and existential quantification are eagerly applied first, until the sequent's conclusion is atomic. Associated rules are given in Fig. 8.

The $\otimes R$ rule relies on the I/O model to efficiently decompose available resources into left and right subgoals. In the right rule for linear implication, **bl-** $\multimap$, we try to achieve the subgoal G by using the assumption D. We should note that assumptions in Δ_O can not depend on the introduced assumption D. We need such a restriction, because the output context Δ_O is not a subset of Δ_I anymore.

In the **bl-**$\exists$ rule, a term t is provided and then substituted for the variable x.

Figure 9 shows the reduction rule for classical implication. Occurrences of the $D \supset G$ formula as a goal allow adding the resource D to the set of unrestricted resources G. Same as the **bl-** $\multimap$ rule, assumptions in Δ_O can not depend on D.

Appendix B: Resolution Rules for the BL Proof Theory

In Fig. 10, we give both linear and unrestricted resolution rules for the BL language. Both rules have the important side condition that focusing on any of the forbidden resources is not allowed with $u \notin F$. The input context of the first subgoal is the union of the input context Δ_I of the conclusion sequent, and the output context $\Delta_{O,0}$ from the residuation judgement.

Apart from these side conditions, the AddLb() function makes sure that, if the label of a resource in the final output context $\Delta_{O,n}$ is in the forbidden list F, then the dependence list of this resource is increased with L (dependence list) of D (focused resource). We formally define the AddLb() as

$$\text{AddLb}(L, \Delta, F) := \left\{ (u_i : {}^{\bar{L}_i} D_i) \mid (u_i : {}^{L_i} D_i) \in \Delta, \text{ and } \bar{L}_i = \left\{ \begin{array}{ll} L_i \cup L & \text{if } u_i \in F \\ L_i & \text{otherwise} \end{array} \right\} \right\}.$$

Fig. 8 Right sequent rules for multiplicative connectives in the **BL** proof theory

$$\frac{\Gamma; \Delta_I \setminus \Delta_M \overset{F}{\Longrightarrow} G_1 \quad \Gamma; \Delta_M \setminus \Delta_O \overset{F}{\Longrightarrow} G_2}{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} G_1 \otimes G_2}\ \textbf{bl-}\otimes$$

$$\frac{\Gamma; \Delta_I, (u : {}^{[u]} D) \setminus \Delta_O \overset{F}{\Longrightarrow} G \qquad \text{if } (v : {}^{L} D_k) \in \Delta_O \text{ and } v \notin \Delta_I, \text{ then } u \notin L}{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} D \multimap G}\ \textbf{bl-}\multimap$$

$$\frac{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} [t/x]G}{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} \exists x.\, G}\ \textbf{bl-}\exists$$

Fig. 9 Right sequent rules for classical implication in the **BL** proof theory

$$\frac{\Gamma, (u : {}^{[u]} D); \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} G \qquad \text{if } (v : {}^{L} D_k) \in \Delta_O, \text{ then } u \notin L}{\Gamma; \Delta_I \setminus \Delta_O \overset{F}{\Longrightarrow} D \supset G}\ \textbf{bl-}\supset$$

Fig. 10 Resolution rules for atomic goals for the BL proof system for LPL

$$\frac{u : {}^{L}D \gg a \setminus (g_1 : G_1^{F_1}, ..., g_n : G_n^{F_n}) \triangleright \Delta_{O,0} \quad \ldots \quad \Gamma; \Delta_{I,k} \setminus \Delta_{O,k} \overset{F_k \cup F}{\Longrightarrow} G_k \quad \ldots}{\Gamma; \Delta_I, (u : {}^{L}D) \setminus \mathrm{AddLb}(L, \Delta_{O,n}, F) \overset{F}{\Longrightarrow} a} \textbf{ bl-lin}$$

$$\frac{v : {}^{L\cup\{v\}}D \gg a \setminus (g_1 : G_1^{F_1}, ..., g_n : G_n^{F_n}) \triangleright \Delta_{O,0} \quad \ldots \quad \Gamma; \Delta_{I,k} \setminus \Delta_{O,k} \overset{F_k \cup F}{\Longrightarrow} G_k \quad \ldots}{\Gamma, (u : {}^{L}D); \Delta_I \setminus \mathrm{AddLb}(L, \Delta_{O,n}, F) \overset{F}{\Longrightarrow} a} \textbf{ bl-int}$$

Side conditions: $k = 1, .., n$, $u \notin F$, $\Delta_{I,1} = \Delta_I, \Delta_{O,0}$.

Appendix C: Residuation Rules for the BL Proof Theory

Finally, Fig. 11 shows all the residuation rules in the BL proof system for LPL. The **bl-atm** rule succeeds if the atomic resource a' can be unified with the atomic goal (a). The **d-** $\multimap$ rule attempts to achieve the atomic goal a with D, and then incorporates the goal G_1 to goals to achieve them later. Each label in Δ_O is incorporated to the G_1's forbidden list, because those assumptions are not allowed to be used to achieve that goal. The **d-**$\otimes_1$ rule uses D_1 to achieve a and adds D_2 to Δ_O. The **d-**$\otimes_2$ rule uses D_2 to achieve the atomic a, and adds the other resource to Δ_O. The **d-**$\forall$ rule achieves a by substituting all variables x with the term t in D. This term is expected to be globally chosen later through unification.

An important property of this newly defined residuation judgment for LPL is that the property $\Delta_O \subseteq \Delta_I$ is not necessarily satisfied, which deviated substantially from the backchaining proof theory provided for LHHF in [16]. In our case, Δ_O might have sub-formulas of certain resources in Δ_I, which is the conclusion of enabling simultaneous conjunction on left side of a formula. This observation is important in proving the soundness and completeness of the BL proof theory.

Fig. 11 Residuation rules for atomic goals for the BL proof system for LPL

$$\frac{\mathrm{unify}(a', a)}{u : {}^{L}(a') \gg a \setminus \cdot \triangleright \cdot} \textbf{ d-atm}$$

$$\frac{v : {}^{L\cup\{v\}}D \gg a \setminus \Delta_G \triangleright \Delta_O}{u : {}^{L}(G_1 \multimap D) \gg a \setminus (j : G_1^{Lb(\Delta_O)}), \Delta_G \triangleright \Delta_O} \textbf{ d-}\multimap$$

$$\frac{v_1 : {}^{L\cup\{v_1\}}D_1 \gg a \setminus \Delta_G \triangleright \Delta_O}{u : {}^{L}(D_1 \otimes D_2) \gg a \setminus \Delta_G \triangleright \Delta_O, (v_2 : {}^{L\cup\{v_2\}}D_2)} \textbf{ d-}\otimes_1$$

$$\frac{v_1 : {}^{L\cup\{v1\}}D_2 \gg a \setminus \Delta_G \triangleright \Delta_O}{u : {}^{L}(D_1 \otimes D_2) \gg a \setminus \Delta_G \triangleright \Delta_O, (v_2 : {}^{L\cup\{v_2\}}D_1)} \textbf{ d-}\otimes_2$$

$$\frac{v : {}^{L\cup\{v\}}([t/x]D) \gg a \setminus \Delta_G \triangleright \Delta_O}{u : {}^{L}(\forall x.\ D) \gg a \setminus \Delta_G \triangleright \Delta_O} \textbf{ d-}\forall$$

References

1. Arslan, O., Saranli, U.: Reactive planning and control of planar spring-mass running on rough terrain. IEEE Trans. Robot. **28**(3), 567–579 (2012)
2. Belta, C., Bicchi, A., Egerstedt, M., Frazzoli, E., Klavins, E., Pappas, G.: Symbolic planning and control of robot motion - grand challenges of robotics. IEEE Robot. Autom. Mag. **14**(1), 61–70 (2007)
3. Bimbó, K.: The decidability of the intensional fragment of classical linear logic. Theor. Comput. Sci. **597**, 1–17 (2015)
4. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1), 1636–1642 (1995)
5. Brachman, R., Levesque, H.: Knowledge Representation and Reasoning. Morgan Kaufmann Publishers Inc., San Francisco (2004)
6. Cervesato, I., Hodas, J.S., Pfenning, F.: Efficient resource management for linear logic proof search. Theor. Comput. Sci. **232**(1-2), 133–163 (2000)
7. Chrpa, L.: Linear logic in planning. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 26–29 (2006)

8. Conner, D., Choset, H., Rizzi, A.: Flow-through policies for hybrid controller synthesis applied to fully actuated systems. IEEE Trans. Robot. **25**(1), 136–146 (2009)
9. Dimopoulos, Y., Nebel, B., Koehler, J.: Encoding planning problems in nonmonotonic logic programs. In: Steel, S., Alami, R. (eds.) Recent Advances in AI Planning, Lecture Notes in Computer Science, pp. 169–181. Springer (1348)
10. Dixon, L., Bundy, A., Smaill, A.: Verified planning by deductive synthesis in intuitionistic linear logic. In: Proceedings of the Workshop on Verification and Validation of Planning and Scheduling Systems, p. 10 (2009)
11. Dixon, L., Smaill, A., Tsang, T.: Plans, actions and dialogue using linear logic. J. Log. Lang. Inf. **18**, 251–289 (2009)
12. Do, M.B., Kambhampati, S.: Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. Artif. Intell. **132**(2), 151–182 (2001)
13. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Answer set planning under action costs. J. Artif. Intell. Res. **19**, 25–71 (2003)
14. Girard, J.Y.: Linear logic. Theor. Comput. Sci. **50**(1), 1–102 (1987)
15. Gupta, N., Nau, D.S.: On the complexity of blocks-world planning. Artif. Intell. **56**(2-3), 223–254 (1992)
16. Hodas, J.S., Miller, D.: Logic programming in a fragment of intuitionistic linear logic. Inf. Comput. **110**(2), 327–365 (1994)
17. Jacopin, E.: Classical AI planning as theorem proving: the case of a fragment of linear logic. In: AAAI Fall Symposium on Automated Deduction in Nonstandard Logics, Palo Alto, CA (1993)
18. Kahramanogullari, O.: Towards planning as concurrency. In: Artificial Intelligence and Applications, pp 387–393 (2005)
19. Kahramanoğulları, O.: On linear logic planning and concurrency. In: Language and Automata Theory and Applications, pp. 250–262. Springer (2008)
20. Kanovich, M., Vauzeilles, J.: The classical AI planning problems in the mirror of horn linear logic: semantics, expressibility, complexity. Math. Struct. Comput. Sci. **11**(6), 689–716 (2001)
21. Kanovich, M.I., Vauzeilles, J.: Linear logic as a tool for planning under temporal uncertainty. Theor. Comput. Sci. **412**(20), 2072–2092 (2011)
22. Kortik, S., Saranli, U.: Linear planning logic: an efficient language and theorem prover for robotic task planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3764–3770 (2014)
23. Kungas, P.: Linear Logic Programming for AI Planning. M.Sc., Institute of Cybernetic at Tallinn Technical University (2002)
24. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
25. Lee, J., Palla, R.: Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. J. Artif. Intell. Res. **43**, 571–620 (2012)
26. Lifschitz, V.: Answer set planning. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Computer Science, vol. 1730, pp. 373–374. Springer (1999)
27. Lopez, A., Bacchus, F.: Generalizing GraphPlan by formulating planning as a CSP. In: Proceedings of the International Joint Confernece on Artificial Intelligence, pp. 954–960. Morgan Kaufmann Publishers Inc., San Francisco (2003)
28. Lozano-Perez, T.: Task planning. In: Robot Motion Planning and Control, pp. 473–498. MIT Press, Cambridge (1982)
29. Mantel, H., Otten, J.: lintap: a tableau prover for linear logic. In: Automated Reasoning with Analytic Tableaux and Related Methods, pp. 661–661 (1999)
30. Martínez, D., Alenyáá, G., Torras, C.: Planning robot manipulation to clean planar surfaces. Eng. Appl. Artif. Intell. **39**, 23–32 (2015)
31. Masseron, M., Tollu, C., Vauzeilles, J.: Generating plans in linear logic I. Actions as proofs. Theor. Comput. Sci. **113**(2), 349–370 (1993)
32. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI-91), pp. 634–639 (1991)
33. McCarthy, J.: Situations, Actions, and Causal Laws. Tech. Rep AD0785031. Stanford University, Department of Computer Science (1963)
34. Nau, D.S.: Current trends in automated planning. AI Mag. **28**(4), 43 (2007)
35. Penberthy, J.S., Weld, D.S.: UCPOP: a sound, complete, partial order planner for ADL. In: Proceedings of the International Conference on Knowledge Representation and Reasoning, pp. 103–114 (1992)
36. Pfenning, F.: Lecture Notes on Automated Theorem Proving. Technical report, Carnegie Mellon University (1999)
37. Pfenning, F.: Lecture Notes on Linear Logic. Technical report, Carnegie Mellon University (2002)
38. Reiter, R.: The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. Artif. Intell. Math. Theory Comput. **27**, 359–380 (1991)
39. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Englewood Cliffs (2003)
40. Saranli, U., Buehler, M., Koditschek, D.E.: RHex: a simple and highly mobile robot. Int. J. Robot. Res. **20**(7), 616–631 (2001)
41. Tamura, N.: User's Guide of a Linear Logic Theorem Prover (Llprover). Tech. Rep., Department of Computer and Systems Engineering Faculty of Engineering. Kobe University, Japan (1998)
42. Thompson, S.: Type Theory and Functional Programming. Addison-Wesley, Reading (1991)
43. Vaquero, T.S., Silva, J.R., Beck, J.C.: Post-design analysis for building and refining AI planning systems. Eng. Appl. Artif. Intell. **26**(8), 1967–1979 (2013)

Sitar Kortik received his B.Sc. degree in Computer Engineering from Dokuz Eylül University, İzmir, Turkey in 2006 and his M.Sc. and Ph.D. degrees in Computer Engineering from Bilkent University, Ankara, Turkey in 2010 and 2017, respectively. He was a visiting researcher at the Computer Science Department, Carnegie Mellon University, USA in 2012. Currently, he is working as a research associate in Fraunhofer Institute for Manufacturing Engineering and Automation (IPA), Robot and Assistive Systems Department, Stuttgart, Germany. His research interests include artificial intelligence, automated reasoning, robotic task planning, industrial robotics, applications of linear logic and theorem proving to domain independent task planning and multiagent systems.

Uluc Saranli received his B.Sc. degree in Electrical and Electronics Engineering from the Middle East Technical University, Ankara, Turkey in 1996 and his M.Sc. and Ph.D. degrees in Computer Science from the University of Michigan, Ann Arbor in 1998 and 2002, respectively. He then joined the Robotics Institute in Carnegie Mellon University as a postdoctoral fellow until 2005. Subsequently, he served as an Assistant Professor in the Department of Computer Engineering in Bilkent University, Ankara, Turkey until 2011, after which he joined the Department of Computer Engineering in the Middle East Technical University as an Associate Professor. His research interests include the analysis and control of dynamic locomotion with legged robots, nonlinear dynamical systems, embedded systems, software architectures for robot programming and control and formal methods applied to planning and robotic autonomy.