Implementing Efficient Resource Management for Linear Logic Programming (Preliminary Version)

Pablo López and Jeff Polakow

Universidad de Málaga and National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Verification and Semantics (CVS), CREST, Japan Science and Technology Agency (JST)

Implementing Efficient Resource Management for Linear Logic Programming

Pablo López[1] and Jeff Polakow[2]

[1] Universidad de Málaga, lopez@lcc.uma.es
[2] National Institute of Advanced Industrial Science and Technology (AIST), Research Center for Verification and Semantics (CVS), CREST, Japan Science and Technology Agency (JST), j-polakow@aist.go.jp**

Abstract. The Tag-Frame system of resource management [1] reunited two divergent threads of linear logic programming research by achieving the efficient proof search behaviour of abstract systems, such as [2], while using a low-level tag-based approach, as in [3], suitable for specifying an abstract machine. However, Tag-Frame relies on set operations which are linear in the size of the sets. Furthermore, as noted in [4], Tag-Frame is not as efficient as it could be. We present a new tag-based derivation system which relies solely on low-level concepts to implement efficient resource management, where most linear time operations have been replaced by constant time ones. Though motivated and informed by the Tag-Frame system, we derive our system directly from, and prove its correctness with respect to the system of Cervesato et al. [2]. An abstract machine based on the new system has been implemented by Tamura and Banbara [5], and its performance compared to their previous machine [6].

1 Introduction

In the early nineties, work by Andreoli [7], and Hodas and Miller [8] showed that *goal-directed, focussed* proof search is complete for a fragment of linear logic [9]. Although this result qualified linear logic as an *abstract logic programming language* [10], the enormous amount of nondeterminism in goal-directed proof search, due to the need to split resources (linear hypotheses) between premises, rendered a naive implementation useless. Hodas and Miller [8] also introduced the I/O system, which Hodas further refined to the $I/O_{\top}$ system [11] to implement the first feasible method of managing resources; this work gave birth to Lolli, a linear logic programming language which conservatively extends λProlog [12].

Subsequent work on improving the implementation of Lolli[3] split into two general categories: refining resource management strategies to further remove nondeterminism from proof search [2, 13]; and investigating low-level implementation techniques suitable for extending the WAM [3, 6, 14]. The work on resource management largely succeeded in removing unnecessary nondeterminism, at least that related to linearity, from proof search. However, the proposed

** Work carried out while a JSPS Fellow at Kobe University.
[3] For this paper we only consider pure Lolli, i.e. no extra-logical operations.

systems rely upon high-level operations such as context intersection and union which are linear in the size of the contexts. On the other hand, the work on compilation associates tags with each resource and then maintains linearity constraints by manipulating individual formula tags, rather than whole contexts. This approach produced very fast implementations, as evidenced by the Lolli-Cop theorem prover [15], but did not manage to capture the more ideal proof search behavior of the abstract systems– in particular, these systems lack the notion of strict resources and thus may diverge where the more abstract systems will simply fail.

These divergent research paths were reunited with the Tag-Frame system of Hodas et al. [1] which captures the behavior of the resource management systems using the low-level techniques of compiled implementations. Specifically, the system abstracts the linear context into a set of tags, each representing some portion of the context. Linearity constraints are then enforced by carefully keeping track of which tags may and must be consumed at any point in the proof. However, as argued by Polakow in [4], the machinery for managing tags presented in [1] both requires more work than necessary and obfuscates the proof search algorithm.

This paper presents a new proof search system for Lolli which improves upon the Tag-Frame system in efficiency. Although informed by our experience with the Tag-Frame system, we derive our new system, which we call $\mathcal{TF}\!\!\not\downarrow$, directly from RM_3, the efficient resource management system of Cervesato et al. [2].

The rest of this paper is organized as follows. Section 2 formally introduces the formula language of Lolli. Section 3 reviews developments in resource management up to the Tag-Frame system. Section 4 presents an inefficient, tag-based version of RM_3 which illustrates the main intuition behind $\mathcal{TF}\!\!\not\downarrow$ and serves as an intermediate system in our proof of correctness. Sections 5 refines our intermediate system to a new, very efficient system based on the intuition that tags are pointers. Section 6 presents one further refinement which results in $\mathcal{TF}\!\!\not\downarrow$, our final system. Section 7 presents LLP-TF, an abstract machine based on $\mathcal{TF}\!\!\not\downarrow$ and discusses its performance. Finally, section 8 offers some conclusions and further work.

2 Formulas and Residuation

We now formally introduce our formula language. Due to space limitations, we limit ourselves to a fragment of Lolli sufficient to illustrate all the features of our system; our system easily extends to the full formula language of Lolli. We divide our formulas into *goal* formulas, G, which may appear as goals, and *definite clause* formulas, D, which may appear as hypotheses. The grammars for goal and clause formulas are as follows:

$$\begin{array}{lcl} G & ::= & A \mid D \multimap G \mid G_1 \,\&\, G_2 \mid \top \mid G_1 \otimes G_2 \mid \mathbf{1} \\ D & ::= & A \mid G \multimap D \mid \top \end{array}$$

where A represents atomic predicates. Note that D is restricted to asynchronous [7] formulas which allows goal-directed proof search to be complete.

In the usual presentation of sequent systems, there are right rules which act on the goal formula, and left rules which act upon hypotheses. In goal-directed proof search, we only use right rules when the goal is non-atomic. When solving an atomic goal, we choose a hypothesis, whose head matches (unifies with) our goal, to focus on, i.e. to apply left rules generating new goals.

As shown by Cervesato [16], the new goals which will be generated by focussing on a hypothesis may be eagerly generated by *residuating*, or "logically compiling," the hypothesis into one new goal formula. We note that this basic idea underlies the earlier notion of a *backchaining* rule present in the original presentation of Lolli [8]. For our presentation, we opt to dispense with left rules in favor of residuation.

We make use of the following judgement to residuate a clause formula, D, whose head matches a given atom, A, into a goal formula, G:

$$D \gg A \diagdown G$$

The rules for residuation are as follows (where $\doteq$ signifies unification.):

$$\frac{A' \doteq A}{A' \gg A \diagdown \mathbf{1}} \qquad \frac{D \gg A \diagdown G}{G' \multimap D \gg A \diagdown G \otimes G'} \qquad \text{(no rule for } \top\text{)}$$

3 A Review of Resource Management

The chief source of nondeterminism in linear logic programming (after restricting to goal-directed, focussed proof search) arises from the need to distribute resources among the premises of multiplicative rules. When designing Lolli, Hodas and Miller realized that resources may be lazily distributed during proof search; when solving a goal with two premises, *all* currently available resources can be given to the first premise and those left-over can be further passed to the second premise. The I/O system [8] follows this intuition and its sequents have the form:

$$\Delta_I \diagdown \Delta_O \Longrightarrow G$$

where Δ_I contains the input linear hypotheses while Δ_O contains the output linear hypotheses, G is the goal to be proved.

In the I/O system, the actual linear context consumed by the derivation corresponds to $\Delta_I - \Delta_O$. Thus we can write the $\otimes$ rule as:

$$\frac{\Delta_I \diagdown \Delta_M \Longrightarrow G_1 \qquad \Delta_M \diagdown \Delta_O \Longrightarrow G_2}{\Delta_I \diagdown \Delta_O \Longrightarrow G_1 \otimes G_2} \otimes_R$$

Linearity constraints are maintained by checking that the output context does not contain a formula which must be consumed, i.e.:

$$\frac{(\Delta_I \cup \{D\}) \diagdown \Delta_O \Longrightarrow G \qquad D \notin \Delta_O}{\Delta_I \diagdown \Delta_O \Longrightarrow D \multimap G} \multimap_R$$

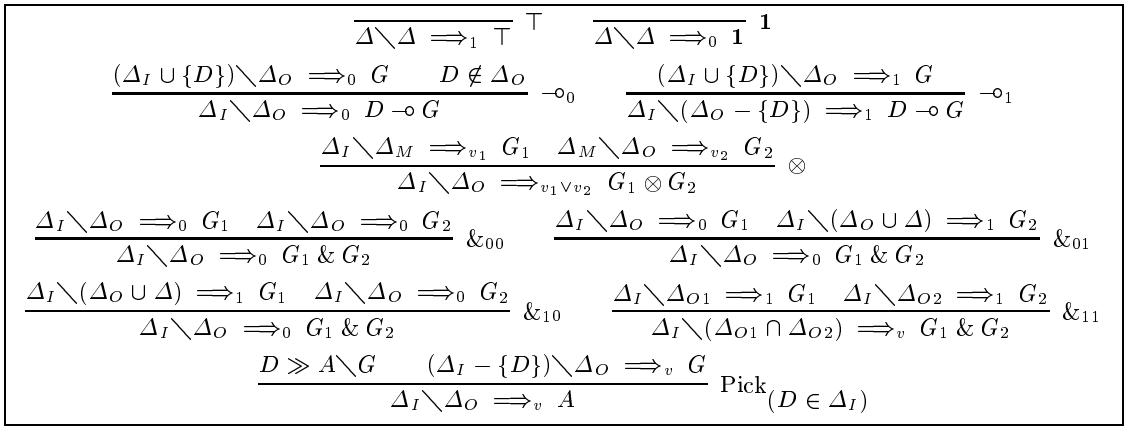

Fig. 1. The I/O$_\top$ system

However, rather than remove the nondeterminism, the idea of lazy splitting only pushes it into the $\top$ (additive unit) rule which can, in this setting, consume an arbitrary subset of the input:

$$\frac{\Delta_I \supseteq \Delta_O}{\Delta_I \setminus \Delta_O \implies \top}\ \top_R$$

3.1 The I/O$_\top$ System

Hodas removed nondeterminism from the $\top$ rule by creating the I/O$_\top$ system [11]. This system decorates sequents with a boolean flag indicating whether a $\top$ was encountered which could implicitly consume extra resources. Thus I/O$_\top$ sequents have the form:

$$\Delta_I \setminus \Delta_O \implies_v G$$

where v is a boolean, the $\top$ flag, and everything else is the same as in I/O.

In the I/O$_\top$ system, the $\top$ rule becomes lazy; rather than consume any of its input, it simply sets the $\top$ flag. When the $\top$ flag is set, linearity constraints may be relaxed since any unconsumed resources could have been consumed by the $\top$. Figure 1 presents the derivation rules for I/O$_\top$ adapted to use residuation and our syntax.

While the I/O$_\top$ system removes much of the nondeterminism of resource distribution, its direct implementation as a logic programming interpreter allows needless backtracking due to the inefficient treatment of & (additive conjuction). Since both premises of & must consume exactly the same resources, giving the same input context to both premises potentially creates useless choice points in the derivation of the second premise.

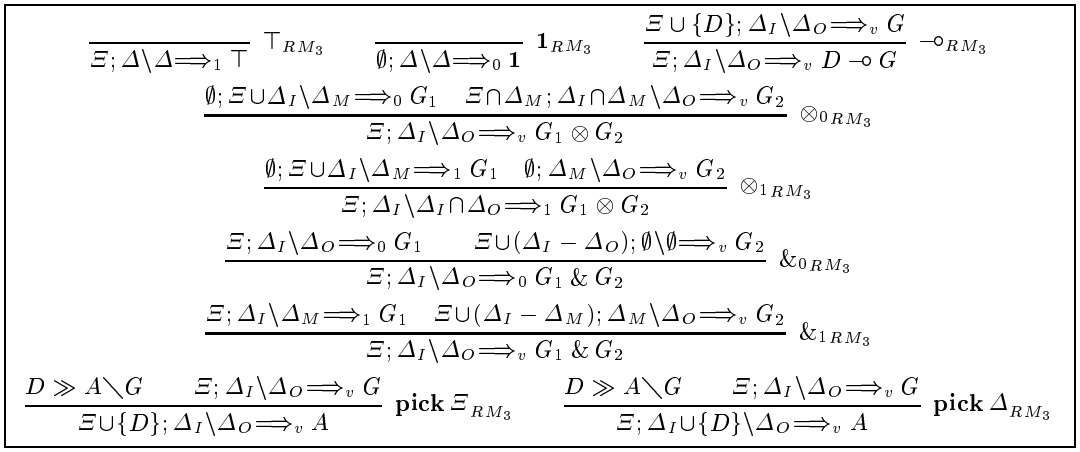

Fig. 2. The RM_3 system

3.2 The RM_3 System

The RM_3 system of Cervesato et al. [2] extends I/O$_\top$ with the ability to keep track of exactly which resources must be consumed at any point in the proof. RM_3 is characterized by the following judgement

$$\Xi; \Delta_I \backslash \Delta_O \Longrightarrow_v G$$

where: Ξ is a set of *strict* linear clause formulas; Δ_I, Δ_O are sets of *lax* linear clause formulas, the input and output contexts respectively; v is the $\top$ flag; and G is the goal formula being derived. All linear hypotheses, i.e. the contents of Ξ, Δ_I, Δ_O, are implicitly uniquely labelled so that we can distinguish different occurrences of the same formula in a set. We assume new labels are tacitly generated as formulas are added to Ξ. Figure 2 gives a version of the derivation rules for RM_3 slightly modified to use residuation.

The basic idea behind RM_3 is that strict resources *must* be consumed while lax resources *may* be consumed. Because we have a lazy $\top_R$ rule, resources can either be explicitly consumed in the **pick** rules, or implicitly consumed in the $\top_R$ rule. Thus Ξ is required to be empty in the $\mathbf{1}_R$ rule which also sets the $\top$ flag to 0. Furthermore, by adding new resources to Ξ in the $\multimap_R$ rule, we guarantee they will be consumed and have no need to check the output context regardless of the $\top$ flag's value. The formulation of the remaining rules serves to insure accurate bookeeping of strict resources.

The RM_3 system specifies a proof search behavior which is very appropriate for linear logic programming– at least for goal-directed, left-to-right proof search. However, the context intersections in the $\otimes$ rules, and context differences in the & rules, render a direct implementation of the system extremely slow. A more concrete, lower-level specification is needed for efficient implementation. In sections 4 through 6 we successively refine RM_3 into such a system. However, we first discuss two earlier refinements of RM_3 which inform our work.

3.3 Frames and Tag-Frames

Upon closer analysis, the strict formulas of RM_3 can be seen to follow a stack-like behavior. The Frame system, $\mathcal{F}$, of López and Pimentel [13], exposes and exploits this behavior to improve upon RM_3's efficiency. In $\mathcal{F}$, the linear contexts, Δ_I and Δ_O, are actually stacks of strict contexts, called frames. This representation alleviates the need for context intersection, in the $\otimes$ rules, to disentangle strict and lax resources. For reference, appendix A contains a presentation of the $\mathcal{F}$ system.

Although significantly more concrete, and efficient, than RM_3, the $\mathcal{F}$ system remains too abstract for direct implementation as a logic programming language. In addition to relying upon context-wide comparisons, i.e. context difference, the system does not maintain the order of linear hypotheses. Such information is necessary to implement the depth first search operational semantics of Prolog-style languages.

To improve the treatment of &, and maintain context order, Hodas et al. created the $\mathcal{TF}$ system [1] which employs low-level implementation techniques of [3] to capture the proof search behavior of $\mathcal{F}$ (and RM_3). Specifically, $\mathcal{TF}$ assigns tags to each resource and then keeps track of exactly which tags must be consumed, i.e. are strict, in the current sub-proof. This requires new delete tags to be generated in the & rules to identify resources consumed in the first premise. Consumption of a resource amounts to changing the resource's tag, rather than explicitly removing the formula from the linear context. The use of individual formula tags allows strict resources to be identified while maintaining the order of linear hypotheses. For reference, appendix B contains a presentation of the $\mathcal{TF}$ system.

Although the $\mathcal{TF}$ could be used as the basis of an abstract machine for Lolli, it is not the ideal candidate. The system still relies upon set membership and set union to manage linearity constraints and requires scanning the linear context[4] in the **1** rule as well as scanning the list of available tags in the **pick** rule. These operations are not only linear in time, but the machinery for managing tags is heavier than need be; see [4] for an alternative formulation.

4 Naive Tag-Based System

We start our development of a low-level and efficient resource management specification with a simple and inefficient tag-based system which captures the same proof search behavior of RM_3. This system, $\mathcal{TF}\!\!\Join$, which maintains context order, ensures that every strict input formula is marked as consumed in the output. The system will use lists of tagged clause formulas to represent linear contexts. We respectively use nil, :: and @ for the empty list, list constructor, and list append function. We also assume each linear hypothesis is implicitly uniquely

[4] This could be optimized to just scanning the strict frame if tags are implemented as pointers to counters.

labelled so that we may distinguish between different occurrences of the same formula. We assume that new labels are generated as the linear context expands.

We make use of the following two functions on lists of tagged formulas:

$$\begin{array}{rcl@{\qquad}rcl} [\mathrm{nil}]_t & = & \emptyset & \mathbb{M}(s,t,\mathrm{nil}) & = & \mathrm{nil} \\ [D^t :: \Delta]_t & = & \{D\} \cup [\Delta]_t & \mathbb{M}(s,t,(D^s :: \Delta)) & = & D^t :: \mathbb{M}(s,t,\Delta) \\ [D^s :: \Delta]_t & = & [\Delta]_t \quad (\text{if } s \neq t) & \mathbb{M}(s,t,(D^{s'} :: \Delta)) & = & D^{s'} :: \mathbb{M}(s,t,\Delta) \quad (\text{if } s \neq s') \end{array}$$

$[\Delta]_t$ returns the set of all formulas in Δ tagged with t. $\mathbb{M}(s,t,\Delta)$ explicitly changes all occurrences in Δ of tag s to t.

$\mathcal{TFM}$ sequents have the form

$$\Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$$

where: Δ_I, Δ_O are lists containing tagged clause formulas, the input and output contexts; s is the strict tag; π is a list of available tags; d is the delete tag; v is the $\top$ flag; and G is the goal formula being derived. Note that list $s :: \pi$ is manipulated as a stack. The basic intuition is that input formulas tagged with s are strict, those tagged with $t \in \pi$ are lax[5], and those with other tags are not available for consumption; furthermore, all formulas consumed in the derivation will be tagged with d in the output. Figure 3 presents the derivation rules for $\mathcal{TFM}$.

4.1 Correctness of $\mathcal{TFM}$

We now prove $\mathcal{TFM}$ sound and complete with respect to RM_3. In the following statements, we assume that all variables are universally quantified at the outermost level, unless explicitly noted otherwise.

We make use of the following function:

$$\begin{array}{rcl} [\![\Delta]\!]_{\mathrm{nil}} & = & \emptyset \\ [\![\Delta]\!]_{t::\pi} & = & [\Delta]_t \cup [\![\Delta]\!]_\pi \end{array}$$

$[\![\Delta]\!]_\pi$ returns the set of all formulas in Δ whose tag is in π.

We begin with a lemma that relates the input and output contexts of a $\mathcal{TFM}$ provable sequent.

Lemma 1 (Properties of $\mathcal{TFM}$).
$\Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *implies*

1. $[\Delta_O]_s = \emptyset$
2. $[\![\Delta_I]\!]_\pi \supseteq [\![\Delta_O]\!]_\pi$
3. $[\Delta_O]_d = [\Delta_I]_s \cup [\Delta_I]_d \cup ([\![\Delta_I]\!]_\pi - [\![\Delta_O]\!]_\pi)$

whenever $s \neq d$, $s \notin \pi$, $d \notin \pi$ and all tags in π are unique.

[5] We overload $\in$ to mean list membership as well as set membership.

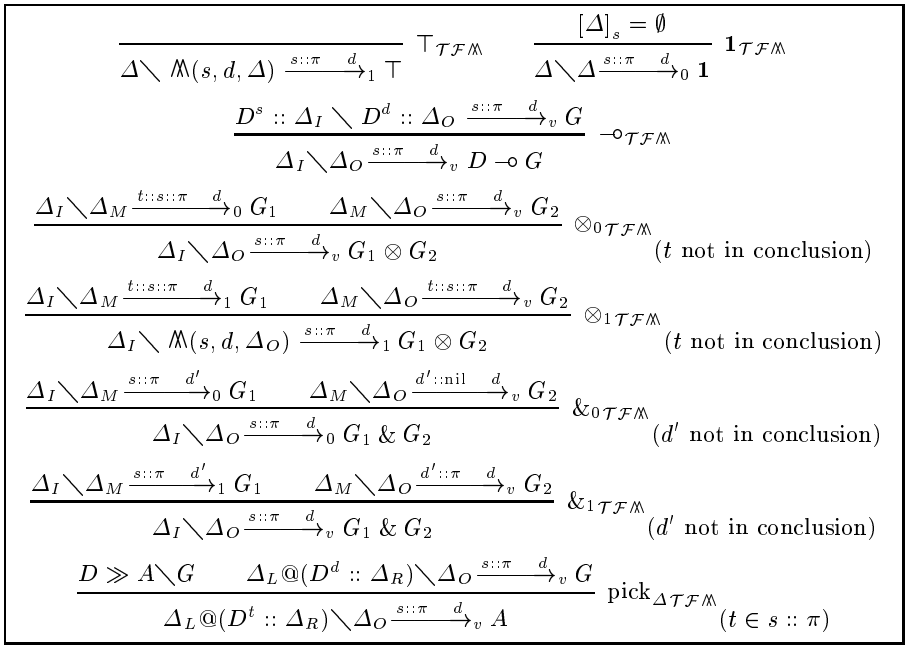

Fig. 3. The $\mathcal{TFM}$ system

Proof. By induction on the structure of the given derivation. See [17] for details.

Part 1 states that there are no strict resources in the output. Part 2 states that the optional output is a subset of the optional input. Finally, part 3 states what resources are marked as consumed in the output.

The proof of correctnes is divided into two parts, a soundness result and a completeness result.

Theorem 1 (Soundness with respect to RM_3).
$\Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *implies*
$[\Delta_I]_s ; [\![\Delta_I]\!]_\pi \setminus [\![\Delta_O]\!]_\pi \Longrightarrow_v G$
whenever $s \neq d$, $s \notin \pi$, $d \notin \pi$ *and all tags in* π *are unique.*

Proof. By induction on the structure of the given derivation, making use of lemma 1. See [17] for details.

Theorem 2 (Completeness with respect to RM_3).
$\Xi; \Delta_I \setminus \Delta_O \Longrightarrow_v G$ *and* $[\Delta'_I]_s = \Xi$ *and* $[\![\Delta'_I]\!]_\pi = \Delta_I$ *implies*
$\exists \Delta'_O . \Delta'_I \diagdown \Delta'_O \xrightarrow{s::\pi \quad d}_v G$ *and* $[\![\Delta'_O]\!]_\pi = \Delta_O$
whenever $s \neq d$, $s \notin \pi$, $d \notin \pi$ *and all tags in* π *are unique.*

Proof. By induction on the structure of the given derivation, making use of lemma 1. See [17] for details.

4.2 Tags as Counters

Exactly four rules of $\mathcal{TFM}$ require (non-constant time) work. The **1** rule requires scanning the context for tag s. The $\top$ rule must traverse the context and explicitly change all the strict tags to delete tags. Likewise, since s is not strict in either premise, the $\otimes_1$ rule must explicitly consume any resources tagged with s leftover from the derivations of its premises. Finally, since consumed formulas remain in the context, the **pick** rule must check that the chosen formula is available for consumption, i.e. that t occurs in $s :: \pi$.

By implementing tags as counters, we can turn the strictness check in the **1** rule to a constant time memory lookup operation. In such an implementation, each tag would be a reference to a memory location containing the number of formulas marked with that tag. The implementations of the $\multimap$ and **pick** rules, as well as the $\mathbb{M}(s, t, \Delta)$ function, would manipulate the counters appropriately to maintain the representation.

5 Indirect Tag-Based System

Suppose we have three tags, s, t and d, and a context $\Delta = D_1^s, D_2^s, D_3^t, D_4^d$ which we want to change to $\Delta' = D_1^d, D_2^d, D_3^t, D_4^d$. Then, letting the tags be memory addresses, we wish to make the following transformation:

By building on the intuition that tags are memory locations, it is possible to effectively accomplish the preceding transformation in constant time by using indirection and considering tags as memory locations which contain either a natural number or another memory location:

If we modify memory lookup to chase down alias chains, s and d will effectively represent the same tag.

In other words, we would like to maintain equivalence classes of tags. The machinery we subsequently add to our derivation rules may be seen as the implementation of a union-find structure.

5.1 Memory and Partial Functions

In order to formalize and prove correct the previous ideas, we will model memory as a partial function from a countably infinite set of locations, $\mathcal{L}$, to the disjoint

union of all possible value types. For our purposes, we will only consider value types to be natural numbers, $\mathbb{N}$, locations, and pairs.

We understand a partial function, σ, from $\mathcal{A}$ to $\mathcal{B}$ to be a set of pairs, (x, y) where $x \in \mathcal{A}$ and $y \in \mathcal{B}$, for which there is *at most one* $b \in \mathcal{B}$ for every $a \in \mathcal{A}$ such that $(a, b) \in \sigma$. We use the following functional notations:

$$\sigma(a) \quad = \quad \begin{cases} b & \text{if } \exists b.\ (a,b) \in \sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\sigma[a := b] \quad = \quad \begin{cases} (\sigma - (a, b')) \cup (a, b) & \text{if } \exists b'.\ \sigma(a) = b' \\ \sigma \cup (a, b) & \text{otherwise} \end{cases}$$

$$\sigma \setminus a \quad = \quad \begin{cases} \sigma - (a, b') & \text{if } \exists b'.\ \sigma(a) = b' \\ \sigma & \text{otherwise} \end{cases}$$

which correspond to memory lookup, memory modification (and allocation), and memory deallocation. Note that our presentation is purely declarative. $\sigma[a := b]$ does not change σ but rather stands for the modified partial function; thus $\sigma[a := \sigma(b)][b := \sigma(a)] \ = \ \sigma[b := \sigma(a)][a := \sigma(b)]$

We make use of the following syntax to denote the domain and codomain of a partial function from $\mathcal{A}$ to $\mathcal{B}$:

$$\mathrm{dom}(\sigma) \ = \ \{a \in \mathcal{A} \mid \exists b \in \mathcal{B}.\ \sigma(a) = b\} \qquad \mathrm{cod}(\sigma) \ = \ \{b \in \mathcal{B} \mid \exists a \in \mathcal{A}.\sigma(a) = b\}$$

5.2 Aliasing and Memory Lookup

For $\mathcal{TF}+$, formula tags will be locations $l \in \mathcal{L}$. Intuitively, these locations are memory addresses which store natural numbers. However, since we want to be able to alias one tag to another, we will model memory as a partial function Σ from $\mathcal{L}$ to $\mathbb{N} \uplus \mathcal{L}$ where $\Sigma(l) \in \mathcal{L}$ implies l is aliased to some other memory location.

We use an explicit lookup judgement to find the current alias of a given tag. In order to improve overall efficiency, we will update the Σ function to point to the current (i.e. last in the chain) alias whenever we have to lookup the value of a location. Our lookup judgement has the following form

$$\Sigma_I \diagdown \Sigma_O \models l \rightsquigarrow l'$$

where: Σ_x are partial functions of type $\mathcal{L} \to \mathbb{N} \uplus \mathcal{L}$; and l, l' are elements of $\mathcal{L}$. The derivation rules for the judgement are as follows:

$$\frac{\Sigma(l) \in \mathbb{N}}{\Sigma \diagdown \Sigma \models l \rightsquigarrow l}\ \text{value} \qquad \frac{\Sigma_I(l) \in \mathcal{L} \qquad \Sigma_I \diagdown \Sigma_O \models \Sigma_I(l) \rightsquigarrow l'}{\Sigma_I \diagdown \Sigma_O[l := l'] \models l \rightsquigarrow l'}\ \text{alias}$$

We use the following notational conveniences, where $l \in \mathcal{L}$ and $n \in \mathbb{N}$, for common memory operations:

$$\Sigma[l + n] \quad = \quad \begin{cases} \Sigma[l := n' + n] & \text{if } \exists n' \in \mathbb{N}.\ \Sigma(l) = n' \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\Sigma[l - n] \quad = \quad \begin{cases} \Sigma[l := n' - n] & \text{if } \exists n' \in \mathbb{N}.\ \Sigma(l) = n' \text{ and } n' \geq n \\ \text{undefined} & \text{otherwise} \end{cases}$$

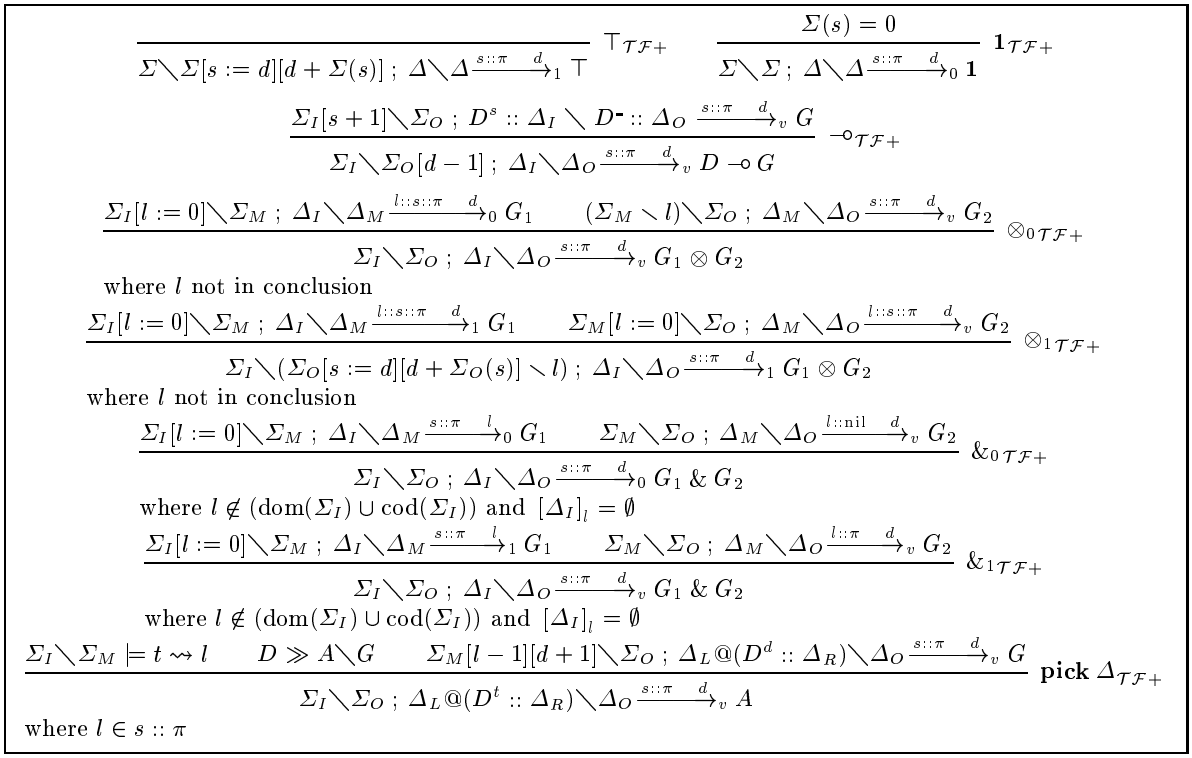

Fig. 4. The $\mathcal{TF}+$ system

5.3 $\mathcal{TF}+$ Sequents

$\mathcal{TF}+$ sequents are of the form

$$\Sigma_I \diagdown \Sigma_O \;;\; \Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$$

where: Σ_I, Σ_O are partial functions of type $\mathcal{L} \to \mathbb{N} \uplus \mathcal{L}$, representing the state of memory before and after the derivation; and everything else remains unchanged from $\mathcal{TFM}$, unique labelling included. It is assumed that $s \neq d$, $s \notin \pi$, and $d \notin \pi$ Figure 4 contains the derivation rules for $\mathcal{TF}+$.

5.4 Correctness of $\mathcal{TF}+$

We now prove $\mathcal{TF}+$ correct with respect to $\mathcal{TFM}$. In all logical statements, all variables are implicitly universally bound at the outermost level unless explicitly stated otherwise. We introduce the following notation:

$$\begin{array}{rcll} \Sigma(\text{nil}) & = & \text{nil} & \\ \Sigma(D^t :: \Delta) & = & D^l :: \Sigma(\Delta) & \text{where} \quad \Sigma \diagdown \_ \models t \rightsquigarrow l \end{array}$$

to apply all aliases to a contex; and:

$$\begin{array}{rcll} \text{nil} - \text{nil} & = & \emptyset & \\ (D^t :: \Delta) - (D^t :: \Delta') & = & \Delta - \Delta' & \\ (D^t :: \Delta) - (D^{t'} :: \Delta') & = & \{t'\} \cup (\Delta - \Delta') & \text{where} \quad t \neq t' \end{array}$$

for the tag difference of two contexts. Additionally, $\#(\mathcal{S})$ denotes the cardinality of a (multi)set $\mathcal{S}$.

For a $\mathcal{TF}+$ sequent to be provable, Σ_I must map every tag to an appropriate counter, possibly through a chain of aliases. We formalize this as follows:

Definition 1 (Well-Formedness).
$\Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *is* **well-formed** *iff*

1. $\forall t \in \mathrm{dom}(\Sigma_I). \exists l \in \mathrm{dom}(\Sigma_I).\ \Sigma_I \diagdown_- \models t \rightsquigarrow l$ *and* $\Sigma_I(l) = \#([\Sigma_I(\Delta_I)]_l)$
2. $\forall t \in d :: s :: \pi.\ \Sigma_I \diagdown_- \models t \rightsquigarrow t$

A derivation is well-formed iff every sequent is well-formed. We now state some fundamental properties of well-formed derivations.

Lemma 2 (Properties of $\mathcal{TF}+$).
A well-formed derivation of $\Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *implies all of the following:*

1. $\forall t \in \mathrm{dom}(\Sigma_O). \exists l \in \mathrm{dom}(\Sigma_O).\ \Sigma_O \diagdown_- \models t \rightsquigarrow l$ *and* $\Sigma_O(l) \ = \ \#([\Sigma_O(\Delta_O)]_l)$
2. $[\Sigma_O(\Delta_O)]_s = \emptyset$
3. $(\Sigma_I(\Delta_I) - \Sigma_O(\Delta_O)) \subseteq \{d\}$

Proof. By structural induction on the given derivation. See [17] for details.

Part 1 states that well-formedness is preserved. Part 2 states that the output contains no strict resources. Part 3 states that a tag in the input can only be changed to denote consumption.

We are now in a position to prove $\mathcal{TF}+$ correct. We break the proof into soundness and completeness results.

Theorem 3 (Soundness with respect to $\mathcal{TF}\mathbb{M}$).
A well-formed derivation $\Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *implies*
$\Sigma_I(\Delta_I) \diagdown \Sigma_O(\Delta_O) \xrightarrow{s::\pi \quad d}_v G$

Proof. By structural induction on the given derivation, making use of lemma 2. See [17] for details.

Theorem 4 (Completeness with respect to $\mathcal{TF}\mathbb{M}$).
$\Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$ *implies*
$\exists \Sigma'_O, \Delta'_O.\ \Sigma'_O(\Delta'_O) = \Delta_O$ *and* $\Sigma'_I \diagdown \Sigma'_O \; ; \; \Delta'_I \diagdown \Delta'_O \xrightarrow{s::\pi \quad d}_v G$
whenever
$\Sigma'_I(\Delta'_I) = \Delta_I$ *and*
$(\forall t \in \mathrm{dom}(\Sigma'_I). \exists l \in \mathrm{dom}(\Sigma'_I). \Sigma'_I \diagdown_- \models t \rightsquigarrow l$ *and* $\Sigma'_I(l) = \#([\Sigma'_I(\Delta'_I)]_l))$ *and*
$\forall t \in d :: s :: \pi.\ \Sigma'_I \diagdown_- \models t \rightsquigarrow t$

Proof. By induction on the structure of the given derivation making use of lemma 2. Note that given Δ_I, s, π, d, it is always possible to construct Σ_I' and Δ_I' such that the resulting $\mathcal{TF}+$ sequent is well-formed. See [17] for details.

6 The $\mathcal{TF}\!\!\not\downarrow$ System

Our final refinement concerns the role of π which serves only to denote available tags. By carrying around a complete stack of available tags, we are forced to search through that stack to determine if a given formula may be focussed upon. We can abstract this stack, or more precisely membership in this stack, by associating an availability value with each tag and then ensuring that all available tags have the same availability value, and all unavailable tags do not have that value.

To carry out this idea, the $\mathcal{TF}\!\!\not\downarrow$ associates with each tag (location) another value which will be used to determine that tag's availability. $\mathcal{TF}\!\!\not\downarrow$ sequents are of the form

$$\Lambda; \Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s \quad d}_v G$$

where: Λ is a partial function of type $\mathcal{L} \to \mathbb{N}$ which tracks each tag's availability; and everything else remains the same as in $\mathcal{TF}+$ sequents. The extension of $\mathcal{TF}+$ rules to $\mathcal{TF}\!\!\not\downarrow$ rules is based on the observation that the strict tag is always available. Figure 5 contains the $\mathcal{TF}\!\!\not\downarrow$ derivation rules. Note that the second premise of the $\&_0$ rule assigns the new strict tag l a new availability number, thus making l the only available tag; in contrast, the $\&_1$ rule assigns l the same availability number as s, thus allowing all tags available in the first premise to also be in consumed the second premise.

The systems $\mathcal{TF}\!\!\not\downarrow$ and $\mathcal{TF}+$ are essentially isomorphic. The following theorem states the logical equivalence of both proof systems.

Theorem 5 (Equivalence of $\mathcal{TF}\!\!\not\downarrow$ and $\mathcal{TF}+$).
$\Lambda; \Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s \quad d}_v G$ *iff* $\Sigma_I \diagdown \Sigma_O \; ; \; \Delta_I \diagdown \Delta_O \xrightarrow{s::\pi \quad d}_v G$
whenever $s \in \mathrm{dom}(\Lambda)$ *and* $\forall t \in \Sigma_I(\Delta_I).t \in \mathrm{dom}(\Lambda)$ *and* $(t \in \pi$ *iff* $\Lambda(t) = \Lambda(s))$

Proof. By induction on the structure of the given derivation. See [17] for details.

7 Implementation and Benchmarks

The first and only published abstract machine for Lolli, LLP, comes from a low-level tag-based resource management system [3, 6]. While this approach yields a fast implementation, as evidenced by LolliCop [15], it fails to preserve the ideal proof search behavior of the more abstract systems. The main design goal for $\mathcal{TF}\!\!\not\downarrow$ was to provide a basis for an efficient low-level implementation while retaining the ideal behavior of more abstract systems.

Tamura and Banbara have recently developed and implemented LLP-TF [5], an abstract machine for Lolli directly based on $\mathcal{TF}\!\!\not\downarrow$. They have also executed

$$\frac{}{\Lambda;\, \Sigma \setminus \Sigma[s := d][d + \Sigma(s)]\,;\; \Delta \setminus \Delta \xrightarrow{s\quad d}_1 \top}\ \top_{\mathcal{TF}\not\downarrow} \qquad \frac{\Sigma(s) = 0}{\Lambda;\, \Sigma \setminus \Sigma\,;\; \Delta \setminus \Delta \xrightarrow{s\quad d}_0 \mathbf{1}}\ \mathbf{1}_{\mathcal{TF}\not\downarrow}$$

$$\frac{\Lambda;\, \Sigma_I[s+1] \setminus \Sigma_O\,;\; D^s :: \Delta_I \setminus D^{\text{-}} :: \Delta_O \xrightarrow{s\quad d}_v G}{\Lambda;\, \Sigma_I \setminus \Sigma_O[d-1]\,;\; \Delta_I \setminus \Delta_O \xrightarrow{s\quad d}_v D \multimap G}\ \multimap_{\mathcal{TF}\not\downarrow}$$

$$\frac{\Lambda[l := \Lambda(s)];\, \Sigma_I[l := 0] \setminus \Sigma_M\,;\; \Delta_I \setminus \Delta_M \xrightarrow{l\quad d}_0 G_1 \qquad \Lambda;\, (\Sigma_M \setminus l) \setminus \Sigma_O\,;\; \Delta_M \setminus \Delta_O \xrightarrow{s\quad d}_v G_2}{\Lambda;\, \Sigma_I \setminus \Sigma_O\,;\; \Delta_I \setminus \Delta_O \xrightarrow{s\quad d}_v G_1 \otimes G_2}\ \otimes_{0\mathcal{TF}\not\downarrow}$$

where l not in conclusion

$$\frac{\Lambda[l := \Lambda(s)];\, \Sigma_I[l := 0] \setminus \Sigma_M\,;\; \Delta_I \setminus \Delta_M \xrightarrow{l\quad d}_1 G_1 \qquad \Lambda[l := \Lambda(s)];\, \Sigma_M[l := 0] \setminus \Sigma_O\,;\; \Delta_M \setminus \Delta_O \xrightarrow{l\quad d}_v G_2}{\Lambda;\, \Sigma_I \setminus (\Sigma_O[s := d][d + \Sigma_O(s)] \setminus l)\,;\; \Delta_I \setminus \Delta_O \xrightarrow{s\quad d}_1 G_1 \otimes G_2}\ \otimes_{1\mathcal{TF}\not\downarrow}$$

where l not in conclusion

$$\frac{\Lambda;\, \Sigma_I[l := 0] \setminus \Sigma_M\,;\; \Delta_I \setminus \Delta_M \xrightarrow{s\quad l}_0 G_1 \qquad \Lambda[l := n];\, \Sigma_M \setminus \Sigma_O\,;\; \Delta_M \setminus \Delta_O \xrightarrow{l\quad d}_v G_2}{\Lambda;\, \Sigma_I \setminus \Sigma_O\,;\; \Delta_I \setminus \Delta_O \xrightarrow{s\quad d}_0 G_1 \,\&\, G_2}\ \&_{0\mathcal{TF}\not\downarrow}$$

where $[\Delta_I]_l = \emptyset,\ l \notin \mathrm{dom}(\Lambda) \cup \mathrm{dom}(\Sigma_I) \cup \mathrm{cod}(\Sigma_I),$ and $n \notin \mathrm{cod}(\Lambda)$

$$\frac{\Lambda;\, \Sigma_I[l := 0] \setminus \Sigma_M\,;\; \Delta_I \setminus \Delta_M \xrightarrow{s\quad l}_1 G_1 \qquad \Lambda[l := \Lambda(s)];\, \Sigma_M \setminus \Sigma_O\,;\; \Delta_M \setminus \Delta_O \xrightarrow{l\quad d}_v G_2}{\Lambda;\, \Sigma_I \setminus \Sigma_O\,;\; \Delta_I \setminus \Delta_O \xrightarrow{s\quad d}_v G_1 \,\&\, G_2}\ \&_{1\mathcal{TF}\not\downarrow}$$

where $[\Delta_I]_l = \emptyset,\ l \notin \mathrm{dom}(\Lambda) \cup \mathrm{dom}(\Sigma_I) \cup \mathrm{cod}(\Sigma_I),$ and $n \notin \mathrm{cod}(\Lambda)$

$$\frac{\Sigma_I \setminus \Sigma_M \models t \rightsquigarrow l \qquad D \gg A \setminus G \qquad \Lambda;\, \Sigma_M[l-1][d+1] \setminus \Sigma_O\,;\; \Delta_L @ (D^d :: \Delta_R) \setminus \Delta_O \xrightarrow{s\quad d}_v G}{\Lambda;\, \Sigma_I \setminus \Sigma_O\,;\; \Delta_L @ (D^t :: \Delta_R) \setminus \Delta_O \xrightarrow{s\quad d}_v A}\ \mathbf{pick}\ \Delta_{\mathcal{TF}\not\downarrow}$$

where $\Lambda(l) = \Lambda(s)$

Fig. 5. The $\mathcal{TF}\not\downarrow$ system

a series of benchmarks to compare the performance of LLP-TF to that of LLP. The results of these experiments are summarized in Fig. 6. The with-test, which tests the performance of & (additive conjunction), shows that, as expected, LLP-TF outperforms LLP for &-intensive tasks. The LolliCoP benchmark exhibits similar performance in both systems, though LLP is slightly faster in general. For the N-Queens program, LLP is still faster than LLP-TF.

The loss of efficiency in LLP-TF is attributed to the treatment of the multiplicative conjunction. Though the tensor rule is in fact constant time, it requires the creation of a new tag and a check of the $\top$ flag, both of which are absent in LLP. However, LLP checks linear constraints in the linear implication rule, thus eager failure is not available and LLP is less complete than LLP-TF[6]. Additionally, LLP relies on a linear time rule for the additive conjunction while LLP-TF's rule is constant time. As a result, it is possible to write programs that are faster in LLP than in LLP-TF, and vice-versa.

In order to improve the performance of LLP-TF, the code generation for tensor goals $G_1 \otimes G_2$ must be optimized. Whenever G_1, G_2 or both are intu-

[6] LLP will sometimes diverge where LLP-TF will simply fail, but not the converse.

		LLP-TF	LLP
with-test	time	0.006	0.506
	speedup	84.33	1.00
LolliCoP	time	1.920	1.794
	speedup	1.00	1.07
12-Queens	time	5.110	3.372
	speedup	1.00	1.51

Fig. 6. LLP-TF benchmarks (CPU time in seconds)

itionistic, optimized code can be generated to avoid the extra cost incurred by the general case. In fact, the current implementation generates optimized code for built-in predicates. For other predicates, static analysis using abstract interpretation seems the most promising technique. It should be noted that this is a compiler optimization independent of both the $\mathcal{TF}\!\!\downarrow$ proof system and its abstract machine. From that point of view, we believe that $\mathcal{TF}\!\!\downarrow$ fulfills its original goal.

8 Conclusions and Further Work

We have presented $\mathcal{TF}\!\!\downarrow$, a new tag-based resource management system for linear logic. This system is based on the intuition that tags are memory locations which store a counter or a reference to another memory location. By using a memory lookup scheme supporting indirections, most linear time operations of previous proposals can be replaced by constant time operations without sacrificing proof search behavior. In particular, the scan of the linear context necessary in the **1** rule of $\mathcal{TF}$[1], has been replaced by a constant-time memory lookup in $\mathcal{TF}\!\!\downarrow$.

The only non-constant time work in the $\mathcal{TF}\!\!\downarrow$ system occurs in the **pick** rule which requires looking up the current alias of a given tag. Since the lookup derivation short circuits the alias chains it follows, only the first attempt to lookup a given tag's alias will require a non-constant amount of work.

The implementation of LLP-TF, an abstract machine based on $\mathcal{TF}\!\!\downarrow$, shows that the proof system is detailed enough to allow a direct, low-level implementation which preserves the ideal proof search behavior of more abstract systems. The performance benchmarks show that LLP-TF is competitive with LLP, though some optimization work remains to be done. In particular, given the pervasive use of tensor in linear logic programming, it is essential to optimize the code generation of tensor goals. We plan to extend the compiler with a static analysis phase to detect intuitionistic predicates.

9 Acknowledgements

The authors are indebted to Professors Naoyuki Tamura and Mutsunori Banbara of Kobe University for implementing an abstract machine based on $\mathcal{TF}\frac{1}{t}$ and measuring its performance.

References

1. Hodas, J., López, P., Polakow, J., Stoilova, L., Pimentel, E.: A tag-frame system of resource management for proof search in linear-logic programming. In Bradfield, J.C., ed.: CSL'02, Edinburgh, Scotland (2002) 167–182
2. Cervesato, I., Hodas, J.S., Pfenning, F.: Efficient resource management for linear logic proof search. Theoretical Computer Science **232** (2000) 133–163
3. Hodas, J.S., Watkins, K., Tamura, N., Kang, K.S.: Efficient implementation of a linear logic programming language. In: JICSLP'98, IEEE Computer Society Press (1998) 145–149
4. Polakow, J.: Linearity constraints as bounded intervals in linear logic programming. In: Proceedings of LRPP'04, Turku, Finland (2004)
5. Tamura, N., Banbara, M.: Llp-tf: an abstract machine for lolli based on tag-frame-fast. http://bach.istc.kobe-u.ac.jp/llp/tf.html (2004)
6. Banbara, M., Tamura, N.: Compiling resources in a linear logic programming language. In Sagonas, K., ed.: JICSLP'98 Post Conference Workshop on Implementation Technologies for Programming Languages based on Logic. (1998) 32–45
7. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation **2** (1992) 297–347
8. Hodas, J.S., Miller, D.: Logic programming in a fragment of intuitionistic linear logic. Information and Computation **110** (1994) 327–365
9. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
10. Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic **51** (1991) 125–157
11. Hodas, J.S.: Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation. PhD thesis, University of Pennsylvania, Department of Computer and Information Science (1994)
12. Miller, D., Nadathur, G.: Higher-order logic programming. In Shapiro, E., ed.: Proceedings of the Third International Logic Programming Conference, London (1986) 448–462
13. López, P., Pimentel, E.: Resource management in linear logic proof search revisited. In Ganzinger, H., McAllester, D., Voronkov, A., eds.: LPAR'99, Tbilisi, Republic of Georgia, Springer-Verlag LNAI 1705 (1999) 304–319
14. Tamura, N., Kaneda, Y.: Extension of wam for a linear-logic programming language. In Ida, T., Ohori, A., Takeichi, M., eds.: Second Fuji International Workshop on Functional and Logic Programming, World Scientific (1996) 33–50
15. Hodas, J.S., Tamura, N.: lolliCOP - a linear logic encoding of a lean connection-method theorem prover for first-order classical logic. In: IJCAR'01, Siena, Italy (2001) 670–684
16. Cervesato, I.: Proof-theoretic foundation of compilation in logic programming languages. In Jaffar, J., ed.: JICSLP'98, Manchester, UK, MIT Press (1998) 115–129
17. López, P., Polakow, J.: Proof of correctness for tag-frame-fast. http://capoeira.scitec.kobe-u.ac.jp/ jp/TFPProofs.ps (2004)

A The $\mathcal{F}$ System

The $\mathcal{F}$ system is characterized by the following judgement

$$\Xi; \Delta_I \backslash \Delta_O \longrightarrow_v G$$

where: Ξ is a set of strict formulas; Δ_I, Δ_O are stacks of sets of lax formulas, the input and output linear contexts; v is the $\top$ flag and G is the goal formula. The notation $\Delta \triangleleft D$ stands for the stack Δ with the formula D inserted into one of its frames. We now give a version of the $\mathcal{F}$ derivation rules adapted to use residuation and our common syntax.

$$\frac{}{\Xi; \Delta \backslash \Delta \longrightarrow_1 \top}\ \top_{\mathcal{F}} \qquad \frac{}{\emptyset; \Delta \backslash \Delta \longrightarrow_0 \mathbf{1}}\ \mathbf{1}_{\mathcal{F}} \qquad \frac{\Xi \cup \{D\}; \Delta_I \backslash \Delta_O \longrightarrow_v G}{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_v D \multimap G}\ \multimap_{\mathcal{F}}$$

$$\frac{\emptyset; \Xi :: \Delta_I \setminus \Xi' :: \Delta_M \longrightarrow_0 G_1 \quad \Xi'; \Delta_M \backslash \Delta_O \longrightarrow_v G_2}{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_v G_1 \otimes G_2}\ \otimes_{0\mathcal{F}}$$

$$\frac{\emptyset; \Xi :: \Delta_I \setminus \Xi' :: \Delta_M \longrightarrow_1 G_1 \quad \emptyset; \Xi' :: \Delta_M \setminus \Xi'' :: \Delta_O \longrightarrow_v G_2}{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_1 G_1 \otimes G_2}\ \otimes_{1\mathcal{F}}$$

$$\frac{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_0 G_1 \qquad \Xi \cup (\Delta_I - \Delta_O); \mathrm{nil} \backslash \mathrm{nil} \longrightarrow_v G_2}{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_0 G_1 \,\&\, G_2}\ \&_{0\mathcal{F}}$$

$$\frac{\Xi; \Delta_I \backslash \Delta_M \longrightarrow_1 G_1 \quad \Xi \cup (\Delta_I - \Delta_M); \Delta_M \backslash \Delta_O \longrightarrow_v G_2}{\Xi; \Delta_I \backslash \Delta_O \longrightarrow_v G_1 \,\&\, G_2}\ \&_{1\mathcal{F}}$$

$$\frac{D \gg A \backslash G \qquad \Xi; \Delta_I \backslash \Delta_O \longrightarrow_v G}{\Xi \cup \{D\}; \Delta_I \backslash \Delta_O \longrightarrow_v A}\ \mathbf{pick}\ \Xi_{\mathcal{F}} \qquad \frac{D \gg A \backslash G \qquad \Xi; \Delta_I \backslash \Delta_O \longrightarrow_v G}{\Xi; \Delta_I \triangleleft D \backslash \Delta_O \longrightarrow_v A}\ \mathbf{pick}\ \Delta_{\mathcal{F}}$$

B $\mathcal{TF}$ System

The $\mathcal{TF}$ system we present here is a cleaner version of the published system [1] in which:

1. only a single consumed tag is necessary,
2. only consumed tags generated in & rules, rather than all current consumed tags, are exported,
3. we use the formula language and residuation judgement of section 2.

In unpublished notes, Hodas, López, and Polakow have proved this new version equivalent to the system presented in [1].

$\mathcal{TF}$ sequents are of the form

$$\Sigma : \ \Delta_I \backslash \Delta_O \xrightarrow[\beta \quad v]{\sigma :: \pi \quad d} G$$

where: Σ is a countably infinite set of available tags; Δ_I, Δ_O are the input and output linear contexts, each containing tagged clause formulas; σ is a set of tags

marking the strict resources; π a stack of frames of tags marking the optional resources; d the consumption, or deletion, tag; β is a set holding the consumption markers introduced by & that are exported, or bleed out; v is the $\top$ flag; G is the goal formula.

$$\frac{}{\Sigma :\ \Delta \diagdown \Delta \xrightarrow[\emptyset \quad 1]{\pi \quad d} \top}\ \top_{\mathcal{TF}'_\beta} \qquad \frac{\forall D^t \in \Delta.\, t \notin \sigma}{\Sigma :\ \Delta \diagdown \Delta \xrightarrow[\emptyset \quad 0]{\sigma :: \pi \quad d} \mathbf{1}}\ \mathbf{1}_{\mathcal{TF}'_\beta}$$

$$\frac{\Sigma : D^t :: \Delta_I \diagdown D^{\text{-}} :: \Delta_O \xrightarrow[\beta \quad v]{\sigma :: \pi \quad d} G}{\Sigma :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta \quad v]{\sigma :: \pi \quad d} D \multimap G}\ \multimap_{\mathcal{TF}'_\beta} \quad (t \in \sigma)$$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 0]{\{t\} :: \pi \quad d} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad v]{\pi \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{t\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_1 \cup \beta_2 \quad v]{\pi \quad d} G_1 \otimes G_2}\ \otimes_{0\mathcal{TF}'_\beta}$$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 1]{\{t\} :: \pi \quad d} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad v]{\{t\} :: \pi \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{t\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_1 \cup \beta_2 \quad 1]{\pi \quad d} G_1 \otimes G_2}\ \otimes_{1\mathcal{TF}'_\beta}$$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 0]{\pi \quad d'} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad 0]{\beta' \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{d'\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_2 \quad 0]{\pi \quad d} G_1 \,\&\, G_2}\ \&_{00\mathcal{TF}'_\beta}$$

where $\beta' = (\{d'\} \cup \beta_1) :: \text{nil}$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 0]{\pi \quad d'} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad 1]{\beta' \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{d'\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_1 \cup \beta_2 \cup \{d'\} \quad 0]{\pi \quad d} G_1 \,\&\, G_2}\ \&_{01\mathcal{TF}'_\beta}$$

where $\beta' = (\{d'\} \cup \beta_1) :: \text{nil}$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 1]{\sigma :: \pi \quad d'} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad 0]{\beta' \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{d'\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_2 \quad 0]{\sigma :: \pi \quad d} G_1 \,\&\, G_2}\ \&_{10\mathcal{TF}'_\beta}$$

where $\beta' = (\sigma \cup \{d'\} \cup \beta_1) :: \pi$

$$\frac{\Sigma_1 :\ \Delta_I \diagdown \Delta_M \xrightarrow[\beta_1 \quad 1]{\sigma :: \pi \quad d'} G_1 \quad \Sigma_2 :\ \Delta_M \diagdown \Delta_O \xrightarrow[\beta_2 \quad 1]{\beta' \quad d} G_2}{\Sigma_1 \cup \Sigma_2 \cup \{d'\} :\ \Delta_I \diagdown \Delta_O \xrightarrow[\beta_1 \cup \beta_2 \cup \{d'\} \quad 1]{\sigma :: \pi \quad d} G_1 \,\&\, G_2}\ \&_{11\mathcal{TF}'_\beta}$$

where $\beta' = (\sigma \cup \{d'\} \cup \beta_1) :: \pi$

$$\frac{D \gg A \diagdown G \qquad \Sigma :\ \Delta_L @ (D^d :: \Delta_R) \diagdown \Delta_O \xrightarrow[\beta \quad v]{\pi \quad d} G}{\Sigma :\ \Delta_L @ (D^t :: \Delta_R) \diagdown \Delta_O \xrightarrow[\beta \quad v]{\pi \quad d} A}\ \textbf{pick}\ \Delta_{\mathcal{TF}'_\beta} \quad \text{where } t \in \pi$$

線型論理プログラミングにおけるリソース管理の高効率な実装 (in English)
(算譜科学研究速報)
発行日：2005 年 1 月 20 日
編集・発行：独立行政法人産業技術総合研究所関西センター尼崎事業所
システム検証研究センター
同連絡先：〒661-0974 兵庫県尼崎市若王寺 3-11-46
e-mail：informatics-inquiry@m.aist.go.jp

Implementing Efficient Resource Management for Linear Logic Programming
(Programming Science Technical Report)
Jan. 20, 2005
Research Center for Verification and Semantics (CVS)
AIST Kansai, Amagasaki Site
National Institute of Advanced Industrial Science and Technology (AIST)
3-11-46 Nakouji, Amagasaki, Hyogo, 661-0974, Japan
e-mail : informatics- inquiry@m.aist.go.jp