Hierarchical Proof Structures

Ewen Denney[1], John Power[2⋆], and Konstantinos Tourlas[2]

[1] USRA/RIACS, NASA Ames Research Center, CA 94035, USA
[2] Laboratory for the Foundations of Computer Science, King's Buildings, University of Edinburgh, EH9 3JZ, SCOTLAND

Abstract. Motivated by structure arising in tactic-based theorem proving, we develop the concept of hierarchical proof tree or *hiproof* by characterising a geometrically natural definition in terms of its family of proof views. We first recall a definition of hierarchical proof tree, explaining its axioms and illustrating by example. We then describe notions involved with proof views. Then we characterise hierarchical proof trees in terms of dags of proof views. Our ultimate goal is to axiomatise the structure required for constructing and navigating tactic-based proofs. This is work in progress towards that end.

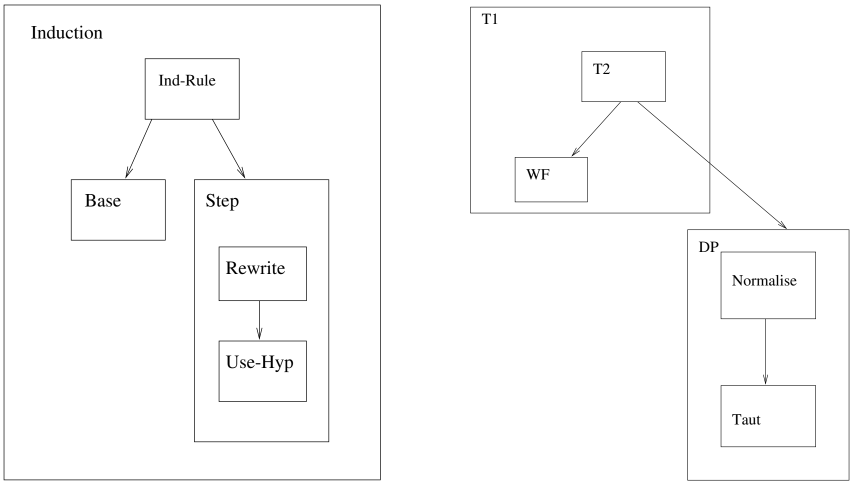
1 Introduction

Consider a proof by induction as represented by Figure 1(a): the nodes are labelled by tactic identifiers, inclusion of one node in another indicates a subtactic relationship, and the arrows represent sequential composition. The diagram is read as follows: the proof consists of invoking an induction tactic, `Induction`. That consists of applying an induction rule, `Ind-Rule`, which then generates two subgoals. The first subgoal is handled by the `Base` tactic, the second by the `Step` tactic. In turn, `Step` is defined as first applying the `Rewrite` tactic, and then the `Use-Hyp` tactic, with `Base`, `Rewrite` and `Use-Hyp` treated as primitive. In contrast to the usual presentations of a proof by induction, the emphasis is on tactics rather than on goals and proof steps.

For a structurally somewhat more complex proof, consider Figure 1(b). At the most abstract level, the proof consists of applying `T1`, and then `DP`. The tactic `T1` first applies `T2`, generating two subgoals, the first of which is handled by `WF`. The second is handled by `DP`, which applies `Normalise` and then `Taut`.

These examples reflect, albeit very abstractly, the hierarchical structure of tactics as appear in proof assistants such as [1–3]. In [4], we took a first abstract step towards developing a definition and mathematical theory of such hierarchy, ultimately aimed towards the development of interfaces, both graphical interfaces for individual theorem provers and interfaces between theorem provers. Our central definition, abstracting from Figures 1(a) and 1(b), was that of a *hierarchical proof tree* or *hiproof*. We analysed an appropriate choice of axioms for hiproofs, repeated here in Section 2, then we studied the relationship between a hiproof and its underlying ordinary proof, gave a notion of refinement of

⋆ John Power has been supported by EPSRC grant no. GR/586372/01.



(a) (b)

Fig. 1. Two hierarchical proofs

hiproofs, and characterised hiproofs in terms more amenable to implementation. In particular, using a subtle notion of map, we showed that the obvious inclusion of a category of ordinary proof trees into one of hierarchical proof trees has a left adjoint, with that left adjoint yielding a natural notion of the *skeleton* of a hierarchical proof.

For the purposes of this paper, we shall refer to the notion of hierarchical proof we developed in [4] as a *hiproof of type 1*. In [4] we also introduced *type 2 hiproofs*, which are an equivalent formulation of type 1 hiproofs designed to facilitate implementation. We do not discuss type 2 hiproofs in this paper, but we want to give a third equivalent formulation of the notion. So, for overall consistency, we shall refer to the central new construct of this paper as a *type 3 hiproof*. One of the overall goals of this work is to study a diversity of possible formulations of hierarchical proof structure in order to understand the common structure behind hierarchy in tactic-based theorem proving.

Now consider a hiproof of type 1. A user is unlikely to view all the information it contains at once: the main point of structuring it hierarchically is that the proof can be viewed at different levels of abstraction in a sense we shall make precise. In particular, consider the hiproof in Figure 1(b). At the highest level of abstraction it can be thought of as the two step proof, `T1` followed by `DP`. We can think of this as an abstract proof, where `T1` and `DP` have no internal structure, and so are regarded as atomic steps. At the lowest level of abstraction, on the other hand, the proof tree is formed from the atomic steps `T2`, `WP`, `Normalise`, and `Taut`, in the obvious way. This is the *skeleton* of the hiproof, as defined in

[4], where it was characterised as a naturally arising left adjoint. Rather than consider the whole skeleton directly, we could have unfolded just one of the top-level tactics, `T1` or `DP`, thus yielding four possible "views" of this hiproof (see Figure 2) in total. More generally, we could unfold any abstract node (which may itself be a tree of abstract nodes), replacing the node with its immediate contents, yielding another tree.

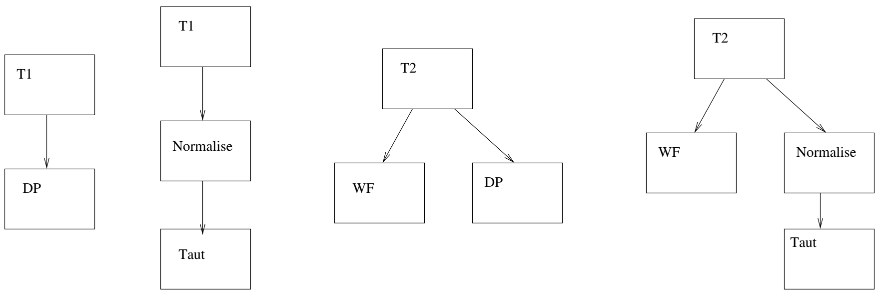


Fig. 2. Four views of a single hiproof

We call the proof trees that result from such sequences of unfolding *proof views*. The skeleton of a hiproof is the special case where no nodes are abstract. A proof view is a tree of inferences where some of the nodes may be abstract steps, i.e., tactics. Sets of proof trees can be seen as a dynamic interpretation of a tactic-based proof, where the possible traces of the proof's unfolding embody its hierarchy. This raises the question: can we take such views as primitive? In other words, can we reformulate the notion of hiproof in terms of a set of proof trees which are self-consistent in some sense? In this paper, we formalise the various relevant concepts and constructions, and formulate the theorem.

In Section 2, we recall the definition of type 1 hiproof and explain its axioms, illustrated by examples. In Section 3, we define the notion of type 3 hiproof. In Section 4, we show how to construct a type 1 hiproof from a type 3 hiproof. And in Section 5, we give the converse, constructing a type 3 hiproof from a type 1 hiproof.

2 Hierarchical proof trees

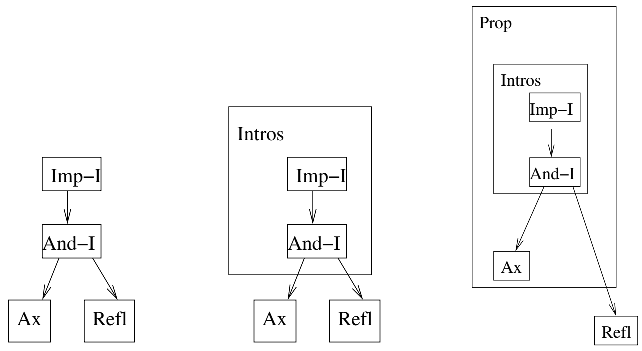
In this section, we recall the notion of a hierarchical proof tree or type 1 hiproof from [4]. To motivate the definition, we first analyse, by means of an example, the relationship between tactics and standard notions of formal proof such as proofs in natural deduction style.

Example 1. Consider a natural deduction proof of $A \Rightarrow A \wedge (x = x)$, as in Figure 3. The obvious (backwards) proof is implication introduction, followed by conjunction introduction, and then applying axiom and reflexivity to the two subgoals. The essential information of the proof is the sequence of inference

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\texttt{Ax} \qquad \dfrac{}{A \vdash x = x}\,\texttt{Refl}}{A \vdash A \wedge (x = x)}\,\texttt{And-I}}{\vdash A \Rightarrow A \wedge (x = x)}\,\texttt{Imp-I}$$

Fig. 3. A simple natural deduction proof

rules, with the order of those rules represented by a proof tree as in Figure 4(a). Typically, however, theorem provers allow the use of higher-level tactics that



(a) (b) (c)

Fig. 4. Introducing hierarchy in proof diagrams by grouping

group together the application of a number of low-level inferences. For example, it is common to have an `Intros` command, which performs all possible introduction rules. We can indicate this on the proof diagram by grouping `Imp-I` and `And-I` together, as in Figure 4(b). We could go further and define a tactic, `Prop`, which first calls `Intros`, and then tries to use axioms wherever possible. This gives the hierarchical structure of Figure 4(c). □

Example 1 shows that proofs can be represented as tactic- (or axiom and inference)-labelled trees with hierarchical structure on the set of nodes. The tree structure is straightforward, but the hierarchical structure and its interaction with the tree structure are more complex. We formalise the hierarchical structure by a partial order, with $v \leq_i w$ represented visually by depicting the node v as sitting inside the node w. The partial order satisfies axioms to the effect that it is generated by a (finite) forest, and it is sometimes convenient to regard it as such. Our hierarchical trees are labelled by tactics, so we henceforth assume that Λ is a fixed non-empty set of *tactic identifiers* or *method identifiers*. We write $isroot_F(v)$ (or $isroot_{\rightarrow}$) for the assertion that there is a tree in forest F with root v, and we write $siblings_F(v, v')$ (or $siblings_{\rightarrow}(v, v')$) if v and v' have the same parent or are both roots.

Definition 1. *A* hierarchical proof tree*, or (type 1)* hiproof *for short, consists of a tuple $\langle V, \leq_i, \rightarrow_s, tac\rangle$, comprising a (necessarily finite) forest* qua poset *$i = \langle V, \leq_i\rangle$ and a forest $s = \langle V, \rightarrow_s\rangle$, together with a function $tac : V \rightarrow \Lambda$ which labels the nodes in V with tactic identifiers in Λ, subject to the following conditions:*

1. *arrows always target outer nodes: whenever $v \rightarrow_s w_1$ and $w_1 <_i w_2$, then $v <_i w_2$*
2. *arrows always emanate from inner nodes: whenever $w_1 \leq_i v$ and $v \rightarrow_s w_2$ then $v = w_1$*
3. *inclusion and sequence are mutually exclusive: whenever $v \leq_i w$ and $v \rightarrow_s^\star w$, then $v = w$*
4. *given any two nodes v and v' which both lie at the top inclusion level, or are both immediately included in the same node, then at most one of v, v' has no incoming $\rightarrow_s$ edge:*

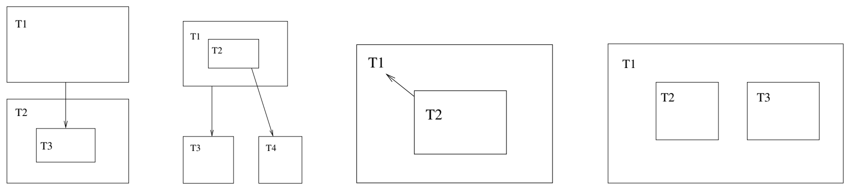
$$\forall v, v' \in V.\ siblings_i(v, v') \wedge isroot_s(v) \wedge isroot_s(v') \implies v = v'.$$

□

Note the subtlety in the first condition, especially in combination with the third: an arrow from a node v can only go to an outer node *relative* to the inclusion level of v. So, for instance, Example 1 satisfies the condition. Observe that the fourth condition together with finiteness imply that there is a unique node that is maximal with respect to $\leq_i$ and has no incoming $\rightarrow_s$ edge, acting as a kind of hierarchical root.

The main theorem justifying the axioms in [4] shows that every hiproof unfolds to give an ordinary proof (its skeleton). But here we analyse the axioms by looking at some non-examples. The axioms are designed to ensure that none of the diagrams in Figure 5 forms a hiproof.

In tactical theorem proving, one tactic is followed by another, which unfolds to give another tactic, and so on. So tactics are invoked 'at the most abstract level.' But Figure 5(a) contradicts that because if `T1` is followed by `T3` and `T2` unfolds to `T3`, the more abstract `T2` should follow `T1`. Equivalently, it would be permissible for `T3` to follow `T1`, but then the fact that `T2` is an abstraction of `T3` would be irrelevant to the proof and should not be added after the composition of `T1` and `T3`. Conversely, when a tactic finishes executing, control flows from the most recently executed tactic, i.e. the innermost, outwards, but Figure 5(b) contradicts that. We want to exclude Figure 5(c) too in order to avoid circularity



(a) (b) (c) (d)

Fig. 5. Four non-examples of hiproofs

of unfolding and sequencing. Finally, Figure 5(d) fails because tactic `T1` should unfold to give a unique subsequent tactic to execute, not two.

The first condition in the definition of hiproof prohibits the inclusion hierarchy from being 'downwards' transcended by composition, e.g. as in Figure 5(a). The second condition precludes Figure 5(b). The third condition precludes Figure 5(c): the similar structure with the arrow pointing the other direction is already precluded by the second condition. And the fourth condition precludes Figure 5(d) as well as the similar non-proof example obtained from Figure 5(d) by removing the node labelled `T1`. For a positive example of a hiproof, consider Figure 1(b).

The main ideas behind the definition of hiproof can be understood in terms of Figures 1(a) and 1(b). Although motivated by diagrams, we have abstracted away from geometry to discrete mathematical structure. The central features are as follows:

- we do not require tactic identifiers to be unique as a tactic may be applied repeatedly in a proof. But we informally refer to proof nodes by their tactic identifiers where there is no ambiguity.
- there are only two relationships that can hold between nodes: inclusion, representing the unfolding of a tactic into its definition, with arrows representing sequential composition. For example, in Figure 1(b), the decision procedure `DP` unfolds to give the composition of `Normalise` with `Taut`.
- hiproofs are essentially tree-like in that subgoals are independent: a tactic acts on a single subgoal. That is not generally the case in tactical theorem proving, and we intend to extend the definition accordingly in future work. Tactics usually return a list of subgoals, but we abstract away from the order on child tactics.

A hiproof, therefore, consists of a finite collection of tactic-labelled nodes, related by inclusion and composition. Although the diagrams represent abstract versions of full proofs, we are interested in how such proofs are constructed, and so we consider partial proofs as well-formed.

3 Hiproofs as families of proof views

A hierarchical proof yields and can be characterised by a collection of non-hierarchical proofs, i.e., by simple inference trees, that are self-consistent in a sense that we make precise in this section. These ordinary proofs generated by the hiproof can be regarded as views of the hiproof at various levels of abstraction given by all possible "unfoldings". Such views may, for instance, appear on a computer screen when one clicks on a particular node of a hiproof.

A first attempt to characterise hiproofs in terms of such views is to try to characterise a hiproof by the set of its partial underlying proofs. But that is not subtle enough as it does not distinguish between the two hiproofs in Figure 6, both of which would be interpreted as the set of two trees, $\{T1,\ T2 \rightarrow T3\}$. So we need to consider the total underlying proofs of a hiproof. The second hiproof in Figure 6 now has interpretation $\{T1 \rightarrow T3,\ T2 \rightarrow T3\}$, and the first is as before.

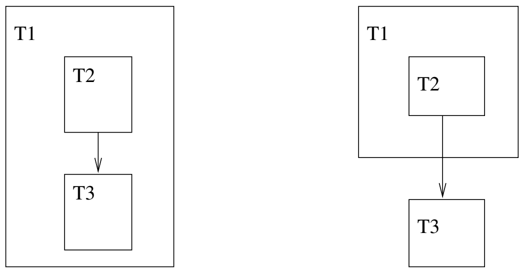


Fig. 6. Two distinct hiproofs with the same underlying proof

But that is still not delicate enough: taking the sets of the underlying proofs of a hiproof does not distinguish between hiproofs with the structures $T1 \leq_i T2$ and $T2 \leq_i T1$. So we replace sets by lists that represent the sequence of unfoldings of a hiproof. But there are several ways in which a hiproof can be unfolded. If we take all possible unfoldings, we obtain the structure of a dag, which, finally, has sufficient structure to provide a characterisation. The main technical part of our work involves characterising those dags that thus arise.

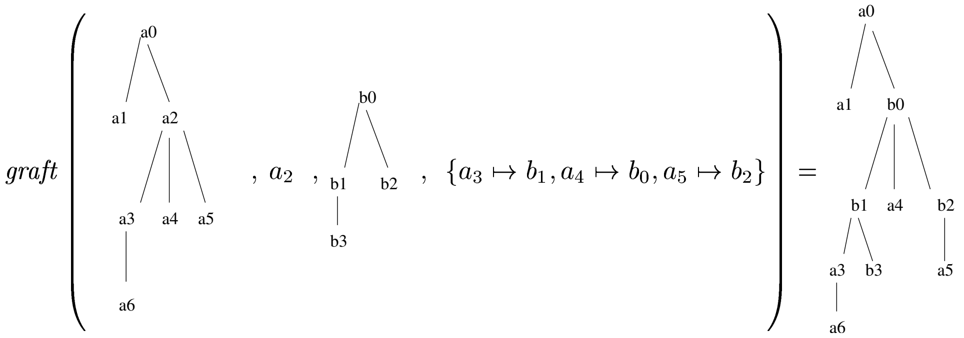
The characterisation requires a delicate operation that grafts a tree t' at a vertex v of a given tree t, with an embedding map, f, from the children of v into the nodes of t, thus generalising the idea of substituting a tree for a leaf vertex in another tree. Formally, the definition is as follows:

Definition 2. *Let $t_A = \langle V_A, \to_A, r_A \rangle$ and $t_B = \langle V_B, \to_B, r_B \rangle$ be (rooted) trees, $v_0 \in V_A$, and f a map from the children of v_0 to V_B. Then graft$(t_A, v_0, t_B, f) \stackrel{\text{def}}{=} \langle V, \to, r \rangle$ is the tree where*

- $V = V_A \setminus \{v_0\} + V_B$,
- $v \to v'$ *if and only if either of the following hold*
  1. $v, v' \in V_A \setminus \{v_0\}$ *and* $v \to_A v'$
  2. $v, v' \in V_B$ *and* $v \to_B v'$
  3. $v \in V_A$, $v' = r_B$ *and* $v \to_a v_0$
  4. v' *is a child of* v_0 *and* $v = f(v')$
- $r = r_B$ *if* $v_0 = r_A$ *or otherwise* $r = r_A$. □

Note that we discard the vertex v_0 instead of the root of the tree being grafted.

Example 2.



It is routine to extend the definition of grafting to incorporate labelling. Labels are taken in the set Λ of tactic identifiers. We do not insist that the labelling functions on the two trees be consistent with each other on v_0 and r_B.

Definition 3. *Let $\langle t_A, l_A \rangle$ and $\langle t_B, l_B \rangle$ be trees labelled over Λ and $v_0 \in V_A$. Then*

$$graft(\langle t_A, l_A \rangle, v_0, \langle t_B, l_B \rangle, f)$$

is the labelled tree $\langle graft(t_A, v_0, t_B, f),\ l \rangle$ where $l : V_A \setminus \{v_0\} + V_B \to \Lambda$ is defined as follows:

1. *whenever* $v \in V_B$, $l(v) = l_B(v)$
2. *whenever* $v \in V_A$ *and* $v \neq v_0$, $l(v) = l_A(v)$. □

For simplicity of presentation we shall tacitly assume that, whenever we write *graft*(t_A, v_0, t_B, f), the sets V_A and V_B of vertices in t_A and t_B are disjoint, i.e., $V_A \cap V_B = \emptyset$, whereby also $v_0 \notin V_B$. So the set of vertices underlying *graft*(t_A, v_0, t_B, f) is $V_A \setminus \{v_0\} \cup V_B$.

In order to give a coherent definition of a type 3 hiproof, we first need to observe a few facts about grafting. They are as follows:

Lemma 1. *(Injectivity of grafting)* $graft(t, v_0, t_1, f) = graft(t, v_0, t_1', f')$ *implies* $t_1 = t_1'$ *and* $f = f'$. □

Lemma 2. *(Commutativity of grafting)*
$graft(graft(t_0, v_1, t_1, f_1), v_2, t_2, f_2) = graft(graft(t_0, v_2, t_2, f_2), v_1, t_1, f_1)$. □

Lemma 3. *(Independence of grafts) Let* $t_0 = \langle V_0, \to_0, r_0 \rangle$ *be a tree and* v_1, v_2 *be* distinct *vertices in* V_0. *Then*

$$graft(graft(t_0, v_1, t_1, f_1), v_2, t_2, f_2) = graft(graft(t_0, v_2, t_2', f_2'), v_1, t_1', f_1')$$

implies $t_i = t_i'$ *and* $f_i = f_i'$, *for* $i = 1, 2$. □

We can now formally define type 3 hiproofs:

Definition 4. *A hiproof of type 3 is a tuple* $\langle U, \rightsquigarrow, \tau, \beta \rangle$, *where*

- $\langle U, \rightsquigarrow \rangle$ *is a dag,*
- $\tau : U \to \Lambda\text{-}Tree$ *is a function, assigning to each vertex* $u \in U$ *a* Λ*-labelled tree* $\tau(u)$, *and*
- β *assigns to each pair* $\langle u, u' \rangle \in \rightsquigarrow$ *a vertex in* $\tau(u)$.

Writing $u \overset{v}{\rightsquigarrow} u'$ *to mean the existence of* $u, u' \in U$ *such that* $\langle u, u' \rangle \in \rightsquigarrow$ *and* $\beta(\langle u, u' \rangle) = v$, *the above data is subject to the following conditions:*

1. $\langle U, \rightsquigarrow \rangle$ *has a source, which we denote by* $u_\top$
2. *if* $u \overset{v}{\rightsquigarrow} u'$, *there exists a labelled tree* γ *and a map* f *such that* $\tau(u') = graft(\tau(u), v, \gamma, f)$
3. *if* $u_0 \overset{v}{\rightsquigarrow} u_1$ *and* $u_0 \overset{v}{\rightsquigarrow} u_2$, *then* $u_1 = u_2$, *and*
4. *if* $u_0 \overset{v_1}{\rightsquigarrow} u_1$ *and* $u_0 \overset{v_2}{\rightsquigarrow} u_2$, *there exists* u_3 *such that* $u_1 \overset{v_2}{\rightsquigarrow} u_3$ *and* $u_2 \overset{v_1}{\rightsquigarrow} u_3$. □

This definition formulates the discussion at the beginning of the section, that a hiproof can be represented as a dag of proof trees. The top is the most abstract proof, the bottom (as we will see) is the most concrete proof, i.e. the underlying skeleton. The second and third conditions give the meaning of the dag in terms of the grafting of proof trees, while the fourth is a completeness condition: if you can unfold nodes separately, then you can unfold them together.

Proposition 1. *A dag* $\langle U, \rightsquigarrow \rangle$ *satisfying the conditions on a dag in Definition 4 has a (necessarily unique) sink* $u_\bot$.

Proof. (Sketch) This follows from general theorems on abstract rewriting systems satisfying the diamond property, as the rewriting system in question is strongly and uniquely normalising. □

4 From hiproofs of type 3 to hiproofs of type 1

In this section, we construct a type 1 hiproof from a type 3 hiproof. This first requires some notation, then a subtle construction combining a type 1 hiproof with a tree: we build our final type 1 hiproof by an inductive process, requiring several intermediary type 1 hiproofs as inductive steps, hence the need to combine a type 1 hiproof inductively with a tree.

First observe that by Lemma 1, whenever $u \overset{v}{\rightsquigarrow} u'$ holds in a type 3 hiproof, the labelled tree γ and map f given by the third condition are unique up to isomorphism such that $\tau(u') = \mathit{graft}(\tau(u), v, \gamma, f)$. We shall therefore write $u \overset{v,\gamma,f}{\rightsquigarrow} u'$, instead of $u \overset{v}{\rightsquigarrow} u'$, when knowledge of the unique γ and f is required. By Lemma 3, we have the following:

Proposition 2. *For $v_1 \neq v_2$, whenever $u_0 \overset{v_1,\gamma_1,f_1}{\rightsquigarrow} u_1 \overset{v_2,\gamma_2,f_2}{\rightsquigarrow} u_3$ and $u_0 \overset{v_2,\gamma_2',f_1'}{\rightsquigarrow} u_2 \overset{v_1,\gamma_1',f_2'}{\rightsquigarrow} u_3$ in a type 3 hiproof, one has $\gamma_1 = \gamma_1'$, $\gamma_2 = \gamma_2'$, $f_1 = f_1'$ and $f_2 = f_2'$.* □

We extend the notation to finite lists of $\langle v, \gamma \rangle$ pairs. Then for every such list $\sigma = \langle v_0, \gamma_0 \rangle, \ldots, \langle v_{n-1}, \gamma_{n-1} \rangle$, we write

$$u \overset{\sigma}{\rightsquigarrow} u'$$

to mean

$$u \overset{v_0,\gamma_0}{\rightsquigarrow} u_0 \overset{v_1,\gamma_1}{\rightsquigarrow} \ldots \overset{v_{n-1},\gamma_{n-1}}{\rightsquigarrow} u_n \quad \text{and} \quad u_n = u' \ .$$

As usual, we write $\langle v, \gamma \rangle :: \sigma$ for the list with head $\langle v, \gamma \rangle$ and tail σ.

Now we inductively present a family F of operations on hiproofs of type 1, indexed by paths $u \overset{\sigma}{\rightsquigarrow} u'$ in a hiproof of type 3. The idea is to gradually build up a (type 1) hiproof by combining all the trees along a path in a hiproof of type 3. We need an operation $h \triangleleft^{v_0} \gamma$ extending a type 1 hiproof h using the labelled tree γ at the node v_0. This is defined as follows:

Definition 5. *Let $h = \langle V_h, \leq_i, \rightarrow_s, \mathit{tac} \rangle$ be a hiproof of type 1, and let $v_0 \in V_h$ be $\leq_i$-minimal. Then given a Λ-labelled tree $\gamma = \langle V_\gamma, \rightarrow, r, l \rangle$, and a map f from the children of v_0 (with respect to $\rightarrow_s$) into V_γ, define $h \triangleleft^{v_0,f} \gamma$ to be $\langle V', \leq_i', \rightarrow_s', \mathit{tac}' \rangle$ where*

- *$V = V_h + V_\gamma$*
- *$v \leq_i' u$ if and only if either of the following hold*
  1. *$v, u \in V_h$ and $v \leq_i u$*
  2. *$v \in V_\gamma$, $u \in V_h$ and $v_0 \leq_i u$*
- *$v \rightarrow_s' u$ if and only if either of the following hold*
  1. *$v, u \in V_\gamma$ and $v \rightarrow u$*
  2. *$u \in V_h$, $v \in V_\gamma$, u is a child of v_0, and $f(u) = v$*
  3. *$v, u \in V_h$, $v \neq v_0$ and $v \rightarrow_s u$*
- *$\mathit{tac}' = \mathit{tac} + l$.* □

The difference between this construction and that of $graft(h, v_0, \gamma, f)$ is that here h can be an arbitrary hiproof (whereas for grafting h must be a tree) and we keep the node v_0 and put γ inside it (since v_0 is $\leq_i$-minimal it has no 'contents') rather than replacing v_0 by γ. Again, when we write $h \triangleleft^{v_0,f} \gamma$, we shall tacitly assume that the sets V_h and V_γ underlying h and γ respectively, are disjoint. This allows us to regard the set of vertices underlying $h \triangleleft^{v_0} \gamma$ as being simply $V_h \cup V_\gamma$. And by "suitable" labelled trees, we mean γ_1 and γ_2 for which $V_h \cap V_{\gamma_1} = \emptyset$, $V_h \cap V_{\gamma_2} = \emptyset$ and $V_{\gamma_1} \cap V_{\gamma_2} = \emptyset$.

Lemma 4. *(Commutativity of $\triangleleft$) If v_1 and v_2 are distinct vertices in a type 1 hiproof h, then $h \triangleleft^{v_1,f_1} \gamma_1 \triangleleft^{v_2,f_2} \gamma_2 = h \triangleleft^{v_2,f_2} \gamma_2 \triangleleft^{v_1,f_1} \gamma_1$ for any suitable Λ-labelled trees.* □

Proposition 3. *(Well-definedness of $\triangleleft$) Let $h = \langle V, \leq_i, \rightarrow_s, tac\rangle$ be a hiproof of type 1 and γ any suitable Λ-labelled tree. $h \triangleleft^{v,f} \gamma$ is a hiproof of type 1 whenever $v \in V$ is $\leq_i$-minimal and f is a suitable map.* □

Now let $u \overset{\sigma}{\rightsquigarrow} u'$ be a path in a hiproof of type 3 and h any hiproof of type 1. We proceed to define F_σ by induction on the length of the list σ:

- $F_u(h) = h$
- $F_{u \overset{v,\gamma,f}{\rightsquigarrow} u' \overset{\sigma'}{\rightsquigarrow} u''}(h) = F_{u' \overset{\sigma'}{\rightsquigarrow} u''}(h \triangleleft^{v,f} \gamma)$.

Any labelled tree may be trivially regarded as a "flat" hiproof of type 1. Formally:

Definition 6. *We define the embedding, $\mathcal{E}$: **Tree** $\rightarrow$ **Hiproof**$_1$, as the map which takes $\gamma = \langle V, \rightarrow, r, l\rangle$ to $\langle V, \mathrm{id}_V, \rightarrow, l\rangle$.* □

However, we will often blur the distinction and regard trees as type 1 hiproofs when convenient.

We showed in [4] that $\mathcal{E}$ extends to a functor from a category whose objects are proof trees to one whose objects are type 1 hiproofs, and proved that it has a left adjoint, characterising a natural construction of the skeleton, or underlying ordinary proof, of a type 1 hiproof.

Corollary 1. *$F_{u \overset{\sigma}{\rightsquigarrow} u'}(\tau(u))$ is always a well-formed hiproof of type 1.* □

Writing $\sigma \sim \sigma'$ to mean that (v, γ, f)-lists σ and σ' are permutations of one another, we finally have:

Theorem 1. *Whenever $u \overset{\sigma}{\rightsquigarrow} u_\perp$ and $u \overset{\sigma'}{\rightsquigarrow} u_\perp$ are paths in a type 3 hiproof, one has*

1. *$\sigma \sim \sigma'$; and*
2. *for all h, $F_{u \overset{\sigma}{\rightsquigarrow} u_\perp}(h) = F_{u \overset{\sigma'}{\rightsquigarrow} u_\perp}(h)$.*

□

Theorem 1 shows that no matter which path is used from a type 3 hiproof, F combines the trees along that path into the same type 1 hiproof. We can now define μ_{31}.

Definition 7. *$\mu_{31}(h_3) = F_\sigma(\mathcal{E}(u_\top))$, where $u_\top \overset{\sigma}{\rightsquigarrow} u_\perp$ is any path in h_3.* □

5 From type 1 to type 3 hiproofs

In this section, we give a converse to our construction of a type 1 hiproof from a type 3 hiproof. We first recall the definition of the skeleton of a type 1 hiproof from [4].

Definition 8. *Let $h = \langle V, \leq_i, \rightarrow_s, t\rangle$ be a type 1 hiproof. We define the* skeleton *of h, written $\mathtt{sk}_1(h)$, to be the Λ-labelled tree $\langle V_T, \rightarrow_T, r\rangle$, corresponding to the finite poset $T = \langle V_T, \leq_T\rangle$, where V_T are the leaves of $\leq_i$, and $v_1 \leq v_2$ if and only if there exists a $v \in V$ such that $v_2 \leq_i v$ and $v_1 \rightarrow_s v$.* □

For example, the skeleton of Figure 1(b) is the rightmost tree in Figure 2.

Proposition 4. *The definition above gives a well-formed tree.*

Proof. We must show that for all $v \in V_T$, there exists a unique path from r to v. There must exist a $(>^1_i \cup \rightarrow_s)$-path from the root of h_1 to v, by induction. This path must be unique since each vertex has a unique predecessor. □

Definition 9. *Let $h_3 = \langle U, \rightsquigarrow, \tau, \beta\rangle$ be a type 3 hiproof. We define the* skeleton *of h_3, written $\mathtt{sk}_3(h_3)$, to be the tree, $\tau(u_\perp)$, where $u_\perp$ is the sink guaranteed by Proposition 1.* □

The next result shows that this is a reasonable definition of skeleton. In other words, it corresponds to skeletons of type 1.

Proposition 5. *For all hiproofs h_3 of type 3, $\mathtt{sk}_3(h_3) = \mathtt{sk}_1(\mu_{31}(h_3))$ and for all hiproofs h_1 of type 1, $\mathtt{sk}_1(h_1) = \mathtt{sk}_3(\mu_{13}(h_1))$.* □

In order to define the translation from type 1 to type 3, we first need to define a notion of abstraction of hiproofs at a node.

Definition 10. *Let $h_1 = \langle V, \leq_i, \rightarrow_s, t\rangle$ be a type 1 hiproof, and let $z \in V$. We define the* abstraction *of h_1 at z, written $\mathtt{abs}_1(z, h_1)$, to be the type 1 hiproof $\langle V', \leq'_i, \rightarrow'_s, t'\rangle$, where $V' = \{v \in V | v \not<_i z\}$, $\leq'_i$ and t' are the restrictions of $\leq_i$ and t to V', and $v_1 \rightarrow'_s v_2$ if and only if $v_1, v_2 \in V'$, and either $v_1 \neq z$ and $v_1 \rightarrow_s v_2$, or $v_1 = z$ and there exists a $v \leq_i z$ such that $v \rightarrow_s v_2$.* □

This is, in some sense, dual to the skeleton since it throws away the contents of a node. We say that h' is an abstraction of h if h' is the abstraction of h at v for some v.

To define a map from type 1 to type 3, we must extend the definition of abstraction for type 1 to sets of nodes. First note that if h_1 nodes z_1 and z_2 are $\leq_i$-incomparable, then $\mathtt{abs}_1(z_1, \mathtt{abs}_1(z_2, h_1)) = \mathtt{abs}_1(z_2, \mathtt{abs}_1(z_1, h_1))$.

Let I be a $\leq_i$-incomparable set of nodes in h_1. Then, $\mathtt{abs}_1(I, h_1)$ is well-defined via

$$\mathtt{abs}_1(\{\}, h_1) = h_1$$

$$\mathtt{abs}_1(I \cup \{z\}, h_1) = \mathtt{abs}_1(z, \mathtt{abs}_1(I, h_1)), \text{ for } z \notin I.$$

Abstraction can also be defined for type 3 hiproofs but this is not needed here to define the translation.

Definition 11. *Define* $\mu_{13} : \mathbf{Hiproof}_1 \to \mathbf{Hiproof}_3$ *as the function sending each hiproof* $h_1 = \langle V, \leq_i, \to_s, tac\rangle$ *of type 1 to the hiproof* $\langle U, \rightsquigarrow, \tau, \beta\rangle$ *of type 3 given by the following data:*

- *The component trees are the skeletons of the abstractions of* h_1 *for all* $\leq_i$*-incomparable subsets*
- $t_1 \overset{v}{\rightsquigarrow} t_2$ *if and only if* $t_1 = \mathtt{sk}_1(\mathtt{abs}_1(I_1, h_1))$, $t_2 = \mathtt{sk}_1(\mathtt{abs}_1(I_2, h_1))$, *where* $I_1 = I \cup \{v\}$, $v \notin I$, $I_2 = I \cup \{v' \,|\, v' <^1_i v\}$.

□

We can show that $u_\top = \mathtt{sk}_1(\mathtt{abs}_1(I_{max}, h_1))$, where I_{max} is the set of $\leq_i$-maximal nodes, and $u_\perp$ is the skeleton of h_1, i.e. $\mathtt{sk}_1(\mathtt{abs}_1(\{\}, h_1))$. When $t = \mathtt{sk}_1(\mathtt{abs}_1(I, h))$, we will write t as t_I. Note that I is not unique, in general, since any node which is $\leq_i$-minimal, for example, can be added with no effect.

Proposition 6. μ_{13} *is well-defined.*

Proof. To show that μ_{13} *is well-defined, we must show that* $\mu_{13}(h_1)$ *is a valid type 3 hiproof.*

First we characterise $\rightsquigarrow^*$*. Define* $I <^1_i I'$ *if and only if* $I' = I_0 \cup \{v\}$, $v \notin I_0$, *and* $I = I_0 \cup \{v' \,|\, v' <^1_i v\}$. *Then define* $I \leq_i I'$ *if and only if* $I\ (<'_i)^*\ I'$. *Now, we have that* $I \leq_i I'$ *implies* $\forall i \in I \,.\, \exists i' \in I' \,.\, i \leq_i i'$ *(though not in the other direction). Clearly,* $t_I \rightsquigarrow^* t_{I'}$ *if and only if* $I' \leq_i I$.

Next, we show that $\leq_i$ *is a partial order. It is reflexive and transitive by definition. To see that it is antisymmetric, suppose* $I \leq_i I'$ *and* $I' \leq_i I$. *Let* $v \in I$. *Then there exists a* $v' \in I'$ *such that* $v \leq_i v'$, *and a* $v_2 \in I$ *such that* $v \leq_i v_2$. *Hence* $v = v'$ *(by antisymmetry of* $\leq_i$*), and so* $I \subseteq I'$. *Likewise,* $I' \subseteq I$, *so* $I = I'$, *and we have shown antisymmetry.*

This shows that $\mu_{13}(h_1)$ *is a dag. We must also check the type 3 conditions:*

1. $u_\top$ *is the top, since each* $I \leq_i \{i \,|\, i \text{ is } \leq_i -maximal\}$.
2. *Define the* shell *of a hiproof,* $\mathtt{shell}(h)$, *as* $\mathtt{abs}_1(I_{max}, h)$, *and* $\mathtt{sub}(h, v)$ *to be the hiproof 'inside the node* v *in* h*', i.e. the hiproof rooted at the top node included in* v, *and with no nodes outwith* v.
   Then, if $t_1 = \mathtt{sk}_1(\mathtt{abs}_1(I \cup \{v\}, h)) \overset{v}{\rightsquigarrow} t_2 = \mathtt{sk}_1(\mathtt{abs}_1(I \cup \{v'|v' <^1_i v\}, h))$, *then it can be shown that* $t_2 = graft(t_1, v, \gamma, f)$, *where* $\gamma = \mathtt{tree}(h, v)$, *is defined as* $\mathtt{shell}(\mathtt{sub}(h, v))$, *and* f, *mapping the children of* v *in* t_1 *to nodes in* t_2, *is given by* $v_1 \mapsto v_2$ *if and only if* v_2 *is the parent of* v_1 *in* t_2, *and so* $v_2 \in \gamma$.
3. *By the definition of* $\overset{v}{\rightsquigarrow}$, $u_0 = \mathtt{sk}_1(\mathtt{abs}_1(I \cup \{v\}, h)) \overset{v}{\rightsquigarrow} u_1, u_2$ *implies* $u_1, u_2 = \mathtt{sk}_1(\mathtt{abs}_1(I \cup V', h))$, *where* $V' = \{v' \,|\, v' <'_i v\}$.
4. *Suppose* $t \overset{v_1}{\rightsquigarrow} t_1$ *and* $t \overset{v_2}{\rightsquigarrow} t_2$, *then we have* $t = \mathtt{sk}_1(\mathtt{abs}_1(I \cup \{v_1, v_2\}, h)) \overset{v_1}{\rightsquigarrow} \mathtt{sk}_1(\mathtt{abs}_1(I \cup V'_1 \cup \{v_2\}, h))$, *and* $t \overset{v_2}{\rightsquigarrow} \mathtt{sk}_1(\mathtt{abs}_1(I \cup \{v_1\} \cup V'_2, h))$. *Then these come together confluently as* $\mathtt{sk}_1(\mathtt{abs}_1(I \cup V'_1 \cup V'_2, h))$.

□

One needs to do a little work to show that the constructions yielding type 1 and type 3 hiproofs from each other are mutually inverse up to coherent isomorphism: the latter point requires some straightforward category theory to make precise.

6 Conclusions

We have presented definitions of two structures arising in tactic-based theorem proving. The primary definition encapsulates our graphical intuition for this form of hierarchical proofs, and the other corresponds more closely to dynamic traces of unfoldings or proof views.

In work that we have not described in this paper, we have also looked at ordered hiproofs, where the underlying proof structures are ordered trees (in contrast to the unordered trees considered here), and hierarchical structures induced from proofs forests, which we call *hitacs*. Indeed, there is a wide range of hi-structures which can be induced from a correspondingly wide range of tree-based proof structures. We aim to develop the necessary categorical machinery for placing them in a common framework. Finally, we have also developed a term language and equational calculus, as well as the natural connections to the semantic structures presented here.

References

1. Cheikhrouhou, L., Sorge, V.: PDS — A Three-Dimensional Data Structure for Proof Plans. In: Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000), Monastir, Tunisia (2000)
2. Kapur, D., Nie, X., Musser, D.R.: An overview of the Tecton proof system. Theoretical Computer Science **133** (1994) 307–340
3. Richardson, J.D.C., Smaill, A., Green, I.: System description: proof planning in higher-order logic with Lambda-Clam. In: 15th International Conference on Automated Deduction. (1998) 129–133
4. Denney, E., Power, J., Tourlas, K.: Hiproofs: A hierarchical notion of proof tree. In: Proceedings of Mathematical Foundations of Programing Semantics (MFPS). Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier (2005)