Hiproofs: A hierarchical notion of proof tree

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

[a] *RIACS, NASA Ames Research Center, Moffett Field, CA 94035, USA*

[b] *Laboratory for the Foundations of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, Scotland*

Abstract

Motivated by the concerns of theorem-proving, we generalise the notion of proof tree to that of hierarchical proof tree. Hierarchical trees extend ordinary trees by adding partial order structure to the set of nodes: that allows us to visualise a node as a rectangle in the plane rather than as a point, letting us use the containment relation to express structure additional to that given by a tree. A hierarchical proof tree, or hiproof for short, is a hierarchical tree with nodes labelled by tactics. We motivate the details of our definition by reference to the behaviour of tactics in tactical theorem proving. We characterise the construction of the ordinary proof tree underlying a hierarchical proof tree as a left adjoint. We then analyse the notion of proof refinement with respect to hierarchy, and we give a characterisation of hiproofs that is more directly suited to implementation.

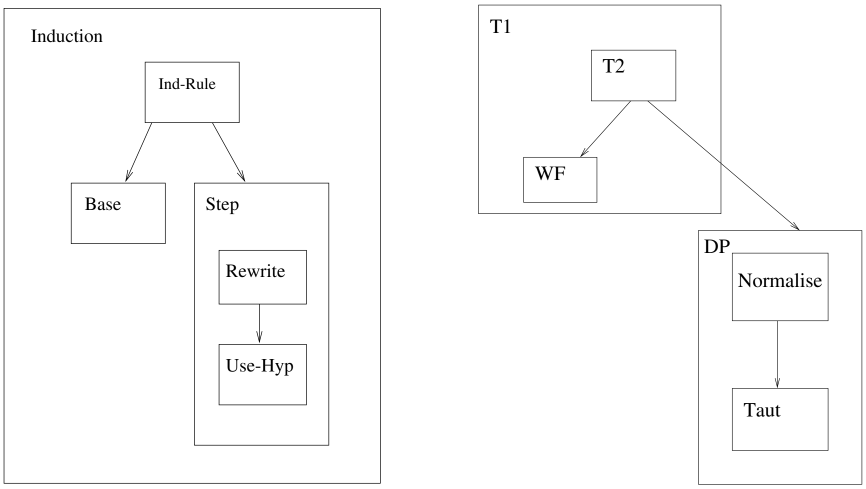
Key words: proof tree, hierarchical proof tree, skeleton, adjoint, refinement, tactical theorem proving

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 sub-tactic 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

[1] This work has been supported by EPSRC grants nos. GR/M45030 (Ewen Denney) GR/N64571/01 and GR/586372/01 (John Power) and GR/N12480 (Konstantinos Tourlas). The first author did most of this work while in Edinburgh.

[2] Email: `edenney@email.arc.nasa.gov`, `ajp@inf.ed.ac.uk`, `kosutamu@yahoo.co.uk`



(a) (b)

Fig. 1. Two hierarchical proofs

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 [5,8,11]. In this paper, we take 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 based on common tactic structure (which is generally independent of the underlying logic). Our central definition, abstracting from Figures 1(a) and 1(b), is that of a *hierarchical proof tree* or *hiproof*. We analyse an appropriate choice of axioms for hiproofs in Section 2, the relationship between a hiproof and its underlying ordinary proof in Section 3, and a notion of refinement of hiproofs in Section 4. Finally, we characterise hiproofs in terms more amenable to implementation in Section 5.

Compared to the hierarchical structures typically implemented in modern theorem provers, our work is an abstract first step: hiproofs abstract away many concrete and operational features. The key abstractions are as follows:

- we only model tactics, not goals. Hierarchy *per se* is independent of the underlying logic. Moreover, tactics alone support rich structure, and we seek the simplest possible framework in which to study it.

- we consider a tactic as a black box, giving no implementation details of it beyond a record of which other tactics define it. We treat inference rules and axioms as primitive tactics.
- we only model the static structure of tactics, not their dynamics. Our diagrams represent only the sequence of tactic applications leading to a proof, not the tactic definitions themselves or information about proof search.
- we consider tree structure rather than dags. Implementations often use dags, but formal logic generally does not. Our aim is to study the structure of proof rather than particular implementations, allowing us independence from specific notions of basic proof and implementation technology.

The central result of the paper, in Section 3, characterises the *skeleton*, or underlying ordinary proof, of a hiproof, as a left adjoint to the inclusion of ordinary proofs in hiproofs. One wants to lift constructions on ordinary proofs via the skeleton functor to constructions on hiproofs. So the central results of Sections 4 and 5, for refinement and towards implementation respectively, show that the relevant constructions respect skeletons.

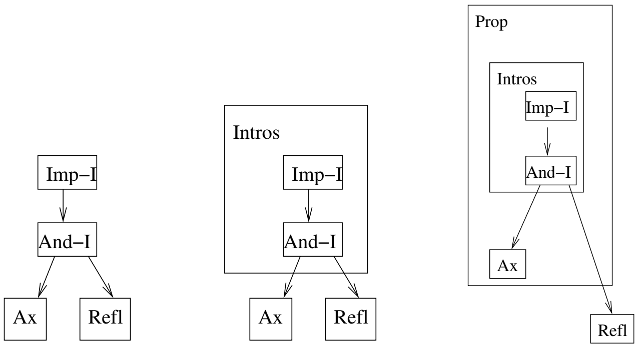
The use of diagrams in logic is far from new [2]. Fitch-style boxed natural deduction is one way to draw the boxes on a given proof, contrasting with the situation here. But hierarchical proofs along our lines appear in proof planning [4]. For example, two different representations appear in [9]. We know of little analysis of algebraic features of proofs, but see [10], which studies the dynamics of a representation language: we only address statics here, but one needs such statics in order to study dynamics.

More generally, any system (such as Coq, HOL and PVS) in which tactics may be defined from other tactics leads naturally to the kind of hierarchical proof here. In particular, the notion of hierarchy we formalise relates to that underlying the Lambda Clam family of proof planners [11], although Lambda Clam does not yet allow tactics to leave open goals as we do in Figure 1(b) for example. The Tecton system [8] also supports hierarchical proof: its hierarchy is 'two level' while ours allows arbitrary nesting. The PDS data structure [5], implemented in Omega [3] and other systems, is of similar generality to our hiproofs but is less abstract and includes implementation features. A PDS consists of *names*, *sequents*, and elements called *justifications* and *reasons*. Of those, named nodes and justification elements have counterparts in hiproofs. Sequents and reasons implement goals and back-tracking, and so have no counterparts in hiproofs. Otherwise, a PDS (one which, moreover, happens to be a tree at the lowest level) corresponds closely to a concrete characterisation of hiproof.

In a different direction, hiproofs are closely related to higraphs (see for instance [1]), but the notions of refinement differ. This paper is to be understood as a first abstract step towards hierarchy in proofs, with no attempt to give the implementation structure of the languages and systems mentioned above.

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

Fig. 2. A simple natural deduction proof.



(a) (b) (c)

Fig. 3. Introducing hierarchy in proof diagrams by grouping.

2 Hierarchical proof trees

In this section, we define the notion of a hierarchical proof tree or hiproof. To motivate it, 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 2.1 Consider a natural deduction proof of $A \Rightarrow A \wedge (x = x)$, as in Figure 2. 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 rules, with the order of those rules represented by a proof tree as in Figure 3(a). Typically, however, theorem provers allow the use of higher-level tactics that 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 3(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 3(c). □

Example 2.1 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 $v \leq_{\mathrm{i}} w$ 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. Standard definitions of trees and forests are given in the appendix.

Definition 2.2 A *hierarchical proof tree*, or *hiproof* for short, consists of a (necessarily finite) forest *qua poset* $i = \langle V, \leq_{\mathrm{i}} \rangle$ and a forest $s = \langle V, \rightarrow_{\mathrm{s}} \rangle$, together with a function $t : V \rightarrow \Lambda$ which labels the nodes in V with tactic identifiers in Λ, subject to the following conditions:

(i) arrows always target outer nodes: whenever $v \rightarrow_{\mathrm{s}} w_1$ and $w_1 <_{\mathrm{i}} w_2$, then $v <_{\mathrm{i}} w_2$

(ii) arrows always emanate from inner nodes: whenever $w_1 \leq_{\mathrm{i}} v$ and $v \rightarrow_{\mathrm{s}} w_2$ then $v = w_1$

(iii) inclusion and sequence are mutually exclusive: whenever $v \leq_{\mathrm{i}} w$ and $v \rightarrow_{\mathrm{s}}^{\star} w$, then $v = w$

(iv) 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 $\rightarrow_{\mathrm{s}}$ edge:

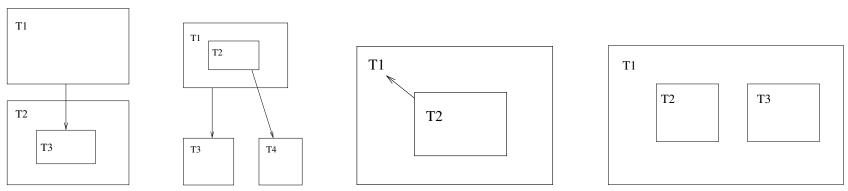
$$\forall v, v' \in V.\ siblings_i(v, v') \wedge isroot_s(v) \wedge isroot_s(v') \implies v = v'.$$

$\square$

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 2.1 satisfies the condition. Observe that the fourth condition together with finiteness imply that there is a unique node that is maximal with respect to $\leq_{\mathrm{i}}$ and has no incoming $\rightarrow_{\mathrm{s}}$ edge, acting as a kind of hierarchical root.

The main theorem justifying the axioms is Theorem 3.9, which shows that every hiproof unfolds to give an ordinary proof. But before developing that result, we shall analyse the axioms by looking at some non-examples. The axioms are designed to ensure that none of the diagrams in Figure 4 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 4(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,



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

Fig. 4. Four non-examples of hiproofs

control flows from the most recently executed tactic, i.e. the innermost, outwards, but Figure 4(b) contradicts that. We want to exclude Figure 4(c) too in order to avoid circularity of unfolding and sequencing. Finally, Figure 4(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 4(a). The second condition precludes Figure 4(b). The third condition precludes Figure 4(c): the similar structure with the arrow pointing the other direction is already precluded by the second condition. And the fourth condition precludes Figure 4(d) as well as the similar non-proof example obtained from Figure 4(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 Relating proof trees and hierarchical proof trees

In this section, we relate hierarchical proof trees with proof trees. Not only are proofs instances of hiproofs with trivial hierarchy, but, more importantly, every hiproof unfolds to give an ordinary proof, which we call its skeleton. As Example 2.1 illustrates, the relationship between ordinary proofs and hiproofs underlies our semantic understanding of the behaviour of hiproofs. So we should like to characterise our definition of skeleton axiomatically, and we do that here by showing that it acts as a left adjoint to the canonical inclusion. We retain the terminological conventions of Section 2.

Definition 3.1 A *proof tree* consists of a tree, $\langle V, \rightarrow, r \rangle$, together with a tactic labeling function, $t : V \rightarrow \Lambda$. □

Definition 3.2 A *proof tree map* $\alpha : \langle V, \rightarrow, r, t \rangle \rightarrow \langle V', \rightarrow', r', t' \rangle$ is a function $\alpha : V \rightarrow V'$ such that α preserves roots and satisfies the following:

- $u \rightarrow v \Rightarrow \alpha(u) \rightarrow'^{*} \alpha(v)$
- $t(u) = t'(\alpha(u))$. □

Proof trees together with proof tree maps form a category we denote by **Proof**. The category **Proof** has finite products and finite coproducts. Such category theoretic structures allow one to build complex proofs from less complex ones, and they provide transformations between proofs. The coproduct of a pair of proof trees is given by joining their roots. The product involves nodes with pairs of labellings, cf. Section 5. There are also sophisticated monoidal-like structures that, given a proof $\langle V, \rightarrow, r \rangle$ and a family of proofs whose roots agree with the leaves of $\langle V, \rightarrow, r \rangle$, allow one to attach the latter proofs at each of the leaves of $\langle V, \rightarrow, r \rangle$, cf [7] and Section 4. In further work, we want to lift such constructions from proof trees to hierarchical proof trees. But first we need to establish the precise nature of the relationship between the two notions, and we do that by finding an adjunction.

In order to find an interesting adjunction, characterising a natural notion of skeleton of a hiproof, we need a delicate definition of the notion of hiproof map, using a refinement of the partial order $\leq_i$.

Definition 3.3 In a hiproof, let $\leq_f$ denote the suborder of $\leq_i$ generated by putting $u \leq_f v$ if u is a child of v and u has no incoming $\rightarrow_s$ edge. □

It follows from the fourth condition of Definition 5.4 that $\leq_f$ is always a finite set of finite chains.

Definition 3.4 A *hiproof map* $\alpha : \langle V, \leq_i, \rightarrow_s, t \rangle \rightarrow \langle V', \leq_i', \rightarrow_s', t' \rangle$ is a function $\alpha : V \rightarrow V'$ such that α preserves $\rightarrow$-roots and satisfies the following:

- $u \leq_f v \Rightarrow \alpha(u) \leq_f' \alpha(v)$
- $u \rightarrow_s v \Rightarrow \alpha(u) \rightarrow_s'^{*} \alpha(v)$
- $t(u) = t'(\alpha(u))$. □

Hiproofs and hiproof maps form a category **Hiproof**. Proofs can naturally be considered as flat hiproofs as follows:

Definition 3.5 Let $\mathcal{E}$: **Proof** $\to$ **Hiproof** send the proof $\gamma = \langle V, \to, r, t\rangle$ to $\langle V, \mathrm{id}_V, \to, t\rangle$. □

Proposition 3.6 *The functor $\mathcal{E}$ is fully faithful and preserves finite products and finite coproducts.* □

The most important relationship between proofs and hiproofs is given by the unfolding of a hiproof into an ordinary proof, called its skeleton, as we discussed in analysing the non-examples of Figure 4. In order to describe and characterise the skeleton in general, we use the following correspondence between trees and partial orders: note, in the following, that $v_1 \leq v_2$ means that, in the corresponding tree, there is a path in the opposite direction, i.e., from v_2 to v_1.

Proposition 3.7 *To give a tree is equivalent to giving a finite poset $T = \langle V, \leq\rangle$ that satisfies the 'non-sharing' condition*

$$\forall\, x, y, z \in F.\;\; x \leq y \;\; \textit{and} \;\; x \leq z \;\; \textit{implies} \;\; y \leq z \; \textit{or} \; z \leq y \;\; ,$$

and has a top element $\top$. The resulting tree is $\langle V, \to, \top\rangle$, where $v \to v'$ whenever $v' \in cover_{\leq}(v)$, and the cover of a node is defined to be the set of nodes immediately below it. □

Corollary 3.8 *To give a forest is equivalent to giving a* finite *poset $\langle V, \leq\rangle$ subject to the 'non-sharing' condition of Prop. 3.7: $\forall\, x, y, z \in V.\;\; x \leq y$ and $x \leq z$ implies $y \leq z$ or $z \leq y$.* □

Now we can define and characterise the skeleton of a hiproof as a left adjoint as follows:

Theorem 3.9 *The functor $\mathcal{E}$ has a left adjoint, denoted by* `sk`*, sending a hiproof $h = \langle V, \leq_i, \to_s, t\rangle$ to what we call its* skeleton*, given as follows: the Λ-labelled tree $\langle V_T, \to_T, r\rangle$, corresponding to the finite poset $T = \langle V_T, \leq_T\rangle$, where V_T is the set of leaves of $\leq_i$, and $v_1 \leq v_2$ if and only if there exists $v \in V$ such that $v_2 \leq_i v$ and $v_1 \to_s v$.* □

Example 3.10 The skeleton of the hiproof in Figure 1(b) is given by Figure 5, and the skeleton of the hiproof depicted in Figure 3(c) is the proof depicted in Figure 3(a). □

The skeleton of a hiproof gives us a notion of an unfolding of a hiproof, given by the tree of underlying atomic tactics. If we think of atomic tactics as inferences and axioms, this gives us a standard non-hierarchical proof. We sometimes regard a skeleton as a proof and sometimes as a flat hiproof: formally, this amounts to applying the composite functor $\mathcal{E}$`sk` and freely using the fully faithfulness of $\mathcal{E}$, which allows us to regard **Proof** as a full subcategory of **Hiproof**. Any construction or transformation we make of a hiproof,

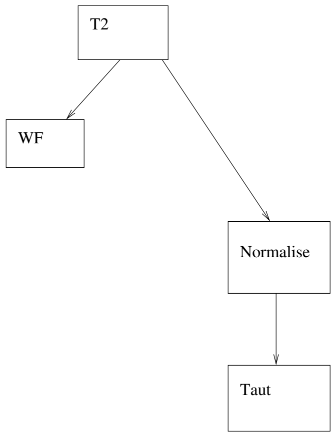


Fig. 5. Skeleton of a hiproof

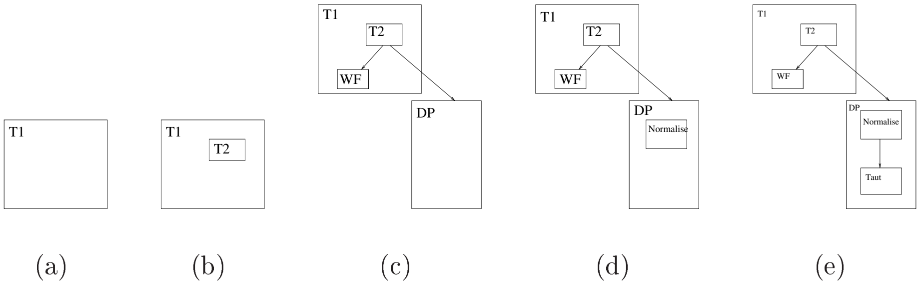


Fig. 6. Refinement of hiproofs

as we consider in the following sections, needs to be justified by preservation of, or a corresponding construction on, its skeleton.

4 Hiproof refinement

The fundamental construction one wants to make on a hiproof is refinement: the idea is that h_1 refines to h_2 when h_2 extends the proof in h_1, where extension is to be understood informally as "proving more". So our goal here is to formalise that. Proofs can grow in two ways: either by a tactic calling a subtactic, or by applying a new tactic to a subgoal. These correspond, respectively, to inclusion of and sequential composition with a tactic. Since we want to formalise a semantic, rather than operational, notion of refinement, our definition of refinement amounts to allowing trees to grow arbitrarily 'at the bottom' and, in the case of forests, adding additional trees. The definitions in this section make this precise.

Example 4.1 Figure 6 shows a refinement, from left to right, of a hiproof. Refinement generates a partial order, and this is not the only possible sequence.

We first define refinement for trees and forests, the simple structures from which hiproofs are constructed. We need a few supplementary definitions.

Definition 4.2 A rooted subtree T' of a tree $T = \langle V, \rightarrow, r\rangle$ is a tree $\langle V', \rightarrow', r\rangle$ where

- V' is an upwards-closed non-empty subset of V: whenever $v' \in V'$ then for all v'' such that $v'' \rightarrow v'$ one also has $v'' \in V$; and thus also $r \in V'$
- $\rightarrow'$ is the restriction of $\rightarrow$ to $V' \times V'$. □

Henceforth, we refer to rooted subtrees simply as subtrees. Intuitively, refining a tree amounts to the addition, at possibly any level below the root, of any (finite) number of new nodes. Thus the original tree is always a subtree of any tree that refines it:

Definition 4.3 A tree $\langle V_1, \rightarrow_1, r_1\rangle$ refines to the tree $\langle V_2, \rightarrow_2, r_2\rangle$, written

$$\langle V_1, \rightarrow_1, r_1\rangle \sqsubseteq_{\mathrm{T}} \langle V_2, \rightarrow_2, r_2\rangle \ ,$$

if the former is a subtree of the latter. □

Definition 4.4 A forest F_1 refines to the forest F_2, written $F_1 \sqsubseteq_{\mathrm{F}} F_2$, if there exists an injective function $\iota : F_1 \rightarrow F_2$ such that for all trees $T \in F_1$, $T \sqsubseteq_{\mathrm{T}} \iota(T)$ □

In practice, it is easier to use the following characterisations of forest refinement, given by regarding a forest as a graph or poset:

Proposition 4.5 *Given forests $F_1 = \langle V_1, \rightarrow_1\rangle$ and $F_2 = \langle V_2, \rightarrow_2\rangle$, $F_1 \sqsubseteq_{\mathrm{F}} F_2$ if and only if $V_1 \subseteq V_2$ and for all $v, v' \in V_1$, $isroot_{F_1}(v) \iff isroot_{F_2}(v)$ and $v \rightarrow_1 v' \iff v \rightarrow_2 v'$.* □

Proposition 4.6 *Given forests $F_1 = \langle V_1, \leq_1\rangle$ and $F_2 = \langle V_2, \leq_2\rangle$, $F_1 \sqsubseteq_{\mathrm{F}} F_2$ if and only if for all $v, v' \in V_1$, $isroot_{F_1}(v) \iff isroot_{F_2}(v)$ and $v \in cover_{F_1}(v') \implies v \in cover_{F_2}(v')$.* □

Thus forest refinement can be characterised in terms of preservation of roots and inclusion of the corresponding order. This justifies the way we define hiproof refinement.

Definition 4.7 A hiproof $h = \langle V, \leq_{\mathrm{i}}, \rightarrow_{\mathrm{s}}, t\rangle$ is said to refine to the hiproof $h' = \langle V', \leq'_{\mathrm{i}}, \rightarrow'_{\mathrm{s}}, t'\rangle$, written $h \sqsubseteq_1 h'$, if $\langle V, \leq_{\mathrm{i}}\rangle \sqsubseteq_{\mathrm{F}} \langle V', \leq'_{\mathrm{i}}\rangle$, $\langle V, \rightarrow_{\mathrm{s}}\rangle \sqsubseteq_{\mathrm{F}} \langle V', \rightarrow'_{\mathrm{s}}\rangle$ and labels are preserved, i.e. $t \subseteq t'$ (regarding each of t and t' as a finite set of pairs, e.g. $\langle v, t(v)\rangle$). □

Refinement for ordinary proofs is defined by tree refinement, as in Definition 4.3, subject to respect for labelling. It follows by inspection of the definitions that refinement for hiproofs restricts along $\mathcal{E}$ to refinement for ordinary proofs. More importantly, `sk` sends hiproof refinement to ordinary refinement as follows.

Theorem 4.8 *If a hiproof h refines to h', then $\mathtt{sk}(h)$ refines to $\mathtt{sk}(h')$.* □

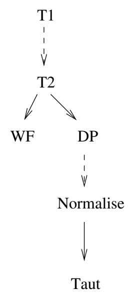
We shall not prove this in this section, as a stronger result follows from our characterisation of hiproofs in terms of ordinary labelled trees with more complex labelling in Section 5. So we shall formulate the stronger statement later.

5 A concrete characterisation of hiproofs

In this section, we characterise hiproofs in terms more amenable to implementation. The definition of hiproof consists of two forests. But they can be combined into a single tree with more complex labelling by dint of the following. Let R^+ denote the transitive closure of a binary relation R.

Proposition 5.1 *No 'composite cycles' exist in a hiproof: writing $v <^1_{\mathrm{i}} v'$ whenever $v \in cover_{\leq_{\mathrm{i}}}(v')$, then for all $v, v' \in V$, whenever $v\,(>^1_{\mathrm{i}} \cup \rightarrow_{\mathrm{s}})^+\,v'$ one has $v \neq v'$.* □

So, as a first attempt at a characterisation of hiproofs in such terms, we might try to represent Figure 1(b) as follows, cf. [9]:



The solid lines denote composition and the dashed lines inclusion. But this representation does not distinguish the similar hiproof in which `DP` is a subtactic of `T1`. That can be resolved by pairing tactics with their level in the inclusion hierarchy: in Figure 1(b), `T1` and `DP` have level 0, `T2` has level 1, and so on. But then we do not need to distinguish between two kinds of arrow, as that information is determined by the respective levels of adjacent nodes. This motivates the following definition:

Definition 5.2 A *hiproof of type 2* is a tree $\langle V, \rightarrow, r\rangle$ together with functions: $t : V \rightarrow \Lambda$ and $l : V \rightarrow \mathbb{N}$, subject to the following conditions:

(i) $l(r) = 0$

(ii) if $v \rightarrow v'$, then $l(v') \leq l(v) + 1$

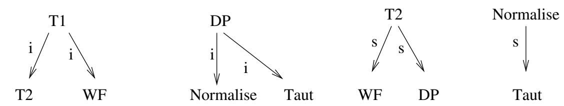
(iii) if $v \rightarrow v_1$, $v \rightarrow v_2$ and $l(v_1) = l(v) + 1$, then $v_1 = v_2$. □

We shall often identify a node v with a pair $\langle \lambda, l\rangle$, thereby implicitly asserting that $t(v) = \lambda$ and $l(v) = l$. The function $l : V \rightarrow \mathbb{N}$ sends a node to what we call its *inclusion level*. So the first condition of the definition asserts that the root of the tree lies at inclusion level 0. The second condition states that nodes are only (directly) connected to those nodes that they directly include or with which they are composed. In the latter case, the node can 'escape' to an arbitrarily lower inclusion level. The third condition asserts uniqueness of children if one increases inclusion level.

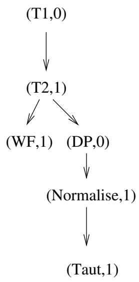
In Definition 5.2, both composition and inclusion depth are implicit in the structure of the nodes. So, in terms of cognitive properties, the diagrams arising from hiproofs of type 2 seem less suitable for human users than the diagrams arising from hiproofs of type 1. In the latter, two distinct visual relations, spatial containment and edge connectivity, are used to represent tactic inclusion and composition (see Figure 1(b)), thus giving less scope for confusion. In contrast, owing to their economy, type 2 hiproofs have advantages as internal, machine-oriented representations.

With a little thought about levels, hiproofs of type 2 readily form a category we denote by $\mathbf{Hiproof}_2$. We sometimes refer to the hiproofs of Definition 2.2 as hiproofs of type 1.

Example 5.3 Figure 1(b) not only forms a hiproof but also a hiproof of type 2. The partial order information in Figure 1(b) may be unfolded as follows:



where the labels on the arrows distinguish between the inclusion and composition forests. If we recombine this data using all the s-labelled arrows but only those i-labelled arrows whose codomain has no incoming s-labelled edge, we obtain a hiproof of type 2 as follows:



where nodes are informally represented by their tactic identifiers and inclusion levels. Now compare this hiproof of type 2 with Figure 5, which is the skeleton of the hiproof of Figure 1(b). In terms of our hiproof of type 2, the skeleton is determined by those nodes v for which $l(v)$ is locally maximum, so if $v \to v'$ and $l(v') \leq l(v)$, it follows that v is in the skeleton. $\square$

Example 5.3 suggests that any proof may be equivalently represented in either type of hiproof. Indeed, the two definitions of hiproof are related by an isomorphism of categories. Moreover, we can directly and naturally describe the skeleton of a hiproof in terms of hiproofs of type 2, so the correspondence between the two notions of hiproof respects the underlying proof structure.

Definition 5.4 Define the functor μ_{12} : **Hiproof** $\to$ $\mathbf{Hiproof}_2$ by sending a hiproof $\langle V, \leq_{\mathrm{i}}, \to_{\mathrm{s}}, t\rangle$ of type 1 to the hiproof $\langle V, \to, r, t, l\rangle$ of type 2 given by the following data:

- $l(v)$ is defined to equal 0 whenever $\mathit{isroot}_{\leq_{\mathrm{i}}}(v)$, and, inductively, to equal $l(\mathit{parent}_{\leq_{\mathrm{i}}}(v)) + 1$ otherwise; (explicitly in forest-qua-poset notation: for each $v' \neq \top$, $\mathit{parent}_{\leq_{\mathrm{i}}}(v')$ is the unique v such that $v' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v)$);
- $v \to v'$ whenever $v \to_{\mathrm{s}} v'$ or, $v' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v)$ and $\mathit{isroot}_{\to_{\mathrm{s}}}(v')$; and
- r is the root of the hiproof (see the remark after the definition). □

Definition 5.5 Define the functor μ_{21} : $\mathbf{Hiproof}_2$ $\to$ **Hiproof** by sending a hiproof $\langle V, \to, r, t, l\rangle$ of type 2 to the hiproof $\langle V, \leq_{\mathrm{i}}, \to_{\mathrm{s}}, t\rangle$ of type 1 given by the following data:

- $v \to_{\mathrm{s}} v'$ whenever $v \to v'$ and $l(v') \leq l(v)$
- $\leq_{\mathrm{i}}$ is the reflexive and transitive closure of $<^1_{\mathrm{i}}$, the latter being defined thus: $v <^1_{\mathrm{i}} v'$ whenever a (non-empty) path $v' = v_0 \to \ldots \to v_n \to v_{n+1} = v$ exists such that $l(v_1) = l(v_0) + 1$ and $l(v_i) = l(v_{i+1})$ for $1 \leq i \leq n$. □

The proofs of well-definedness of μ_{12} and μ_{21} amount to Propositions A.3 and A.5 in Appendix A. We now show that they are mutually inverse.

Theorem 5.6 *The functors* μ_{12} : **Hiproof** $\to$ $\mathbf{Hiproof}_2$ *and* μ_{21} : $\mathbf{Hiproof}_2$ $\to$ **Hiproof** *are mutually inverse.*

Proof. We only sketch one direction of the argument as the other is similar. Let $h_1 = \langle V, \leq_{\mathrm{i}}, \to_{\mathrm{s}}, t\rangle$, $h_2 = \mu_{12}(h_1) = \langle V_2, \to, r, t_2\rangle$ and $h_1' = \mu_{21}(h_2) = \langle V', \leq_{\mathrm{i}}', \to_{\mathrm{s}}', t'\rangle$. It follows directly from the definitions that $V = V' = V_2$ and $t = t_2 = t'$.

We first show $\leq_{\mathrm{i}}' \subseteq \leq_{\mathrm{i}}$ (required to show $\leq_{\mathrm{i}}' = \leq_{\mathrm{i}}$). Assume $v \leq_{\mathrm{i}}' v'$ and proceed by induction on the length of the path linking v' to v in the forest $\langle V, \leq_{\mathrm{i}}'\rangle$. The base case is trivial. For the inductive case, assume $v \leq_{\mathrm{i}} v''$ by the induction hypothesis and $v'' \in \mathit{cover}_{\leq_{\mathrm{i}}'}(v')$. From the latter and Def. 5.5, a path $v' = v_0 \to \ldots \to v_n \to v_{n+1} = v''$ must exist in h_2 such that $l(v_1) = l(v_0) + 1$ and $l(v_{i+1}) = l(v_i)$ for $1 \leq i \leq n$. It follows from Def. 5.4 that $v' = v_0 = \mathit{parent}_{\leq_{\mathrm{i}}}(v_1)$ or, equivalently, $v_1 \in \mathit{cover}_{\leq_{\mathrm{i}}}(v')$. From $v_i \to v_{i+1}$ and $l(v_i) = l(v_{i+1})$, it follows that v_i and v_{i+1}, for i ranging as above, share $v' = v_0$ as their parent in $\leq_{\mathrm{i}}$. So $v'' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v')$. Together with the inductive hypothesis $v'' \leq_{\mathrm{i}} v'$, this proves $v \leq_{\mathrm{i}} v'$.

The proof of $\leq_{\mathrm{i}} \subseteq \leq_{\mathrm{i}}'$ is similar. And the equality $\to_{\mathrm{s}} = \to_{\mathrm{s}}'$ can also be proved similarly. □

We now turn to skeletons. We need to characterise the construction of the skeleton in terms of hiproofs of type 2. Given a type 2 hiproof $h_2 = \langle V, \to, r, t, l\rangle$, we call a node an *inclusion node* if it has a child with greater inclusion level.

Theorem 5.7 *Given a type 2 hiproof $h_2 = \langle V, \rightarrow, r, t, l\rangle$, the following data, which we denote by* $\mathtt{sk}_2(h_2)$*, agrees, via* μ_{12}*, with* $\mathtt{sk}$*: the* Λ*-labelled tree* $\langle V_T, \rightarrow_T, r\rangle$ *where* V_T *is the set of non-inclusion nodes of* h_2*, and* $v \rightarrow_T v'$ *if and only if* $v \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow v'$*, where* $v_1, \ldots, v_n$ *are inclusion nodes,* r *is the maximum non-inclusion node, and labelling is given by the restriction of the labelling function to* V_T*.* □

Proof. By well-founded induction on the hiproofs. At each stage one extends the hiproof by one leaf and shows that $\mathtt{sk}$ and $\mathtt{sk}_2$ and the $\mu\_$'s respect the extension. □

Finally, as promised at the end of Section 4, we formulate refinement in terms of hiproofs of type 2 and show that the formulation agrees, relative to the equivalence, with our formulation of refinement for hiproofs of type 1 in Definition 4.7.

Definition 5.8 A hiproof $h = \langle V, \rightarrow, r, t, l\rangle$ of type 2 refines to a hiproof $h' = \langle V', \rightarrow, r', t', l'\rangle$ of the same type, written $h \sqsubseteq_2 h'$, if and only if $\langle V, \rightarrow, r\rangle \sqsubseteq_{\mathrm{T}} \langle V', \rightarrow', r'\rangle$ and, moreover, $t \subseteq t'$ and $l \subseteq l'$. □

Theorem 5.9 *Let* h_1 *and* h_1' *be hiproofs. Then,* $h_1 \sqsubseteq_1 h_1'$ *holds if and only if* $\mu_{12}(h_1) \sqsubseteq_2 \mu_{12}(h_1')$ *does.* □

This shows that the two formulations of hiproofs are equivalent with respect to refinement. A proof of Theorem 4.8 follows directly.

6 Conclusions and Further Work

We have introduced and begun to develop a notion of hierarchical proof tree or hiproof, abstractly reflecting the use of tactics in theorem proving. We have presented axioms that allow one to unfold a hiproof to yield an ordinary proof, and we have illustrated the axioms by examples and non-examples. We have outlined how refinement works and we have given an alternative presentation of the definition that is better suited to implementation.

In practice, tactics possess much more complex structure than we have addressed here, where we have restricted ourselves to simple notions of tactic and proof. We regard this work as just a first step towards providing a semantic foundation for the topic. We believe that, suitably developed, the study of general properties and operations on hiproofs and extensions of the notion will help to give a principled way in which to design interfaces that are independent of the specifics of hiproof representation and allow one to reason about the correctness of implementation. We are actively investigating the application of hiproofs as a foundation for prover protocols, using an operational semantics which formalises how hiproofs are constructed from sequences of tactic applications.

To develop hiproofs as we have defined them, we next seek to define natural operations on hiproofs that are supported by the various proof assistants.

Examples are the various abstraction operations. Such 'zooming' operations have been considered for higraphs and statecharts [1,6], and there are natural operations to consider on them. We also plan to characterise the relationship between our semantic structures and the underlying logic, introducing a notion of stepwise refinement.

Acknowledgements The first author thanks Alan Bundy for his encouragement and interest in this work.

References

[1] Anderson, S., J. Power and K. Tourlas, *Zooming out of higraph-based diagrams: syntactic and semantic issues*, in: *Proceedings of CATS 2002, the Australasian Symposium on Theory of Computing*, Electronic Notes in Theoretical Computer Science (ENTCS) **61** (2002).

[2] Barwise, J. and E. Hammer, *Diagrams and the concept of logical system*, in: G. Allwein and J. Barwise, editors, *Logical Reasoning with Diagrams*, Oxford University Press, 1996 pp. 49–78.

[3] Benzmüller, C., L. Cheikhrouhou, D. Fehrer, A. Fiedler, X. Huang, M. Kerber, M. Kohlhase, K. Konrad, A. Meier, E. Melis, W. Schaarschmidt, J. Siekmann and V. Sorge, *Omega: Towards a mathematical assistant*, in: *Proceedings of CADE-14*, LNAI **1249** (1997).

[4] Bundy, A., *Proof planning*, in: B. Drabble, editor, *Proceedings of the 3rd International Conference on AI Planning Systems, (AIPS) 1996*, 1996, pp. 261–267, also available as DAI Research Report 886.

[5] Cheikhrouhou, L. and V. Sorge, *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.

[6] Harel, D., *On visual formalisms*, Communications of the ACM **31** (1988), pp. 514–530.

[7] Hermida, C., M. Makkai and A. J. Power, *Higher dimensional multigraphs*, in: *Proceedings of 13th LICS* (1998), pp. 199–206.

[8] Kapur, D., X. Nie and D. R. Musser, *An overview of the Tecton proof system*, Theoretical Computer Science **133** (1994), pp. 307–340.

[9] Richardson, J. and A. Bundy, *Proof Planning Methods as Schemas*, DAI Technical Report, Division of Informatics, University of Edinburgh (1999).

[10] Richardson, J. D. C. and A. Smaill, *Continuations of proof strategies*, in: *International Joint Conference on Automated Reasoning, IJCAR 2001 — Short Papers*, 2001, technical Report DII 11/01, Dipartimento di Ingegneria dell'Informazione, Università di Siena, Italy.

[11] Richardson, J. D. C., A. Smaill and I. Green, *System description: proof planning in higher-order logic with Lambda-Clam*, in: *15th International Conference on Automated Deduction*, 1998, pp. 129–133.

A Definitions and Technical proofs

Definition A.1 A *tree* $T = \langle V, \rightarrow, r\rangle$ is a finite dag $\langle V, \rightarrow\rangle$ together with a distinguished vertex $r \in V$, called the *root*, such that there is exactly one path from r to every other vertex $v \neq r$. For every edge $\langle v, v'\rangle \in\rightarrow$, which we shall conventionally write as $v \rightarrow v'$, one says that v' is a *child* of v or, equivalently, that v' has *parent* v. The vertices V in a tree are conventionally also called *nodes*. □

Definition A.2 A *forest* F is a finite set $\{T_1, \ldots, T_n\}$ of trees $T_j = \langle V_j, \rightarrow_j, r_j\rangle$. We shall write $v \rightarrow_F v'$ (or just $v \rightarrow v'$ when F understood) to mean that there exists tree T_j in F such that $v, v' \in V_j$ and $v \rightarrow_j v'$. Consequently we shall often also write the forest F as $\langle V, \rightarrow_F\rangle$ where V is the disjoint union of all V_j. □

Proposition A.3 *μ_{12} is well-defined, i.e. each $\mu_{12}(\langle V, \leq_{\mathrm{i}}, \rightarrow_{\mathrm{s}}, t\rangle)$ conforms to Def. 5.2.*

Proof. By Prop. 5.1 and observing that $\rightarrow^+ \subseteq (>^1_{\mathrm{i}} \cup \rightarrow_{\mathrm{s}})^+$, it follows that $\langle V, \rightarrow\rangle$ is an acyclic graph. Moreover, whenever $v_1 \rightarrow v$ and $v_2 \rightarrow v$ one has $v_1 = v_2$: for the only possible cases are $v_1 \rightarrow_{\mathrm{s}} v$ and $v_1 \rightarrow_{\mathrm{s}} v$, or, $v \in \mathit{cover}_{\leq_{\mathrm{i}}}(v_1)$ and $v \in \mathit{cover}_{\leq_{\mathrm{i}}}(v_2)$; $v_1 = v_2$ immediately follows from $\langle V, \rightarrow_{\mathrm{s}}\rangle$ and $\langle V, \leq_{\mathrm{i}}\rangle$ being forests. Thus, whenever a path $v_0 \rightarrow^\star v$ exists, it must be unique. We show that $r \rightarrow^\star_{\mathrm{s}} v$ for all $v \in V$ by induction on $d(v)$, the 'depth' of v with respect to $\rightarrow_{\mathrm{s}}$, which is defined thus: $d(v) = d(v') + 1$ whenever $v' \rightarrow v$ and $d(v) = 0$ otherwise. When $d(v) = 0$ then clearly $\mathit{isroot}_{\rightarrow_{\mathrm{s}}}(v)$ and $\mathit{siblings}_{\leq_{\mathrm{i}}}(v, r)$ (in the sense that $\mathit{isroot}_{\leq_{\mathrm{i}}}(v)$). Now $r = v$ and $r \rightarrow^\star v$ holds trivially. In the inductive case assume true for v' and $v' \rightarrow v$. Then the induction hypothesis yields $r \rightarrow v'$ and so, transitively, also $r \rightarrow^\star v$.

Showing that $l(v') \leq l(v) + 1$ whenever $v \rightarrow v'$ proceeds by case analysis. Case $v' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v)$ and $\mathit{isroot}_{\rightarrow_{\mathrm{s}}}(v')$ is immediate. When $v \rightarrow_{\mathrm{s}} v'$ one examines whether $\mathit{isroot}_{\leq_{\mathrm{i}}}(v')$ or not. When so, $l(v') = 0 \leq l(v) + 1$. When $v' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v'')$ for some v'', condition i yields $v' \in \mathit{cover}_{\leq_{\mathrm{i}}}(v'')$, from which $l(v) = l(v'') + 1 = l(v')$ follows.

Assume $v \rightarrow v_1$ and $v \rightarrow v_2$ such that $l(v_1) = l(v) + 1$. Then one must have $v = \mathit{parent}_{\leq_{\mathrm{i}}}(v_1)$ and hence $v_1 \leq_{\mathrm{i}} v$ in the type 1 hiproof. Further, we distinguish two cases: first, if $l(v_2) = l(v_1)$, one has $\mathit{siblings}_{\leq_{\mathrm{i}}}(v_1, v_2)$ while also $\mathit{isroot}_{\rightarrow_{\mathrm{s}}}(v_1) \wedge \mathit{isroot}_{\rightarrow_{\mathrm{s}}}(v_2)$. Then condition iv of Def. 2.2 establishes $v_1 = v_2$ as required. Similarly the case $l(v_2) \leq l(v)$ means $v \rightarrow_{\mathrm{s}} v_2$ while $v_1 \leq_{\mathrm{i}} v$, hence $v_1 = v_2$ by the first condition in the definition of type 1 hiproofs. Finally, that $l(r) = 0$ is obvious. □

Lemma A.4 *In the context of Def. 5.5, $isroot_{\rightarrow_{\rm s}}(v)$ is equivalent to $\forall v_0 \in V.\ (v_0 \rightarrow v \implies l(v_0) + 1 \leq l(v))$.*

Proof. Using the definition of $\rightarrow_{\rm s}$ and the tautology $(p \implies q) \iff (\neg p \vee q)$:

$$\begin{aligned} isroot_{\rightarrow_{\rm s}}(v) &\iff \nexists v_0.\ (v_0 \rightarrow v \ \wedge \ l(v) \leq l(v_0)) \\ &\iff \forall v_0.\ (v_0 \not\rightarrow v \ \vee \ l(v) > l(v_0)) \\ &\iff \forall v_0.\ (v_0 \not\rightarrow v \ \vee \ l(v) \geq l(v_0) + 1) \\ &\iff \forall v_0.\ (v_0 \rightarrow v \implies l(v_0) + 1 \leq l(v)) \end{aligned}$$

□

Proposition A.5 *μ_{21} is well-defined, i.e. each $\mu_{21}(\langle V, \rightarrow, r, t, l\rangle)$ is a hiproof of type 1.*

Proof. (Sketch) $<_{\rm i}$ is manifestly irreflexive and transitive. On the other hand, $\leq_{\rm i}$ is clearly antisymmetric, as follows from observing that $\leq_{\rm i} \subseteq (\rightarrow^\star)^{-1}$ while $\langle V, \rightarrow, r\rangle$ is a tree. Thus, the definition of $\leq_{\rm i}$ as the reflexive closure of $<^1_{\rm i}$ makes $\langle V, \leq_{\rm i}\rangle$ a poset.

The non-sharing condition, needed by Corol. 3.8 to show $\langle V, \leq_{\rm i}\rangle$ a forest, as required, also follows from $\langle V, \rightarrow, r\rangle$ being a tree and $\leq_{\rm i} \subseteq (\rightarrow^\star)^{-1}$:

to assume $v \leq_{\rm i} w_1$ and $v \leq_{\rm i} w_2$ while $w_1 \neq w_2$ would mean the existence of two distinct paths in $\langle V, \rightarrow, r\rangle$ from the root r to v, one via w_1 and the other via w_2, thus contradicting the fact of $\langle V, \rightarrow, r\rangle$ being a tree. One must therefore admit that, whenever $v \leq_{\rm i} w_1$ and $v \leq_{\rm i} w_2$, w_1 must equal w_2.

To show $\langle V, \rightarrow_{\rm s}\rangle$ a forest, as required, we shall show that $\langle V, (\rightarrow^\star_{\rm s})^{-1}\rangle$ is a forest-qua-poset and appeal to Corol. 3.8. The poset structure of $\langle V, (\rightarrow^\star_{\rm s})^{-1}\rangle$ is immediate as $\rightarrow$ is clearly antisymmetric. Again, observing that $\rightarrow^\star_{\rm s} \subseteq \rightarrow^\star$, the 'non-sharing' condition required by Corol. 3.8 follows, as above, from $\langle V, \rightarrow, r\rangle$ being a tree.

To show that $\leq_{\rm i}$ and $\rightarrow_{\rm s}$ are mutually exclusive in the sense of condition (iii) of Def. 2.2, consider first the case of $v \leq_{\rm i} w$ and $v \rightarrow^\star_{\rm s} w$: as the former implies $w \rightarrow^\star v$ and the latter implies $v \rightarrow^\star_{\rm s} w$, $v = w$ follows easily from the acyclicity of the tree $\langle V, \rightarrow, r\rangle$. In the case of $v \leq_{\rm i} w$ (hence also $l(w) \leq l(v)$) while $w \rightarrow^\star_{\rm s} v$ (and hence $l(v) \leq l(w)$), one must have $l(v) = l(w)$ and so, according to the definition of $\leq_{\rm i}$, $v = w$.

To establish condition i, first observe that $w_1 \in cover{\leq_{\rm i}}(w_2)$ means the existence of a non-empty path $w_2 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n \rightarrow w_1$ in the tree $\langle V, \rightarrow, r\rangle$ such that $l(v_1) = l(w_2) + 1$ and $l(v_1) = \ldots = l(v_n) = l(w_1)$. Assuming also that $v \rightarrow_{\rm s} w_1$, i.e. also $v \rightarrow w_1$, forces $v = v_n$, for $\langle V, \rightarrow, r\rangle$ is a tree. That $v \in cover_{\leq_{\rm i}}(w_2)$ now follows immediately from Def. 5.5.

For condition ii, suppose that $w_1 \leq_i v$ and $v \rightarrow_s w_2$. We must show that $v = w_1$. By the definition of μ_{21}, we have that $w_1\,(<^1_i)^*\,v$ and $v \rightarrow w_2$, with $l(w_2) \leq l(v)$. Suppose that the path from w_1 to v is non-empty, i.e. $w_1 <^1_i w'_0\,(<^1_i)^*\,v$, for some w'_0. Then, by definition of $<^1_i$, there exists w_0 such that $v \rightarrow w_0$ and $l(w_0) = l(v) + 1$. Now, by condition iii of Def. 5.2, we have that $w_2 = w_0$, but this is impossible because they have different levels.

Therefore the path from w_1 to v must be be empty, and so $w_1 = v$.

For showing condition iv assume first that $\mathit{siblings}_{\leq_\mathrm{i}}(v, v')$. Then either $v = v'$, or else (by unfolding the definition of $\mathit{cover}_{\leq_\mathrm{i}}$) there must exist v_0 such that, at least, $v_0 \to v$, and $v_0 \to v'$. Thus, by condition ii of Def. 5.2, $l(v) \leq l(v_0) + 1$ and $l(v') \leq l(v_0) + 1$ also hold. Assuming further that $\mathit{isroot}_{\to_\mathrm{s}}(v) \wedge \mathit{isroot}_{\to_\mathrm{s}}(v')$, $v_0 \to v$ and $v_0 \to v'$ additionally yield, by Lemma A.4, that $l(v_0) + 1 \leq l(v)$ and $l(v_0) + 1 \leq l(v')$. Hence, $l(v) = l(v') = l(v_0) + 1$ and, by condition (iii) of Def. 5.2, it now follows that $v_1 = v_2$, as required. $\square$