

**SOFTWARE VERIFICATION RESEARCH CENTRE
SCHOOL OF INFORMATION TECHNOLOGY
THE UNIVERSITY OF QUEENSLAND**

**Queensland 4072
Australia**

TECHNICAL REPORT

No. 97-15

**Improving Angel's Parallel Operator:
Gumtree's Approach**

**Andrew Martin, Ray Nickson,
and Mark Utting**

December 1997

Phone: +61 7 3365 1003

Fax: +61 7 3365 1533

<http://svrc.it.uq.edu.au>

Note: Most SVRC technical reports are available via anonymous ftp, from [svrc.it.uq.edu.au](ftp://svrc.it.uq.edu.au) in the directory `/pub/techreports`. Abstracts and compressed postscript files are available via <http://svrc.it.uq.edu.au>

Improving Angel's Parallel Operator: Gumtree's Approach

Andrew Martin,¹ Ray Nickson,²
and Mark Utting³

¹ Software Verification Research Centre
The University of Queensland

² School of Mathematical and Computing Sciences,
Victoria University of Wellington

³ Department of Computer Science,
The University of Waikato

Abstract

We describe some features of the tactic language implemented in the theorem prover Ergo 5. This is a variant of the generic tactic language *Angel*. We have adapted the language by changing the semantics of its parallel composition operator, the operator by which different tactics are applied to different branches in a proof tree. The paper includes a denotational semantics for this operator, and a collection of derived tactics which use it, together with a collection of algebraic laws which they obey.

Keywords

Tactic, Tactical, Denotational Semantics, Algebraic Laws, Interactive theorem proving

1 Introduction

Theorem-proving tools have traditionally used their implementation language as a *tactic language* in which users can write procedures to assist in the discovery of proofs. In the LCF family of tools, this language is ML, with certain tactic combinators (tacticals) pre-defined. Various versions of *Ergo* (Nickson, Traynor, and Utting, 1996) have used Qu-Prolog (Robinson and Hagen, 1997) in the same way.

In the latest version of Ergo, (Ergo 5), we have decided to adopt a more generic approach. The tool is implemented in Qu-Prolog, but all user interactions are via a high-level tactic language which has been described independently (as *Angel*) by Martin, Gardiner, and Woodcock (1996). This is a simple language which is described formally but without any operational bias. With growing interest in porting of theories and results between proof tools, a generic means of describing proof procedures seems most valuable.

Our variant of *Angel* is called *Gumtree*. A single language is used for tactic programming, user interactions, and for the display of proof records. This report

describes the chief changes made to Angel in its implementation as Gumtree. A companion paper (Martin, Nickson, and Utting, 1997) concentrates to a greater extent on the implementation of Gumtree.

The design of Gumtree has been undertaken in keeping with the style of *Angel*—without immediate reference to its implementation. Moreover, care has been taken to preserve the abstract semantics of the *Angel* constructs. The paper of Martin et al. (1996) describes both a denotational semantics for the language, and a large collection of axioms (presented as tactic equivalences) which can be used in reasoning about tactics and transforming them.

We have adapted Angel in two regards. The treatment of parallelism—i.e. the way tactics are to be applied when a proof bifurcates—seemed inappropriate for our purposes, and so is changed quite markedly.⁴ Our new definition of parallel satisfies simple and powerful laws, including associativity, and also allows us to incorporate arity checking into the tactic language. This arity checking can often be performed at compile-time, which helps to detect errors in tactics.

The implementation also allows some non-declarative constructs, allowing proof nodes to be deferred for later consideration. We describe these briefly, but they play no interesting part in the abstract model.

In keeping with the Angel style of implementation-independence, in this paper, we will avoid operational descriptions of tactics as much as possible. Section 2 gives a description of Angel, and Section 3 describes the design decisions made in adapting this to Gumtree. Section 4 provides an abstract denotational model for the new constructs, and Section 5 uses these constructs to define some useful derived tactics, and discuss some of their algebraic properties. The final section concludes by discussing this paper’s contribution and its context with regard to related and possible future work.

2 Angel

Since the Edinburgh LCF project first described *tactics* as programs for directing proof tools (Gordon, Milner, and Wadsworth, 1979), the notion has become very widespread. Many systems implement some form of tactic language for directing proofs. Sometimes, following the LCF style, tactics construct possible proofs which are later validated. In other systems the tactics form an extension of the set of primitive inference rules. The soundness of each tactic application is assured by ensuring that its only interaction with the proof under construction is by application of the primitive inference rules.

Angel (Martin et al., 1996) is a generic tactic language. Initially it was intended to support proofs in the second of these styles, that is, providing a framework for the composition of primitive inference rules in the construction of backwards (goal-directed) proofs, but it turns out to be more general than this. Term rewriting, for

⁴ Observe that our use of the word ‘parallel’ refers to application of different tactics in separate branches of a proof tree, and not (necessarily) to the parallel computation of the corresponding rule applications.

example, which in Cambridge LCF (Paulson, 1987) is directed by a separate set of operators from those which describe tactics, could also be described using Angel.

Because Angel is a small language, its semantics are quite clean and easy to reason about. Nevertheless it is able to describe a large class of useful algorithms. The language is named Angel because it can be described by making tactics angelically nondeterministic (Martin et al., 1996). That is, a tactic is a relation between goals, rather than a single function. Thus we may arrange that a compound tactic will fail only if there is no possible path from input to output: an implementation must avoid dead-end paths, which is typically achieved by a backtracking search. The angelic style removes the need for the tactic programmer to program backtracking explicitly.

Primitive/basic inference rules (or rewrites) are referenced using atomic tactics of the form ‘**rule** R ’. Other atomic tactics are **skip** (which always succeeds, leaving its goal unchanged) and **fail**, which always fails. Tactics may be sequentially composed (‘;’), or placed in alternation (‘|’). For efficiency reasons, it may be appropriate to limit the scope of the angelic choice, so ‘! t ’ restricts the nondeterminism to the scope of t . A number of laws illustrate the interactions of the basic Angel tactic constructs. To avoid using too many parentheses, we adopt a precedence convention: ‘**rule**’ binds tightest of all, followed by ‘!’, then ‘;’, then ‘|’. Later, ‘||’ is introduced; this binds weakest of all.

The semantics of the language admits the following laws:

$$\begin{array}{l|l}
 \mathbf{skip} ; t = t & t_1 | (t_2 | t_3) = (t_1 | t_2) | t_3 \\
 t ; \mathbf{skip} = t & t_1 ; (t_2 ; t_3) = (t_1 ; t_2) ; t_3 \\
 t | \mathbf{fail} = t & (t_1 | t_2) ; t_3 = (t_1 ; t_3) | (t_2 ; t_3) \\
 \mathbf{fail} | t = t & !t_1 ; (t_2 | t_3) = !t_1 ; t_2 | !t_1 ; t_3 \\
 t ; \mathbf{fail} = \mathbf{fail} & !t_1 ; !t_2 = !(t_1 ; t_2) \\
 \mathbf{fail} ; t = \mathbf{fail} & !(t_1 ; t_2) = !(t_1 ; !t_2) \\
 !\mathbf{skip} = \mathbf{skip} & !(t_1 | t_2) = !(t_1 | t_2) \\
 !\mathbf{fail} = \mathbf{fail} & !(t_1 | t_2) = !(t_1 | !t_2) \\
 !\mathbf{rule} \ r = \mathbf{rule} \ r & !(t_1 | t_1 ; t_2) = !t_1 \\
 & !(t_1 | t_2 | t_1) = !(t_1 | t_2) .
 \end{array}$$

The distributive law is perhaps the most indicative of the backtracking semantics: we have $(t_1 | t_2) ; t_3 = (t_1 ; t_3) | (t_2 ; t_3)$. This is not the case in (say) LCF or HOL, although Isabelle has a construct of this form (**APPEND** is akin to Angel’s ‘|’).

From these laws (and others), together with a suitable account of fixpoint theory for the recursive tactics—see the Angel paper (Martin et al., 1996) for details and a proof of completeness—we may prove desirable laws about derived tactics. For example, if *exhaust* t is the tactic which applies t as many times as possible (**REPEAT** t in LCF), we may prove that

$$!(\mathit{exhaust} \ t) ; !(\mathit{exhaust} \ t) = !(\mathit{exhaust} \ t) .$$

2.1 Structural Combinators

In addition to these basic constructs, Angel also provides for *structural combinators* which permit tactics to be applied to sub-expressions of the goal. The set of structural combinators provided by an Angel implementation will generally depend on the nature of the inference system that underlies Angel.

In a theorem prover based on term rewriting, such as Ergo 4 (Nickson et al., 1996), the structural combinators will typically reflect the object-level structure of terms. Such a tool will have object-level terms for, say, conjunction, implication, and disjunction. In such systems it is generally possible to decompose goals, applying tactics independently to the components. Each such object-level connective \oplus will have an associated structural combinator \boxplus . If tactics t_1 and t_2 are respectively applicable to goals α and β , the combined tactic $t_1 \boxplus t_2$ will be applicable to the compound goal $\alpha \oplus \beta$. For example, if we have that t_1 applied to α gives ρ , and that t_2 applied to β gives η , then $t_1 \boxplus t_2$ applied to $\alpha \wedge \beta$ will be $\rho \wedge \eta$. In this style, there is no distinction between forwards and backwards proof, or indeed between proof and other rewriting activities.

Structural combinators are defined in such a way as to make them obey useful rules. The most striking is the *abides* law:

$$(t_1 \boxplus t_2) ; (t_3 \boxplus t_4) = (t_1 ; t_3) \boxplus (t_2 ; t_4) .$$

This law is subject to certain (rather strong) side-conditions. It is sufficient for t_2 to be deterministic ($t_2 = !t_2$), or for both t_1 and t_3 to be deterministic ($t_1 = !t_1$ and $t_3 = !t_3$).

2.2 Parallel Combinator for Tree-Structured Proofs

In a theorem prover supporting backwards inference in the style of natural deduction, such as Ergo 5, (Utting, 1996), the input of a typical fundamental inference step is a goal to be proved (the *conclusion* of the inference rule), and the result is a set or sequence of *premisses*. The previous account of Angel (Martin et al., 1996) describes a *parallel* tactic combinator for dealing with proof trees which branch in this way. A slightly different approach has been taken in Ergo 5; this is described in Section 3, below.

Ergo 5 implements and extends Angel (as the *Gumtree* interface—see below). This is accomplished in such a way as to preserve the truth of most of the tactic transformation laws mentioned previously. We expect that these will permit tactic optimisation, as well as enhancing users' understanding of the semantics of the tactics they write.

3 Parallelism—some choices

In this section we explain the design choices which brought us to the meaning for parallel composition which we have implemented in Gumtree.

3.1 Angel’s Parallel

The previous account of parallel (Martin et al., 1996) treats the bifurcation as an instance of the general theory of structural combinators.⁵ When the application of a rule such as and-introduction ($\frac{A \quad B}{A \wedge B}$) causes a fork in the proof tree, the resulting pair of subgoals is seen as a single goal, consisting of two subgoals in parallel composition. We write this parallel composition of goals using $\&$, and thus the goal $A \wedge B$ is transformed into the goal $A \& B$.

In order to avoid difficulty in associating goals with tactics in the presence of several such parallel compositions, this operator and the corresponding tactic structural combinator ‘ \boxtimes ’ are both (non-associative) binary operators.

A further complication is that an inference rule may produce *no* subgoal (when the proof of a particular branch is complete, for example), and the tactic definition must take account of this. Therefore, defining inference rules *nullidL* and *nullidR* which transform goals of the form $g \& ()$ and $() \& g$ respectively into g , we define the general-purpose tactic parallel combinator as

$$t_1 \parallel t_2 = t_1 \boxtimes t_2 ; !(nullidL \mid nullidR \mid \mathbf{skip}) .$$

This combinator turns out to be fairly useful, but unpredictable. The tactic-writer needs to know the structure of the goal at all points which makes writing general-purpose tactics more difficult. If basic rules return more than two antecedents, then either a ternary version of ‘ \parallel ’ is needed, or a convention that the resulting instances of $\&$ associate to the right (or left).

\boxtimes of course obeys the generic laws for structural combinators. These are useful in some situations, but as with the *abides* law mentioned above, do not interact well with nondeterminism. \parallel is more restrictive; its laws have additional side-conditions because of the more complex definition given above.

For these reasons, we have decided to develop a more general parallel operator that supports modular tactic development, rather than adopting the Angel parallel operator.

3.2 Making parallel associative

To generalise Angel’s parallel, we chose to make it associative. Another way of viewing this is that instead of viewing the collection of open goals as a binary tree, we view it as a list, and likewise view as a list the tactics being applied. This approach removes the need to treat the null goal specially: when an inference rule returns no subgoals, this fact is not made explicit in the list.

Example Suppose tactic t takes some goal P and transforms it into subgoals P_1 , P_2 and P_3 . Using Angel’s parallel, the new goal must be either $P_1 \& (P_2 \& P_3)$ or $(P_1 \& P_2) \& P_3$ and *only one* of the tactics $t_1 \parallel (t_2 \parallel t_3)$ or $(t_1 \parallel t_2) \parallel t_3$ will be applicable. In order to know which one, the user must know something of the structure of t . In the Guntree approach, it would suffice to write $t_1 \parallel t_2 \parallel t_3$.

⁵ We are interested only in *finitely*-branching proof trees.

3.3 Tactic Arity

The form of parallel we describe is very general. Two difficulties can be foreseen: (a) problems of efficiency, and (b) added complication leading to errors in tactic-writing. To mitigate these problems, we chose to introduce machinery for tactic arity calculation and checking.

We write $t^{m \rightarrow n}$ to stand for an arbitrary (deterministic) tactic with arity $m \rightarrow n$. An inference rule with n premisses has arity $1 \rightarrow n$. The sequential composition $t_1^{m \rightarrow n}; t_2^{p \rightarrow q}$ has arity $m \rightarrow q$, if $n = p$. The composition will fail otherwise.

Choice complicates the arity calculation since the different branches of the tactic may have different arities. Therefore, in general, we must allow a tactic to take a sequence of arities. The alternation $t_1^{m \rightarrow n} | t_2^{p \rightarrow q}$ can take arity $m \rightarrow n$ or $p \rightarrow q$. We will write this as $(t_1 | t_2)^{m \rightarrow n | p \rightarrow q}$.

More generally, then, if M and N are sequences of arities, the sequential composition $t_1^M; t_2^N$ may succeed only if $\pi_2 M \cap \pi_1 N \neq \emptyset$ (where π_i is a selection operator lifted over sequences of arities, so $\pi_2(1 \rightarrow 3 | 2 \rightarrow 4) = \{3, 4\}$). In this case, the arity of $t_1; t_2$ is given by the list comprehension $[m_1 \mapsto n_2 | m_1 \rightarrow m_2 \in M, n_1 \rightarrow n_2 \in N, m_2 = n_1]$.

$!t^M$ has arity/arities M , because at compile-time it is generally not possible to determine which branch of a choice will succeed at runtime, so all possibilities must be considered. When tactics are placed in parallel composition, their arities will be added—in every possible way (the behaviour is a *product*; see Section 4.2). Some examples will clarify this.

Examples

1. We can observe that

$$t_1^{1 \rightarrow 2}; t_2^{3 \rightarrow 1}$$

must fail. This can be detected at compile-time.

2. Moreover,

$$(t_1^{1 \rightarrow 2} | t_2^{1 \rightarrow 3}); t_3^{3 \rightarrow 1}$$

may be compiled directly to $t_2^{1 \rightarrow 3}; t_3^{3 \rightarrow 1}$, omitting the nondeterministic choice. Such analysis can dramatically reduce the possible nondeterminism, and thereby increase the over-all efficiency.

3. Given $t_1^{m \rightarrow n}$, $t_2^{p \rightarrow q}$, and $t_3^{r \rightarrow s}$, we obtain

$$(t_1 \parallel (t_2 | t_3))^{m+p \rightarrow n+q | m+r \rightarrow n+s} .$$

3.4 How to associate tactics with goals

There remains a problem of how to associate tactics with goals. For example, if the tactic $t_1 \parallel t_2$ is applied to three goals, which of t_1 and t_2 should get two goals, and which should get one? One approach is to force each tactic in a parallel composition to have arity $1 \rightarrow n$. In this way, each tactic in the parallel composition is applied to the corresponding subgoal in the goal list. This appears to be overly-restrictive,

however; we will generally wish to write tactics with internal structure. This scheme would mean that if we defined t_1 by $t_1 = t_2 \parallel t_3$, we could not later apply $t_1 \parallel t_4$, because t_1 has arity $2 \rightarrow n$.

A variation on this scheme is to require that the tactics in the parallel composition should have fixed arities, but even this seems overly restrictive: it would prohibit use of $\text{try } t_1 \parallel \text{try } t_4$, say, where t_1 is defined as before, and try (which is discussed in Section 5.3) is the tactical which applies its argument if possible, and behaves like **skip** otherwise. This is because **skip** has *any* suitable arity $n \rightarrow n$.

We avoid the problems with these schemes by allowing the match of subgoals to tactics to be nondeterministic, as at other choice points in the language. Any association of goals with tactics (provided that the order of both lists is preserved) is permitted. The allocation chosen will be one which gets the proof the furthest. Of course, in many cases, the tactics in the parallel composition will be deterministic (or, at least, of fixed arity), and so much of the nondeterminism can be resolved at compile time.

Although the allocation of goals to tactics is nondeterministic, this semantics does not introduce additional nondeterminism: if t_1 and t_2 are deterministic, then $t_1 \parallel t_2$ will also be deterministic. The nondeterminism of the composition is constrained to be no worse than that of the individual tactic components. The behaviour of *cut* tactics within a parallel composition is, however, surprising at first sight; see Section 5.3.

3.5 Arity Coercion

In order to annotate parallel tactics, or to reduce nondeterminism, we define special tactics **zero** and **one**, which succeed respectively when applied to a goal consisting of zero subgoals and one subgoal.

At first it seems that **one** has the behaviour expected of **skip**.⁶ However, **skip** must be an identity of sequential composition, and **one** is too restrictive for this. (For example, if r is a primitive inference rule which produces two subgoals (thus **rule** r has arity $1 \rightarrow 2$) then **rule** r ; **one** = **fail**.) Thus, **skip** must correspond to an arbitrary number of instances of **one**, in parallel composition (that is, *every*(**one**) : see Section 5.3). When a parallel composition is formed, if it is possible that there will be fewer tactics than matching goals, we may finish the parallel composition with an instance of **skip**, so that any left-over goals are passed through unchanged.

In Section 5.1, we define *two* as **one** \parallel **one**, and, more generally *fix* n , a tactic which succeeds iff the current goal has exactly n subgoals. These fixed arity tactics can be used as coercions to reduce nondeterminism, and increase scope for partial evaluation at compile-time.

We considered a semantics where *implicit* arity coercion takes place. If $t^{m \rightarrow n}$ is applied in a context where a tactic of arity $p \rightarrow q$ is demanded, and if $p \geq m$ and $p - m = q - n$, then t is coerced to $t \parallel \text{skip}^{(p-m) \rightarrow (p-m)}$. One can similarly imagine

⁶ And maybe **zero** and **fail** are identical.

a semantics in which having more tactics in parallel than subgoals was not a reason for failure, but rather the unused tactics were discarded.

Clearly no arity-checking would be possible/useful in either of these circumstances. We decided not to adopt these approaches, as both would tend to lead to sloppy programming. Instead, a tactic must correspond closely to the proof being constructed.

3.6 Deferring subgoals

This treatment for parallel subgoals is very appropriate for writing tactics off-line, constructing proof procedures, etc. However, in interactive use, it will often be cumbersome to have to set up a parallel composition when one in fact wishes to consider the subgoals one at a time. To solve this problem, Ergo 5 has two levels of ‘current state’. The current subgoal context is as described here; there is also a secondary set of ‘deferred goals’, and a tactic primitive **defer** which removes a subgoal from the current context into that set. Correspondingly, the tactic primitive **select** moves goals from that set into the current subgoal state. Thus the user may at any time defer all the current context, and select from the set of deferred goals another (sequence of) member(s) for consideration.

This two-level state is not modelled in the following section. From the perspective of tactic semantics, the **defer** action is a ‘magic’ rule (arity $1 \rightarrow 0$) which proves any subgoal; the effect on the proof tree is different, of course, from a goal which really completes a branch. The tactic **select** is by contrast a catastrophic rule, introducing a goal where there was none before (arity $0 \rightarrow 1$). Use of **defer** and **select** in tactic scripts is possible, but the result is akin to ‘tactic programming with *gotos*’, and is discouraged.

4 The Semantics of Gumtree’s Parallel

4.1 Angel’s Denotational Semantics

The basic tactic combinators in Section 2 can be defined using a simple theory of lists (sequences).⁷ We take as given a set of goals G . A tactic can be regarded as a function from G , to lists of goals. These lists denote *alternative* goals. (The analysis of parallel goals will also give rise to lists of (sub-)goals, but these are modelled *within* G —see below.) Thus tactic failure is modelled as a function returning an empty list of alternatives. **skip** constructs a singleton list:

$$\begin{aligned} \mathbf{fail} \ g &= [] \\ \mathbf{skip} \ g &= [g] . \end{aligned}$$

⁷ A theory of infinite lists and fixpoints is needed to give semantics for recursively defined tactics. The details are found elsewhere (Martin et al., 1996). A subsequent paper will show how these definitions can be generalised to use an arbitrary monad.

Sequential composition entails applying the second tactic to all the outcomes of the first. Alternation is simply concatenation, and the cut operator takes the front of the list:

$$\begin{aligned}(t_1 ; t_2) g &= (++) \cdot t_2 * \cdot t_1) g \\(t_1 | t_2) g &= t_1 g ++ t_2 g \\(!t) g &= \text{if}(t g = []) \text{ then } [] \text{ else } [\text{head}(t g)] \text{ fi} .\end{aligned}$$

Where $++$ is the list concatenation operator, $++/$ is distributed concatenation, and $f * xs$ applies function f to each element of the list xs . Functional composition is denoted by the centred dot (Bird, 1989).

The combinator \boxplus is defined as the composition of a function *break* which takes terms of the form $g_1 \oplus g_2$ and returns the list $[g_1, g_2]$, a function *cross* which applies tactics to goals, and a function to recombine the subgoal lists into goals (\oplus -separated sequences of subgoals). *cross* forms a cartesian product of the resulting lists so that all possible combinations of alternatives are available:⁸

$$(t_1 \boxplus t_2) g = (++) \cdot \text{combine} * \cdot \text{cross} [t_1, t_2] \cdot \text{break}) g$$

$$\begin{aligned}\text{break} (g_1 \oplus g_2) &= [g_1, g_2] \\ \text{combine} [g_1, g_2] &= g_1 \oplus g_2 \\ \text{cross}[t_1, t_2][g_1, g_2] &= (t_1 g_1) \times (t_2 g_2) .\end{aligned}$$

For tree-structured proofs, we suppose that G has internal structure: there is a distinguished empty goal $()$, there are atomic goals g , and goals of the form $g_1 \& g_2$ (where g_1 and g_2 are any members of g). A corresponding general tactic \boxtimes is constructed as explained in Section 3.1.

4.2 Extension for Gumtree's Parallel

The definitions above can be extended to cover Gumtree's parallel (and any other structural combinator with a similar list behaviour) simply by generalising the definition of *break* to return a list of alternative breaks for the goal,⁹ and arranging for *cross* to be applied to each break-down of the goal:

$$(t_1 \parallel \dots \parallel t_n) g = (\text{combine} * \cdot ++/ \cdot (\text{cross} [t_1, \dots, t_n]) * \cdot \text{break}) g .$$

Assume that we have a distinguished null goal $()$, and that the operator ' $\&$ ' combines a subgoal and a goal to give a goal (thus the structure of goals is exactly isomorphic to lists). Ignoring instances of $()$, the required behaviour of *break* is:

$$\begin{aligned}\text{break} (g_1 \& g_2) &= [[[g_1], [g_2]], [[g_1], [g_2]]] \\ \text{break} (g_1 \& g_2 \& g_3) &= [[[g_1, g_2, g_3]], [[g_1], [g_2], [g_3]], [[g_1], [g_2, g_3]], [[g_1, g_2], [g_3]]] .\end{aligned}$$

⁸ The full definition of *cross* is in the Angel paper (Martin et al., 1996).

⁹ This version of *break* is essentially Bird's *parts* function, which computes list partitions.

In fact, *break* must also allow an arbitrary number of ()s at any point. One specification is as follows:

$$\begin{aligned} \mathit{break} \ g &= \mathit{ordered} \{ \mathit{gs} \mid \mathit{combine} \ \mathit{gs} = g \} \\ a = \mathit{ordered} \ s &\iff s = \mathit{ran} \ a \wedge \\ &(\forall i, j : \mathit{dom} \ a \bullet i < j \Rightarrow a.i \prec a.j) . \end{aligned}$$

The ordering relation \prec is defined on lists of lists:

$$\begin{aligned} [] \prec (b : bs) &\iff \mathit{true} \\ (a : as) \prec [] &\iff \mathit{false} \\ (a : as) \prec (b : bs) &\iff \mathit{length} \ a < \mathit{length} \ b \vee \\ &(a = b \wedge as \prec bs) . \end{aligned}$$

Observe that *combine* is simply a fold with ‘&’.

Finally, we must define,

$$\begin{aligned} \mathbf{zero} \ g &= \mathbf{if} (g = ()) \ \mathbf{then} \ [] \ \mathbf{else} \ [] \ \mathbf{fi} \\ \mathbf{one} \ g &= \mathbf{if} (g \ \mathit{atomic}) \ \mathbf{then} \ [g] \ \mathbf{else} \ [] \ \mathbf{fi} . \end{aligned}$$

4.3 Properties of this definition

Chiefly because the list cartesian product used in the definition of *cross* is associative, we may prove the following law, which demonstrates that our parallel operator has the properties we desired.

Law 1. $(t_1 \parallel t_2) \parallel t_3 = t_1 \parallel t_2 \parallel t_3 = t_1 \parallel (t_2 \parallel t_3)$

zero is an identity element for parallel, as intended.

Law 2. $t \parallel \mathbf{zero} = t = \mathbf{zero} \parallel t$

5 Derived Tactics

In this section, we describe some derived tactics using the basic tactics and tacticals defined above. Laws are given which characterise the behaviour of these derived tactics, with the aim both of aiding understanding, and permitting reasoning about these tactics.

In the definition of recursive tactics with parameters, we will often use the style popular in functional programming, giving a base case (e.g. with the parameter as zero), and an inductive case (parameter $n + 1$).

$$\begin{aligned} f(0) &\hat{=} e_1 \\ f(n + 1) &\hat{=} e_2(n) . \end{aligned}$$

(In our implementation of Gumtree, these must actually be written $f(n) \hat{=} \mathbf{if} (n = 0) \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2(n - 1) \ \mathbf{fi}$, but that detail need not concern us here.)

5.1 Fixed-arity tactics

The tactic **skip** is fully general; it will apply to any sequence of goals, and succeed, leaving them unchanged. Sometimes it is useful to have less general versions of **skip** which match specific numbers of subgoals. Above, we have defined **zero** which succeeds when the current list of subgoals is empty, and fails otherwise, and **one** which succeeds (leaving the goal unchanged) when there is exactly one current subgoal. Then we may define

$$two \hat{=} \mathbf{one} \parallel \mathbf{one} ,$$

and more generally

$$\begin{aligned} fix\ 0 &\hat{=} \mathbf{zero} \\ fix(n + 1) &\hat{=} fix\ n \parallel \mathbf{one} . \end{aligned}$$

Then we may prove some laws:

Law 3. $fix\ n \parallel fix\ m = fix(n + m)$

Law 4. $(fix\ n \mid fix\ m) \parallel fix\ p = fix(n + p) \mid fix(m + p)$

Law 5. $fix\ n ; fix\ m = \begin{cases} \mathbf{fail} , & (n \neq m) \\ fix\ n , & (n = m) \end{cases}$

Law 6. $rule\ r = \mathbf{one} ; rule\ r$

Law 7. $\mathbf{one} ; t_1 \parallel \mathbf{one} ; t_2 = two ; (\mathbf{one} ; t_1 \parallel \mathbf{one} ; t_2)$

Law 8. $fix\ n ; t = \mathbf{fail}$ for ‘almost all’ t

In law 8, ‘almost all t ’ refers to those tactics t which terminate and are unable to take arity $n \rightarrow m$ (for some m). Thus laws 5 and 8 are special cases of the more general law

Law 9. $t_1 ; t_2 = \mathbf{fail}$ provided $\pi_2 t_1 \cap \pi_1 t_2 = \emptyset$

(where $\pi_1 t$ determines the set of possible source arities of tactics t , and $\pi_2 t$ determines its set of possible target arities). This illustrates a general feature of sequential composition discussed in Section 3.3: in general, it reduces the scope for nondeterminism, by intersecting the respective target and source sets of its component tactics.

In the laws of *Angel*, ‘!’ is used to enforce determinism where necessary. For example, the left-distribution law for ‘;’ over ‘|’ holds only in the presence of a ‘!’. Likewise, fixed-arity tactics can be used as coercions to do much the same for tactics of varying arity. Thus whereas it is not generally true that $!(t_1 \parallel t_2) = !t_1 \parallel !t_2$, we do have

Law 10. $!(\mathbf{one} ; t_1 \parallel \mathbf{one} ; t_2) = \mathbf{one} ; !t_1 \parallel \mathbf{one} ; !t_2$

and, more generally,

Law 11. $!(fix\ n_1 ; t_1 \parallel fix\ n_2 ; t_2) = fix\ n_1 ; !t_1 \parallel fix\ n_2 ; !t_2$

5.2 Constraining Arity

The tactics of the previous subsection introduce precise constraints on the number of subgoals in parallel composition. Sometimes, a more relaxed arity constraint will be appropriate:

$$\begin{aligned} \mathit{atmost}\ 0 &\hat{=} \mathbf{zero} \\ \mathit{atmost}(n + 1) &\hat{=} \mathit{atmost}\ n \mid \mathit{fix}(n + 1) \end{aligned}$$

$$\begin{aligned} \mathit{atleast}\ 0 &\hat{=} \mathbf{skip} \\ \mathit{atleast}(n + 1) &\hat{=} \mathit{atleast}\ n \parallel \mathbf{one} . \end{aligned}$$

A more succinct definition of *atleast* is given by the law:

Law 12. $\mathit{atleast}\ n = \mathbf{skip} \parallel \mathit{fix}\ n$

and a simple inductive proof verifies the equivalence of these definitions:

Proof. Base case:

$$\begin{aligned} \mathit{atleast}\ 0 & \\ = \mathbf{skip} & \quad [\text{definition of } \mathit{atleast}] \\ = \mathbf{skip} \parallel \mathbf{zero} & \quad [\text{law 2}] \\ = \mathbf{skip} \parallel \mathit{fix}\ 0 & \quad [\text{definition of } \mathit{fix}] \end{aligned}$$

Step case:

$$\begin{aligned} \mathit{atleast}(n + 1) & \\ = \mathit{atleast}\ n \parallel \mathbf{one} & \quad [\text{definition of } \mathit{atleast}] \\ = (\mathbf{skip} \parallel \mathit{fix}\ n) \parallel \mathbf{one} & \quad [\text{inductive hypothesis}] \\ = \mathbf{skip} \parallel (\mathit{fix}\ n \parallel \mathbf{one}) & \quad [\text{law 1}] \\ = \mathbf{skip} \parallel \mathit{fix}(n + 1) & \quad [\text{definition of } \mathit{fix}] \end{aligned}$$

5.3 Repeated application

Certain tacticals arise in most tactic-based systems. *try* t attempts to apply t , but succeeds whether t is successful or not. *exhaust* t applies t repeatedly, as many times as possible. *repeat* $n\ t$ applies t exactly n times, failing if this is not possible:¹⁰

$$\begin{aligned} \mathit{try}\ t &\hat{=} t \mid \mathbf{skip} \\ \mathit{exhaust}\ t &\hat{=} t ; \mathit{exhaust}\ t \mid \mathbf{skip} \\ \mathit{repeat}\ 0\ t &\hat{=} \mathbf{skip} \\ \mathit{repeat}\ (n + 1)\ t &\hat{=} t ; \mathit{repeat}\ n\ t . \end{aligned}$$

Variations on these, such as exhaustive application subject to a minimum or maximum number of applications, are also possible.

¹⁰ In the LCF community, *exhaust* is known as REPEAT (Gordon et al., 1979).

We may similarly define tactics which apply their arguments to the current parallel goals. *every* t will apply t to every available subgoal; *copy* n t will apply n instances of t in parallel:

$$\begin{aligned} \textit{every } t &\hat{=} (t \parallel \textit{every } t) \mid \mathbf{zero} \\ \textit{copy } 0 \ t &\hat{=} \mathbf{zero} \\ \textit{copy } (n + 1) \ t &\hat{=} t \parallel \textit{copy } n \ t . \end{aligned}$$

For $t^{1 \rightarrow p}$, we may prove by induction that

Law 13. $\textit{fix } n ; \textit{every } t = \textit{copy } n \ t$

We may also prove

Law 14. $\textit{copy } n \ t \parallel \textit{copy } m \ t = \textit{copy } (n + m) \ t$

and use these two properties together to prove

Law 15. $\textit{fix } n ; \textit{every } t \parallel \textit{fix } m ; \textit{every } t = \textit{fix } (n + m) ; \textit{every } t$

This law is reminiscent of one proved by Martin et al. (1996),

$$!(\textit{exhaust } t) ; !(\textit{exhaust } t) = !(\textit{exhaust } t) .$$

Because the behaviour of **skip** is to succeed on any number of subgoals, leaving each of those subgoals unchanged, we have that

Law 16. $\mathbf{skip} = \textit{every}(\mathbf{one})$

At first sight, this result is surprising. *every* is written using a large nondeterministic choice, yet **skip** is deterministic. This law is satisfied, however, because in any given context, *exactly one* of the nondeterministic branches of *every*(**one**) will be applicable.

Moreover, it seems odd that, for an arbitrary t (which may be of variable arity) both $(t \parallel \mathbf{skip})$ and $(t \parallel !\mathbf{skip})$ should have the same meaning. We generally think of *cut* as denoting ‘make the nondeterministic choice only once’; if the choice turns out to have been the wrong one (that is, if the tactic backtracks), then the tactic will fail. On this reasoning, the **!skip** should choose only once how many branches (subgoals) to match—corresponding to one particular arity for t —and fail if a different arity for t is in fact needed.

However, though our semantics for parallel composition does not increase the nondeterminism present in the tactics, it does *promote* it. In this way, for *each* decomposition of the goal produced by *break*, **!skip** is able to choose a different number of subgoals to match (where necessary), and so behaves exactly like **skip**.

every t is a very exacting tactic: for it to succeed, t must succeed on every subgoal (perhaps taking several subgoals at a time).¹¹ It will sometimes be appropriate to

¹¹ We observe in passing that *every*(**zero**) diverges; it is the tactic which *Angel* calls **abort**.

be able to apply t to every subgoal, succeeding regardless of whether or not t is successful. This is accomplished for $t^{1 \rightarrow m}$ by *tryevery* t :

$$\text{tryevery } t \hat{=} \text{every}(t \mid \mathbf{one}) .$$

For t with an arity $n \rightarrow m$, the **one** might be replaced by *fix* n (to consider subgoals in distinct groups of n) or *atmost* n (to consider all possible groupings). The former will succeed only if the total number of subgoals is a multiple of n .

Sometimes it will be appropriate for a tactic like *tryevery* t to fail if *all* of the applications of t fail, but to succeed otherwise (in the context of a loop, for example). This is achieved by *some* t :

$$\begin{aligned} \text{any } t &\hat{=} \mathbf{skip} \parallel t \parallel \mathbf{skip} \\ \text{some } t &\hat{=} \text{every}(\text{any } t) . \end{aligned}$$

The operation of *some* t may not be immediately clear. Expanding its definition gives

$$\mathbf{skip} \parallel t \parallel \mathbf{skip} \parallel \dots \parallel \mathbf{skip} \parallel t \parallel \mathbf{skip} \parallel \mathbf{zero} ,$$

where the number of instances of $(\mathbf{skip} \parallel t \parallel \mathbf{skip})$ is bounded above by the number of subgoals to which t is applicable. This expanded tactic will be in alternation with others in which t is applied fewer times.

The companion paper to this one (Martin et al., 1997) defines and uses a tactic for breadth-first application of some tactic t —this is simply the exhaustive application of *some* t :

$$\text{bfs } t \hat{=} \text{exhaust}(\text{some } t) .$$

We may also define a depth-first application of t , and expect that for a deterministic t the outcome should be the same as that for the breadth-first application. The relative efficiencies will depend on the nature of t :

$$\text{dfs } t \hat{=} \text{every}(t ; \text{dfs } t \mid \mathbf{one}) .$$

A more useful tactic may be a similar one which additionally restricts to some bound the depth to which the search should be carried.

6 Example

As a brief example, we describe the tactics used in a proof using a Gentzen-style system for classical propositional logic. The theorem to be proved is

$$A \vee (B \iff C) \Rightarrow ((A \vee B) \iff (A \vee C)) .$$

Clearly, this is not a technically challenging deduction, however we use it merely to illustrate the application of our tactics without having to explain an otherwise unfamiliar object logic.

We begin the proof by using rules of \Rightarrow -introduction, and \Leftrightarrow -introduction. The resulting consequent is a conjunction, so we use \wedge -introduction, followed by \Rightarrow -introduction in each subgoal. The tactic is

```

rule(implies_intro);
rule(iff_intro);
rule(and_intro);
(rule(implies_intro) || rule(implies_intro)) .

```

The resulting subgoals have a lot of structure in their antecedents. We define a tactic which will apply any of the antecedent introduction rules (*or_left*, etc.),

```

lefts  $\hat{=}$  rule(or_left)
          | rule(and_left)
          | rule(implies_left)
          | rule(iff_left) ,

```

and apply this exhaustively using *bfs*:

```

!(bfs(lefts)) .

```

The result is twenty subgoals, mostly of the form

$$A \vdash B \vee (C \vee (A \vee C)) .$$

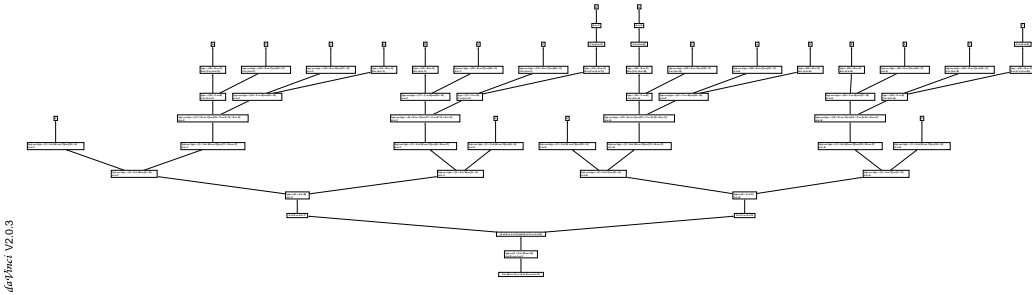
To solve these, it suffices to try each of the possible \vee -introduction rules, followed by the rule of assumption:

```

every(exhaust(rule(or_intro_L) | rule(or_intro_R)) ; rule(assump)) .

```

Application of this tactic completes the proof. A tree representation of the proof, produced automatically by Ergo is shown below. Each arc represents the application of one primitive inference rule.



7 Conclusions

In this paper, we have described how we have adapted the generic tactic language *Angel* to be the language for all aspects of the interface to the theorem-prover Ergo 5.

In doing so, we have made a significant enhancement to the semantics of Angel, by allowing the definition of associative structural combinators. Our application of these is to lift tactics over the parallel branches of a proof tree. This associative parallel tactical not only satisfies simple laws but also makes it possible to perform arity checking on tactics. To our knowledge, this is the first time arity checking has been applied to tactics in an interactive theorem prover.

We have shown that using this combinator, we can define many useful common control structures. By retaining an independent semantics for these combinators, we promote the implementation of portable proof procedures, as well as making the meaning clear and providing scope for efficiency improvements via correctness-preserving transformations.

7.1 Related Work

The tactic language of Isabelle (Paulson, 1994) is clearly close to Angel (save in concrete syntax). Isabelle handles parallel subgoals by subgoal numbering, so

SELECT_GOAL *tac i*

applies *tac* to subgoal *i*. We could achieve something similar by

selectGoal tac i $\hat{=}$ *fix*(*i* - 1) || *tac* || **skip** .

Our derived tactics *every*, *tryevery*, and *some* are similar in effect to Isabelle’s **ALLGOALS**, **TRYALL**, and **SOMEGOAL**.

Conversely, the Isabelle subgoal selection mechanisms could be used to define an operator like our parallel, though the Isabelle community has not yet done this. Thus, the two approaches appear to be of similar utility. The simple algebra of the parallel combinator makes it a very manageable construct to use in designing tactics.

7.2 Future Work

As already mentioned, work is in progress to present an account of the Angel semantics using an arbitrary monad instead of the angelic nondeterministic one chosen here. If, for example, sets are used instead of lists, many more distributive properties hold, but the operationally desirable properties associated with ordering the outcomes are lost.

We also hope to undertake some sizeable case studies using Gumtree. A promising approach appears to be attempting to ‘port’ some Isabelle proofs to Ergo 5.

Window inference is the proof paradigm underlying Ergo 4. We are deriving window *opening rules* in Ergo 5, and using Gumtree to define tacticals which employ these—in the style of structural combinators—so that we can continue to support this style of proof.

Acknowledgements

Most of the work described here was undertaken whilst all of the authors were on projects at the Software Verification Research Centre.

References

- Bird, R. S. (1989). Lectures on constructive functional programming, in M. Broy (ed.), *Constructive Methods in Computing Science*, NATO ASI Series F, Springer-Verlag, pp. 151–216. Also available as Technical Monograph PRG-69, Oxford University Computing Laboratory, 1988.
- Gordon, M. J. C., Milner, R., and Wadsworth, C. P. (1979). *Edinburgh LCF: A Mechanised Logic of Computation*, Vol. 78 of *LNCS*, Springer-Verlag.
- Martin, A., Nickson, R., and Utting, M. (1997). A tactic language for Ergo, in L. Groves and S. Reeves (eds), *Formal Methods Pacific '97*, Springer Series in Discrete Mathematics and Theoretical Computer Science, Springer-Verlag, Singapore. Also appears as TR97-16, Software Verification Research Centre, The University of Queensland, QLD 4072, Australia.
- Martin, A. P., Gardiner, P. H. B., and Woodcock, J. C. P. (1996). A tactic calculus, *Formal Aspects of Computing* 8(4): 479–489. An abridged version appears in the printed journal; the full version is available in the electronic supplement to Formal Aspects of Computing, <ftp://ftp.cs.man.ac.uk/pub/fac>.
- Nickson, R., Traynor, O., and Utting, M. (1996). Cogito Ergo Sum: Providing structured theorem prover support for specification formalisms, in K. Ramamohanarao (ed.), *Proceedings of the Nineteenth Australasian Computer Science Conference (ACSC'96)*, Vol. 18(1) of *Australian Computer Science Communications*, pp. 149–158.
- Paulson, L. C. (1987). *Logic and Computation—Interactive Proof with Cambridge LCF*, Cambridge University Press.
- Paulson, L. C. (1994). *Isabelle: a generic theorem prover*, Vol. 828 of *Lecture notes in Computer Science*, Springer Verlag, Berlin; New York. ‘With contributions by Tobias Nipkow’.
- Robinson, P. and Hagen, R. (1997). Qu-Prolog 4.2 reference manual, *Technical Report 97-11*, Software Verification Research Centre, The University of Queensland.
- Utting, M. (1996). An architecture for a unified refinement/proof tool, *Fifth Australasian Refinement Workshop*, Software Verification Research Centre, The University of Queensland.

This document was processed on 21st January 1998 at 11:12 with L^AT_EX and an adapted version of the LNCS style.