Heuristics for Proof Development in Athena

David R. Musser

February 4, 2005

Contents

1 Introduction

Proofs can be developed by a *goal-driven* or *top-down* strategy — starting with the desired conclusion (the *goal*) and reducing it by some deduction to one or more *subgoals*, further reducing each of the subgoals, and so on, until subgoals are generated that are identical to the original premises.

Alternatively, proofs can be developed by a *premise-driven* or *bottom-up* strategy — starting with the premises and deducing from them new premises, combining those with the original premises to deduce still more new premises, and so on, until the goal is deduced.

In actual practice, it is useful to combine these strategies, working both top-down and bottom-up during the same proof. The Athena language and proof-checking system [1, 2] supports both top-down and bottom-up proof development with the deduction constructs provided in the language and with its primitive methods. This document presents both kinds of strategies as "proof templates" that one can apply to goals or premises to suggest a deduction or method one can use to further the proof. Proof development using these strategies isn't an entirely algorithmic process since there is often more than one template that could be used, and while in some cases either path might ultimately lead to a complete proof, in other cases one would reach a dead end or go in circles. These templates should therefore be regarded as heuristics that can aid in proof development and will often lead to success in proofs if applied with the kind of skill and foresight that come from experience.

In these templates, by a "premise" we mean any proposition in Athena's current assumption base, including those that are added to it during the current deduction; and by the "goal" we mean the conclusion (another proposition) one is trying to prove, including (sub)goals produced by deduction steps. For stating goals we use the `conclude` method from the auxiliary methods file `rewriting.ath` (available from http://www.cs.rpi.edu/~{}musser/gsd/athena; see also [3]).

```
(!(conclude ⟨proposition⟩)
   ⟨deduction⟩)
```

Like Athena's built-in operator `BY`, this sets up ⟨*proposition*⟩ as a proposition to be proved by the deduction that follows. The use of `conclude` instead of `BY`, however, helps in developing proofs because one can trace the progress of the proof (by doing `(set! tracing true)`, which causes each argument of a `conclude` call to be printed).

2 Goal-Driven Proof Development

For each syntactic form that a goal proposition can take, we suggest a type of deduction that can reduce it to one or more subgoals.

2.1 Proof by contradiction

`(!(conclude (not P))`
`  (!by-contradiction`
`     (assume P`
`        ...)))`
	The goal in the proof within the `by-contradiction` call is to show that `false` follows from P and any other premises. To use proof by contradiction, the proof goal doesn't have to be in the form `(not P)`; see the next proof template.

`(!(conclude P)`
`  (!by-contradiction`
`     (assume (not P)`
`        ...)))`
	In either of these forms, the inner deduction usually takes the form, for some proposition Q,
	`(dseq`
	`  ...`
	`  (!(conclude Q)`
	`    ...)`
	`  ...`
	`  (!(conclude (not Q))`
	`    ...)`
	`  (!absurd Q (not Q)))`

2.2 Proof of an implication

`(!(conclude (if P Q))`
`  (assume P`
`     (!(conclude Q)`
`       ...)))`
	Except when used for a toplevel goal of the form `(if P Q)`, it's often best to omit the goal part of this template and just write the `(assume ...)` deduction; this works well (in terms of readability) when doing proofs within `equiv`, `cases` or `cd` deductions (see below).

2.3 Proof of an equivalence

`(!(conclude (iff P Q))`
`  (!equiv`
`     (assume P`
`        (!(conclude Q)`
`          ...))`
`     (assume Q`
`        (!(conclude P)`
`          ...))))`
	We prove each implication `(if P Q)` and `(if Q P)` and combine them with `equiv`.

2.4 Proof by case-splitting

(!(conclude P) (!cases (assume Q (!(conclude P) ...) (assume (not Q) (!(conclude P) ...)))	For this to work out, Q must be chosen so that one proof of P can be found when Q is assumed and a different proof of P can be found when (not Q) is assumed. Note that within one or both of the subcases there could be further case splitting by using this proof template again (or the next one, cd) at that level.
(!(conclude P) (!cd (or Q R) (assume Q (!(conclude P) ...) (assume R (!(conclude P) ...)))	This form of case-splitting is the method of choice when one has a disjunction (or Q R) as a premise or can deduce it, and when one proof of P can be found when Q is assumed and a different proof of P can be found when R is assumed. Note that within one or both of the subcases there could be further case splitting by using this proof template again (or by using cases) at that level.

2.5 Proof of a conjunction

(!(conclude (and P Q)) (dseq (!(conclude P) ...) (!(conclude Q) ...) (!both P Q)))	The top level statement of the goal (and P Q) might be omitted if the inner deductions are not too long, since (!both P Q) makes it clear the conjunction is the goal. One can also put the both call at the top with the form (!both (!(conclude P) ...) (!(conclude Q) ...)) which avoids having to repeat P and Q. However, if the proof of Q depends on P, this won't work since P will *not* be in the assumption base during the proof of Q, so one must use the sequential form in that situation.

2.6 Proof of a disjunction

`(!(conclude (or P Q))`
`  (!either`
`    (!(conclude P)`
`     ...)`
`    Q))`
	Alternatively, prove Q. Usually though such a proof using `either` will be a subcase of a larger proof; see the next proof template.

`(!(conclude (or P Q))`
`  (!cases`
`      (assume P`
`        (!either P Q))`
`      (assume (not P)`
`          (!either`
`               P`
`               (!(conclude Q)`
`                   ...)))))`
	In both of the subcases, `(or` P Q`)` is proved, but under different assumptions, which are then taken into account by the `cases` method. Note that in the P case, Q is not necessarily in the assumption base, but it doesn't have to be since the other argument of `either`, P, is in the assumption base. Similarly, in the `(not` P`)` case, P doesn't have to be in the assumption base (in fact, it shouldn't be, since `(not` P`)` is). Alternatively, the proof could be broken into cases Q and `(not` Q`)`, or R and `(not` R`)` for some other proposition R.

2.7 Proof of a universally quantified proposition

`(!(conclude (forall ?x P))`
`  (pick-any v`
`      (!(conclude P')`
`        ...)))`
	Here P' should be the proposition that results from replacing all free occurrences of $?x$ in P with v. The identifier v that is chosen to use in the `pick-any` must not occur free within P. In some cases it may be better to omit the inner use of `conclude`.

2.8 Proof of an existentially quantified proposition

`(!(conclude (exists ?x P))`
`  (dseq`
`      (!(conclude P')`
`         ...)`
`      (!egen (exists ?x P) t)))`
	Here P' should be the proposition that results from replacing all free occurrences of $?x$ in P with some term t (t can be any term for which it is possible to prove the resulting P'). If the proof of P' is successful, `egen` generalizes it to the existentially quantified proposition.

3 Premise-Driven Proof Development

For each syntactic form of premise we suggest a type of deduction that can take advantage of it. Keep in mind that by a "premise" we mean any proposition in Athena's current assumption base, including those that are added to it during the current deduction.

3.1 Using a conjunction

If there is a premise P of the form
 	
`(and` Q R`)`
 	
consider using
 	
`(!left-and` P`)` and `(!right-and` P`)`
	Use these as needed to bring the conjuncts Q and R into the assumption base.

3.2 Using a disjunction

If there is a premise P of the form
 	
`(or` Q R`)`
 	
consider using
 	
`(!cd (or` Q R`)` ...`)`
	See "proof by case-splitting" (Section 2.4).

3.3 Using an implication

If there is a premise P of the form
 	
`(if` Q R`)`
 	
and Q is also a premise, consider using
 	
`(!mp` P Q`)`
	This is "modus ponens," which puts R into the assumption base. The implication P might also be useful as an argument to `cd`, `cases`, or `equiv`.

3.4 Using an equivalence

If there is a premise P of the form
 	
`(iff` Q R`)`
 	
consider using
 	
`(!left-iff` P`)` and `(!right-iff` P`)`
	Use these as needed to bring the implications `(if` Q R`)` and `(if` R Q`)` into the assumption base, respectively.

3.5 Using a universally-quantified premise

If there is a premise P of the form (for $?x$ Q) consider using (!uspec P t)	This puts Q' in the assumption base, where Q' is the result of substituting the term t for all free occurrences of $?x$ in Q. Here t is a term chosen so that the resulting Q' will be useful in further steps of the proof. If P has the form (for $?x_1$ $?x_2 \ldots ?x_n$ Q), one can specialize some or all of the variables at once with (!uspec* P $[t_1\ t_2 \ldots t_m]$) where $m \leq n$; terms $t_1, t_2, \ldots, t_m$ will be substituted for $?x_1, ?x_2, \ldots, ?x_m$, respectively.

3.6 Using an existentially-quantified premise

If there is a premise P of the form (exists $?x$ Q) consider using (pick-witness v P (dlet ((n Q')) ...)))	Within the enclosed deduction the proposition Q' will be in the assumption base, where Q' is the result of substituting the identifier v for all free occurrences of $?x$ in Q. The use of dlet as shown to introduce a name n for Q' is not strictly necessary but writing out Q' explicitly can make it easier to see how to proceed with the proof (and for others to read it), and naming it makes it easier to refer to. Note that v must not be allowed to "escape" the enclosed deduction, since there is no way to express in a proposition the constraint that it was chosen to represent some value for which Q holds. It is usually eliminated by using it in an egen method application to conclude another existentially quantified proposition, or by using it within a (!by-contradiction (assume R ...)) deduction in which R is independent of v.

4 Example

Suppose we want to prove that for an arbitrary binary relation R,

$$\exists y \forall R(x, y) \supset \forall x \exists y R(x, y).$$

In Athena we set up this goal with

```
(domain D)

(declare R (-> (D D) Boolean))

(define exists-forall
  (exists ?y
    (forall ?x (R ?x ?y))))

(define forall-exists
  (forall ?x
    (exists ?y (R ?x ?y))))

(define goal (if exists-forall forall-exists))

(!(conclude goal)
  ...)
```

We begin working on the proof top-down (goal driven). Since the goal is of the form `(if` P Q`)`, we apply template 2.2 (Proof of an implication):

```
(!(conclude goal
  (assume
      exists-forall
    (!(conclude forall-exists)
       ...)))
```

Now the new (sub)goal is of the form `(forall` $?v$ P_1`)`, so we apply template 2.7 (Proof of a universally quantified proposition):

```
(!(conclude goal)
  (assume
     exists-forall
     (!(conclude forall-exists)
       (pick-any x
        ...))))
```

Within the `pick-any` we have a new goal $P_1 =$ `(exists ?y (R x ?y))`, which suggests applying template 2.8 (Proof of an existentially quantified proposition). However, the recommended method is `egen`, which will require having a "witness," and to obtain such a witness we need to make use of our existentially quantified premise, `(exists ?y (forall ?x (R ?x ?y)))`. This bottom-up (premise-driven) step is done with template 3.6 (Using an existentially quantified premise):

```
(!(conclude goal)
  (assume
      exists-forall
    (!(conclude forall-exists)
      (pick-any x
         (pick-witness z exists-forall
                       (dlet ((z-for-all (forall ?x (R ?x z))))
                         ...))))))
```

Now we have a premise of the form `(forall ?x (R ?x z))`, which suggests continuing bottom-up and making use of it with template 3.5 (Using a universally-quantified premise):

```
(!(conclude goal)
  (assume
     exists-forall
   (!(conclude forall-exists)
     (pick-any x
         (pick-witness z exists-forall
                       (dlet ((z-for-all (forall ?x (R ?x z))))
                         (dseq
                          (!(conclude (R x z))
                            (!uspec z-for-all x))
                            ... )))))))
```

This works, because it shows that `z` can serve as the witness we need for using `egen`:

```
(!(conclude goal)
  (assume
     exists-forall
   (!(conclude forall-exists)
     (pick-any x
         (pick-witness z exists-forall
                       (dlet ((z-forall (forall ?x (R ?x z))))
                         (dseq
                          (!(conclude (R x z))
                            (!uspec z-forall x))
                          (!egen (exists ?y (R x ?y))
                                 z))))))))
```

That completes the proof; Athena's response is:

```
Theorem: (if (exists ?y:D
               (forall ?x:D
                 (R ?x ?y)))
             (forall ?x:D
               (exists ?y:D
                 (R ?x ?y))))
```

References

[1] Konstantine Arkoudas. *Denotational Proof Languages*. PhD thesis, MIT, 2000. 1

[2] Konstantine Arkoudas. An Athena tutorial, 2004. http://www.cag.csail.mit.edu/~kostas/dpls/athena. 1

[3] David R. Musser and Aytekin Vargun. Proving theorems with Athena, September 2003, revised January 2005. http://www.cs.rpi.edu/~musser/gsd/athena/. 1