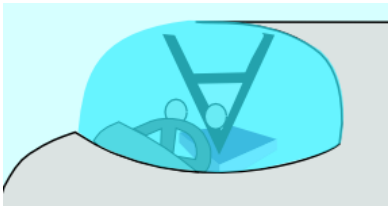
Proofs in the Pilot's Seat: Toward Verified Simultaneous Maneuvers in the Next-Generation Airborne Collision Avoidance System

Brandon Bohrer (bbohrer@cs.cmu.edu)

Contents



1 Abstract

ACAS X is a collision-avoidance system currently being developed by the Federal Aviation Administration, which when completed will be deployed on large commercial aircraft worldwide. In this work I present a model for encounters between pairs of ACAS-X equipped aircraft, where each aircraft can perform arbitrary vertical maneuvers. I present a decision procedure (expressed in first-order logic) showing whether an arbitrary maneuver is safe. The safety theorem for my states that any maneuver approved by my decision procedure is safe (will not lead to a mid-air collision). This theorem will allow us to integrate our safety analysis into the ACAS X implementation with confidence, detecting encounters where the existing logic would make provably unsafe advisories and replacing them with provably safe advisories. I reduce the safety proof to 32 cases, $2 - \epsilon$ of which are verified in KeYmaera X and the rest of which are left as future work. The two cases I have completed consist of 600 lines of Scala code using the KeYmaera X libraries, much of which is likely reusable for the remaining 30 cases. The proof effort includes a number of novel tactics (code that implements a complex proof operation using of simpler operations) that are reusable in future proofs.

2 Introduction

ACAS X is an aircraft collision avoidance system currently being developed to replace the TCAS system used by the Federal Aviation Administration. As in TCAS, ACAS X works by giving an *advisory* to the human pilot informing them what vertical maneuver, if any, should be made to avoid an imminent collision. Our lab, in collaboration with researchers at Johns Hopkins University, is analyzing the safety of advisories given by ACAS X using formal methods. Previous work [7] has analyzed encounters where one aircraft (the *ownship*) is maneuvering vertically in response to an ACAS X advisory, while the second aircraft (the *intruder*) is moving in a straight line in the absence of an advisory.

The task at hand is to extend this analysis to support encounters where both aircraft are maneuvering vertically. This level of generality is essential for several reasons:

1. Currently we cannot make safety statements about encounters in which both aircraft are equipped with ACAS X, which will be an extremely common scenario, nor can we say anything about encounters in which the

2. We intend to use our safety analysis to detect encounters in which the existing ACAS X logic would make a provably unsafe advisory and replace it with a provably safe advisory. We can only make such a decision if we can reason about all maneuvers the intruder might make. Because this decision-making logic is safety critical, it is essential to provide the strongest correctness guarantees possible. Such correctness guarantees are provided by the formal approach taken in this project.

Our collaborators have made some progress in this direction. In particular, ongoing work [3] describes a general approach for proving first-order arithmetic propositions that arise during the safety analysis, and which are too difficult to handle with current automated solvers. The same work also provides a Coq proof that builds upon this approach to show that certain conditions are safe.

What's lacking in the above work is an *understanding* of what has or has not been proved. The safety theorems that have been proven so far do not contain a clear model of pilot behavior, but rather consist mostly of first-order real arithmetic formulas that are hard to interpret.

Not knowing what you proved is a serious problem. We intend to use this safety analysis as part of a safety-critical production system. If our verification result does not correspond with the code we deploy, we will be able to say nothing about the correctness of the deployed system, possibly leaving airline passengers in mortal danger.

My project addresses this lack of understanding by reformulating our safety theorems in differential dynamic logic (d$\mathcal{L}$) with an explicit model for pilot behavior, then proving safety formally in KeYmaera X. In contrast with previous approaches, differential dynamic logic enables me to write a concise, explicit model of pilot behavior which can easily be compared with the designers' expectations by inspection.

More specifically, my safety proof defines a formula in first-order logic describing a *safe region* for each maneuver in which no collisions will occur if the given maneuver is followed. This proof can then be combined with ongoing work [3] to provide a propositional formula for the safe region, from which we can trivially generate correct code that determines whether a given maneuver is safe in a given state. This, in turn, allows us to confidently integrate the generated code into the ACAS X implementation in the following fashion:

1. Run the existing ACAS X logic, which suggests an advisory.
2. Use the generated code to compute whether the advisory is safe.
3. If the suggested advisory is potentially unsafe, use the generated code to compute whether some other advisory is completely safe (there are only 18 advisories, so this is feasible).
4. If some other advisory is completely safe, override the ACAS X suggestion with the safe advisory. Otherwise, use the advisory suggested by ACAS X, since we have not found an advisory that is clearly safer.

This is a technically demanding project that pushes the frontier of what can be verified with differential dynamic logic. Looking at the number of variables in a model can give us a sense of the complexity, since proof automation tools generally scale poorly in terms of the number of variables. The model presented here has 30 variables, compared to 15 for the most complicated model from the previous ACAS X verification effort [3, 7–10]. As the model becomes more complex, so do the proofs. As I need to prove statements about increasingly complex arithmetic, an increasing fraction of these proofs has to be done manually instead of by automated tools. Furthermore, the size of this proof stresses the limits of the KeYmaera X, suggesting a number of future improvements for the prover, some of I which I implement as a small library in this work.

3 Model, Safety Properties and Analysis

I follow the modeling approach of [3, 7–10], but extend it to include encounters where both the ownship and intruder can maneuver vertically.

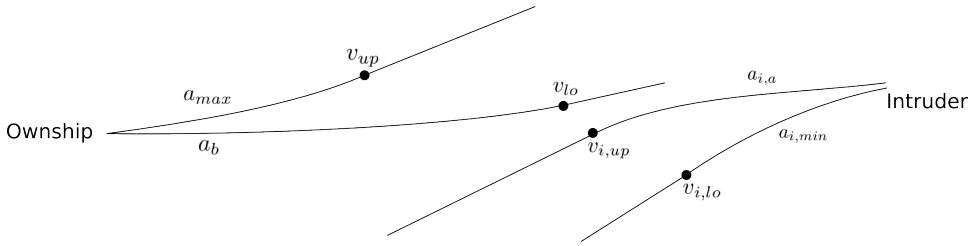


Figure 1: Safe Pair of Maneuvers

Continuous Dynamics We model the vertical motion of each aircraft as a series of parabolas: that is the vertical acceleration of an aircraft can only discretely, when the pilot makes a new control decison, and is constant throughout the continuous evolution of the system. The vertical positions are represented as a height h for the ownship and h_i for the intruder, with velocties v and v_i and accelerations a and a_i.

The model of horizontal motion is very simplistic: We assume the aircraft are moving head-on in the horizontal plane, which means their horizontal position can be fully described by the distance (range) r between them. Furthermore we assume they're moving with a constant horizontal velocity v_r.

These continous dynamics can be summarized in the following differential equation:

$$r' = -vr, hi' = vi, h' = v, v' = a, vi' = ai$$

Safety The notion of safety I investigate in this project is the absence of near midair collisions (NMAC). An NMAC, defined as vertical separation of less than 500 feet and lateral separation of less than 0.5 nautical miles, means two aircraft have come close enough that there is serious risk of an actual collision, and is considered a serious incident by regulatory agencies.

Formally, we say an NMAC occurs if a cylindrical *puck* centered on the center of the ownship intersects the center of the intruder (who is modelled as a point). We write h_p for the half-height of the puck and r_p for the radius of the puck. We leave these values as parameters instead of hard-coding 500 feet and 0.5 nautical miles both because these values must include the size of the aircraft themselves and because this gives us freedom to experiment with different levels of safety in our analysis.

Discrete Control: Maneuvers As mentioned above, the pilot for each aircraft can choose new accelerations throughout an encounter. The accelerations they are allowed to pick from are defined by a *maneuver*. In ACAS X there are exactly 18 maneuvers allowed, some of which are only allowed as a second advisory after the first advisory has shown itself to be insufficient.

We model a more general notion of maneuver so that the work can be more widely applicable, but our notion of maneuver is chosen to ensure it can express the maneuvers needed by ACAS X.

A maneuver (for the ownship) specifies an upper bound v_{up} and lower bound v_{lo} on the velocity v. If the ownship is not within these velocity bounds at the start of the maneuver, they accelerate to satisfy the bounds. There are four acceleration bounds $a_{min}, a_a, a_b, a_{max}$ where a_{min} and a_{min} are absolute bounds that must be obeyed at all times and a_a and $_b$ specify the minimum acceleration

with which the pilot must decelerate to achieve $v \leq v_{up}$ or accelerate to achieve $v_{lo} \leq v$, respectively if one of those conditions does not hold initially.

It is worth noting that at any given control point (i.e. any one decision made by the pilot) We can use v_{lo}, v, and v_{up} to simplify the acceleration bounds to two bounds a_{lo} and a_{up} such that $a_{lo} \leq a \leq a_{up}$ must hold throughout the next phase of the continuous dynamics. We will take advantage of this to simplify the proof, but cannot make the same simplification in the model, because a_{lo} and a_{hi} can change throughout the maneuver. For example, if a pilot is initially at $v > v_{up}$ they will have $a_{up} = a_a$, but if they then decelerate so $v < v_{up}$ they will then have $a_{up} = a_{max}$.

We say a maneuver is *well-formed* if the a and v bounds are consistent with each other, e.g. $a_{min} \leq a_a < 0 < a_b \leq a_{max}$. The intruder has its own set of variables: $v_i, v_{i,up}, v_{i,lo}, a_i, a_{i,min}, a_{i,a}, a_{i,b}, a_{i,min}, a_{i,lo}, a_{i,up}$ and follows their own maneuver which works the same way as the ownship's maneuver.

Discrete Control: Safe Regions An essential design question in this project is to design a safety predicate `?Safe` that determines whether a given maneuver is safe. We wish to find a predicate that is as *efficient* as possible, meaning true in as many safe states as possible, while also being *sound*, meaning false in *every* unsafe state. In this section, I derive such a safety predicate. In the proof section I present my progress toward a formal proof of safety for this predicate.

The system is safe if both aircraft are following *valid* maneuvers and either the ownship is safely above the intruder for the entire maneuver or safely below the intruder for the entire maneuver.

```
Safe <->
 ManeuverOk(v, vlo, vup, alo, aup) &
 ManeuverOk(vi, vilo, viup, ailo, aiup) &
 (OwnAbove | OwnBelow)
```

We say a maneuver is valid if it is always possible for us to accelerate into the allowable velocity range and all the expected inequalities hold:

```
ManeuverOk(v, vlo, vup, amin, aa, ab, amax) <->
  vlo <= vup & amin <= aa & aa <= 0 & 0 <= ab & ab <= amax
```

In order to define what it means to be safely above or safely below another aircraft, we must first model the trajectory of an aircraft. The predicate $A(t, hn, h0, v, vlim, alim)$ says that our height at time t is h_n if we follow a trajectory that starts with height h_0 and velocity v then accelerates at a_{lim} until v_{lim} is achieved and then maintains constant velocity (or maintains constant velocity at the start if $\frac{v}{a_{lim}} > \frac{v_{lim}}{a_{lim}}$).

```
  A(t,hn,h0, v,vlim,alim) <->  rn = rv*t
    &((0 <= t & t <  max(0, (vlim-v)/alim)
              & hn = h0 + alim/2*t^2 + v*t)
    |           t >= h0 + max(0, (vlim-v)/alim)
              & hn = (max(vup, v)/aup)*t - (max(0,(vlim-v))/(2*alim))^2)
```

Next we need to define what the acceleration limit a_{lim} is in each case (this is exactly the dimension reduction from $a_{min}, a_a, a_b, a_{max}$ to a_{lo}, a_{up} that I mentioned earlier):

```
isAlo(a, amin, ab, v, vmin) <->
  (a = amin & v >= vmin)
 |(a = ab   & v <= vmin)

isAup(a, amax, aa, v, vmax) <->
  (a = amax & v <= vmax)
 |(a = aa   & v >= vmax)
```

We can then describe whether one plane is above the other:

```
  OwnAbove <-> \exists alim \exist ailim \forall t \forall rn \forall hn \forall ihn
    rn = rv*t
  & isAlo(alim, amin, ab, v, vlo)
  & isAup(ailim, aimax, aia, vi, viup)
  & A(t, hn, h0, v, vlo, alo)
  & A(t, ihn, ih0, iv, ivup)
    -> abs(r - rn) > rp | (h + ihn - hn) < -hp

OwnBelow <-> \exists alim \exist ailim \forall t \forall rn \forall hn \forall ihn
    rn = rv*t
  & isAlo(ailim, aimin, aib, vi, vilo)
  & isAup(alim, amax, aia, v, vup)
  & A(t, hn, v, vup, aup)
  & A(t, ihn, iv, ivlo)
    -> abs(r - rn) > rp | (h + ihn - hn) > hp
```

(Idealized) d$\mathcal{L}$model Now that we have derived a safe region, it may be useful to see how the different parts of the model fit together. Here is the complete hybrid program:

```
 {/* Either keep the current maneuver by doing nothing
    ("?true" does nothing) or assign a new safe maneuver */
  {?true; ++
  amin :=*; amax :=*; aa :=*; ab :=*; vlo :=*;  vup :=*;
  aimin :=*; aimax :=*; aia :=*; aib :=*;  vilo :=*; viup :=*;
  /* Test that the maneuver is both well-formed and safe */
  ?Safe;
  /* Assign arbitrary accelerations. The constraints on the
   following ODE ensure the accelerations are valid.  */
  { a :=*; ai :=*;
  /* Evolve continuously with linear horizontal motion,
     quadratic vertical motion and a clock t */
  {r' = -vr, hi' = vi, h' = v, v' = a, vi' = ai
   & (a >= amin & a <= amax)  /* Valid ownship velocity */
   & (v >= vlo | a >= ab) /* If velocity is too low, go up */
   & (v <= vup | a <= aa) /* If velocity is too high, go down */
   & (ai >= aimin & ai <= aimax) /* Valid intruder velocity */
   & (vi >= vilo | ai >= aib) /* If velocity is too low, go up */
```

```
  & (vi <= viup | ai <= aia) /* If velocity is too high, go down*/
 }}* /* This loop allows the pilot to change acceleration during
     a maneuver */
}* /* This loop allows ACAS X to choose multiple
      maneuvers during an encounter. */
```

And then our safety theorem simply states

$$r_p >= 0 \land h_p > 0 \land v_r >= 0 \implies [\alpha](|h_i - h| > h_p || r| > r_p)$$

In the interest of full disclosure, my current proof is about a simplified version of this model. I discuss the simplification in the proof section where they are most relevant. I fully intend to address these simplification in future work, but simplifications are often necessary in the early stages of a project.

4 Model Limitations

Even ignoring the simplifications that I make in the proof, this model has limitations both in *efficiency* and in *safety*. Efficiency issues will lead our analysis to report false alarms when we assess the safety of ACAS X. This can be an issue because if we have too many false alarms, it is difficult to sort through test cases and determine which cases are false alarms and which cases demonstrate real bugs in ACAS X. That being said, efficiency issues are less frightening because we can easily detect them before the system is ever deployed, giving us a chance to address them, whereas safety issues are more concerning because if we do not find them during the design phase, they could lead to midair collisions in the final version of ACAS X.

An important safety limitation is the fact that it is sometimes physically impossible for a pilot to obey the advice given to it by ACAS X, usually because the aircraft is near its service ceiling and adverse weather conditions limit the potential climb rate to less than the climb rate requested by ACAS X. This has ramifications if we wish to use the safety analysis to inform control decisions. The dangerous case is if the original ACAS X logic suggests a maneuver that has a low, non-zero probability of collision, but can safely be executed by the aircraft while our safety analysis suggests a maneuver that would have no probability of collision *if followed out successfully*, but has *higher* probability of collision because the aircraft physically cannot follow our assumptions. We believe this limitation does not make us worse off than other approaches because both TCAS and ACAS X ignore this issue entirely. It's worth noting that climb advisories are still often good advice when an aircraft is at its service ceiling. In practice, aircraft that are at their ceiling can maintain climbs for short periods of time at the expense of lost speed. Since collision avoidance maneuvers are short-lived, this can give the intruder craft extra space to maneuver around the ownship without putting the ownship at significant additional risk.

We also ignore that present TCAS (and possibly ACAS X) can only operate within a certain range of velocities (TCAS does not work if the vertical closing rate exceeds 10,000ft/s). We deem this limitation acceptable because not only does it seem rare in practice, but it is unlikely to have an effect on which advisories ACAS X should give. ACAS X never gives advisories that violate these bounds on their own, so the only situation this would matter is if the intruder is extremely non-compliant and our choice of advisory would be the deciding factor in whether or not we remain within ACAS X's operating range.

A particularly serious safety problem is that of non-compliant or overcompliant pilots. Our safety analysis must necessarily make some assumptions on pilot behavior to avoid excessive false positives, but this must be balanced by the fact that almost all assumptions will be broken in practice. It is a standard assumption that during coordinated encounters, intruder aircraft comply with their ACAS X advisories, but this is not always the case. The 2002 Überlingen plane crash occurred in a encounter where one pilot complied with TCAS and the other did not, setting them on a collision course. This would have been avoided if the compliant pilot's TCAS reversed the advisory under the assumption that the ownship will continue to comply and the intruder will continue not to comply. At the time TCAS did not make such a reversal, which was a contributing factor to the crash [19] and was the impetus for a revision to the TCAS collision avoidance logic (TCAS 7.1). [16]. Our analysis does not give us an analysis of this scenario for free, but the Überlingen crash shows it is of serious practical concern. I suspect that with additional work, our approach might be extensible to such an analysis.

In practice, *overcompliance* is a common phenomenon: pilots will maneuver in the requested direction, but in an excessive magnitude. This behavior is understandable given the short window in which pilots have to react and the high stress involved in an TCAS alert. In two-aircraft encounters, overcompliance is often harmless, but in multi-threat encounters it can be a source of danger. For our model, a partial solution to this problem would be to add an extra parameter describing the maximum allowed overcompliance to an advisory. This would allow us to recognize the overcompliance problem and measure how the safety of the system changes as overcompliance increases. Such a model is only truly useful when combined with empirical measurements of how much overcompliance occurs in practice. Because TCAS alerts are often thoroughly documented, such data may actually be available.

While this effort focuses on vertical motion, we also make a potentially-unsafe simplification in the horizontal dimension. We assume the derivative of range (horizontal distance between aircraft) is constant. This is an invalid assumption for several reasons:

- The plane may have already been accelerating horizontally when the encounter began, and under the "don't maneuver horizontally" assumption, that will continue to be the case.
- As the plane maneuvers vertically, this may change their horizontal velocity.
- Unless the planes are moving perfectly head-on, the range does not have constant derivative even under the assumption that there is no horizontal acceleration. To see this most simply, consider the scenario in Figure 2 where two aircraft are the closest point of approach (which in this example is an NMAC). At this point, the derivative of range is 0.

One safety issue that seems inherent to the nature of collision-avoidance is that we cannot guarantee absolute safety in the presence of multiple intruders and a finite sensor range. In the most extreme case, consider an "ambush" encounter where a horde of intruder aircraft enters your sensor range from every angle and altitude all at the same time. Clearly you cannot avoid a collision if you're completely surrounded. This issue may arise even in simpler encounters as well.

Compared with TCAS, this model also has efficiency issues. It remains to determine whether the same applies for ACAS X. TCAS makes the assumption that pilots can respond more quickly to follow-up advisories (2.5 seconds) than to initial advisories (5 seconds), but none of the existing models allow for this variation in response time (and my current model does not even handle delay at all). TCAS also enforces a 5 second delay between consecutive reversals, presumably to prevent pilot confusion. This differs from ACAS X which does not allow multiple reversals at all.

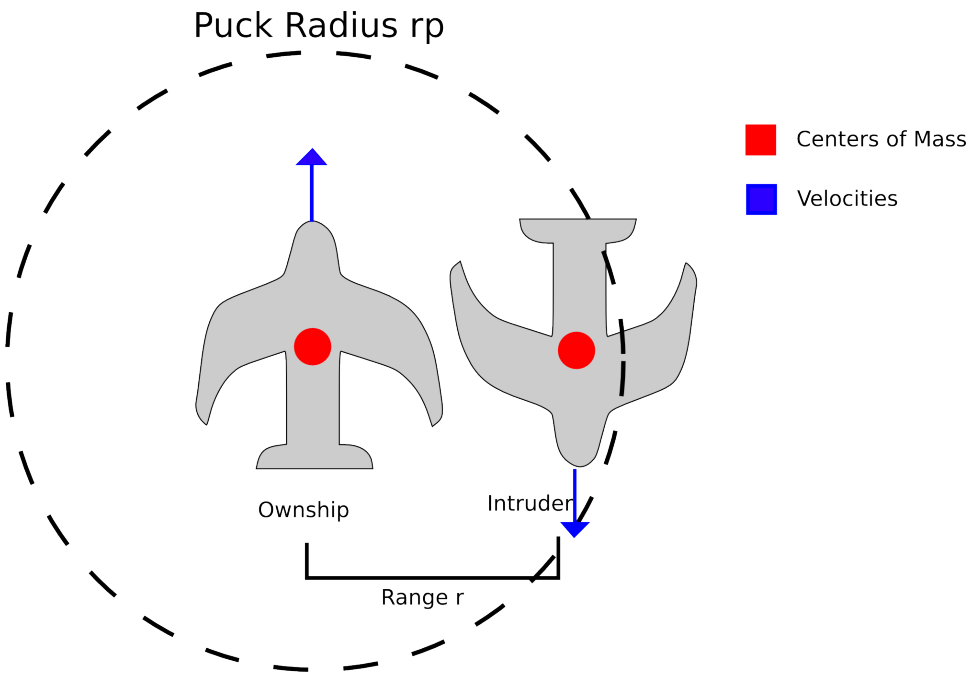


Figure 2: NMAC at Closest Point of Approach, Parallel Flight

5 Custom Tactics Library

Before going into the proof itself in depth, it is worth discussing the tactic[1] library that I built to make the proof easier, and which I hope to use as the basis for standard libraries in KeYmaera X My custom tactics library contains the following components:

1. Arithmetic lemmas
2. In-place proof steps
3. Maintainability-enhancing tactics.

5.1 Arithmetic Lemmas

My library includes the following arithemetic facts. These facts all prove automatically, making this by far the simplest part of the library. However, this listing should make a good summary of the low-level arithmetic steps needed in the proof.

```
hi - h <= -hp -> abs(hi-h)>=hp
∀a∀b∀t ((t < a & t < b) | (t < a & t >= b) |
    (t >= a & t < b) | (t >= a & t >= b))
∀a∀b∀c ((a <= b & b <= c) -> a <= c)
∀a∀b (a = b -> a <= b)
∀x (x = x)
∀a∀b∀c∀d ((a <= c & b >= d) -> a - b <= c - d)
∀a∀b∀c ((b <= c) -> a + b <= a + c)
∀a∀b∀c∀d ((a <= c & b <= d) -> a + b <= c + d)
∀a∀b∀c ((b >= c) -> a + b >= a + c)
∀x∀y (y >= x -> max(x,y) = y)
∀x∀y∀z (z >= max(x, y) -> z >= y)
∀x∀y∀z (z >= max(x, y) -> z >= x)
∀x∀y (max(x,y) = x | max(x,y) = y)
```

5.2 In-place proof steps

It turns out that the lemmas given above are quite verbose to use in proofs by themselves. For example, if we wish to use the transitivity lemma to reduce an open proof goal $x \leq z$ located at position `pos` to proving just $x \leq y \wedge y \leq z$, we need the following proof:

```
1  cutL (transitivityFormula) <(
2      allL("x".asFormula)('Llast) &
3      allL("y".asFormula)('Llast) &
4      allL("z".asFormula)('Llast) &
5      implyL('Llast) <(
6         hide(pos),
7         closeId) partial
8    , cohide('Rlast) &
```

[1] A tactic is a piece of code which implements a proof or a part of a proof by building it up out of basic proof rules provided by the theorem prover implementation.

```
9      trasitivityTactic)
```

You need not worry about the details of this proof, but the point stands that such a simple, intuitive proof step should not require 9 steps. Most of these steps will always be the same, except the following steps which depend on the goal located at `pos`:

```
2      allL("x".asFormula)('Llast) &
4      allL("z".asFormula)('Llast) &
```

and the following step, where the formula y has to be supplied by the person writing the proof:

```
3      allL("y".asFormula)('Llast) &
```

My tactic library reduces this to a simple one-line tactic call: `trans("y".asFormula)(pos)` and does the same for a number of arithmetic lemmas used throughout the proof. Their names in the Scala code are as follows:

```
trans
eqToLeq
refl
leqLoosenDiff
leqSumLeftConst
leqSumPiecewise
geqSumLeftConst
```

Some of these tactics, such as `trans` and `refl` are clear candidates for a standard library. It less clear whether the other tactics should be part of a standard library or where they would fit in. As a follow-up project I plan on developing a suite of arithmetic tactics as part of the KeYmaera X standard library.

The remaining tactics in this section, `implyGoalWith` and `implyWithAssumption`, implement the following proof rules:

$$\frac{\Gamma \vdash P, \Delta \quad \text{tactic } e \text{ proves } \cdot \vdash P \implies Q}{\Gamma \vdash Q, \Delta} \qquad \frac{\Gamma, Q \vdash \Delta \quad \text{tactic } e \text{ proves } \cdot \vdash P \implies Q}{\Gamma, P \vdash \Delta}$$

These tactics are essentially two variants of modus ponens, one of which operates on assumptions and one of which operates on conclusions. These both follow readily from built-in proof rules. Furthermore, to avoid introducing extra proof branches (which can make the proof a bit harder to read) these both accept tactics e as an argument which prove the side branches. This is most useful in the common case where $P \implies Q$ is some lemma that has already been proved with some tactic e and will be used many times over.

5.3 Maintainability-Enhancing Tactics

Some of the existing tactics in KeYmaera X, while they get the job done, are not very effective from a maintainability standpoint. The vast majority of these tactics require specifying the affected formulas by index, which is both incredibly unreadable and subject to change frequently as the proof or model change during debugging. One design goal for future library development is to reduce the usage of explicit indices and attempt to make tactics less fragile overall to minor bugfixes in a proof or model. The following tactics are an initial step in that direction:

- multiCohide: Before handing an arithmetic problem off to an automated tool, we almost always want to get rid of unnecessary assumptions, leaving only the ones that are actually relevant to the proof. This is because these tools do not scale well as formulas get more complicated, especially as the number of variables increases (solving first-order arithmetic problems is $O(2^{2^n})$ in the number of variables). Existing tactics are not well equipped for this: our only options are to either remove assumptions one at a time, specifying the position of each assumption we remove (which is incredibly fragile) or remove every assumption except a single assumption of our choosing (which is insufficient in the most common case where more than one assumption is relevant). This tactic is a simple extension of the existing rules which takes a list of the assumption we want to keep, then scans through the assumptions removing all the ones we didn't explictly ask to keep. So far this has made the proof much more maintainable and I believe will make a useful member of the standard library.

- rewriteEqsOver: This solves a similar problem for equality rewriting: often we have some equations that define some of the variables in the system based on known values, e.g. $v = v0 + a * t$. Because the right-hand side often changes as we debug the model, it is useful to have a tactic that locates these equations using only the left-hand side (i.e. v) and rewrites every v to $v0 + a * t$. That is exactly what this tactic does.

6 Proof Approach and Challeneges

Truth in Advertising The model I'm proving makes two significant simplifications from the one presented here:

1. My proof does not look at horizontal motion at all, assuming the aircraft are always in horizontal conflict and thus need to be vertically separated forever.

2. My proof does not allow the pilot to change accelerations during a maneuver.

These are both real limitations that I have every intention of resolving in future work, but this proof is simply big enough that it makes sense to add extra restrictions to the initial version of the proof.

Proof My safety proof is based on simple proof techniques, but applied at large scale. The proof breaks up into three main phases:

1. Solve the continuous dynamics for both aircraft.

2. Reduce the safety of the solved system to a finite number of cases.

3. Use first-order logic and arithmetic lemmas to slog through the resulting cases. This is a large majority of the proof effort.

6.1 Solving continuous dynamics

Solving the continuous dynamics is by far the easiest part of the proof. The horizontal dynamics (which are not even used in the current iteration of the proof) are linear: $r = r0 + rv * t$ and the vertical dynamics of each intruder is a quadractic: $h = h_0 + v_0 * t + a * \frac{t^2}{2}$ for the ownship

and $h_i = h_{i,0} + v_{i,0} * t + a_i * \frac{t^2}{2}$ for the intruder. One technical detail is that the existing ODE solving tactics are unusably slow for systems this complex, so I simply prove that these equations are invariants of the system by hand.

6.2 Reduce Safety to Finitely Many Cases

I break the proof into cases by following the structure of the safety predicate `Safe`. In particular, there are three places where the predicate branches, each of which corresponds to branches in the proof:

- If the system is safe, either the ownship is safely above the intruder or the intruder is safely above the ownship (2 choices)
- In each case, other the ownship can accelerate freely (e.g. up to a_{max}) or its acceleration is restricted (e.g. $a \leq a_a$ because we need to accelerate until $v \leq v_{max}$). The same goes for the intruder ship (4 choices total). One might expect to see 16 choices here because the ownship and intruder each have both an upper and lower bound on their acceleration, but remember that if the ownship is above the intruder, we only care about the ownship's lower bound and the intruder's upper bound (or vice versa when we're below the intruder). Thus there are 2 bounds to consider and 4 total choices.
- In each case, the trajectory of each ship looks like a quadratic function followed by a linear function. The proof will differ based on whether we are currently in the linear part or the quadratic part (4 choices).

Since $2 * 4 * 4 = 32$, we have a total of 32 cases. In each case, the goal is to show that the real trajectories for both ships obey the bounds established by the safe region, which then implies the trajectories are safe.

6.3 Solve each case

In each case it suffices to show that the real trajectories obey the bounds given by the safe region. In principle these proofs are straigtforward, but the details can take significant proof effort. The interesting case (as appears in case 2 out of 32) is when wee are currently in the linear part of the safety bound, but the trajectory itself is quadratic. See the picture below for a visual proof, which is made into a rigorous arithmetic proof in KeYmaera X.

The proof for the linear-quadratic case proceeds by computing the point t_1 where the bound switches from quadratic to linear, then splitting the trajectory at that point. We compute the heights of regions A, B, C, D from Figure 3, then have $A \geq B$ because $a \leq a_{max}$ and $C \geq D$ because $a \cdot \frac{(t-t_1)}{2} + (v_0 + a \cdot t_1) \leq v_0 + a \cdot (t_1 + (t - t_1)) = v \leq v_{max}$. Thus $A + C \geq B + D$ so the trajectory is always within the bound.

At present I am almost-done proving the second of the 32 cases. As a gain more experience with this proof I intend to build up my tactics library further so I can reuse as much code as possible between different cases of the proof.

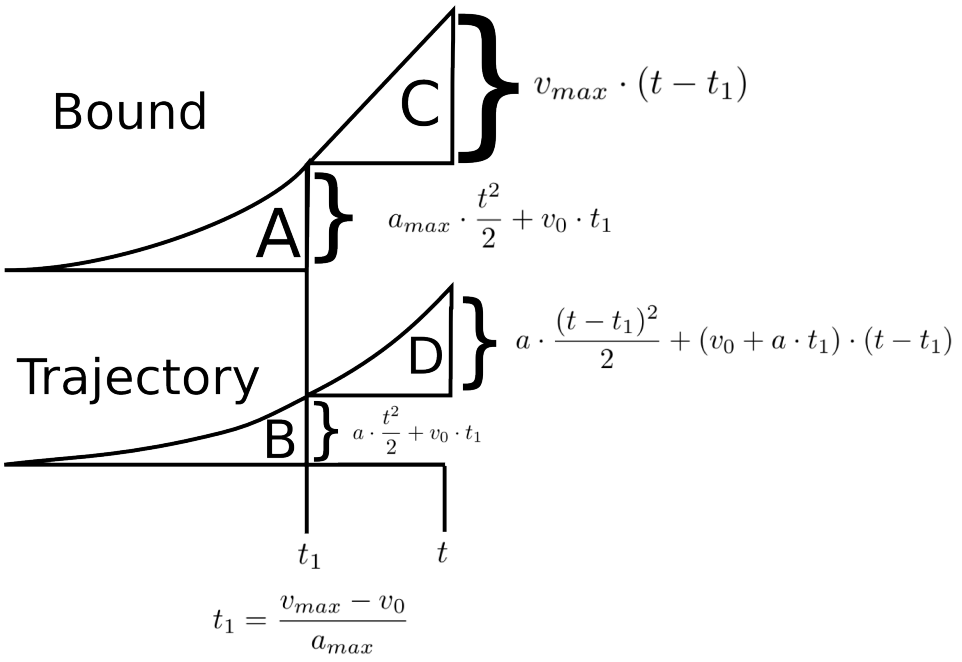


Figure 3: Quadratic Trajectory Obeys Linear Part of Upper Bound

7 Related Work

This work builds directly on the current effort to verify ACAS X [3, 7–10]. Previous work has analyzed vertical maneuvering in the encounter where an intruder aircraft travels in a straight line and the ownship travels in a quadratic parabola, determining which maneuvers for the ownship are safe (i.e. safe for all time even if the aircraft never maneuvers again) and safeable (i.e. safe assuming the ownship makes the appropriate follow-up maneuvers in the future). Their work accounts for pilot and system delay whereas I do not yet. They do not consider intruders which can move in a parabola, which is a focus of this project.

Several collision-avoidance maneuvers have been verified before [18, 21, 22]. However, the goal of this project is not to verify the most complicated maneuvers possible (and in particular our maneuvers are strictly vertical), but rather to verify the safety of a system which might pick arbitrarily from a variety of maneuvers.

Kochenderfer describes the design of the ACAS X collision avoidance logic [11, 12, 15], which is developed by first modeling an encounter as a Markov Decision Process (MDP), assigning a reward function that specifies the desired tradeoff between safety and operational suitability, solving for the optimal policy with respect to the reward function and making collision avoidance decisions according to that policy. Holland shows that the safety and operational suitability of ACAS X compare favorably to TCAS [6] when evaluated using several models based on historical flight data from the US [13, 14]. The tradeoffs between safety and operational suitability are informed by planned changes to the structure of air traffic worldwide. Initiatives such as SESAR [17] and NextGen [1] include plans to pack planes much more tightly in the near future, which places unprecedented demands on collision avoidance.

The safety and suitability of the original ACAS system have been studied as well, with the conclusion that ACAS reduces the risk of collision by anywhere from a factor of 3 to a factor of 5, but leads to excessive alerts in common scenarios [2, 20, 23]. The behavior and requirements of ACAS are extensively documented. Publicly-available documentation for ACAS was essential to our evaluation of the limitations of our approach [4].

8 Conclusion

In this paper I presented a model for aircraft encounters where two ACAS X-equipped aircraft are maneuvering vertically in response to ACAS X advisories. I derived a safe region for these encounters. In preparation for the proof, I developed a library of tactics which will be extended in future work. I made significant progress toward a formal proof of safety using the KeYmaera X Scala libraries, but much work remains for a complete proof.

Completing this proof and removing the simplifications in my model is the main future work. As demonstrated by the accompanying Scala code, the cases I have solved so far already constitute significant progress, and as the proof section of this report suggests, there is much in common with the remaining cases. However, the devil is in the details and it requires a closer look at the remaining cases before I can generalize the proof, which itself a nontrivial task.

Once these proofs are complete, the impact of this project is that we will be able to integrate the safety analysis as part of the implementation of ACAS X, gaining confidence that we are giving safer advisories for collision avoidance than ever before, thus making the skies safer for everyone.

References

[1] Federal Aviation Administration. Next generation air transportation system (NextGen). `https://www.faa.gov/nextgen/`, 2016. Accessed: 2016-04-02.

[2] T. Arino et. al. Studies on the safety of ACAS II in Europe. Technical Rep. ACASA/WP-1.8/210D, 2002.

[3] Y. Kouskoulas et. al. Safe advisories for ACAS X in the presence of curved trajectories and non-deterministic intruder behavior. Unpublished - in progress, March 2016.

[4] Eurocontrol. Overview of ACAS II, 2014.

[5] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413, pages 279–294. Springer, 2014.

[6] Jessica E. Holland, Mykel J. Kochenderfer, and Wesley A. Olson. Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance. Chicago, Illinois, 2013.

[7] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, and Erik Zawadzki André Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. In Christel Baier and Cesare Tinelli, editors, *TACAS*, volume 9035 of *LNCS*, pages 21–36. Springer, 2015.

[8] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. Formal verification of ACAS X, an industrial airborne collision avoidance system. In Alain Girault and Nan Guan, editors, *EMSOFT*, pages 127–136. IEEE Press, 2015.

[9] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Garnder, Aurora Schmidt, Erik Zawadzki, and André Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. Technical Report CMU-CS-14-138, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2014.

[10] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Aurora Schmidt, Ryan Gardner, Stefan Mitsch, and André Platzer. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system, 2015.

[11] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. A decision-theoretic approach to developing robust collision avoidance logic. Madeira Island, Portugal, 2010.

[12] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. Robust airborne collision avoidance through dynamic programming. Project Report ATC-371, Massachusetts Institute of Technology, Lincoln Laboratory, 2011.

[13] Mykel J. Kochenderfer, Matthew W. M. Edwards, Leo P. Espindle, James K. Kuchar, and J. D. Griffith. Airspace encounter models for estimating collision risk. 33(2):487–499, 2010.

[14] Mykel J. Kochenderfer, Leo P. Espindle, James K. Kuchar, and J. D. Griffith. Correlated encounter model for cooperative aircraft in the national airspace system. Project Report ATC-344, Massachusetts Institute of Technology, Lincoln Laboratory, 2008.

[15] Mykel J. Kochenderfer, Jessica E. Holland, and James P. Chryssanthacopoulos. Next generation airborne collision avoidance system. *Lincoln Laboratory Journal*, 19(1):17–33, 2012.

[16] John Law. Controller and pilot ACAS regulation and training. volume 5 of *ACAS II Bulletins*. Eurocontrol.

[17] Hubert Lepori and Dennis Hart. The future ATM system: MET and OGC, 2009.

[18] Sarah M. Loos, David W. Renshaw, and André Platzer. Formal verification of distributed aircraft controllers. In Calin Belta and Franjo Ivancic, editors, *Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013*, pages 125–130. ACM, 2013.

[19] German Federal Bureau of Aircraft Accidents Investigation. Investigation report AX001-1-2. `http://www.bfu-web.de/EN/Publications/Investigation%20Report/2002/Report_02_AX001-1-2_Ueberlingen_Report.pdf?__blob=publicationFile`, May 2004.

[20] W. A. Olson and J. E. Olszta. TCAS operational performance assessment in the U.S. national airspace. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 4.A.2–1–4.A.2–11, Oct 2010.

[21] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009.

[22] David W. Renshaw, Sarah Loos, and André Platzer. Mechanized safety proofs for disc-constrained aircraft. Technical Report CMU-CS-12-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 2012.

[23] Thierry Arino Stéphan Chabert. Safety benefit of ACAS II phase 1 and phase 2 in the new European airspace environment. Technical Rep. ACAS/02-022, 2002.

Appendix A: Model

```
Functions.
 R abs(R).
 R max(R,R).
 R rp. /* Radius of conflict puck */
 R hp. /* Height of conflict puck */
 R vr. /* Relative radial velocity */
End.

ProgramVariables.
 R r. /* Current radius of cylinder between ships (range) */
 R a. /* Acceleration of ownship */
 R ai./* Acceleration of intruder */
 R v. /* Velocity of ownship */
 R v0.
 R vi./* Velocity of intruder */
 R vi0./* Velocity of intruder */
 R h. /* Height of ownship */
 R h0. /* Height of ownship */
 R hi./* Height of intruder */
 R hi0./* Height of intruder */
 R amin. /* Minimum ownship acceleration throughout maneuver */
 R amax. /* Maximum ownship acceleration throughout maneuver */
 R aa.   /* Minimum acceleration with which ownship may approach vup
            during maneuver */
 R ab.   /* Minimum acceleration with which ownship may approach vlo
            during maneuver */
 R vlo.  /* Bottom of target range for ownship velocity */
 R vup.  /* Top of target range for ownship velocity */
 R aimin./* Minimum intruder acceleration throughout maneuver */
 R aimax./* Maximum intruder acceleration throughout maneuver*/
 R aia.  /* Minimum acceleration with which intruder may approach viup
            during maneuver */
 R aib.  /* Minimum acceleration with which intruder may approach vilo
            during maneuver */
 R vilo. /* Bottom of target range for intruder velocity */
 R viup. /* Top of target range for ownship velocity */
 R alo.  /* Lower bound on ownship acceleration for particular maneuver
            + starting configuration */
 R aup.  /* Upper bound on ownship acceleration for particular maneuver
            + starting configuration */
 R aiup. /* Lower bound on intruder acceleration for particular maneuver
            + starting configuration */
 R ailo. /* Upper bound on intruder acceleration for particular maneuver
            + starting configuration */
 R t.    /* Time elapsed since last control decision */
 R time.    /* Time elapsed since last control decision */
 R rn.   /* Radial distance in nominal trajectory */
 R hn.   /* Ownship height in nominal trajectory */
 R hin.  /* Intruder height in nominal trajectory */
 R hrel. /* Relative height trajectory */
End.

Problem.
  rp >= 0 & hp > 0 & vr >= 0
 ->
 [
  /* Assign an arbitrary maneuver */
  amin :=*; amax :=*; aa :=*; ab :=*; vlo :=*;  vup :=*;
```

```
aimin :=*; aimax :=*; aia :=*; aib :=*;  vilo :=*; viup :=*;
t := 0; v0 := v; vi0 := vi; h0 := h; hi0 := hi;
/* Test that the maneuver is both well-formed and safe */
?(
vlo <= vup & amin <= aa & aa < 0 & 0 < ab & ab <= amax &
vilo <= viup & aimin <= aia & aia < 0 & 0 < aib & aib <= aimax &
((\exists alo \exists aiup (
  ((alo  = amin  & v0  >= vlo)  | (alo  = ab  & v0 <= vlo))
& ((aiup = aimax & vi0 <= viup) | (aiup = aia & vi0 >= viup))
& (\forall time \forall hrel ((0 <= time
  &((time <  max(0, (vlo-v0)/alo) & time <  max(0, (viup-vi0)/aiup)
   & hrel = hi0 + aiup/2*time^2 + vi0*time - (h0 + alo/2*time^2 + v0*time))
  | (time <  max(0, (vlo-v0)/alo) & time >=  max(0, (viup-vi0)/aiup)
   & hrel = (hi0 + (viup*time - (max(0, viup - vi0)^2/(2*aiup))))
            - (h0 + (v0*time + alo/2*time^2)))
  | (time >= max(0, (vlo-v0)/alo) & time < max(0, (viup-vi0)/aiup)
   & hrel = hi0 + aiup/2*time^2 + vi0*time - (h0 + alo/2*time^2 + v0*time))
  | (time >= max(0, (vlo-v0)/alo) & time >= max(0, (viup-vi0)/aiup)
   & hrel = hi0 + aiup/2*time^2 + vi0*time
      - (h0+(max(0,(viup-vi0))/(2*aiup))^2) - alo/2*time^2+v0*time)))
  -> hrel <= -hp))))
|(\exists aup \exists ailo (
  ((ailo = aimin & v0 >= vilo)|(ailo = aib  & vi0 <= vilo))
& ((aup = amax   & v0 <= vup) |(aiup = aa   & vi0 >= viup))
& (\forall time \forall hrel ((0 <= time
  &((time <  max(0, (vup-v0)/aup) & time <  max(0, (viup-vi0)/ailo)
   & hrel = hi0 + (ailo/2*time^2 + vi0*time)
          - (h0 + aup/2*time^2 + v0*time))
  | (time <  max(0, (vup-v0)/aup) & time >=  max(0, (viup-vi0)/ailo)
   & hrel = hi0 + (max(viup, vi0)/ailo)*time -
            (h0 + (max(0,(viup-vi0))/(2*ailo))^2
               -(aup/2*time^2 + v0*time)))
  | (time >=  (max(0, (vup-v0)/aup)) & time <  max(0, (viup-vi0)/ailo)
   & hrel = hi0 + (ailo/2*time^2 + vi0*time)
          - (h0 + (max(vup, v0)/aup)*time
               - (max(0,(vup-v0))/(2*aup))^2))
  | (time >=  max(0, (vup-v0)/aup) & time >=  max(0, (viup-vi0)/ailo)
   & hrel = hi0 + (max(viup, vi0)/ailo)*time -
            (h0 + (max(0,(viup-vi0))/(2*ailo))^2 -
                (max(vup, v0)/aup)*time - (max(0,(vup-v0))/(2*aup))^2))))

-> hrel >= hp)))))
);
/* Assign arbitrary accelerations */
a :=*; ai :=*;
/* Evolve continuously with linear horizontal motion,
   quadratic vertical motion and a clock t */
{r' = -vr, hi' = vi, vi' = ai, h' = v, v' = a, t' = 1
 & (a >= amin & a <= amax)
 & (v >= vlo | a >= ab) /* If velocity is too low, go up */
 & (v <= vup | a <= aa) /* If velocity is too high, go down */
 & (ai >= aimin & ai <= aimax)
 & (vi >= vilo | ai >= aib) /* If velocity is too low, go up */
 & (vi <= viup | ai <= aia) /* If velocity is too high, go down*/
}] abs(hi - h) >= hp
End.
```

Appendix B: Scala Code

```
package edu.cmu.cs.ls.keymaerax.btactics.acasxhstp

import edu.cmu.cs.ls.keymaerax.btactics.{DebuggingTactics, DifferentialTactics, EqualityTactics, TacticTestBase}
import edu.cmu.cs.ls.keymaerax.core._
import edu.cmu.cs.ls.keymaerax.tactics.ProvabilityTestHelper
import org.scalatest.{BeforeAndAfterEach, FlatSpec, Matchers}
import testHelper.ParserFactory._
import edu.cmu.cs.ls.keymaerax.bellerophon._
import edu.cmu.cs.ls.keymaerax.btactics.TactixLibrary._
import edu.cmu.cs.ls.keymaerax.btactics.TacticFactory._
import edu.cmu.cs.ls.keymaerax.parser.StringConverter._
import edu.cmu.cs.ls.keymaerax.tags.SlowTest
import edu.cmu.cs.ls.keymaerax.tools.CounterExampleTool

import scala.collection.immutable.Map
import scala.collection.parallel.immutable

/**
  * Created by bbohrer on 4/17/16.
  */
@SlowTest
class SimultaneousManeuvers extends TacticTestBase {

  def debug(s:String) = { DebuggingTactics.debug(s, true)}
  def simpleSeq(f:Formula) =  Sequent(Nil, collection.immutable.IndexedSeq(), collection.immutable.IndexedSeq(f))


  val noRandos = (composeb(1) & randomb(1) & allR(1)) *@ TheType()
  val equateAssigns = (composeb(1) & useAt("[:=] assign equality")(1) & allR(1) & implyR(1)) *@ TheType()
  val postWeaken = implyR(1) & (andL('Llast) *@ TheType())

  val abs_below_fml = "hi - h <= -hp -> abs(hi-h)>=hp".asFormula
  val abs_below_seq = simpleSeq(abs_below_fml)
  def abs_below_tac(implicit qeTool:QETool) = QE

  "Absolute value lemma" should "be provable" in withMathematica { implicit qeTool =>
    proveBy(abs_below_seq, abs_below_tac) shouldBe 'proved
  }

  def implyGoalWith(fml:Formula,e:BelleExpr)(implicit qeTool:QETool):DependentPositionTactic = {
    "implyGoalWith" by ((pos, sequent) => {
      fml match {
        case Imply(p, q) =>
          assert(q == sequent(pos.checkTop))
          assert(pos.isSucc)
          cut(fml) <(implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial, cohide('Rlast) & e) partial
    }})
  }

  "implyGoalWith" should "use P-> Q to change a Q goal into a P goal" in withMathematica { implicit qeTool =>
    val example = "1 + hi - h <= -hp -> abs(hi-h)>=hp".asFormula
    proveBy (example, implyR(1) &  implyGoalWith(abs_below_fml, abs_below_tac)(qeTool)(1) & debug("preQE") & QE)
  }

  /* Turn P assumption into Q assumption with P -> Q */
  def implyWithAssumption(fml:Formula, e:BelleExpr)(implicit qeTool:QETool):DependentPositionTactic = {
```

```
  "implyWithAssumption" by ((pos, sequent) => {
    fml match {
      case Imply(p, q) =>
        assert(p == sequent(pos.checkTop))
        assert(pos.isAnte)
        cut(fml) <(implyL('Llast) <(closeId, nil partial) partial,  cohide('Rlast) & e) partial
    }
  })
}

"implyWithAssumption" should "use P-> Q to change a P assumption into a Q assumption" in withMathematica
  { implicit qeTool =>
  val example = "1 > 2 -> 1 > 3".asFormula
  proveBy (example, implyR(1) &
    mplyWithAssumption("(1>2 -> false)".asFormula, QE)(qeTool)(-1) & debug("preQE") & QE)
}

val TCAStCase = ("\\forall a \\forall b \\forall t ((t < a & t < b) | " +
  "(t < a & t >= b) | (t >= a & t < b) | (t >= a & t >= b))").asFormula
def TCAStCaseTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"TCAStCaseTac" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(TCAStCase, TCAStCaseTac) shouldBe 'proved
}

val specificCase = ("\\forall t ((t < max((0,(vlo-v0)/alo)) & t < max((0,(viup-vi0)/aiup))) |" +
  " (t < max((0,(vlo-v0)/alo)) & t >= max((0,(viup-vi0)/aiup))) |" +
  " (t >= max((0,(vlo-v0)/alo)) & t < max((0,(viup-vi0)/aiup))) |" +
  " (t >= max((0,(vlo-v0)/alo)) & t >= max((0,(viup-vi0)/aiup))))").asFormula
def specificCaseTac(implicit qeTool:QETool) =
  cut(TCAStCase) <(
      allL("max((0,(vlo-v0)/alo))".asTerm)(-1) &
      allL("max((0,(viup-vi0)/aiup))".asTerm)(-1) &
      closeId
    , cohide('Rlast) & TCAStCaseTac
  )

"specific case" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(specificCase, specificCaseTac) shouldBe 'proved
}

val leqTrans = "\\forall a \\forall b \\forall c ((a <= b & b <= c) -> a <= c)".asFormula
def leqTransTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"transitivity of <=" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(leqTrans, leqTransTac) shouldBe 'proved
}

def trans(intermediate:Term)(implicit qETool: QETool):DependentPositionTactic =
  "trans" by ((pos, sequent) => sequent(pos.checkTop) match {
    case LessEqual(left, right) =>
      cut(leqTrans) <(
        allL(left)('Llast) & allL(intermediate)('Llast) &
          allL(right)('Llast) & implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
        , cohide('Rlast) & leqTransTac
        ) partial
  } )
```

```
"transitivity example" should "be provable" in withMathematica { implicit qeTool =>
  val exampleFml = "0 <= 2".asFormula
  proveBy(exampleFml, trans("1".asTerm)(qeTool)(1) & andR(1) <(QE, QE)) shouldBe 'proved
}

val eqToLeqFml = "\\forall a \\forall b (a = b -> a <= b)".asFormula
def eqToLeqTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"= -> <=" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(eqToLeqFml, eqToLeqTac) shouldBe 'proved
}

def eqToLeq(implicit qETool: QETool):DependentPositionTactic =
  "eqToLeq" by ((pos, sequent) => sequent(pos.checkTop) match {
    case LessEqual(left, right) =>
      cut(eqToLeqFml) <(
        allL(left)('Llast) & allL(right)('Llast) & implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
        , eqToLeqTac
        ) partial
  } )

"eqToLeq example" should "be provable" in withMathematica { implicit qeTool =>
  val exampleFml = "0 <= 0".asFormula
  proveBy(exampleFml, eqToLeq(qeTool)(1) & QE) shouldBe 'proved
}

val reflFml = "\\forall x (x = x)".asFormula
def reflTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"refl" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(reflFml, reflTac) shouldBe 'proved
}

def refl(implicit qETool: QETool):DependentPositionTactic =
  "refl" by ((pos, sequent) => sequent(pos.checkTop) match {
    case Equal(left, right) =>
      assert(left.equals(right))
      cut(reflFml) <(
        allL(left)('Llast) & closeId
        , reflTac
        )
  } )

"refl tactic" should "prove fancy terms" in withMathematica { implicit qeTool =>
  val term = "x^y/z^w/a^b/c^(d/e)^(-1.15*2^j)".asTerm
  proveBy(Equal(term,term), refl(qeTool)(1)) shouldBe 'proved
}

def bigAnd(tax:List[BelleExpr]):BelleExpr = {
  tax.foldRight(nil)((be:BelleExpr, acc:BelleExpr) => be & acc)
}

def multiCohideL(fmls:List[Formula]):DependentTactic = {
  "multiCohideL" by (sequent => {
    val tacs =
      sequent.ante.toList.flatMap({case fml =>
        if (fmls.contains(fml)) {
          Nil
```

```
        } else {
          List(hide(Find.FindL(0, Some(fml))))
        }})
    bigAnd(tacs)
   })
}

"multiCohideL" should "hide just the right stuff" in withMathematica { implicit qeTool =>
  val example = "(x>y^2 & y>z & a >z^5 & b > z-(x^(1/2)) & w > 0) -> y + w > z + 0".asFormula
  proveBy(example, implyR(1) & andL('Llast) *@ TheType() &
    multiCohideL(List("y>z".asFormula, "w > 0".asFormula)) & QE) shouldBe 'proved
}

/* Search for equalities in antecedent that set each of the given variables and rewrite them in the succedent
 * Makes theorem proving great again */
def rewriteEqsOver(vars:List[Term]): DependentPositionTactic = {
  "rewriteEqsOver" by ((pos, sequent) =>   {
      val applicableFmls = sequent.ante.toList.zipWithIndex.flatMap({case (fml, i) =>
      fml match {
        case Equal(l, r) =>
          if (vars.contains(l)) {
            List(i)
          } else {
            Nil
          }
        case _ => Nil
      }})
      bigAnd(applicableFmls.map({case fml => eqL2R(AntePosition(fml+1))(pos.checkTop)}))})}

val leqLoosenDiffFml = "\\forall a \\forall b \\forall c \\forall d ((a <= c & b >= d) -> a - b <= c - d)".asFormula
def leqLoosenDiffTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"leqLoosenDiffFml" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(leqLoosenDiffFml, leqLoosenDiffTac) shouldBe 'proved
}

def leqLoosenDiff(implicit qeTool:QETool):DependentPositionTactic = {
  "leqLoosenDiff" by ((pos, sequent) => sequent(pos.checkTop) match {
    case LessEqual(Minus(a,b), Minus(c,d)) =>
    cut(leqLoosenDiffFml) <(
      allL(a)('Llast) & allL(b)('Llast) & allL(c)('Llast) & allL(d)('Llast) &
        implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
      , cohide('Rlast) & leqLoosenDiffTac
      ) partial
  })
}

"leqLoosenDiffTac" should "loosen up" in withMathematica { implicit qeTool =>
  val example = "5 - 4 <= 6 - 3".asFormula
  proveBy(example, leqLoosenDiff(qeTool)(1) & andR(1) <(QE, QE)) shouldBe 'proved
}

val leqSumLeftConstFml = "\\forall a \\forall b \\forall c ((b <= c) -> a + b <= a + c)".asFormula
def leqSumLeftConstTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"leqSumLeftConst" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(leqSumLeftConstFml, leqSumLeftConstTac) shouldBe 'proved
}
```

```
def leqSumLeftConst(implicit qeTool:QETool):DependentPositionTactic = {
  "leqSumLeftConst" by ((pos, sequent) => sequent(pos.checkTop) match {
    case LessEqual(Plus(a,b), Plus(c,d)) =>
      assert(a == c)
      cut(leqSumLeftConstFml) <(
        allL(a)('Llast) & allL(b)('Llast) & allL(d)('Llast) &
          implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
        , cohide('Rlast) & leqSumLeftConstTac
        ) partial
  })
}

"leqSumLeftConst" should "prove stuff" in withMathematica { implicit qeTool =>
  val example = "x + y <= x +  (y + 25)".asFormula
  proveBy(example, leqSumLeftConst(qeTool)(1) & QE) shouldBe 'proved
}

val leqSumPiecewiseFml = "\\forall a \\forall b \\forall c \\forall d ((a <= c & b <= d) -> a + b <= c + d)".asFormula
def leqSumPiecewiseTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"leqSumPiecewiseFml" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(leqSumPiecewiseFml, leqSumPiecewiseTac) shouldBe 'proved
}

def leqSumPiecewise(implicit qeTool:QETool):DependentPositionTactic = {
  "leqSumPiecewise" by ((pos, sequent) => sequent(pos.checkTop) match {
    case LessEqual(Plus(a,b), Plus(c,d)) =>
      cut(leqSumPiecewiseFml) <(
        allL(a)('Llast) & allL(b)('Llast) & allL(c)('Llast) & allL(d)('Llast) &
          implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
        , cohide('Rlast) & leqSumPiecewiseTac
        ) partial
  })
}

"leqSumPiecewise" should "prove stuff" in withMathematica { implicit qeTool =>
  val example = "x + y <= (x + 1) +  (y + 2)".asFormula
  proveBy(example, leqSumPiecewise(qeTool)(1) & andR(1) <(QE, QE)) shouldBe 'proved
}

val geqSumLeftConstFml = "\\forall a \\forall b \\forall c ((b >= c) -> a + b >= a + c)".asFormula
def geqSumLeftConstTac(implicit qeTool:QETool) = (allR(1) *@ TheType()) & QE

"geqSumLeftConst" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(leqSumLeftConstFml, leqSumLeftConstTac) shouldBe 'proved
}

def geqSumLeftConst(implicit qeTool:QETool):DependentPositionTactic = {
  "geqSumLeftConst" by ((pos, sequent) => sequent(pos.checkTop) match {
    case GreaterEqual(Plus(a,b), Plus(c,d)) =>
      assert(a == c)
      cut(geqSumLeftConstFml) <(
        allL(a)('Llast) & allL(b)('Llast) & allL(d)('Llast) &
          implyL('Llast) <(hide(pos.checkTop) partial, closeId) partial
        , cohide('Rlast) & geqSumLeftConstTac
        ) partial
  })
```

```
}

"geqSumLeftConst" should "prove stuff" in withMathematica { implicit qeTool =>
  val example = "x + (y + 1) >= x +  y".asFormula
  proveBy(example, geqSumLeftConst(qeTool)(1) & QE) shouldBe 'proved
}

"trivial statements about max(x,y)" should "prove with QE" in withMathematica{ implicit qeTool =>
  proveBy("max(1,2) = 2".asFormula, QE) shouldBe 'proved
}


/* @TODO: This makes Mathematica sad - it seems we can't handle nontrivial usages of max
* val case2Lemma1Sequent =
  Sequent(Nil,
    collection.immutable.IndexedSeq(
      "t >= 0".asFormula,
      "t1 >= t".asFormula,
      "aimax >= 0".asFormula,
      "t1 = (viup - vi0)/aimax".asFormula
    ),
    collection.immutable.IndexedSeq(
      "viup*t - max(0, ((aimax*t1^2)/2)) = vi0*t1 + (aimax *t1^2/2) + ((t-t1) * viup)".asFormula
    ))

* */

val maxRightGeqFml = "\\forall x \\forall y (y >= x -> max(x,y) = y)".asFormula
def maxRightGeqTac(implicit qeTool:QETool) = QE

"maxRightGeqFml" should "be true" in withMathematica { implicit qeTool =>
  proveBy(maxRightGeqFml, maxRightGeqTac) shouldBe 'proved
}


val geqMaxGeqRightFml = "\\forall x \\forall y \\forall z (z >= max(x, y) -> z >= y)".asFormula
def geqMaxGeqRightTac(implicit qeTool:QETool) = QE

"geqMaxGeqRight" should "be true" in withMathematica { implicit qeTool =>
  proveBy(geqMaxGeqRightFml, geqMaxGeqRightTac) shouldBe 'proved
}

val geqMaxGeqLeftFml = ("\\forall x \\forall y \\forall " +
  "z (z >= max(x, y) -> z >= x)").asFormula
def geqMaxGeqLeftTac(implicit qeTool:QETool) = QE

"geqMaxGeqLeft" should "be true" in withMathematica { implicit qeTool =>
  proveBy(geqMaxGeqLeftFml, geqMaxGeqLeftTac) shouldBe 'proved
}

val maxLEMFml = "\\forall x \\forall y (max(x,y) = x | max(x,y) = y)".asFormula
def maxLEMTac(implicit qeTool:QETool) = QE

/* Lemmas needed for case 2 of big proof. Generalize these once I see what is
 needed for the other proofs. */
val case2Lemma1Fml = ("\\forall t1 ((aimax > 0 & t1 = (viup - vi0)/aimax) -> " +
  "(viup*t - max(0, ((aimax*t1^2)/2)) = (vi0*t1 + (aimax *t1^2/2)) + " +
  "(((t-t1) * viup))))").asFormula
```

```
def case2Lemma1Tac(implicit qeTool: QETool) = {
  allR(1) &
    implyR(1) &
    andL(-1) &
    cut(maxRightGeqFml) <(
      allL("0".asTerm)('Llast) &
        allL("((aimax*t1^2)/2)".asTerm)('Llast) &
        implyL('Llast) <(
          /* Show case */
          hideR(1) &
            multiCohideL(List("aimax > 0".asFormula)) &
            QE
          , /* Use case */
          /* Substitute for max and t1 then hide them */
          eqL2R(AntePosition(3))(1) &
            eqL2R(AntePosition(2))(1) &
            hideL(-2, "t1=(viup-vi0)/aimax".asFormula) &
            hideL(-2, "max((0,aimax*t1^2/2))=aimax*t1^2/2".asFormula) &
            QE)
      , cohide('Rlast) & maxRightGeqTac)
}

"lemma 1" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(case2Lemma1Fml, case2Lemma1Tac)
}

val case2Lemma2Fml = ("\\forall t1 (vi0 * t + ai * t^2/2 = (vi0 * t1 + ai*t1^2/2)" +
  " + ((vi0 + ai * t1) * (t - t1) + ai*(t-t1)^2/2))").asFormula
def case2Lemma2Tac(implicit qeTool:QETool) = {
  QE
}

"lemma 2" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(case2Lemma2Fml, case2Lemma2Tac)
}

val case2Lemma3Fml = ("\\forall t1 \\forall ai \\forall aimax (ai <= aimax -> " +
  "vi0 * t1 + ai * t1^2/2 <= vi0*t1 + aimax * t1^2/2)").asFormula
def case2Lemma3Tac(implicit qeTool:QETool) = {
  QE
}

"lemma 3" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(case2Lemma3Fml, case2Lemma3Tac)
}

val case2Lemma4Fml = ("\\forall t1 ((vi0 <= viup & vi0 + ai*t<= viup & (t - t1) " +
  ">= 0 & t1 >= 0) -> (vi0 + ai*t1)*(t-t1) + ai*(t-t1)^2/2<= (t-t1)*viup)").asFormula
def case2Lemma4Tac(implicit qeTool:QETool) = {
  QE
}

"lemma 4" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(case2Lemma4Fml, case2Lemma4Tac)
}

val case2Lemma5Fml = ("(aimax > 0 & ai <= aimax & (t - ((viup-vi0)/aimax)) >= 0  " +
  "& vi0+ai*t<=viup & vi0 <= viup) -> vi0*t+ai*t^2/2 <= " +
```

```
  "viup*t - max(0, ((aimax*((viup-vi0)/aimax)^2)/2))").asFormula
def case2Lemma5Tac(implicit qeTool: QETool) = {
  implyR(1) &
  andL('Llast) &
  andL('Llast) &
  andL('Llast) &
  andL('Llast) &
  cut(case2Lemma2Fml) <(
    allL("(viup-vi0)/aimax".asTerm)('Llast) &
    eqL2R(AntePosition(6))(1) &
    hideL(-6) &
    cut (case2Lemma1Fml) <(
      allL("(viup-vi0)/aimax".asTerm)('Llast) &
      implyL('Llast) <(
        /* Show */
        hide(1) &
        andR (1) <(closeId, refl(qeTool)(1))
        , /* Use */
        eqL2R(AntePosition(6))(1) &
        hideL(-6) &
        leqSumPiecewise(qeTool)(1) & andR(1) <(
          cut(case2Lemma3Fml) <(
            allL("(viup-vi0)/aimax".asTerm)('Llast) &
            allL("ai".asTerm)('Llast) &
            allL("aimax".asTerm)('Llast) &
            implyL('Llast) <(closeId, eqL2R(-6)(-7) & closeId)
            ,cohide('Rlast) & case2Lemma3Tac
            )
          , cut(case2Lemma4Fml) <(
              allL("(viup-vi0)/aimax".asTerm)('Llast) &
              implyL('Llast) <(
                /* Show */
                hide(1) & andR(1) <(closeId, andR(1)
                  <(closeId, andR(1) <(closeId, QE)))
                /* Use */
                , closeId
              )
            , cohide('Rlast) & case2Lemma4Tac
          )))
      ,cohide('Rlast) & case2Lemma1Tac
    )
  ,cohide('Rlast) & case2Lemma2Tac
  )
}

"lemma 5" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(case2Lemma5Fml, case2Lemma5Tac)
}

/*"a > 0 -> max(0,a*(d/a)^2/2) = max(0,d)^2/(2*a)".asFormula*/
val sillyMaxLemmaFml = "max(0, x)^2 = max(0,x^2)".asFormula
def sillyMaxLemmaTac(implicit qeTool:QETool) = QE

"sillyLemma" should "be provable" in withMathematica { implicit qeTool =>
  proveBy(sillyMaxLemmaFml, sillyMaxLemmaTac)
}

val sequentAfterWeakening =
```

```
  Sequent(Nil,
    collection.immutable.IndexedSeq(
      "rp>=0".asFormula,
      "hp>0".asFormula,
      "vr>=0".asFormula,
      "vlo<=vup".asFormula,
      "amin<=aa".asFormula,
      "aa<0".asFormula,
      "0<ab".asFormula,
      "ab<=amax".asFormula,
      "vilo<=viup".asFormula,
      "aimin<=aia".asFormula,
      "aia<0".asFormula,
      "0<aib".asFormula,
      "aib<=aimax".asFormula,
      ("((\\exists alo \\exists aiup (    ((alo  = amin  & v0  >= vlo)  |" +
        " (alo  = ab  & v0 <= vlo))  & ((aiup = aimax & vi0 <= viup) |" +
        " (aiup = aia & vi0 >= viup))  & (\\forall time \\forall hrel " +
        "((0 <= time &((time <  max(0, (vlo-v0)/alo) & time <  max(0, (viup-vi0)/aiup) " +
        "& hrel = hi0 + (aiup/2*time^2 + vi0*time) - (h0 + (alo/2*time^2 + v0*time)))    " +
        "| (time <  max(0, (vlo-v0)/alo) & time >= max(0, (viup-vi0)/aiup) " +
        "& hrel = (hi0 + (viup*time - (max(0, viup - vi0)^2/(2*aiup)))) - (h0 + (v0*time + alo/2*time^2)))" +
        "| (time >= max(0, (vlo-v0)/alo) & time < max(0, (viup-vi0)/aiup)" +
        "& hrel = hi0 + aiup/2*time^2 + vi0*time - (h0 + alo/2*time^2 + v0*time))" +
        "| (time >= max(0, (vlo-v0)/alo) & time >= max(0, (viup-vi0)/aiup)" +
        "& hrel = hi0 + aiup/2*time^2 + vi0*time - (h0 + (max(0,(viup-vi0))/(2*aiup))^2) - alo/2*time^2 + v0*time)))" +
        "    -> hrel <= -hp))))  " +
        "|(\\exists aup \\exists ailo (((ailo = aimin & v0 >= vilo)" +
        "|(ailo = aib  & vi0 <= vilo))  & ((aup = amax   & v0 <= vup)" +
        "|(aiup = aa   & vi0 >= viup))  & " +
        "(\\forall time \\forall hrel ((0 <= time &((time <  max(0, (vup-v0)/aup) & time <  max(0, (viup-vi0)/ailo) " +
        "  & hrel = hi0 + (ailo/2*time^2 + vi0*time) - (h0 + aup/2*time^2 + v0*time))" +
        "| (time <  max(0, (vup-v0)/aup) & time >=  max(0, (viup-vi0)/ailo)     " +
        "& hrel = hi0 + (max(viup, vi0)/ailo)*time - (h0 + (max(0,(viup-vi0))/(2*ailo))^2 -(aup/2*time^2 + v0*time)))" +
        "|  (time >=  (max(0, (vup-v0)/aup)) & time <  max(0, (viup-vi0)/ailo)     " +
        "& hrel = hi0 + (ailo/2*time^2 + vi0*time) - (h0 + (max(vup, v0)/aup)*time - (max(0,(vup-v0))/(2*aup))^2))    " +
        "| (time >=  max(0, (vup-v0)/aup) & time >=  max(0, (viup-vi0)/ailo)     " +
        "& hrel = hi0 + (max(viup, vi0)/ailo)*time - (h0 + (max(0,(viup-vi0))/(2*ailo))^2 - (max(vup, v0)/aup)*time - " +
        "(max(0,(vup-v0))/(2*aup))^2))))-> hrel >= hp)))))").asFormula,
      ("(((((a>=amin&a<=amax)&(v>=vlo|a>=ab)&(v<=vup|a<=aa)&(ai>=aimin&ai<=aimax)&(vi>=vilo|ai>=aib)" +
        "&(vi<=viup|ai<=aia))&v=v0+a*t)&vi=vi0+ai*t)&h=h0+(v0*t+a*t^2/2))&hi=hi0+(vi0*t+ai*t^2/2)").asFormula,
      "t>=0".asFormula),
    collection.immutable.IndexedSeq(
      "abs(hi-h)>=hp".asFormula
    ))

def tacticAfterWeakening(implicit qeTool: QETool):BelleExpr = {
  /* Use case */
  debug("DiffCut Use case") &
    andL(-15) &
    orL(-14) <((
      existsL(-14) &
        existsL('Llast) &
        andL('Llast) *@ TheType() &
        //implyR(1) &
        debug("Or else what") &
        andL(-15) &
        andL(-19) &
```

```
andL(-20) &
andL(-21) &
andL(-22) &
andL(-22) &
(andL('Llast) *@ TheType()) &
orL(-16, "alo=amin&v0>=vlo|alo=ab&v0<=vlo".asFormula) <((
  debug("Ownship above lower bound ") &
    orL(-17, "aiup=aimax&vi0<=viup|aiup=aia&vi0>=viup".asFormula) <((
      debug("OA Intruder below upper bound") &
        andL(-16, "alo=amin&v0>=vlo".asFormula) &
        andL(-16, "aiup=aimax&vi0<=viup".asFormula) &
        /* Turn absolute value into negative */
        implyGoalWith(abs_below_fml, abs_below_tac)(qeTool)(1) &
        /* Case on possible time values*/
        cut(specificCase) <(
          /* Use case*/
          allL("t".asTerm)('Llast) &
          /* Consider all cases for time */
          (orL('Llast) <(
            /* case 1 */
            /* Instantiate forall for the safe region */
            allL("t".asTerm)(-16) &
            allL("((hi0 + (aiup/2*t^2 + vi0*t))) - (h0 + (alo/2*t^2 + v0*t))".asTerm)(-16) &
            implyL(-16) <(
              /* Show assumption */
              hide(1) & /* hide old goal */
              andR(1) <(/* time*/
                cohide2(-14,1) & QE,
                /* big disjunct*/
                orR('Rlast) & hideR('Rlast) &
                andR('Rlast) <(
                  /* time bound 1 */
                  andL('Llast) & closeId,
                  /* bound 2 and trajectory */
                  andR('Rlast) <(
                   /*bound 2*/
                   andL('Llast) & closeId,
                   /* trajectory */
                   refl(qeTool)(1)))),
              /* Use conclusion */
              trans("(hi0+(aiup/2*t^2+vi0*t))-(h0+(alo/2*t^2+v0*t))".asTerm)(qeTool)(1) &
              andR(1) <(
                /* Munge arithmetic so QE don't choke */
                /* Rewrite h and hi, then aiup and alo */
                rewriteEqsOver(List("h".asTerm, "hi".asTerm, "aiup".asTerm, "alo".asTerm))(1) &
                multiCohideL (List("t>=0".asFormula, "a >= amin".asFormula, "ai>=aimin&ai<=aimax".asFormula)) &
                QE,
                /* triv */
                closeId
              )
            ),
           orL('Llast) <(
             /* case 2 */
             /* Instantiate forall for the safe region */
             allL("t".asTerm)(-16) &
             allL("(hi0 + (viup*t - (max(0, viup-vi0)^2/(2*aiup)))) - (h0 + (v0*t + alo/2*t^2))".asTerm)(-16) &
             implyL(-16) <(
               /* Show assumption */
```

```
hide(1) & /* hide old goal */
  andR(1) <(/* time*/
    cohide2(-14,1) & QE,
    /* big disjunct*/
    orR('Rlast) & hideR(1) & orR('Rlast) & hideR('Rlast) &
      andR('Rlast) <(
        /* time bound 1 */
        andL('Llast) & closeId,
        /* bound 2 and trajectory */
        andR('Rlast) <(
          /*bound 2*/
          andL('Llast) & closeId,
          /* trajectory */
          refl(qeTool)(1)))),
/* Use conclusion */
trans("(hi0 + (viup*t - (max(0, viup - vi0)^2/(2*aiup)))) - (h0 + (v0*t + alo/2*t^2))"
  .asTerm)(qeTool)(1) &
  andR(1) <(
    /* Munge arithmetic so QE don't choke */
    /* Rewrite h and hi, then aiup and alo */
    rewriteEqsOver(List("h".asTerm, "hi".asTerm, "aiup".asTerm, "alo".asTerm))(1) &
    leqLoosenDiff(qeTool)(1) & andR(1) <(
      leqSumLeftConst(qeTool)(1) &
        multiCohideL (List("t>=0".asFormula, "a >= amin".asFormula,
          "ai>=aimin&ai<=aimax".asFormula, "v0>=vlo".asFormula, "vi0<=viup".asFormula,
        "t < max((0,(vlo-v0)/alo))&t>=max((0,(viup-vi0)/aiup))".asFormula,
        "ai>=aimin&ai<=aimax".asFormula, "0 < aib".asFormula, "aib <= aimax".asFormula,
          "vi0<=viup".asFormula, "vi<=viup|ai<=aia".asFormula, "aiup=aimax".asFormula,
          "alo=amin".asFormula, "vi=vi0+ai*t".asFormula, "aia < 0".asFormula)) &
        /* (aimax > 0 &
           ai <= aimax &
           (t - ((viup-vi0)/aimax)) >= 0 &
           vi0+ai*t<=viup
           & vi0 <= viup)
           -> vi0*t+ai*t^2/2 <= viup*t - max(0, ((aimax*((viup-vi0)/aimax)^2)/2))*/
      cut(case2Lemma5Fml) <(
        implyL('Llast) <(
          /* Show */
          andR('Rlast) <(
          /* aimax > 0 */
          multiCohideL(List("0 < aib".asFormula, "aib <= aimax".asFormula)) & hide(1) & QE
          , andR('Rlast) <(
          /* ai <= aimax */
            andL(-7, "ai>=aimin&ai<=aimax".asFormula) & closeId
          , andR('Rlast) <(
            /* t - ((viup-vi0)/aimax) >= 0 */
            andL(-13, "t < max((0,(vlo-v0)/alo))&t>=max((0,(viup-vi0)/aiup))".asFormula) &
            rewriteEqsOver(List("aiup".asTerm))(-14) &
            multiCohideL(List("t>=max((0,(viup-vi0)/aimax))".asFormula,
              "0 < aib".asFormula, "aib <= aimax".asFormula)) &
            implyWithAssumption("t>=max((0,(viup-vi0)/aimax)) -> t>=(viup-vi0)/aimax".asFormula,
              cut (geqMaxGeqRightFml) <(
                allL("0".asTerm)('Llast) &
                allL("(viup-vi0)/aimax".asTerm)('Llast) &
                allL("t".asTerm)('Llast) &
                closeId
                ,cohide('Rlast) & geqMaxGeqRightTac))(qeTool)(-3) &
             hide(-3) & hide(1) & QE
```

```
, hide(1) & andR(1) <(
  /* vi0 + ai*t <= viup*/
  multiCohideL(List("vi<=viup|ai<=aia".asFormula,
    "0 < aib".asFormula, "aib<=aimax".asFormula,
    "aiup=aimax".asFormula, "aia < 0".asFormula, "vi=vi0+ai*t".asFormula,
    "t < max((0,(vlo-v0)/alo))&t>=max((0,(viup-vi0)/aiup))".asFormula,
    "vi0<=viup".asFormula)) &
  andL(-8) &
  hide(-8) &
  implyWithAssumption("t>=max((0,(viup-vi0)/aiup)) -> t>=0".asFormula,
      cut (geqMaxGeqLeftFml) <(
     allL("0".asTerm)('Llast) &
     allL("(viup-vi0)/aiup".asTerm)('Llast) &
     allL("t".asTerm)('Llast) &
     closeId
   ,cohide('Rlast) & geqMaxGeqLeftTac))(qeTool)(-8) &
  hide(-8) &
   QE
,/* vi0 <= viup */
  closeId))))
/* Use*/
/* @TODO annoying goal */
, cut("max((0,aimax*((viup-vi0)/aimax)^2/2)) = max((0,viup-vi0))^2/(2*aimax)".
asFormula) <(
   /* Use */
   /*andL(1) &*/
   eqR2L(AntePosition(15))(1) &  closeId
   , /* Show */
   cut(maxLEMFml) <(
     allL("0".asTerm)('Llast) &
     allL("aimax*((viup-vi0)/aimax)^2/2".asTerm)('Llast) &
     cut(maxLEMFml) <(
         hide(1) &
         allL("0".asTerm)('Llast) &
         allL("viup-vi0".asTerm)('Llast) &
         /* @TODO: I think this approach is utterly broken.*/
         andL(1) &
         orL('Llast) <(
           eqL2R(AntePosition(16))(1) &
           hide(-16) &
           orL ('Llast) <(
             eqL2R(AntePosition(15))(1) &
               hide(15) &
               multiCohideL(List("0 < aib".asFormula, "aib <= aimax".asFormula)) &
               QE
             , eqL2R(AntePosition(15))(1) &
             hide(15) &
             andL(1) &
             multiCohideL(List("0 < aib".asFormula, "aib <= aimax".asFormula)) &
             QE
             )
           , eqL2R(AntePosition(16))(1) &
           hide(-16) &
           orL ('Llast) <(
             eqL2R(AntePosition(15))(1) &
               hide(15) &
               multiCohideL(List("0 < aib".asFormula, "aib <= aimax".asFormula)) &
               QE
```

```
                                        , eqL2R(AntePosition(15))(1) &
                                        hide(15) &
                                        multiCohideL(List("0 < aib".asFormula, "aib <= aimax".asFormula)) &
                                        QE
                                        )
                                      )

                                      /*orL('Llast) &*/

                                    , maxLEMTac)
                                  , maxLEMTac))
                                )partial
                              , cohide('Rlast) & case2Lemma5Tac) &
                            /* @TODO annoying goal */
                            andL(1) &
                             QE
                            ,geqSumLeftConst(qeTool)(1) & multiCohideL(List("a >= amin".asFormula)) & QE
                            ) partial
                          ,
                          /* triv */
                          closeId
                       ) partial
                      ) partial,
                   orL('Llast) <(
                    /* case 3 */
                    nil partial,
                    /* case 4 */
                    nil partial
                    )partial) partial) partial) partial,
                  /* Show case */
                  cohide('Rlast) & specificCaseTac
                  ) partial
              ) partial,
              debug("OA Intruder above upper bound") partial
              ) partial ) partial, (
            debug("Ownship below lower bound") &
              orL(-17, "aiup=aimax&vi0<=viup|aiup=aia&vi0>=viup".asFormula) <(
                debug("OB Intruder below upper bound") partial,
                debug("OB Intruder above upper bound") partial
                ) partial ) partial
            ) &
          debug("Ownship safely above intruder") partial
        ) partial,
        debug("Intruder safely above ownship") partial
        ) partial
}
"Weakened part of simulateous manevuers proof" should "be provable" in
  withMathematica { implicit qeTool =>
    proveBy(sequentAfterWeakening, tacticAfterWeakening)
}

  "Simultaneous maneuvers" should "be provable" in withMathematica
    { implicit qeTool =>
  val s = parseToSequent(getClass.getResourceAsStream
    ("/examples/casestudies/acasx/simultaneous-maneuvers.kyx"))

  val tactic =
    implyR('_) &
```

```
  andL('_) *@ TheType() &
  //chase(1) &
  noRandos &
  equateAssigns &
  composeb(1) &
  testb(1) &
  implyR('R) &
  noRandos &
  allR('R) *@ TheType() &
  andL(-9) &
  andL(-10) &
  andL(-11) &
  andL(-12) &
  andL(-13) &
  andL(-14) &
  andL(-15) &
  andL(-16) &
  andL(-17) &
  andL(-18) &
  diffCut("v = v0 + a * t".asFormula)(1) <(
  diffCut("vi = vi0 + ai * t".asFormula)(1) <(
  diffCut("h = h0 + (v0 * t + a * t^2/2)".asFormula)(1) <(
  diffCut("hi = hi0 + (vi0 * t + ai * t^2/2)".asFormula)(1) <(
  diffCut("t >= 0".asFormula)(1) <(
  diffWeaken(1) & postWeaken &
  (tacticAfterWeakening partial)
  ,
    /* Show case for t >= 0 */
    //debug("show case for h ") &
    hideL(-5) *@ TheType() &  hideL(-1) & hideL(-1) & hideL(-1) &
      //debug("solving for height") &
      diffInd(qeTool)(1)
    ) partial,
    /* Show case for hi = hi0 + vi0 * t + ai * t^2/2 */
      //debug("show case for hi ") &
      hideL(-9) *@ TheType() &  hideL(-1) & hideL(-1) & hideL(-1) &
      //debug("solving for intruder height") &
      diffInd(qeTool)(1)
    ) partial,
    /* Show case for h = h0 + v0 * t + a * t^2/2 */
      //debug("show case for h ") &
      hideL(-8) *@ TheType() &  hideL(-1) & hideL(-1) & hideL(-1) &
      //debug("solving for height") &
      diffInd(qeTool)(1)
    ) partial,
    /* Show case for vi = vi0 + ai * t */
      //debug("show case for vi") &
      hideL(-7) *@ TheType() &  hideL(-1) & hideL(-1) & hideL(-1) &
      diffInd(qeTool)(1)
    ) partial,
      /* Show case for v = v0 + a * t */
      //debug("show case for v ") &
      hideL(-6) *@ TheType() & hideL(-1) & hideL(-1) & hideL(-1) &
      diffInd(qeTool)(1)
    )&
  debug("Down to the ODE")

proveBy(s, tactic)
```

```
  }
}
```