Incremental Development of a Protocol for Free Flight

Zhuang Li[1,2], HuaiKou Miao"[2],Yang Liu[1,2]

' School of Computer Engineering & Science, Shanghai University,Shanghai, China
[2] Shanghai Key Laboratory of Computer Software Evaluating & Testing, 201112 Shanghai, China
{lizhuang, hluniao, yangl_shul@shu.edu.cn

Abstract. This paper gives a formal description of the free flight in Z notation, refines an operation schema in Z to loop style in refinement calculus, proposes a new concept refinement premise, gives the proof rules, at last get the protocol which safety and liveness property hold for free flight under some assumptions, and presents the proof of the correctness. The contributions of this paper are formal development of a protocol for free flight and proposal of enforced premise to complement the traditional refinement calculus.

Keywords: free flight, Z, refinement calculus, enforced premise, loop refinement.

1 Introduction

Free flight is a developing air traffic control method that uses no centralized control (e.g. air traffic controllers). Instead, parts of airspace are reserved dynamically and automatically in a distributed way using computer communication to ensure the required separation between aircraft. The Conflict Detection and Resolution (CD&R) problems in free flight[1] have been studied by various research groups in last several years. Algorithms for avoiding collusion have been developed, for example in [2], [3]. In these papers the methods of simulation or model checking was used to verify correctness for an arbitrary number of airplanes. In this paper, we propose a formal approach to developing a protocol in a refinement way with the combination of Z notation[4] and Refinement calculus[5]. Refinement is a formal top-town style development methodology with rigorous correctness preservation.

We develop a protocol for avoiding multi-aircraft collusion which can't be reduced into pair-wise situation[6]. We consider the problem of conflict resolution in the horizontal plane, using only information about each aircraft's current position, velocity, and heading. Two goals is considered, one is the aircraft doesn't conflict, the other is they get their destinations.

2 Original Formal Specification

We begin by introducing the abstract types *AIRCRAFT* and *POSITION*.

$$[AIRPLANE, POSITION]$$

In the following axiomatic schema, *orig* is a function from the airplane to its original location(departure airport). *dest*, is a function from the airplane to its destination location(landing airport). *A* represents the set of all airplanes in our problem domain. *d* is a function from two locations to the distance between them. *rdest* represents the radius of the radar's coverage in the destination airport.

$$orig : AIRCRAFT \rightarrow POSITION$$
$$dest : AIRCRAFT \rightarrow POSITION$$
$$A : \mathbb{P}\, AIRCRAFT$$
$$d : POSITION \times POSITION \rightarrow \mathbb{N}$$
$$rdest : \mathbb{N}$$
$$\text{dom } orig = \text{dom } dest = A$$

Airspace

$$\top pos : AIRCRAFT \rightarrow POSITION$$

Airspace'

$$\forall a : A \bullet pas' a = orig\ a$$

Fly

AAirspace

$$\forall a : A \bullet pos\ a = orig\ a = d((pos'\ a), (dest\ a)) < rdest$$

Airspace is the state schema. Schema *Init* shows all airplanes are located at their origins initially. In schema *Fly*, each airplane in *A* takes off from the *orig(a)*, and at the end, each airplane gets its destination airport. *d((pos'a),(desta)) < rdest* represents airplane *a* gets its destination airport, because when an airplane is in the coverage area of the radars in its destination airport, we believe airplane lands safe under the control from air traffic control system command center.

3 Enforced Premise and Loop Refinement Theory

The Refinement theory foundations can be found in [5], and we only introduce our contribution of enforced premise and its refinement theory in this section. Here we refine an operation to a loop style in refinement calculus in the combination of Z and refinement calculus[7].

Lemma 1 For two predicate transformers, *A* and *A'* , predicate *q* on domain X , ACA'L\Vq:IP *: X • A.q < •g•[8]*

Definition 1 *tr(A):* trace of loop *A*

We depict the states in a loop and describe its property, for observable variables set *V* to denote(v_i, v_2, ..., v„). In loop, the wp(weakest precondition) semantic depends on the pre predicate and post predicate. We add internal evolution of the observable variables to be a trace, a loop *A* has the trace (V_1, V_2, ..., *V„),* in which V_I denotes the state of *V* before loop, V, denotes the state after loop. And V_2, ..., V_n — ldenote the internal states. We denote the set of (V1, V2, ..., V_n) as *tr(A).*

Definition 2 enforced premise ep

For two loops *A= do g —> S od* and *A' = do* 9_1 —> ...09n *od,* where *A = g S, A, = 9, -+ S*$_i$. g_1 n n g_n = *0,* g_1 ∪ ∪ g_n = *g* and consecutive execution of *A,* is a refinement of *A* we call the property the enforced premise. i.e. if 9_1 n n A_n = *0,* and 9_1 U U g_n = *g* and A E *do* A_i *od* hold then the enforced premise between *A* and *A'* holds. And when 91 n n g_n = 0, then we call that A1, A_n is excluded. A E *do A, od* holds. We can get

A E *do* A_i *od*
A.q < (do A_i *od).q*
3 m E N • *([9]; S).q < (([*g_i*]; S*$_i$*)m;* formulae (1)

Enforced premise makes *tr(A')* has a relation with *tr(A)* that we can select the states with keeping the sequence to be a subset of *tr(A).* The meaning of the enforced premise is that if *ep* is satisfied, *AL A'* can be guaranteed. In the development process, the developer can replace each S, in *A* with S, of a small granularity that change the observable variables many times until getting the same state with *S,*.

Lemma 2 Let *g* and *q* be predicates and *S* a monotonic predicate transformer. Then *while g do S od.q = (px • (g* n *s.x)* u n *q))* [9].

Theorem 1 If there is enforced premise between *A* and *A'* then *A E A'* holds.

Proof According to Lemma 1, we need to prove V *q:PX • A.q < Ai.q*

do g S od.q
while g do S od.q
{ lemma 3 }
x • (9 n *s.x)* u(-'9n *q))*
< {the definition of eq and monotonicity of fixed points}
x • (g n *(([gi]; Si)n; [—Ti]).x)* n *(g U q))*
{exclution of *gA*$_1$, *9A*$_2$, ...,*gA*$_n$*}*
do g_i —> S_n *od*

Lemma 3: Loop correctness[5][10].

Assume that 9_1, $g_{„}$, *p,* and *q* are predicates and *S,* are monotonic statements for i = 1, n. Furthermore, assume that {r„, I w E W} is a ranked collection of predicates, then
p{1 do gi *Sod ftq* provided that

(1) p C r

(2) (V *w* e W • n_u n gi {1 Si }$r_{<}$w), for i = 1, ...n and

(3) r *(1¯ig q* where *g = gi* V ••• *gn*

Corollary 1 for *A = do* g_l S_n *od* and *A'* = *Sir •Ogn od* then *1)*{*1 do* g_i —> —> *Sn od q* provided that

(1) p C r

(2) (Vw E W • r_u, n gi Si w), for i = 1, ...n

(3) m e N • *([g]; S).q = (([gi]; Si)m; [-1gi]).q,* for i = 1, ...n

(4) r *Cq* where *g = V gnn* n *An* = 0

Proof From lemma 3, (1),(2),(4) guarantee *A 11* . From theorem 1,(4) and *gi V ... V* gn,gi n n A_n = 0 guarantees *A* E *A'*, so *pfl*

Corollary 1 is used in loops refinement, and it can be also used when we refine an operation schema into a loop. We can refine the operation schema to a simple loop *A* first which p fl *A q* can be easily proved, and then prove enforced premise exists in the simple loop *A* and the loop *A'* which we really want, we can get *A' 1)¯g.*

4 Refinement

We are now going to proceed with a refinement of our initial model. We break down the *Fly* operation schema into two kinds of small step *FreeStep* and *ConflictStep.*

We only consider the exact conflict[6],which means all aircrafts involved in a conflict come into conflict at a single point in space and time. The inexact conflict situation can be proved as the exact conflict. Dynamically coordinate is allowed, i.e. a new aircraft get involved in coordination under the assumption that distance between the possible conflict aircrafts is larger than 3 *rsafty.* When the aircraft finishes the adjusting trajectory, it acts the back phase which the effect is to bring the aircraft to original trajectory(the line segment from the origination to destination) and with heading *(orig(a), dest(a)).*

In Free Flight, safety property is that we want all the aircrafts are safe, which means the distance between arbitrary two aircrafts is larger than the separate area radius (*rsafty).*

3 a, ac : A • *dis(a, ac)* < *rsafty.* Liveness property for Free Flight is that all the aircrafts get their destination eventually, i.e. at the end of the protocol V a : *A* • *d((pos a), (dest a))* < *rdest* holds. We add them in the following refined specification.

4.1 Specification

We first introduce the added constants.

VECTOR : POSITION x POSITION
CONFLICT == {FALSE, TRUE, UNKNOWN}
vector : POSITION x *POSITION* —> *VECTOR*
rahead :N rsafty : N
computeGroup : AIRPLANE —> P *AIRPLANE*

rahead is the radius that the aircraft can look ahead. *rsafty* is the safety radius. If there are more than one aircrafts in *rsafty,* then they will collide. *computeGroup* is a function from a aircraft a to an set of aircrafts that need to readjust the trajectory with a. We don't give out the concrete definition here, the set of aircrafts is taken as the input of the input of multiple-aircraft conflict algorithm in reference [6]. *conflict(a) = TRUE* means aircraft a is free. Aircraft a is free if a is in the original trajectory and with the heading form origination to destination, and there is no aircrafts in a's looking ahead time area which will collide with a. *Conflict(a) = FALSE* means aircraft a is in readjusting trajectory or back trajectory.

Airspacel
Airspace
heading : AIRCRAFT —> VECTOR
conflict : AIRCRAFT CONFLICT
group : AIRCRAFT -4 P *AIRPLANE*
sequence : AIRCRAFT seqPOSITION

Initl
Airspace
Va:A•
pos' (a) = orig(a) A
heading' (a) = vector((orig(a), dest(a)) A
conflict'(a) = UNKNOWN A
group' a = 0 A
sequence(a) = 0

sequence(a) is used to store the planed trajectory computed by the multiple-aircraft conflict algorithm in reference [6]. And *resetSequence* re-compute the trajectory with the now group and set the result to *sequence(a). group(a)* is used to store the input to the resolution last time. *computeGroup(a) # group(a)* means aircraft a adjusts the trajectory to avoid collision with aircrafts last time, and now there is a new aircraft get involved in the collision. So the trajectory of a need to be reset by *resetSequence(a). computeGroup(a) = group(a) 0* means this time the aircrafts which a adjusts to avoid is the same as last time. *computeGroup(a) = group(a) = 0* means that a is free.

Detect
Airspacel

V a, *ac : A I (conflict(a) = UNKNOWN* A *d(pos(a),pos(ac)) < rahead* A *(d(pos(a), dest(a)) > rdest* A *computeGroup(a) group(a)) —> group'(a) = computeGroup(a)* A *resetSegence(a)* A *conflict(a) = TRUE V a, ac : A I (conflict(a) = UNKNOWN* A *d(pos(a),pos(ac)) < rahead* A *(d(pos(a), dest(a)) > rdest* A *computeGroup(a) = group(a) # 0) conflict(a) = TRUE*
V a, ac : A I (conflict(a) = UNKNOWN A *d(pos(a),pos(ac)) < rahead* A *(d(pos(a), dest(a)) > rdest* A *computeGroup(a) = group(a) = 0) conflict(a) = FALSE*

FreeStep shows aircraft a flies forward one unit with unchanged direction. *ConflictStep* shows that next position of a comes from the *seq(a)* . Both *FreeStep* and *ConflictStep* include *conflict(a) = UNKNOWN.*

FreeStep _
6,Airspacel

$\forall a{:}Al$ *conflict(a) = FALSE (pos'(a) = pos(a) + heading* * 1 A *conflict'(a) = UNKNOWN)*

— ConflictStep
Airspace 1

conflict(a) = TRUE —>
(pos' (a) = head(seg(a)), seg' (a) = tail(seq(a)) A *conflict'(a) = UNKNOWN)*

4.2 Proof

To prove specification described in section 4.1 refines the original specification in section 2.

A' = do pre Detectl; ConflictStep Detectl; ConflictStep pre Detectl; FreeStep —> Detectl; FreeStep
od

We translate the schemas in Z notation into statements in refinement calculus.

init = [true, pos(a) = orig(a)] *Fly = [pos(a) = orig(a), d(pos(a), dest(a)) < rdest]*

initl = *[true, pos' (a) = orig(a)* A *heading' (a) = vector((orig(a), dest(a))* A *conflict'(a) = UNKNOWN* A *group' a* = 0A *sequence(a)* = 0]

The proof goal is *Init; Fly* C *Initl; A' ,* So the proof obligation is

$$P \text{ fl } A' \; IN \qquad \text{formulae (2)}$$

where *p* is,

pos(a) = orig(a) A *heading(a) = vector((orig(a), dest(a))* A *conflict(a) = UNKNOWN* A *groupa = 0* A *sequence(a) = 0,*

And $_q$ i_s *d(pos(a), dest(a)) < rdest.* We can't prove (2) using Lemma 2 because the variant *d(pos'(a), dest(a)) < d(pos(a), dest(a))* doesn't hold after each execution of *Detect; ConflictStep* . We can model the simple loop *A* which the body is *conflict(a) = UNKNOWN —> d(pos'(a), des(a)) < d(pos(a), des(a))* . So *p* II *A q* holds which can be proved with lemma 3.

And there is enforce premise between *A* and *A'.*

gl = pre Detect; ConflictStep and S_i *= Detect; ConflictStep*

92 = pre Detect; FreeStep and S2 = *D ectecl; FreeStep*

g = conflict(a) = UNKNOWN and *S = d(pos'(a), des(a)) < d(pos(a), des(a))*

3 m E N • $([g]; S).q = (([g_i]; S_i)m; [-'g_i]).q$, for i = 1, ...n holds. For !h. and S_i, *m.* is the execution times of *ConflictStep* until aircraft a turns to be free and if a can't turn to be free before bypassing the destination airport, then m doesn't exist. For 92 and 82, M is 1. So according to Corollary 1, formulae (2) holds under the enforced premise that the aircraft can turn to be free before the aircraft bypass the airport.

In each step, the safety property is preserved in the algorithm of reference [6], and when the aircraft readjust, the solution can be used due to our assumption that the distance between aircraft is larger than 3 *rsafty* each time readjust. For liveness, all the aircraft get their destination as in above proof of refinement under the enforced premise.

5 Conclusion

Based on an algorithm of avoiding collision, we developed a protocol for free flight in a refinement way, which guarantees the safety and liveness under enforced premise that the airplane must turn to be free before it bypass the airport. In the process of refinement proof, we complemented traditional loop correctness proof rules with introducing enforced premise, and gave the loops refinement rules and the rules for proving a loop refines an operation.

6 Related works

Reference [2] used a simulation way, which could raise confidence in an algorithm, but couldn't provide guarantees. Reference [3] used the model checking to verify the property of the model, but it was less design-oriented, and was not in the top-down way. Reference [6] gave proof of collision algorithm which was the part of our paper, but didn't give the whole process from the aircraft taking off to landing which is this paper's work. Reference [9] also modeled the system in Z, designed in a refinement way, but it assumed the aircraft could fly nearer to its destination airport in each step. It is not practical, and that why the enforced premise come into being. And In our paper, we refine an operation to be a loop, which is easy to transform Z schema to programming language. Reference [5] and [10] gave the proof basis and proof rules for loop, and both restrict the variant decrease rigidly, we relax the restriction through enforced premise.

ACKNOWLEDGMENT. This work is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 60970007, 61073050 and 61170044, Shanghai Leading Academic Discipline Project (Project Number: J50103). Thanks to J.W.Sanders, his extensive discussions around my work and interesting explorations in operations have been very helpful to this study.

References

1 Hoekstra,J.M.: Designing for Safety: the Free Flight Air Traffic Management Concept. PhD thesis, Technische Universiteit Delft, (2001)

2 Simon, P., Bil, C.: Free flight air traffic management optimisation using discrete particle simulations. In: Harigae,M. (eds.) Asia-Pacific International Symposium on Aerospace Technology APISAT,pp.171--178.The Japan Society Aeronautical Space Sciences, Tokyo (2009)

3 Platzer,A., Clarke E.M.: Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study. In: Cavalcanti,A.and Dams,D. (eds.) FM 2009: FORMAL METHODS. LNCS, vol. 5850, pp. 547--562. Springer, Heidelberg (2009)

4 Woodcock,J., Davies,J.: Using Z: specification, refinement, and proof. Prentice-Hall, New Jersey(1996)

5 Back, R.J.R., Wright,J.V.: Refinement Calculus:A Systematic Introduction. Springer-Verlag, Heidelberg (1998) 6. Hwang, I., Kim, J., Tomlin, C.: Protocol-Based Conflict Resolution for Air Traffic Control. J. Air Traffic Control Quarterly.15(1),1--34 (2007)

7 Lindsay,G.: refinement and the Z schema calculus. J. Electronic Notes in Theoretical Computer Science.70(3),70--93(2002)

8 Back, R.J.R.: Refinement Calculus, Lattices and Higher Order Logic. TR. California Institute of Technology(1992)

9 S. Eder and G. Smith. An approach to formal verification of free-flight separation. In: Trustworthy Self-Organizing Systems (TSOS 2010), pp. 166-171.1EEE Computer Society Press, New Jersey(2010)

10 Morgan,C. The specification statement.J. ACM Transactions on Programming Languages and Systems .10(3), 403--419(1988)