Universiteit Gent
Faculteit Ingenieurswetenschappen
Formele Systeemmodellering voor Software

Project Formele Systeemmodellering voor Software
Deel 1 & deel 2

Dieter Decaestecker
Tom Naessens
Nick Van Haver

2de master Computerwetenschappen

Vrijdag 2 januari, 2015

Inhoudsopgave

Hoofdstuk 1

Ontwerp en verificatie van een index in ACL2

1.1 Voorafgaand

Vooraleer we over gaan op de bespreking van de specificaties met bijhorende concrete implementatie staan we eerst nog even stil bij de gebruikte datastructuur om deze index voor te stellen en enkele hulpfuncties.

Dit verslag dient als een tekstuele handleiding bij het eerste deel van het project. De formele specificaties, implementaties en verificaties kunnen gevonden worden in bijgevoegd bestand, `Bijlages/group3_index.lisp`.

1.1.1 Hulpfuncties

Doorheen het volledige programma maken we gebruik van twee hulpfuncties, namelijk `is-ordered` en `is-ordered-string`. Deze functies controleren respectievelijk of een lijst van atomen van klein naar groot is gesorteerd, en of een lijst van strings alfabetisch is gesorteerd.

Deze functies gaan na of de lijst uit nul of één elementen bestaat, of wanneer de lijst meer dan één element bevat gaan deze na of het eerste kleiner is dan het tweede element, en roept dan recursief zichzelf op met de "staart" van de lijst. Voor atomen wordt hier de functie `<=` gebruikt, voor `strings` gebruiken we `alphorder` als operator.

1.1.2 Datastructuur

Lisp heeft hier een handige datastructuur voor ingebouwd zitten, namelijk een "association list", of ook wel "alist" [1]. Via de ingebouwde `alistp` functie kunnen we controleren of de opgebouwde index steeds een geldige alist is [2]. Om de correcte structuur en eigenschappen van de index te controleren hebben we volgende predikaat formeel gedefinieerd:

```
(defun indexp (index)
  (cond ((not (alistp index)) nil)
        ((endp index) t)
        ((not (stringp (caar index))) nil)
        ((not (is-ordered-string (strip-cars index))) nil)
        ((not (Nat-listp (cdar index))) nil)
        ((not (is-ordered (cdar index))) nil)
        (t (indexp (cdr index)))))
```

1.2 Formele specificaties met bijhorende implementatie

Aangezien de formele specificaties rechtstreeks in ACL2 geïmplementeerd zijn volgt hier een overzicht van alle functies, met een korte beschrijving van hun werkwijze. De functies zelf kunnen in het bijgevoegde ACL2 bestand teruggevonden worden onder de vetgedrukte naamgeving.

- **`createIndex`:** Functie die een lege index aanmaakt, geeft simpelweg `nil` terug.
- **`getWords`:** Deze functie geeft, gegeven een index, met behulp van de ingebouwde `strip-cars` functie alle woorden terug uit de lijst.
- **`getPages`:** Deze functie geeft, gegeven een woord en een index, alle pagina's uit de index terug bijhorende aan het gegeven woord. De implementatie van deze functie is iets ingewikkelder, en recursief geïmplementeerd.

  Wanneer de lijst leeg is, geven we `nil` terug. Dit dient ook als basisgeval voor onze recursie. Aangezien de woorden alfabetisch zijn toegevoegd in de index kunnen we de woorden één voor één recursief opvragen en vergelijken. Wanneer we een match vinden geven we de bijhorende lijst van paginanummers terug. Wanneer het huidige woord voor het gevraagde woord komt, geven we `nil` terug, aangezien we zeker weten dat het woord later niet meer voor komt wegens het alfabetisch gesorteerd zijn van de lijst. Als geen van de voorgaande gevallen getriggerd zijn roepen we de functie recursief op met de staart van de lijst.

- **`addPage`:** Deze functie voegt, gegeven een pagina en een lijst van reeds gesorteerde paginanummers, een nieuw paginanummer toe aan deze lijst, op zo'n manier dat deze lijst nog steeds geordend is na toevoeging. Deze functie is een hulpfunctie die later gebruikt wordt in de `addEntry` functie. Voor de implementatie gaan we analoog te werk aan de vorige functie (`getPages`), met een uitzondering op teruggeven waarden.

  We gaan recursief te werk waarbij we enkele checks uitvoeren. Wanneer de pagina leeg is, of we op het eind van de lijst zijn gekomen door recursie geven we het opgegeven paginanummer terug. Als het eerste element groter is dan het opgegeven paginanummer geven we een nieuwe lijst terug, opgemaakt door het opgegeven paginanummer te concateneren met de overblijvende lijst. Wanneer het opgegeven paginanummer gelijk is aan het eerste paginanummer is dit een duplicaat, en geven we enkel de overblijvende lijst terug. In alle andere gevallen roepen we de functie recursief op met de staart van de lijst.

- **`addEntry`:** Deze functie voegt, gegeven een woord, een paginanummer en een index, het opgegeven paginanummer toe aan de lijst van paginanummers voor het opgegeven woord als dit woord reeds bestaat, en voegt anders alphabetisch het opgegeven woord samen met het opgegeven paginanummer toe aan de index. Ook deze functie is recursief geïmplementeerd en steunt op op de `addPage` functie.

  Bij het basisgeval voor deze recursie krijgen we een woord in combinatie met een paginanummer en een lege lijst (= index), waarbij we een nieuw associatief koppel van het woord met het paginummer teruggeven. Wanneer het opgegeven woord echter gelijk is aan het eerste element uit de index voegen we aan de hand van de

`addPage` functie het paginanummer toe voor dit woord, en geven we deze geüpdatete combinatie, samengevoegd met de resterende index terug. Wanneer het opgegeven woord alfabetisch gezien voor het eerste woord in de huidige index ligt geven we een nieuwe combinatie van het opgegeven woord met het opgegeven paginanummer terug, geconcateneerd met het overblijvende deel van de index. In alle andere gevallen roepen we de functie opnieuw aan, maar ditmaal met de "staart" van de index.

1.3 Beschrijving formele verificatie

De correcte werking van de bovenstaande functies dienen natuurlijk geverifieerd te worden op hun correcte werking. Daarom hebben we in ACL2 ook een aantal theorema's toegevoegd. Hier volgt een lijst van de geïmplementeerde theorema's met uitleg. De formele verificatie kan teruggevonden worden in het bestand, `Bijlages/group3_index.lisp.a2s`.

- **`createIndex-correctness`:** Dit theorema gaat na of de index, gemaakt door de `createIndex` functie een correcte index teruggeeft. Hierbij testen we of een opgegeven index voldoet aan alle voorwaarden en restricties van de datastructuur uit Sectie 1.1.2.

- **`getWords-ordered`:** Dit theorema test, gegeven een index, of de woorden teruggegeven door de functie `getWords` in alfabetische volgorde staan. Hierbij maken we gebruik van de hulpfunctie `is-ordered-string`.

- **`getWords-get-words`:** Dit theorema controleert of, gegeven een index, de `getWords` functie wel degelijke de woorden uit deze index haalt. Deze verificatie is eigenlijk overbodig aangezien de `getWords` functie uit één statement bestaat, die al door ACL2 zelf wordt geverifieerd.

- **`getPages-ordered`:** Dit theorema test of, gegeven een woord en een index, de `getPages` functie een geordende lijst van paginanummers teruggeeft. Hiervoor passen we de hulpfunctie `is-ordered` toe op de `getPages` functie.

- **`getPages-gets-pages`:** (Opgelet: Dit theorema is helemaal onderaan het bestand gedeclareerd aangezien ACL2 anders problemen gaf bij het verifiëren van andere functies.) Dit theorema verifieert dat de `getPages` functie wel degelijk de lijst van paginanummers teruggeeft die bij het opgegeven woord horen. Dit doen we aan de hand van de ingebouwde `assoc-equal` functie, die gegeven een sleutel, het bijhorende sleutel en waarde paar uit een `alist` haalt [3]. Als we hierna de ingebouwde `cdr` functie toepassen krijgen we de lijst van paginanummers geassocieerd met het opgeven woord. Aan de hand van de `equal` functie vergelijken we vervolgens de gelijkheid van deze twee lijsten.

  Aangezien ACL2 hier een verificatiefout opwierp omdat hij probeerde te generaliseren hebben we hier de hint "do not generalize" meegegeven, waarna dit theorema wel geverifieerd kon worden.

- **`addPage-ordered`:** Dit theorema test of, gegeven een paginanummer en een lijst van geordende paginanummers, de lijst geordend is na toevoeging van het opgegeven paginanummer. Ook hier maken we gebruik van de hulpfunctie `is-ordered`.

- **`addPage-adds`:** Dit theorema controleert of de `addPage` functie wel degelijk een woord toevoegt aan een lijst van paginanummers. Dit doen we door te testen of een getal dat nog geen deel uitmaakt van de lijst, wel deel uitmaakt van de lijst na het toevoegen.

- **`addEntry-adds`:** Dit theorema test of, gegeven een correcte index, een natuurlijk getal en een woord, het woord wel degelijk wordt toegevoegd als het daarvoor nog niet bestond in de index. Hiervoor eisen we de correctheid van de gegeven argumenten, met de aanvulling dat het opgegeven woord nog niet voorkomt in de woorden in de index, en gaan we aan de hand van de `member` en `getWords` functie na of het opgegeven woord wel voorkomt in de lijst van de woorden in die index.

- **`addEntry-adds-non-existing-pages-to-existing-words`:** Dit theorema is een uitbreiding van voorgaand theorema waarbij gecontroleerd wordt of een pagina die nog geen deel uitmaakt van de pagina's bij een opgegeven woord na toevoeging van een nieuwe entry via de `addEntry` functie wel deel uitmaakt van die paginanummerlijst. Hiervoor breidden we het `addEntry-adds` theorema uit met de extra restrictie dat de index het woord reeds bevat en met de postcondities dat het nieuw toegevoegde paginanummer nu wel deel uitmaakt van de paginanummers bij het opgegeven woord, en dat deze lijst bovendien één element meer bevat dan voorheen.

- **`addEntry-doesnt-add-existing-pages-to-existing-words`:** Dit theorema is een kleine variatie op het voorgaande theorema. Hier eisen we dat het betreffende paginanummer wel reeds deel uitmaakt van de lijst van paginanummers bij het betreffende woord. Als resultaat controleren we of de originele lijst van paginanummers nog steeds `equal` is aan de lijst van paginanummes na het oproepen van de `addEntry` functie.

- **`addEntry-still-index`:** Dit eenvoudig theorema test of een index nog steeds een correcte index is na toevoeging van een nieuwe entry door te verifiëren dat de index nog steeds voldoet aan de datastructuur gedefinieerd in Sectie 1.1.2.

  Net zoals bij `getPages-gets-pages` wierp ACL2 hier een verificatiefout op aangezien hij probeerde te generaliseren. Hier hebben we dan ook de hint "do not generalize" aan meegegeven, waarna dit theorema wel geverifieerd kon worden.

1.4 Bijlages

- **`Bijlages/group3_index.lisp`:** Formele specificatie-, implementatie- en verificatiefuncties en -theorema's van de index

- **`Bijlages/group3_index.lisp.a2s`:** Volledige `a2s` output van de ACL2 sessie bij het evalueren van alle functies en theorema's

Hoofdstuk 2

Specificatie en verificatie in TLA+/TLC

2.1 Voorafgaand

Ook dit deel van het verslag dient als een tekstuele handleiding bij het tweede deel van het project. De formele specificaties en verificaties kunnen gevonden worden in bijgevoegd bestand, `Bijlages/Luchthaven.tla`. Om de constanten te definiëren leveren we ook een configuratiebestand mee, `Bijlages/Luchthaven.cfg`.

Vooraleer we over gaan op de bespreking van alle acties met bijhorende verificatie lichten we even ons extra assumpties, het gebruikte luchthavenmodel, de onderverdeling in zones en de gebruikte variabelen toe.

2.1.1 Assumpties

Een veiligheidsassumptie die we toevoegen is dat het deel tussen de taxibaan en het platform maar door één vliegtuig kan gebruikt worden, op eender welk moment. Voor het platform zelf geldt dezelfde aanname. Pas wanneer een vliegtuig in een gate is of op het vertrekkend gedeelte van de taxibaan is aangekomen, wordt het platform vrijgegeven.

2.1.2 Beschrijving Formele specificatie

Ons vliegveld kunnen we in het algemeen voorstellen als een wachtlijnsysteem, met enkele buffers. De "invoer" en de "uitvoer" van dit systeem verlopen langs één "kanaal" dat niet tezelfdertijd kan worden gebruikt voor meerdere invoer- en uitvoeroperaties, noch voor de combinatie van één invoer en één uitvoer op eenzelfde moment.

Zones

Meer concreet bestaat ons vliegveld uit de onderstaande onderdelen. Voor de naamgeving gebruiken we dezelfde als in het bijgevoegde bestand. Deze komen allemaal behalve de `taxiway` overeen met het schema uit de opgave.

- **`runway`:** De `runway` is de landings-/startbaan van het vliegveld. Deze kan maar door één vliegtuig tegelijk gebruikt worden, voor of het opstijgen, of het landen van het betreffende vliegtuig.

- **`taxiwayIn`:** Na het landen schuiven vliegtuigen door naar de `taxiwayIn`. Dit is het deel van de taxibaan aan de rechterkant van het schema. Dit gedeelte bestaat uit enkele plaatsen in een soort van aanschuifsysteem om het `platform` te betreden. Voor de rest van het verslag zullen we dit het inkomende gedeelte van de taxibaan noemen, of kortweg ingaande taxibaan.
- **`platform`:** Het platform vormt de verbinding tussen de `gates` en de `taxiway`. Net als de `runway` is dit een zone waar maar één vliegtuig is toegelaten op één moment. Vliegtuigen die zich begeven naar of weg van de gates gebruiken deze zone.
- **`gates`:** Vliegtuigen wachten in de `gates` tot ze terug mogen opstijgen. Dit wordt ook als een bufferzone aanzien.
- **`taxiwayOut`:** Vliegtuigen die zich in de gates bevinden en willen opstijgen begeven zich terug via het `platform` naar de `taxiwayOut`. Ook dit is een buffer waar een aantal vliegtuigen kunnen wachten alvorens toegelaten te worden tot de `runway`. Voor de rest van het verslag zullen we dit het uitgaande gedeelte van de taxibaan noemen, of kortweg uitgaande taxibaan.

Modellering één-vliegtuig-zones

De zones waar maar één vliegtuig is toegelaten op één moment worden gemodelleerd aan de hand van een status-variabele. Deze kan drie waarden aannemen: `out` voor vliegtuigen op hun weg naar de `taxiwayOut` of opstijgende vliegtuigen, `in` voor vliegtuigen die landen op de `runway` of zich naar de gates begeven en `free` voor zones waar zich geen vliegtuigen op bevinden.

Modellering bufferzones

De bufferzones hierboven besproken (`taxiwayIn`, `gates` en `taxiwayOut`) modelleren we niet volledig. We houden niet bij welk vliegtuig zich waar in de buffer bevindt, maar enkel de aantallen vliegtuigen in elke buffer. Hierbij gebruiken we respectievelijk de variabelen `taxiwayInFreePlaces`, `gatesInUse` en `taxiwayOutFreePlaces`.

Modellering van vliegtuigen die willen landen

Als laatste hebben we nog een manier nodig om vliegtuigen die willen landen te modelleren. Dit doen we aan de hand van de `land` variabele die de waarde `1` aan neemt als er een vliegtuig wil landen, en `0` wanneer dit niet het geval is.

2.2 Formele specificatie

2.2.1 Constanten

Om luchthavens van allerlei groottes te kunnen voorstellen bieden we enkele constanten aan die in het configuratiebestand kunnen gespecificeerd worden. Deze variabelen zijn `AmountOfGates`, `TaxiInSize` en `TaxiOutSize` en stellen respectievelijk het aantal gates, het aantal vliegtuigen dat de ingaande taxibaan kan bevatten en het aantal vliegtuigen dat de uitgaande taxibaan kan bevatten voor.

2.2.2 Acties

- **`NewPlane`:** Dit is een triviale actie die aangeeft of er een vliegtuig is dat wil landen. Dit gebeurt door de `land` variable op waarde `1` te zetten.

- **`Arrival`:** De `arrival` actie laat een vliegtuig landen, op behoud van enkele voorwaarden: er moet een vliegtuig zijn dat wil landen, er moet een plaats vrij zijn om het vliegtuig te kunnen ontvangen op de inkomende taxibaan en er moet een gate vrij zijn. Bovendien moet natuurlijk ook de landingsbaan vrij zijn. Als aan al deze voorwaarden voldaan is reserveren we zowel een plaats op de inkomende taxi als in de gates, reserveren we de landingsbaan en geven we aan dat dat het vliegtuig niet meer wil landen (aangezien het al aan het landen is) door `land` terug op `0` te zetten.

- **`RunwayToTaxi`:** Deze actie geeft de `runway` terug vrij door het landende vliegtuig naar het inkomende gedeelte van de taxibaan te leiden. Aangezien de plaats op deze taxibaan reeds eerder gereserveerd werd, zodat we het vrijgeven van de taxibaan kunnen garanderen, hoeven we in deze actie geen plaats meer te reserveren.

- **`TaxiToPlatform`:** De `TaxiToPlatform` actie geeft een migratie aan van een vliegtuig van de inkomende taxibaan naar het platform. Hiervoor moet het platform vrij zijn om botsingen te vermijden en moet er natuurlijk een vliegtuig zijn op de ingaande taxibaan. Wanneer aan deze voorwaarden voldaan is geven we aan dat het platform bezet is, en geven we een plaats vrij op de inkomende taxibaan.

- **`PlatformToGate`:** Ook deze actie geeft een beweging van een vliegtuig aan, deze keer van het platform naar een gate. Aangezien gates al bij de landing gereserveerd worden geven we simpelweg het platform vrij, op voorwaarde dat het platform een inkomend vliegtuig bevat.

- **`GateToPlatform`:** Vliegtuigen die willen vertrekken moeten zich eerst naar het platform begeven. Dit gebeurt wanneer er een vliegtuig in de gate zit, er plaatsen op de uitgaande taxibaan vrij zijn en het platform vrij is. Als aan deze voorwaarden voldaan is geven we aan dat het platform bezet is en dat er een gate is vrijgekomen. We reserveren hier ook al meteen een plaats op de uitgaande taxibaan zodat het vliegtuig bij het verlaten van het platform met zekerheid een plaats heeft en dus het platform niet onnodig zal bezet houden.

- **`PlatformToTaxi`:** Deze actie beschrijft wat er dient te gebeuren wanneer een vliegtuig zich van het platform naar de uitgaande taxibaan verplaatst en vindt plaats wanneer de status van het `platform out` is. Aangezien de plaats op de taxibaan al in de voorgaande actie (`GateToPlatform`) is gereserveerd, dienen we hier gewoon het `platform` terug vrij te geven.

- **`TaxiToRunway`:** Een vliegtuig kan zich enkel naar de `runway` verplaatsen wanneer deze vrij is en wanneer er zich een vliegtuig op de uitgaande taxibaan bevindt. Wanneer dit het geval is geven we de plaats vrij op de uitgaande taxibaan en bezetten we de `runway`.

- **`TakesOff`:** Het opstijgen van een vliegtuig gebeurt wanneer er zich een vliegtuig dat wenst op te stijgen op de startbaan bevindt. Bij het opstijgen geven we de startbaan terug vrij.

- **`RejectPlane`:** Deze actie wijst vliegtuigen die wensen te landen af, wanneer er geen vrije gates zijn. We verwijzen deze door naar een andere luchthaven, of naar een later tijdstip door de `land` variabele terug op `0` te zetten.

2.2.3 Veiligheidsvereiste

Zoals eerder vermeld in Sectie 2.1.2 kan de status van een zone waar maar één vliegtuig op eender welk moment dient te worden toegelaten drie statussen aannemen: `free`, `out` en `in`. `free` betekent dat deze zone vrij te betreden is aangezien er zich geen vliegtuig bevindt. `out` en `in` betekenen dat de zone bezet is.

Door als voorwaarde voor het betreden van de zone te eisen dat de betreffende zone vrij is zorgt ervoor dat een bezette zone nooit door twee vliegtuigen tegelijk bezet kan betreden worden, zodat deze nooit botsen.

2.2.4 Fairnessvereiste

Een vliegtuig moet ooit kunnen landen als het dat wil (en als er voldoende vrije plaatsen zijn). Een vliegtuig dat wil opstijgen moet hier ook de kans tot krijgen. Meer bepaald willen we dat alle acties eventueel voorkomen, ook al zijn ze op bepaalde momenten niet `enabled` aangezien bepaalde acties de "enabling conditions" van andere acties kunnen beïnvloeden. In TLA+/TLC realiseren we dit door gebruik te maken van "strong fairness" en leggen we door middel van `SF_vars(actie)` deze voorwaarde op voor alle acties.

Merk wel op dat we hier geen rekening houden met welk vliegtuig precies uit de gates vertrekt, of uit beide delen van de taxibaan. We zorgen er voor dat dit eerlijk gebeurt, blijft gebeuren en dat alles veilig blijft. De selectie van het betreffende vliegtuig laten we over aan externe programma's en algoritmes.

2.3 Beschrijving formele verificatie

Om het opgestelde model te verifiëren definiëren we enkele theorema's. Deze zijn te vinden onderin het bijgevoegde bestand. Hier volgt een kort overzicht van de gedefinieerde theorema's samen met hun tekstuele beschrijving. In bijgevoegd bestand `Bijlages/Luchthaven_verificatieuitvoer.txt` kan de uitvoer van de TLA+/TLC checker gevonden worden.

- **`Types`:** Ten allen tijde dienen de gebruikte variabelen aan bepaalde voorwaarden te voldoen. Zo eisen we dat de `runway` en het `platform` altijd een status hebben overeenkomstig met `out`, `in` of `free`, dat de bufferzones altijd nul tot hun grootte aantal vliegtuigen kunnen bevatten, en dat het landen of vals (`0`) of waar (`1`) kan zijn.

- **Controleren op vastlopen:** Om te vermijden dat het programma vast loopt doordat één-vliegtuig-zones bezet blijven en nooit vrijkomen controleren we of dit het geval kan zijn. Dit doen we aan de hand om te eisen dat voor eender welk moment in de tijd, wanneer de betreffende zone bezet is, te eisen dat er een toekomstig moment is waarop dit niet het geval is. Voor zones waar meerdere vliegtuigen zijn toegelaten zoals de `taxiwayIn`, de `taxiwayOut` en de `gates` eisen we dat er op elk moment waarop deze zone vol zit, een toekomstig moment bestaat waar niet het

maximum aantal plaatsen bezet zijn. De hiervoor gedefinieerde theorema's zijn `RunwayAlwaysBecomesFreeEventually`, `PlatformAlwaysBecomesFreeEventually`, `TaxiwayInAlwaysBecomesFreeEventually`, `TaxiwayOutAlwaysBecomesFreeEventually` en `GatesAlwaysBecomesFreeEventually`.

- **Controleren op crashes:** Er zijn twee zones waar vliegtuigen elkaar kunnen kruisen en daarom ook kunnen botsen. Deze zones zijn de `runway` en het `platform`. De acties die tot beweging kunnen leiden in deze zones zijn respectievelijk `TaxiToPlatform` en `PlatformToTaxi`, en `Arrival` en `TakeOff`. De theorema's `NoPlatformCrashes` en `NoRunwayCrashes` controleren dat op eender wel moment maar één van de twee betreffende acties kan `enabled` zijn.

- **Controleren op overlopen van buffers:** Net als voorgaande controle kan een botsing optreden wanneer er zich te veel vliegtuigen op de bufferzones bevinden. Deze zones zijn de `taxiwayIn`, de `taxiwayOut` en de `gates`. Om dit te controleren kijken we of het vol zijn van deze buffers impliceert dat de "enabling conditions" niet geldig zijn voor de acties die het verplaatsen naar, alsook het reserveren van deze bufferzones (`Arrival` voor de `gates` en de `taxiwayIn`, en `GateToPlatform` voor de `taxiwayOut`).

Enkele theorema's hebben we niet formeel bewezen in TLA+/TLC aangezien deze "by design" niet kunnen voorkomen. Een actie die bijvoorbeeld zijn eigen "enabling conditions" "disablet" kan niet twee keer na elkaar worden opgeroepen. Een voorbeeld hiervan is de `Arrival` actie. Deze heeft als enabling conditie dat de `runway free` moet zijn, en past daarna de status van de `runway` aan naar `in`. Het is vanzelfsprekend dat deze actie geen twee keer na elkaar kan optreden zodat er geen vliegtuig kan landen terwijl er zich een ander vliegtuig op de `runway` bevindt. Analoog hier aan kan er geen vliegtuig opstijgen als er zich nog een vliegtuig op de `runway` bevindt.

2.4 Bijlages

- **`Bijlages/Luchthaven.tla`**: Bevat de implementatie van alle acties en specificaties.

- **`Bijlages/Luchthaven.cfg`**: Het bijhorende configuratiebestand voor bovenstaand bestand. Bevat de parameters om het aantal `gates` en de groottes van de `taxiwayIn` en `taxiwayOut` aan te passen.

- **`Bijlages/Luchthaven_verificatieuitvoer.txt`**: Voorbeelduitvoer van de TLA+/TLC checker voor het bijgevoegde configuratiebestand.

Bibliografie

[1] *15.6. Association Lists.* https://www.cs.cmu.edu/Groups/AI/html/cltl/clm/node153.html.

[2] *ALISTP.html – ACL2 Version 6.3.* http://www.cs.utexas.edu/users/moore/acl2/v6-3/ALISTP.html.

[3] *ASSOC.html – ACL2 Version 6.3.* http://www.cs.utexas.edu/users/moore/acl2/v6-3/ASSOC.html.