UCLouvain

École polytechnique de Louvain

Specification and Verification of a Railway Level Crossing

Auteur: **Thomas Ponchau**
Promoteurs: **Charles Pecheur, Christophe Limbrée**
Lecteur: **Axel Legay**
Année académique 2019–2020
Master [120] : ingénieur civil en informatique

Résumé

Le domaine d'intérêt est la modélisation et la vérification formelle des spécifications de la logique de sécurité d'un passage à niveau ferroviaire. En effet, Infrabel, le gestionnaire de l'infrastructure ferroviaire belge, souhaite renouveler son système de passage à niveau et que les spécifications du nouveau système soient validées par une approche scientifique.

Un passage à niveau est un endroit où une ligne de chemin de fer et une route se rencontrent au même niveau. Un passage à niveau appartient à la catégorie des systèmes de sécurité et doit être conçu selon les normes de sécurité les plus élevées (IEC61508).

L'objectif de cette thèse est de développer un modèle de passage à niveau à partir des spécifications fonctionnelles, et de vérifier les propriétés de sécurité liées à son comportement. Le document de base supportant la modélisation est écrit en langage semi-formel (machines à états et expressions booléennes).

Un modèle du système de passage à niveau est donc réalisé en Event-B. Les propriétés de sécurité sont encodées dans le modèle. Certaines de ces propriétés sont prouvées. Et d'autres propriétés sont démontrées comme étant fausses.

Ce document reprend la façon dont le système a été modélisé avec les choix de modélisation. Il reprend également les méthodes de preuve utilisées pour vérifier ou réfuter une propriété.

Table des matières

Remerciements

Je tenais à remercier mes promoteurs monsieur Charles Pecheur et monsieur Christophe Limbrée qui ont su m'aiguiller et répondre à mes questions durant ce mémoire.

Monsieur Charles Pécheur pour les questions liées directement au mémoire et monsieur Christophe Limbrée pour toutes les questions liées aux passages à niveau.

Je remercie également Monsieur Axel Legay d'avoir accepté de relire mon mémoire.

Chapitre 1

Introduction

1.1 Contexte

Infrabel a décidé de changer son système interne de passage à niveau, et il désire valider son niveau de sécurité.

Infrabel a fourni des spécifications décrivant les variables, ainsi que des propriétés de sécurité. L'objectif du mémoire est de vérifier que le nouveau système spécifié, respecte les propriétés. Dans cette thèse, nous utilisons le langage formel Event-B et le système de preuves Rodin.

Pour valider les spécifications d'un nouveau système, il est important de vérifier qu'elles respectent les propriétés de sécurité.
L'objectif est de prouver la consistance du nouveau système imaginé.

1.2 Approche

La première phase du projet consistait à se familiariser avec les outils nécessaires à la réalisation du projet. Il y a Rodin et ProB. Chacun avait son utilité.

Après cela, il a fallut retranscrire tous les éléments du système en variables dans le modèle, ainsi que définir la façon de mettre à jour celles-ci.

Pour finir, les différentes propriétés que le système devait avoir ont été testées après avoir été implémentées dans le modèle.

Ceci, bien sûr, n'a pas été un processus purement chronologique. En avançant, certains problèmes ont surgis, qui ont obligé à faire des allers-retours.

1.3 Contributions

Un premier modèle du nouveau système de passage à niveau en Event-B a été crée. Sur base de ce modèle, certaines des propriétés de sécurité ont été prouvées et la falsification de quelques autres a été démontrée.

1.4 Travaux connexes

Un des travaux connexes est "From UML to B–a level crossing case study" (une section dans [1]). Ce document traite le sujet des passages à niveau et montre comment on peut traduire du UML (Unified Modelling Language) en B. UML est un langage semi-formel, et B est un langage formel sur lequel on peut effectuer des vérifications. C'est intéressant de partir d'un modèle UML pour en construire un sur lequel on peut faire des vérifications.

L'évolution des langages B et Event-B est retracée dans "Applying a Formal Method in Industry : a 25-Year Trajectory" [2]. Le langage B fut utilisé dans les années 80 pour réaliser la vérification formelle de systèmes ferroviaires. Event-B en est une évolution qui permet de traiter naturellement les systèmes gérés de manière événementielle.

Un autre travail connexe est "Verification of railway interlocking systems" [3]. Ce travail réalise de la vérification pour le système d'enclenchement dans les chemins de fer. "Un enclenchement est un ensemble d'appareils de signalisation. Ceux-ci matérialisent physiquement, dans la zone d'action d'un poste d'aiguillage (jonction, croisement de voies...), une incompatibilité de manœuvre entre différents organes de commande d'appareils de voie ou de signaux. Le but est de n'autoriser le passage d'un mouvement sur les différents appareils de voie, que lorsque toutes les conditions de sécurité nécessaires à ce mouvement sont réalisées. Il peut être mis en œuvre mécaniquement, électriquement ou les deux." [4] Le travail explique d'abord comment construire un modèle d'enclenchement, et puis comment des propriétés de sécurité ont été prouvées grâce au modèle.

Les ouvrages de référence utilisés dans le cadre de ce travail sont le livre "Modeling in Event-B" [5] ainsi que le manuel d'utilisateur de Rodin [6].

Chapitre 2

Matériel de référence

2.1 Passage à niveau

Un passage à niveau est un système qui gère l'interaction entre les usagers de la route et les trains circulant sur un réseau ferroviaire. Dans le cas qui nous occupe, l'interaction se fait via des feux lumineux, des signaux sonores et aussi des barrières dans certains cas. La fonction de sécurité principale d'un passage à niveau est d'avertir l'usager de la route, suffisamment tôt, de l'arrivée d'un train.

Les passages à niveau à modéliser sont de 2 catégories différentes.

Le passage à niveau de 2e catégorie possède des demi-barrières, des feux de signalisation routiers et un signal sonore.

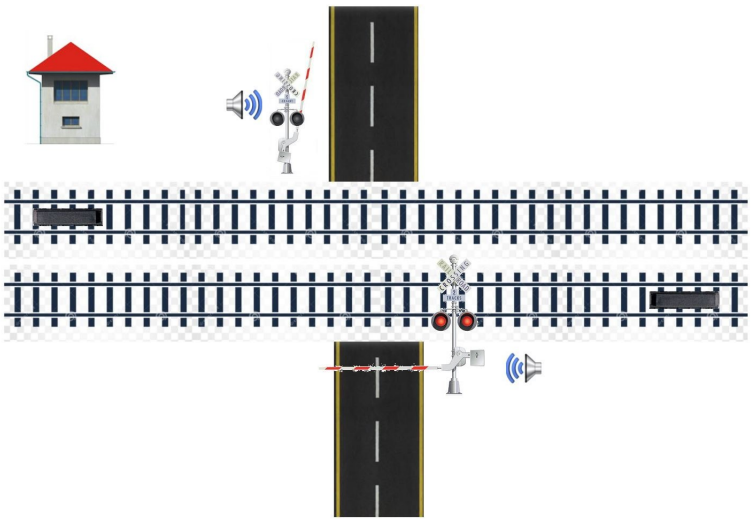

FIGURE 2.1 – Passage à niveau de 2e catégorie

Le signal sonore doit être activé dès que le passage à niveau doit être fermé. Il s'éteint lorsque les barrières sont fermées et reste éteint tant qu'il n'y a pas d'erreurs.

Le passage à niveau de 3e catégorie, ne possède pas de barrières, mais bien des feux de signalisation ainsi que le signal sonore.

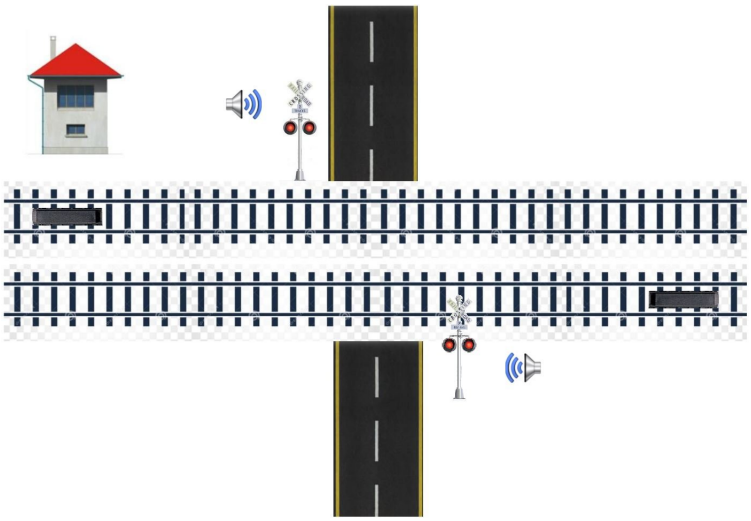

FIGURE 2.2 – Passage à niveau de 3e catégorie

2.2 Documents fournis par Infrabel

Deux documents ont été fournis pour la réalisation du mémoire.

Le premier décrit le fonctionnement du système. Il s'intitule "Protection des itinéraires ferroviaires à l'aide de passages à niveau, Nouvelle génération de passages à niveau, Exigences fonctionnelles et RAMS". Les spécifications décrivent comment la logique interne d'un passage à niveau agit sur les feux et les barrières pour assurer sa fonction de sécurité. La gestion de la logique dépend de l'environnement du passage à niveau et de son état interne. En résumé, si on considère que la fonction principale repose sur un ensemble de sous-fonctions, les propriétés permettent de vérifier le fonctionnement correct de ces sous-fonctions.

Il y a les variables d'entrée, les variables de sortie et les variables internes, ainsi que les machines à états.

- Les variables d'entrée sont des mesures de l'état des barrières, des feux,...
- Les variables de sortie sont les commandes sur les barrières, les feux, le signal sonore, ainsi que les alarmes.

- Les variables internes sont des variables intermédiaires utilisant des variables d'entrée et utiles pour déterminer les variables de sortie.

Le deuxième document spécifie les propriétés de sécurité à vérifier. Il s'intitule "Protection des itinéraires ferroviaires à l'aide de passages à niveau, Nouvelle génération de passages à niveau, Spécification de vérification de sécurité du DSG_RS". Il contient aussi la définition des variables supplémentaires utilisées dans ces propriétés.

2.3 Event-B

Event-B est un langage formel de modélisation et d'analyse au niveau du système. Les principales caractéristiques de l'événement B sont l'utilisation de la théorie des ensembles comme notation de modélisation et l'utilisation de preuves mathématiques pour vérifier leur cohérence.

Event-B est une évolution du langage B développée par Jean-Raymond Abrial. [7] Dans la structure en Event-B, un système s'appelle une machine. Celle-ci peut contenir différents éléments, comme des variables, des variants, des invariants, des events.

Les variables peuvent être booléennes (valoir TRUE ou FALSE), entières, mais elles peuvent également être d'un autre type qui peut être défini par l'utilisateur comme un "set" et on peut définir quelle set de valeurs peut prendre les variable de ce type.

Les invariants sont des prédicats qui doivent être vrai à tout moment de l'exécution. Le type des variables doit être spécifié comme invariant (par ex : $a \in BOOL$)

Les events sont des fonctions. Ils peuvent contenir plusieurs actions. Une action consiste à assigner une nouvelle valeur à une variable. Une petite subtilité à savoir est que les actions ont lieu en parallèle. Quand on utilise des variables dans des équations, c'est toujours l'ancienne valeur de cette variable qui est utilisée. On peut également avoir des paramètres dans les events. Il y a aussi les guards qui sont des conditions sur les paramètres. Ce sont des prédicats tout comme les invariants. Il y aura donc des guards pour définir le type des paramètres (s'ils sont booléens ou entiers).

Il est également possible d'affiner une machine. Cela va étendre la machine, on va pouvoir ajouter de nouvelles variables, de nouveaux invariants, de nouveaux events et redéfinir certains events (les fonctions). Pour la redéfinition des events, on peut ajouter des paramètres, des conditions (guards) et des actions, mais pas modifier ceux déjà présents. La figure 2.3 décrit ce que représente une machine étendue.

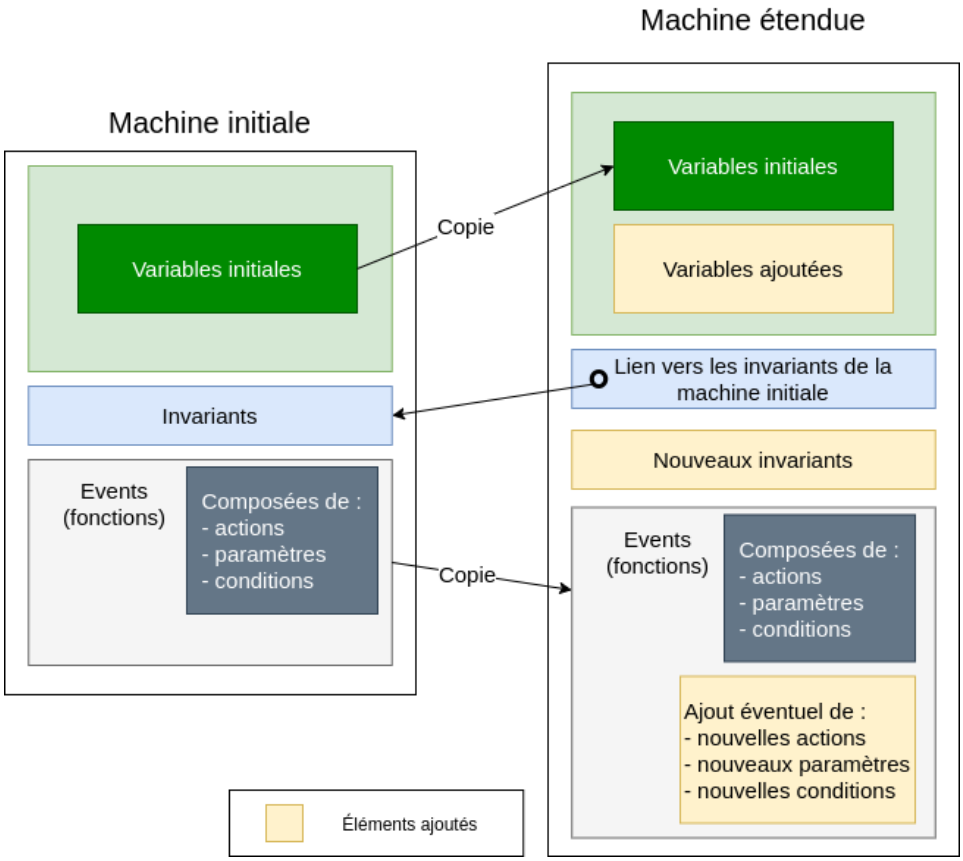

FIGURE 2.3 – Illustration d'une machine étendue

2.4 Rodin

La plateforme Rodin est un outil basé sur Eclipse pour Event-B. Elle est open source, contribue au framework Eclipse et est extensible avec des plugins.

Le développement de Rodin est actuellement soutenu par le projet ICT de l'Union européenne ADVANCE (2011 à 2014).

Il y a plusieurs facettes à Rodin : le code en Event-B, les preuves d'invariant, et ProB si le plug-in est installé (ProB est un model checker à ne pas confondre avec le moteur de preuve de Rodin.). Ce dernier permet de trouver les propriétés qui ne sont pas satisfaites.

Rodin permet de programmer facilement en Event-B.

Il permet surtout de prouver automatiquement, que les propriétés encodées sont correctes ou pas.

Pour prouver les propriétés, on peut utiliser l'outil de preuve de Rodin. Celui-ci permet de prouver que les invariants sont vérifiés à l'initialisation, ainsi que pour chaque event si les invariants sont vrais avant, ils restent vrais après. Cela permet donc de prouver les invariants par récursivité.

Pour faire la preuve des propriétés, on étend la machine en ajoutant dans les invariants toutes les propriétés et ainsi pouvoir les vérifier.

Rodin tente de faire les preuves automatiquement grâce aux invariants et aux formules de mises à jour des variables définies dans les actions et les guards des events.

Rodin permet d'assister la preuve manuellement lorsque celle-ci n'aboutit pas automatiquement.

Deux approches sont possibles.

On peut faire une distinction de cas. On rentre un prédicat et la preuve se scinde en deux parties :

- une première où l'on doit prouver l'objectif avec le prédicat comme hypothèse supplémentaire.
- une deuxième où l'on doit prouver l'objectif avec la négation du prédicat comme hypothèse supplémentaire.

On peut également ajouter une hypothèse. La preuve va se diviser en deux parties :

- une première où l'on doit prouver l'hypothèse sur base de celles déjà présentes.
- une deuxième où l'on doit prouver l'objectif avec l'hypothèse supplémentaire.

2.5 ProB

ProB est un plugin disponible dans Rodin, mais il existe aussi une version de ProB en-dehors de Rodin. Il permet de faire des simulations des systèmes modélisés en Event-B pour prouver que certaines propriétés sont fausses, en trouvant un contre-exemple.

L'utilisation de ProB nécessite d'abord son installation. Quand la version externe a été installé, et lancé, il demandait des fichiers avec une extension qui ne correspondait à aucun fichier créé par Rodin. Mais, il y avait moyen d'installer des nouvelles extensions dans Rodin et ProB faisait partie de la liste de ces extensions. Une fois installé, pour lancer la simulation, il suffit de choisir sur une machine dans l'Event-B Explorer de Rodin et de clicker sur "Start Animation / Model Checking"

À partir de là, on peut faire des simulations en sélectionnant la fonction (event) à exécuter et en choisissant les valeurs des paramètres parmi les valeurs acceptées par les conditions (guard) de la fonction.

On peut également lancer un "model checking" qui vérifie la présence de deadlocks, la violation d'invariants.

Dans un premier temps, ProB a été utilisé pour faire des simulations et vérifier que le modèle fonctionnait et que les différents états possibles pour le passage à niveau (décrits à la section 3.5) étaient atteignables.

Ensuite, il a été utilisé pour vérifier des propriétés violées. En trouvant un exemple d'exécution pour arriver à un moment où la preuve n'était plus vérifiée.

Enfin, pour tenter de trouver le reste des propriétés non vérifiées, on va utiliser le "model checking" disponible dans ProB. Pour éviter que le model checking ne trouve les propriétés déjà découvertes comme étant violées, il faut retirer celles-ci du système.

Chapitre 3

Modélisation

3.1 Principe

- En premier lieu, il faut modéliser les 2 types de passages à niveaux en Event-B. Ceci se fera à l'aide de l'application RODIN.
- En second lieu, il faut encoder les différentes propriétés dans Rodin.
- Rodin possède une fonctionnalité qui permet de vérifier automatiquement une propriété.

  Si celle-ci est satisfaite, Rodin le signale.

  Dans l'autre cas, Rodin s'arrête.
- Il faut ensuite chercher si l'arrêt est dû à un rejet de la propriété ou à l'incapacité de Rodin à la résoudre.

  En cas de rejet, on peut utiliser le plug-in ProB afin de trouver un cas de rejet.

  Sinon, on peut ajouter des conditions dans la preuve afin que Rodin puisse quand même prouver la propriété. Les conditions ajoutées doivent également être prouvées.

3.2 Éléments

Dans la description du système, il y a plusieurs types d'éléments.

Il y a des variables booléennes. Certaines sont des variables d'entrées (par exemple les variables liées à l'état des feux) , d'autres sont des variables de sortie (par exemple les commandes d'allumage de feux), et les dernières sont des variables internes (par exemple pour déterminer si le passage à niveau est commandé à la fermeture ou à l'ouverture).

Il y a également des horloges qui sont liées à certaines variables internes ou variables de sortie.

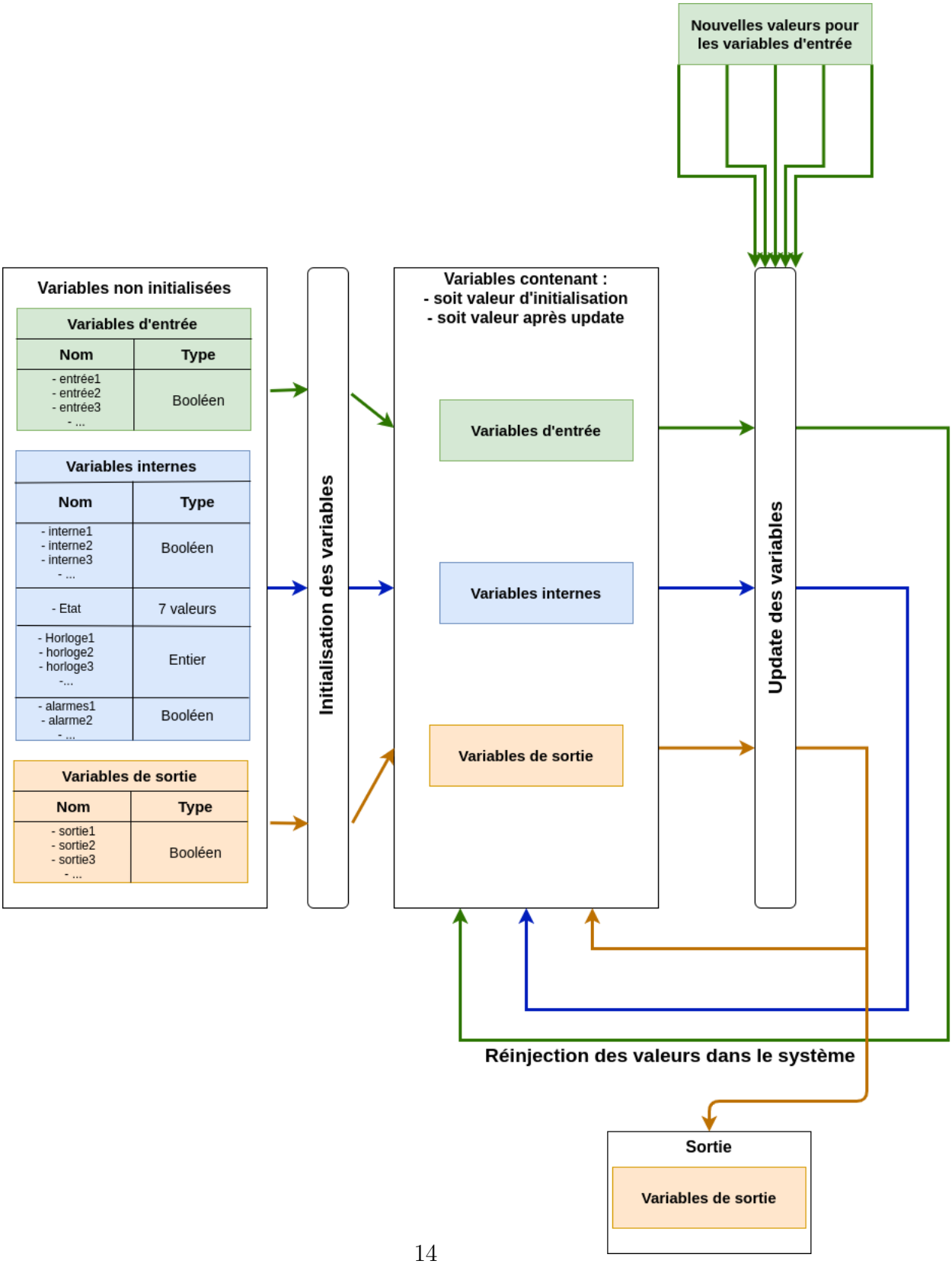

FIGURE 3.1 – Visualisation du système et de ses variables

Le passage à niveau peut être dans différents états qui sont déterminés par des variables d'entrée et des variables internes. Pour les passages à niveau de 2e catégorie, il y a 7 états possibles, et pour ceux de 3e catégorie, il y a 2 états possibles.

Lors de l'exécution du système, toutes les variables sont d'abord initialisées (comme on peut le voir sur le schéma 3.1). Ensuite, les variables sont mises à jour à chaque cycle. Un cycle correspond à un intervalle de temps. Les cycles se déroulent de la manière suivante :

- le système lit l'état des variables d'entrée.
- il calcule la nouvelle valeur pour certaines variables internes, comme par exemple la variable spécifiant si le passage à niveau est commandé à l'ouverture ou à la fermeture.
- le système définit le nouvel état.
- il met à jour les horloges.
- il calcule les valeurs des variables de sortie.

Il y a également plusieurs paramètres de configuration du système (par exemple, les valeurs limites des différentes horloges, le nombre de barrières, le nombre de poteau-feux...). Une valeur leur est assignée à l'initialisation du système et puis la valeur ne change plus.

3.3 Variables

Pour la modélisation, les variables du système sont encodées comme variables dans Rodin en Event-B. La définition du type d'une variable se fait en spécifiant un invariant (par exemple $X \in BOOL$, pour dire que la variable X est de type booléen). Toutes ces variables doivent être mises à jour régulièrement. On implémente une fonction qui devra être exécutée plusieurs fois tout au long de l'exécution du système. On appelle cette fonction `updateVariable`.

Une particularité des fonctions dans Event-B (appelées events), c'est que les actions sont effectuées en parallèles. Cela implique que l'on ne peut pas réutiliser les nouvelles valeurs des autres variables pour mettre à jour une certaine variable. Or certaines variables internes, ont besoin de la mise à jour d'autres variables internes, pour être mise à jour.

L'option choisie pour contourner ce problème, c'est d'introduire :

- la mise à jour des variables comme paramètres de la fonction.
- des conditions (guards) sur les paramètres correspondants à l'équation de ces variables.

Ces conditions servent à vérifier la mise à jour correcte des variables. En effet, les event dans eventB permettent d'ajouter des conditions appelées guards sur les variables et les paramètres de l'event.

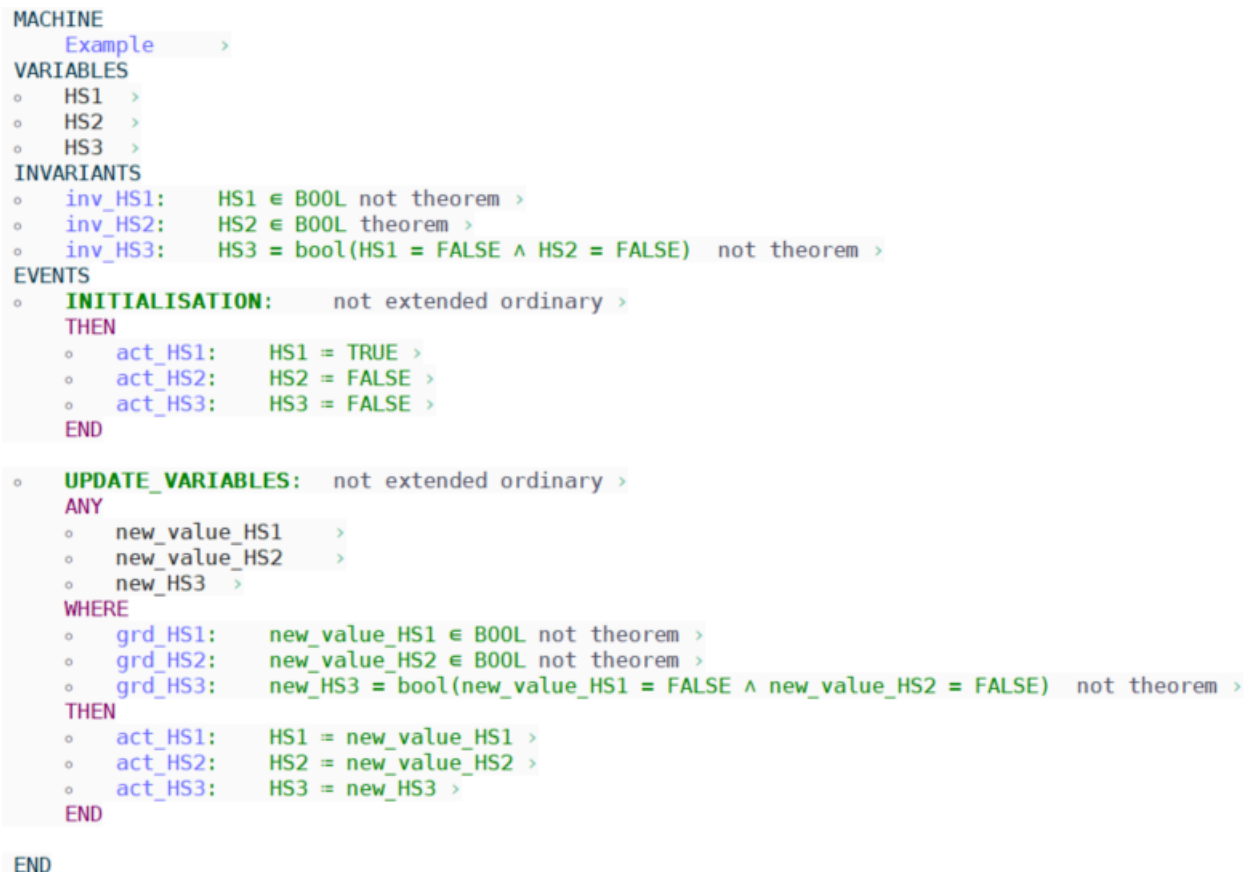

FIGURE 3.2 – Exemple de modélisation pour 2 variables d'entrée (HS1 et HS2) et une variable interne (HS3)

Pour différencier les paramètres qui sont des variables d'entrées, et donc qui n'ont pas de conditions par rapport aux paramètres qui sont des variables internes, un préfixe différent est utilisé. Pour les variables d'entrée, le préfixe `new_value_` est utilisé. par exemple : `new_value_HS1`. Pour les variables internes et de sortie, le préfixe `new_` est utilisé. Par exemple : `new_HS3`.

Les conditions sur les variables sont également réécrite en tant qu'invariant. Cela permet de pouvoir utiliser ces informations pour les preuves dans Rodin. Comme certaines de ces conditions ont besoin de la valeur précédente de certaines variables, des variables avec le suffixe `_prev` pour indiquer que c'est la valeur de la variable au cycle précédent.

Par exemple `etat_prev` stocke la valeur de `etat` du cycle précédent.

3.3.1 Variables d'entrée

Voici la liste des variables d'entrée (les variables précédées de * sont uniquement pour les passages à niveaux de 3e catégorie)

- HS1 est une variable booléenne permettant de préciser si le passage à niveau est en mode automatique.
- HS2 est une variable booléenne permettant de préciser si le passage est en mode gardiennage et commandé fermé.
- KLP1 et KLP2 sont les variables booléennes indiquant si les feux blancs sont allumés.
- KLR11, KLR12, KLR21 et KLR22 sont des variables booléennes indiquant si les feux rouges sont allumés.
- * KOB1 et KOB2 sont des variables booléennes permettant de savoir si les barrières sont contrôlées ouvertes (un angle entre 75° et 90° avec le sol).
- * KFB1 et KFB2 sont des variables booléennes permettant de savoir si les barrières sont contrôlées fermées (un agles entre 0° et 10° avec le sol).
- ZAX1 et ZAX2 sont des variables booléennes qui signalent si un train arrive.
- FT est une variable booléenne qui indique si le passage à niveau est commandé manuellement à la fermeture.
- FX1 et FX2 sont des variables booléennes permettant de commander le passage à niveau à la fermeture (sous certaines conditions).
- CHS1 et CHS2 sont des variables booléennes signalant si les variables FX1 et FX2 ne doivent être tenues en compte pendant un moment.
- NHS1 et NHS2 sont des variables booléennes signalant quand les variables FX1 et FX2 peuvent être réutilisées après une mise hors service par les variables CHS1 et CHS2.
- NT600 est une variable booléenne qui permet de remettre l'horloge T600 (voir paragraphe suivant) à zero.
- A3a est une variable booléenne permettant de signaler la perte de l'alimentation auxiliaire ou une défaillance d'un élément dans la chaine d'alimentation.
- A3b est une variable booléenne permettant de signaler la perte de l'alimentation principale.
- A4a est une variable booléenne permettant de signaler la défaillance d'un élément de la chaîne d'alimentation interne.
- A4b est une variable booléenne permettant de signaler qu'un certain élément n'est plus alimenté avec le niveau de sécurité exigé.
- A20 est une variable booléenne permettant de détecter un dysfonctionnement de l'horloge T15 (décrite dans le paragraphe suivant).
- A21 est une variable booléenne permettant de détecter un dysfonctionnement de l'horloge Tb (décrite dans le paragraphe suivant).
- A22 est une variable booléenne permettant de détecter un dysfonctionnement

de l'horloge T600 (décrite dans le paragraphe suivant).
- A23 est une variable booléenne permettant de détecter un dysfonctionnement de l'horloge T_Loss_KFB (décrite dans le paragraphe suivant).

Les 8 dernières variables sont des alarmes dont le déclenchement ne dépend pas de la modélisation qui a été faite. Elles ont donc été introduites comme variables d'entrée, ce qui veut dire qu'elles peuvent être activée à n'importe quel moment.

3.3.2 Variables internes

Voici la liste de variables internes :
- HS3 est une variable booléenne qui dit si le passage à niveau est en mode gardiennage et commandé ouvert.
- CLX est une variable booléenne qui dit si le passage à niveau est commandé fermé.
- etat est une variable qui précise dans quel état se trouve le passage à niveau. Il y a 2 valeurs possibles pour ceux de 3e catégorie et 7 valeurs possibles pour ceux de 2e catégorie.
- etat_prev est une variable qui stocke la valeur précédente de la variable etat.
- RHSintern est la variable booléenne qui signalent si les variables FX1 et FX2 ne doivent être prises en comptes.
- Partial_opening est une variable booléenne, permettant de savoir, si le passage à niveau a été précédemment en mode "ouverture de barrière", sans être passé en mode "ouvert" entre temps.
- Partial_opening_prev est la variable booléenne permettant de stocker la valeur précédente de Partial_opening.
- * T15 est une variable entière qui correspond à l'horloge qui permet d'attendre un délai avant la fermeture des barrières, lorsque le passage à niveau est commandé à la fermeture.
- * T15_prev est la variable entière contenant la valeur de T15 du cycle précédent.
- T600 est une variable entière qui correspond à l'horloge qui calcule le temps écoulé depuis la dernière fois que le passage à niveau était ouvert lorsqu'il ne l'est pas.
- T600_prev est la variable entière contenant la valeur de T600 du cycle précédent.
- * T_Loss_KFB est la variable entière qui correspond à l'horloge qui calcule le temps écoulé, lorsqu'une barière n'est plus contrôlée fermée alors que le passage à niveau est fermé.
- * T_Loss_KFB_prev est la variable entière contenant la valeur de T_Loss_KFB du cycle précédent.

- Tb est la variable entière qui correspond à l'horloge qui mesure le temps de fermeture des barrières.
- Tb_prev est la variable entière contenant la valeur de Tb du cycle précédent.
- A1, A2, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18, A19 sont des variables booléennes représentant des alarmes qui sont activées dans certaines conditions dépendant des variables d'entrée et des variables internes.
- A est une variable booléenne indiquant si un grand dérangement est présent (sur base d'autres alarmes).
- a est une variable booléenne indiquant si un grand dérangement est présent (sur base d'autres alarmes).

Certaines variables implémentées dans la modélisation sont des paramètres et ne sont donc jamais mise à jour durant les cycles. Ce sont des variables entières liées aux horloges qui sont expliquées dans la section suivante.

- Tstep est l'intervalle de temps entre chaque début de cycle.
- * Ttrans est le délai qu'il faut attendre avant de commencer à fermer les barrières lorsque le passage à niveau est commandé à la fermeture. Elle est liée à l'horloge T15.
- Alarm_600 est le temps maximal accepté où le passage à niveau n'est pas ouvert. Elle est liée à l'horloge T600 ainsi qu'à l'alarme A7 (activée dans le cas où l'horloge dépasse la valeur).
- * Alarm_20 est le temps maximal accepté durant lequel une barrière n'est plus fermée, alors que le passage à niveau est quant à lui fermé. Elle est liée à l'horloge T_Loss_KFB ainsi qu'à l'alarme A9 (activée dans le cas où l'horloge dépasse la valeur).
- Tb_max est le temps maximal accepté de fermeture des barrières. Cette variable est liée à l'horloge Tb ainsi qu'à l'alarme A6 (activée dans le cas où l'horloge dépasse la valeur).

3.3.3 Variables de sortie

Les variables de sortie sont les suivantes.

- KFX1 et KFX2 indiquent si les barrières sont contrôlées fermées.
- DA1 et DA2 indiquent si le signal ferroviaire de couverture est commandé fermé (sous certaines conditions).
- RA1, RA2, Ra1 et Ra2 indiquent si une alarme est générée au poste de signalisation.
- RHS1 et RHS2 indiquent que les entrées FX1 et FX2 ne doivent pas être prises en compte.
- NRHS1 et NRHS2 indiquent que les entrées FX1 et FX2 peuvent être prises en compte.

- CLP indique si les feux blancs doivent être commandés allumés.
- CLR indique si les feux rouges doivent être commandés allumés.
- CB indique si les barrières doivent être commandées fermées.
- CSB indique si le signal sonore est commandé actif.
- CRADAR indique si le radar doit fonctionner.

3.4 Horloges

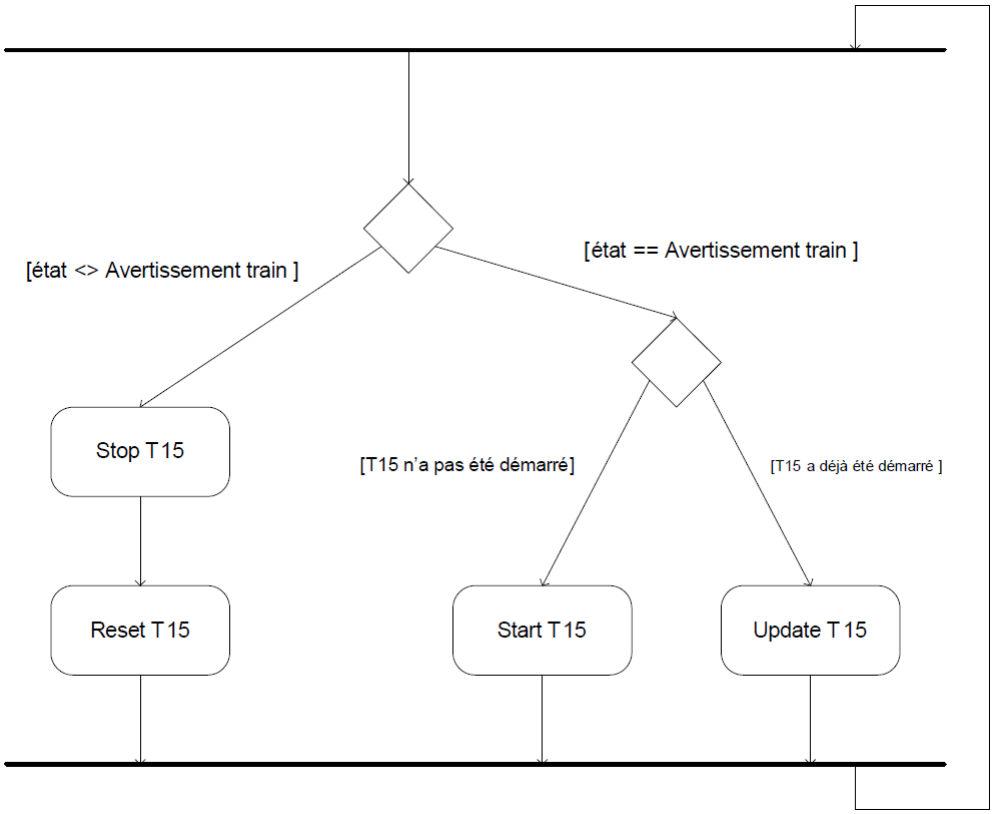

FIGURE 3.3 – Diagramme de fonctionnement de l'horloge T15

Plusieurs horloges sont nécessaires pour permettre à un passage à niveau de fonctionner correctement. Une horloge peut recevoir plusieurs commandes :

- "Start" si l'horloge n'est pas démarrée et que les conditions d'activation sont réunies.
- "Update" si les conditions d'activation sont actives et que l'horloge est déjà démarrée.
- "Stop" quand les conditions d'activation ne sont pas respectées.

- "Reset" qui va remettre l'horloge à 0 et pouvoir la réutiliser par la suite.

Pour modéliser chaque horloge, une variable entière correspondant à la durée écoulée a été utilisée.

Pour déterminer de combien l'"Update" incrémente l'horloge, une constante `Tstep` a été ajoutée. Celle-ci est une variable entière initialisée au début avec l'intervalle de temps entre chaque cycle. Elle n'est pas modifiée par la suite. À chaque exécution de l'event `updateVariable`, les horloges qui doivent être updater sont incrémentées de la même valeur `Tstep` (un seul `Tstep` pour toutes les horloges).

Lors de l'event `updateVariable`, un nouveau paramètre est créé pour chaque horloge (ex : new_T15 pour T15). Pour contrôler la valeur de ce paramètre, plusieurs conditions ont été créées. Elles sont mises sous forme d'implication. Dans la partie gauche, on met une des conditions (par exemple `etat = Avert_techn`) et on assigne la nouvelle valeur à l'horloge (par exemple $new\_T15 = T15 + Tstep$).

Dans la modélisation, pour les commandes "Start" et "Update", la valeur de la nouvelle variable est la somme de la précédente (qui vaudra 0 pour "Start") et de $Tstep$: $new\_TX = TX + Tstep$.
Pour la commande "Reset", qu'il y ait eu une commande "Stop" lors du même cycle ou lors d'un cycle précédent, la nouvelle valeur pour l'horloge vaut 0 : $new\_TX = 0$.
Pour la commande "Stop" sans commande "Reset" juste après dans le même cycle, ou pour le cas où on a aucune des 4 commandes sur le cycle, la valeur de l'horloge ne change pas : $new\_TX = TX$.

On peut voir à la figure 3.4 un petit exemple du code pour la mise à jour de l'horloge T15 dont le fonctionnement est décrit à la figure 3.3.

```
grd_25: new_etat ≠ Avert_train ⇒ new_T15 = 0 not theorem ›
grd_26: new_etat = Avert_train ⇒ new_T15 = T15 + Tstep not theorem
```

FIGURE 3.4 – Mise à jour de la variable T15

3.5 États

3.5.1 3e catégorie

Pour un passage à niveau de 3e catégorie, il n'y a que 2 états possibles : `PNouvert` et `PNfermé`.
Un contexte a été prévu pour créer une partition nommée `ETAT` comme le montre la figure 3.5.
Le type de la variable `etat` est défini dans les invariants comme appartenant à `ETAT`. La variable `etat` peut alors prendre les valeurs de la partition, à savoir `PNouvert` ou `PNfermé`, mais pas les 2 en même temps.

```
CONTEXT
    ctx1    ›
SETS
◦   ETAT    ›
CONSTANTS
◦   PNouvert     not symbolic ›
◦   PNfermé  not symbolic ›
AXIOMS
◦   type:   partition(ETAT,{PNouvert},{PNfermé})  theorem
END
```

FIGURE 3.5 – Définition du contexte 1

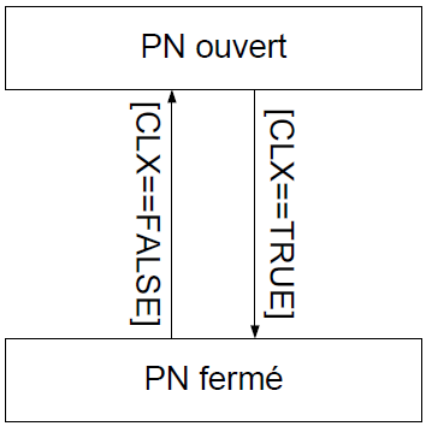

FIGURE 3.6 – Diagramme des transitions d'états pour un passage à niveau de 3e catégorie

La transition de la variable `etat` est décrite dans le diagramme 3.6. Pour définir la nouvelle variable `new_etat`, on se base sur la nouvelle valeur de CLX (une variable interne qui est `TRUE` lorsque le passage à niveau est commandé à la fermeture, et `FALSE` sinon). Lorsque $new\_CLX = TRUE$, alors $new\_etat =$ `PNfermé`, et lorsque $new\_CLX = FALSE$, alors `new_etat = PNouvert`.

3.5.2 2e catégorie

L'ajout de barrières au passage à niveau augmente le nombre d'états possibles. Nous avons donc 7 états à la place de 2 pour les passages à niveaux de 3e catégorie.

- `PNouvert` est l'état lorsque
  - le passage à niveau est commandé ouvert (`CLX = FALSE`)
  - les barrières sont contrôlées en position ouverte ($\prod_{r=1}^{\#r} KOB(r) = TRUE$)
- `Avert_techn` est l'état
  - le passage à niveau est en avertissement technique, c'est à dire qu'il est commandé ouvert (`CLX = FALSE`) mais que, au moins une barrière, n'est pas contrôlée en position ouverte ($\prod_{r=1}^{\#r} KOB(r) == FALSE$)

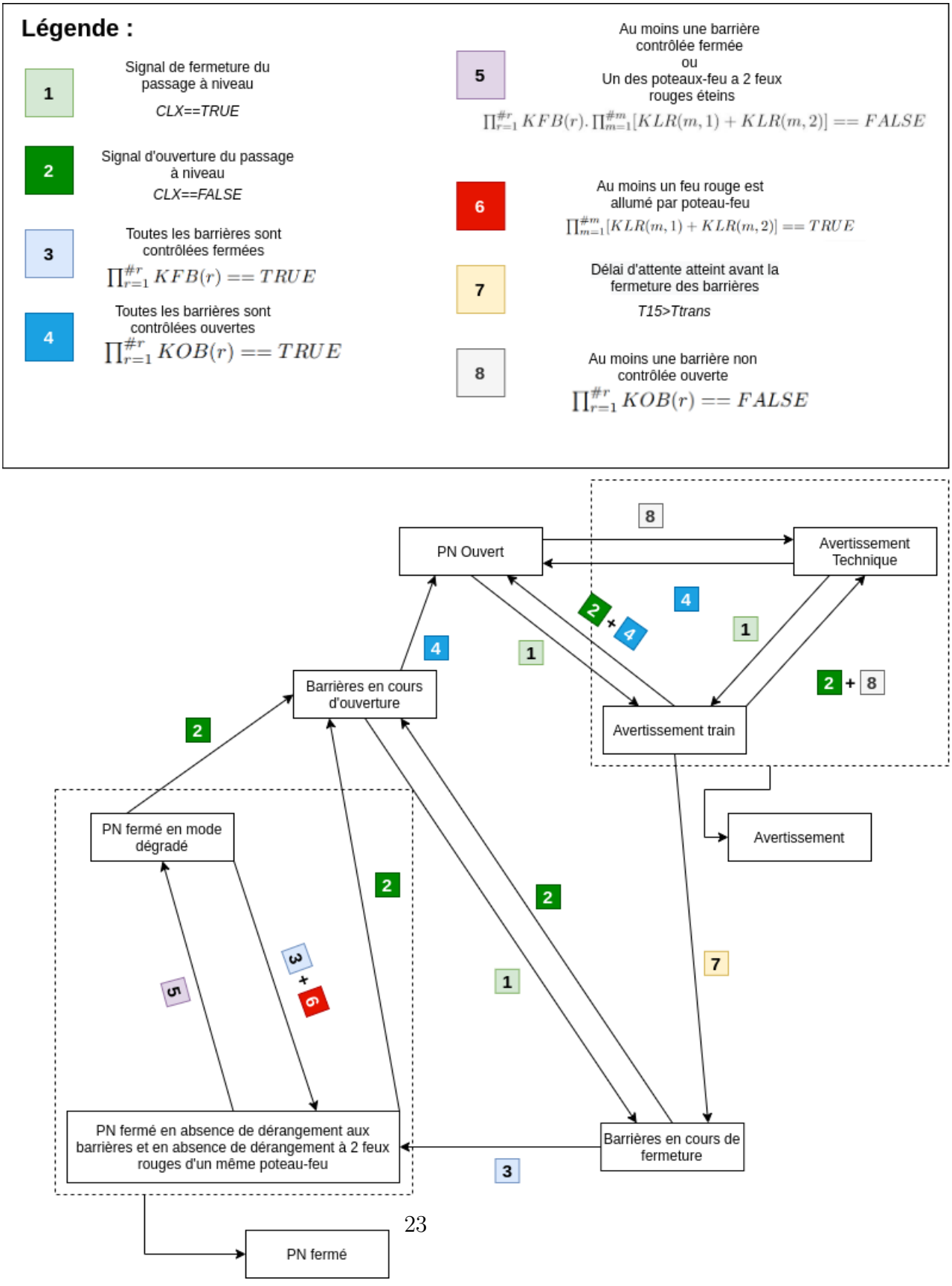

FIGURE 3.7 – Diagramme des transitions d'états pour un passage à niveau de 2e catégorie

- `Avert_train` est l'état où
  - le passage à niveau est commandé à la fermeture ($CLX = TRUE$) (exemple : lorsqu'un train est détecté en approche)
  - les feux rouges et le signal sonore sont déjà actifs
  - les barrières ne se ferment pas encore ($T15 \leq Ttrans$). $Ttrans$ est le temps de transition entre le moment où le passage à niveau est commandé à la fermeture et le moment où les barrières commencent à se fermer.
- `Barrière_fermeture` est l'état lorsque
  - le passage à niveau est commandé à la fermeture ($CLX = TRUE$)
  - les barrières sont en cours de fermeture (donc assez de temps s'est écoulé depuis qu'il est commandé à la fermeture : $T15 > Ttrans$)
- `PNfermé_correct` correspond au passage à niveau
  - commandé à la fermeture
  - les barrières sont contrôlées fermées ($\prod_{r=1}^{\#r} KFB(r) = TRUE$)
  - au moins un feu rouge de chaque poteau-feu est allumé ($\prod_{m=1}^{\#m}[KLR(m,1)+ KLR(m,2))$
- `PNfermé_dégradé` correspond au passage à niveau commandé à la fermeture ($CLX = TRUE$) mais avec une erreur, donc soit :
  - au moins une barrière n'est pas contrôlée fermée ($\prod_{r=1}^{\#r} KFB(r) = FALSE$)
  - au moins un poteau-feu n'a aucun de ses feux rouges allumés ($\prod_{m=1}^{\#m}[KLR(m,1)+ KLR(m,2) = FALSE$).
- `Barrière_ouverture` est l'état où
  - le passage à niveau est commandé à l'ouverture ($CLX = FALSE$)
  - les barrières ne sont pas encore toutes contrôlées ouvertes ($\prod_{r=1}^{\#r} KOB(r) == FALSE$)

Pour chaque barrière, il y a 2 variables d'entrée qui vérifient leurs positions.

- La variable $KOB(r)$ permet de vérifier si la barrière est contrôlée ouverte, c'est-à-dire que l'angle entre la barrière et le sol est compris entre 75° et 90°.
- La variable $KFB(r)$ permet de vérifier si la barrière est contrôlée fermée, c'est-à-dire que l'angle entre la barrière et le sol est compris entre 0° et 10°.

Pour modéliser les 7 états possibles, un nouveau contexte avec une nouvelle partition `ETAT2` a été créé comme on peut voir sur le code 3.8. Ensuite, le type de la variable `etat` est défini comme appartenant à `ETAT2` ce qui permet que `etat` soit toujours une et une seule des 7 valeurs possibles.

```
CONTEXT
    ctx2    ›
SETS
    ETAT2    ›
CONSTANTS
    PNouvert     not symbolic ›
    Avert_techn  not symbolic ›
    Avert_train  not symbolic ›
    Barrière_fermeture   not symbolic ›
    PNfermé_dégradé  not symbolic ›
    PNfermé_correct  not symbolic ›
    Barrière_ouverture   not symbolic ›
AXIOMS
    axm1:   partition(ETAT2,{PNouvert}, {PNfermé_dégradé},
                {PNfermé_correct}, {Avert_techn}, {Avert_train},
                {Barrière_fermeture}, {Barrière_ouverture})  not theorem
END
```

FIGURE 3.8 – Définition du contexte 2

Ensuite, pour la mise à jour de la variable `etat` on utilise

- la variable précédente
- la nouvelle variable `CLX`
- les nouvelles valeurs pour la mesure de l'état des barrières et des feux rouges
- l'ancienne valeur de l'horloge $T15$

Pour implémenter tout cela, des implications ont été faites avec, à gauche, l'état précédent et les autres conditions, et à droite, on assigne le nouvel état.

3.6 Hypothèses

3.6.1 Simplification du modèle

Pour modéliser le système et avoir un système cohérent sans avoir trop de complexité, quelques hypothèses ont été posées.

Ainsi, le passage à niveau de catégorie 2 est modélisé avec les paramètres suivants :

- 2 barrières (#r = 2)
- 2 voies de chemin de fer (#k = 2)
- 2 poteau-feux (#m = 2)

Les hypothèses suivantes ont également été posées :

- existence de 2 variables pour l'enclenchement PLP ($Is\_there\_FX2 = TRUE$)
- existence de 2 variables de détection de train ($Is\_there\_ZAX2 = TRUE$)

Cela permet de simplifier les équations où se trouvent ces paramètres. Ceux-ci permettent de savoir si les variables $FX2$ et $ZAX2$ sont présentes, et si elles doivent être prises en compte dans le modèle et les équations. Avoir les variables $FX2$ et $ZAX2$ dans le modèle, pouvait être intéressant, c'est pourquoi les parmètres Is_there_FX2 et Is_there_ZAX2 ont été fixés à $TRUE$.

Après coup, la présence de $FX2$ et $ZAX2$ n'apporte pas grand chose étant donné qu'elles sont toujours utilisées avec $FX1$ et $ZAX1$ respectivement. Par exemple dans la définition de $CLX = \overline{HS3} \vee \{HS2 \vee (FT \wedge HS1) \vee (HS1 \wedge ((ZAX1 \wedge ZAX2) \vee (FX1 \wedge FX2 \wedge \overline{RHSintern})))\}$
De plus, l'ajout de Is_there_FX2 et Is_there_ZAX2 n'auraient pas posé tant de problème, car ce ne sont pas des variables d'entrée pouvant changer à tout moment, mais des paramètres de configuration.
Certes, les équations sont légèrement plus longues, mais il n'y a pas besoin de mettre à jour les paramètres.
Ils sont juste encodés comme variables et initialisés, puis on ne les touche plus et ils ne doivent pas être mis à jour dans l'event `updateVariable`.

3.6.2 Erreurs détectées dans les spécifications

Dans la définition des propriétés, certaines variables sont utilisées mais n'ont pas été définies dans le premier document. En comprenant ce que doivent représenter les variables, on peut supposer qu'il y a des erreurs et qu'il faut les remplacer par des variables cohérentes.

Pour les propriétés 41, 42, 43 et 44, il y a la variable T qui est présente, or cette variable n'est pas décrite. Ces propriétés servent à vérifier la commande de fermeture des barrières. Dans les propriétés, le T est comparée avec $Ttrans$ ($T > Ttrans$). Dans le modèle, la condition pour passer de l'état `Avert_train` à l'état `Barrière_fermeture` $T15 > Ttrans$. On peut donc conclure que les auteurs des propriétés ont utilisé T pour la variable T15. Celle-ci est l'horloge qui est démarrée dans l'état `Avert_train` et incrémentée tant qu'on reste dans cet état. Lorsqu'elle dépasse la constante $Ttrans$, le système passe en état `Barrière_fermeture`.

Pour les propriétés 62, 93 et 94, il y a également une variable T non définie. Ces propriétés considèrent le passage à niveau en état fermé depuis plus de 600s (pour la propriété 62) ou depuis moins de 600s (pour les propriétés 93 et 94). L'horloge qui est comparée à 600 dans le modèle est $T600$:

- elle sert à vérifier que le passage à niveau ne reste pas trop longtemps fermé
- elle est démarrée dès que le système quitte l'état `PNouvert`
- elle est utilisée dans l'alarme $A7$ qui vérifie si $T600 > Alarm\_600$. $Alarm\_600$ est une constante qui vaut en générale 600.

Les éléments T et 600 ont été remplacées par $T600$ et $Alarm\_600$.

Pour la propriété 113, la variable $Broken\_barrier$ devrait correspondre à la variable $Alarm\_20$. En effet, la propriété teste l'activation de l'alarme $A9$ si la barrière est cassée. $Broken\_barrier$ est comparée à T_Loss_KFB qui est l'horloge qui est démarrée lorsque le passage à niveau est fermé mais que au moins une barrière n'est pas fermée. Dans la définition de l'alarme $A9$, la variable T_Loss_KFB est comparée à $Alarm\_20$ qui est une constante.

3.6.3 ProB

ProB est utile pour faire les simulations.

Le nombre de transitions possibles est fort élevé. En effet, le nombre de nouvelles variables libres (les paramètres avec le préfixe `new_value` pour l'event `updateVariable`) s'élève à 26. Les variables libres sont les variables d'entrée qui ne sont pas calculées sur base d'autres variables. Lorsque ProB générait les états possibles suivants, il trouvait 2^{26} ce qui est beaucoup pour le logiciel sur un petit ordinateur et pour la compréhension de l'effet de chacune des variables.

Pour régler le problème, une variable booléenne appelée `NoAlarm` a été créée. Cette variable n'est pas modifiée lors de l'event `updateVariable`. Elle est initialisée à `TRUE` quand on utilise ProB, et sinon à `FALSE`.
Des conditions ont été ajoutées à l'event `updateVaraible` permettant de réduire la liberté des variables libres lorsque `NoAlarm` est à `TRUE` :

- les variables dédoublées ont été égalées (par ex : `new_value_KBO1 = new_value_KOB2`)
- les variables d'entrée des alarmes ont été mises à `FALSE` (A3a, A3b, A4a, A4b, A21, A22 et A23)
- les autres variables d'entrée ont été initialisées avec leur valeur d'origine. Seule la valeur de 2 ou 3 variables était changée à la fois, en fonction de la partie du modèle que l'on souhaitait parcourir

ProB a permis de découvrir une erreur dans le modèle décrit en Event-B. Pour la mise à jour de la variable Tb, l'état Barrière_fermeture était pris en compte une deuxième fois au lieu de l'état Barrière_ouverture.

3.7 Modélisation des propriétés

Les propriétés sont également encodées dans Rodin.
Une extension des machines de 2e catégorie et de 3e catégorie a été faite pour pouvoir y ajouter les propriétés respectives. La figure 2.3 représente le principe d'une machine étendue.

Dans notre cas, certaines variables doivent être ajoutées à notre machine initiale (annFX, annZAX, NOTannFX, NOTannZAX, memoTb_KFB, preFXHS). Liés à ces variables, il y a des invariants supplémentaires, ainsi que des paramètres, des conditions et des actions en plus dans l'event `updateVariable`.

- annFX permet de savoir si les 2 variables FX (FX1 et FX2) sont vraies en même temps.
- annZAX permet de savoir si les 2 variables ZAX (ZAX1 et ZAX2) sont vraies en même temps.
- NOTannFX est la négation de annFX.
- NOTannZAX est la négation de annZAX.
- memoTb_KFB permet de mémoriser qu'au moins une barrière ne s'est pas fermée à temps et que le défaut n'a pas disparu depuis lors.
- preFXHS indique si les variables FX sont mises hors service.

Les propriétés sont encodées comme des invariants dans la machine. Cela permet de vérifier leur validité car un invariant doit être vrai à tout moment de l'exécution et Rodin permet de prouver les invariants.

Chapitre 4

Vérification

4.1 Méthodologie de preuves

Une fois que toutes les propriétés ont été encodées comme invariant, Rodin parvient à en prouver certaines automatiquement tandis qu'une action manuelle est nécessaire pour prouver les autres.

Quelques propriétés ont été sélectionnées pour être vérifiées grâce à Rodin et une aide manuelle (la sélection est détaillée dans la section suivante).

ProB permet de simuler des exécutions du modèle. Il indique à chaque étape simulée tous les invariants qui ne sont pas respectés.

La violation de certaines propriétés a été découverte avec de simple simulations dans ProB.

Ensuite, lorsqu'une des propriétés à prouver est peut-être fausse, le cas où elle pourrait être violée est reproduit sur ProB ce qui peut prouver que la propriété est fausse.

Certaines propriétés qui ressemblent très fort à d'autres propriétés violées sont sans doute violées également. Du coup, des simulations pour reproduire le cas où elles seraient violées sont réalisées.

4.2 Sélection des propriétés

Il y a 119 propriétés décrites.

	2e catégorie	3e catégorie
nombre de propriété	79	44
propriétés prouvées automatiquement par Rodin	4	21
propriétés prouvés avec Rodin avec une aide manuelle	8	0
propriétés violées trouvée avec ProB	16	2
propriétés non vérifiées	51	21

Étant donné le nombre important de propriétés et qu'elles ne se sont pas toutes prouvées automatiquement, toutes n'ont pas pu être vérifiées. De plus, beaucoup de propriétés se ressemblent. Il a donc fallu en sélectionner plusieurs à prouver, tout en essayant d'être le plus diversifié possible, et en couvrant la plupart des types (par exemple : la catégorie qui vérifie l'allumage ou l'extinction des feux, celle qui vérifie la fermeture des barrières,...).

Les propriétés se présentent sous la forme d'implications. Elles permettent de vérifier l'activation de certaines commandes ou alarmes sous certaines conditions. La partie gauche de l'implication sont les variables d'entrée correspondantes aux conditions, et la partie droite est composé des variables de sortie correspondant aux commandes ou aux alarmes dont l'activation ou la désactivation doit être vérifiée.

Il y a différentes catégories de propriétés pour tester la commande des feux, la commande sonore, la commande des barrières, l'activation des alarmes. Ces différentes commandes et alarmes peuvent être activées dans différentes situations. Il y a donc plusieurs propriétés par catégorie pour tenter d'englober plusieurs cas. Dans chaque catégorie, il y a des propriétés dont le début diffère légèrement. Les débuts des propriétés rassemblent en général les variables d'entrée permettant de définir la valeur de la variable interne CLX. Cette variable est $TRUE$, lorsque le passage à niveau est commandé à la fermeture et sinon $FALSE$.

Les différentes possibilités qui font, que le passage à niveau est commandé à la fermeture sont :

- le passage à niveau est commandé manuellement à la fermeture ($FT = TRUE$)
- le passage à niveau est en mode gardiennage et commandé fermé ($HS2 = TRUE$)
- le passage à niveau est
  - en mode automatique ($HS1 = TRUE$)
  - et la présence d'un train est détecté ($ZAX1.ZAX2 = TRUE$)
  - ou certaines conditions liés à l'enclenchement PLP sont vérifiées ($FX(1).FX(2) = TRUE$ et la variable interne $RHSintern = FALSE$)

Les débuts sont donc en général $\overline{HS1}.HS2.\overline{HS3}$
, $FT.HS1.\overline{HS2}.\overline{HS3}$, $annZAX.HS1.\overline{HS2}.\overline{HS3}$
ou $\overline{preFXHS}.annFX.HS1.\overline{HS2}.\overline{HS3}$
qui sont en fait les différentes possibilités pour rendre $CLX = TRUE$.
Il peut aussi y avoir $NOTannFX.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3}$
ou $\overline{HS1}.\overline{HS2}.HS3$ qui rend la variable interne $CLX = FALSE$.

Les propriétés sélectionnées pour apporter une aide manuelle à la preuve sont les 4, 7, 10, 20, 25, 27, 34, 37, 47, 56, 62, 68, 114.

Certaines de ces propriétés sont décrites dans les sections. Celles non décrites sont prouvées correctes ou bien la preuve n'a pas encore abouti.

4.3 Critères de sécurité logique interne et signalisation routière

4.3.1 Allumer les feux rouges

Propriété 4

Cette propriété vérifie l'activation de la commande pour allumer les feux rouges dans la condition suivante :

- fermeture manuelle en mode "gardiennage" du passage à niveau.

$$\overline{HS1}.HS2.\overline{HS3} = TRUE$$
$$\Rightarrow CLR = TRUE$$

FIGURE 4.1 – Propriété 4

La propriété est automatiquement correcte à l'initialisation.

Il suffit ensuite de prouver la propriété pour l'update des variables. Les nouvelles valeurs des variables doivent donc vérifier cette propriété. L'équation à vérifier devient donc

$$new\_value\_HS1 = FALSE \wedge new\_value\_HS2 = TRUE \wedge new\_HS3 = FALSE$$
$$=> new\_CLR = TRUE$$

Le début de la preuve, faite automatiquement, commence par résoudre l'implication, en ajoutant aux hypothèses, la partie gauche de cette implication, et en remplaçant l'objectif à prouver, par la partie de droite de cette implication. Grâce à l'ajout de ces hypothèses, il y a quelques simplifications qui sont faites automatiquement dans les invariants et les formules de conditions.

Ce qu'il faut faire lorsque la preuve automatique s'arrête, c'est la distinction de cas. On distingue le cas sur la variable `etat` ainsi que certaines valeurs pour pouvoir définir le nouvel etat `new_etat`. On peut voir la finalité des différentes distinctions de cas sur le schéma 4.2.

Une fois que le nouvel état peut être défini, la preuve automatique se termine. Il suffit que le passage à niveau ne soit pas dans l'état `PNouvert` pour que les feux rouges soient allumés (CLR = TRUE) (ce qui est en effet le cas comme on peut voir sur la figure 4.2. L'état peut valoir 7 valeurs différentes (`PNouvert, Avret_techn, Avert_train, Barrière_fermeture, PNfermé_correct, PNfermé_dégradé, Barrière_ouverture`)

Cette propriété est donc correcte pour le modèle validée grâce à Rodin avec une aide manuelle.

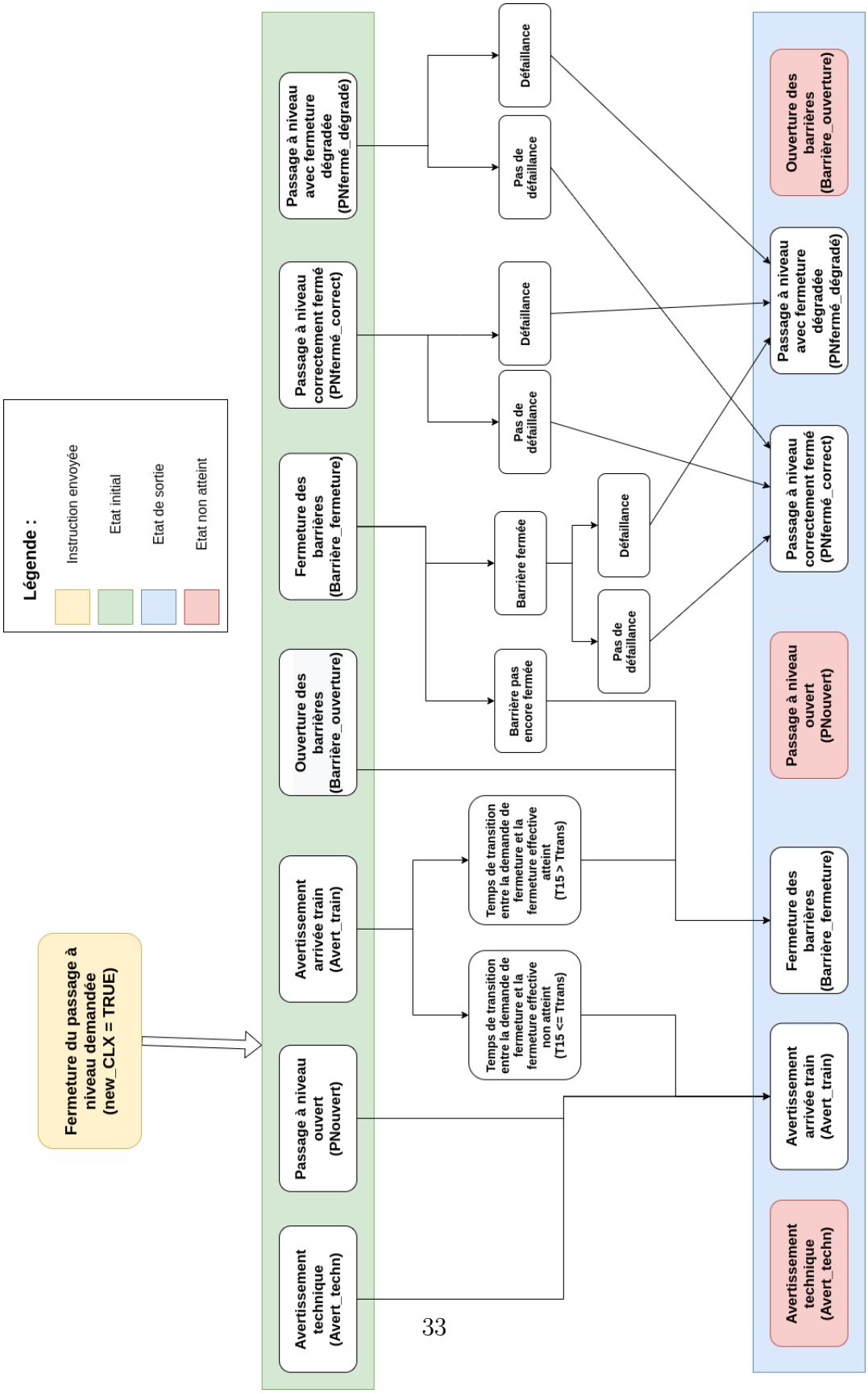

FIGURE 4.2 – Illustration pour la preuve de la propriété 4

4.3.2 Éteindre les feux rouges

Propriété 7

Cette propriété sert à vérifier que les feux rouges sont commandés éteints lorsque les conditions suivantes sont réunies :

- le passage à niveau commandé ouvert par l'SSI (Solid State Interlocking)
- la logique d'annonce locale ne détecte pas de train
- le passage à niveau n'est pas commandé manuellement à la fermeture
- le passage à niveau est en mode "automatique"
- toutes les barrières sont contrôlées ouvertes

$$NOTannFX$$
$$.NOTannZAX$$
$$\overline{FT}.HS1.\overline{HS2}.\overline{HS3}.\left(\prod_{r=1}^{\#r} KOB(r)\right) = TRUE$$
$$\Rightarrow CLR = FALSE$$

FIGURE 4.3 – Propriété 7

Cette propriété est une des propriétés dont la preuve a été essayé avec Rodin, et elle s'est avérée fausse. Pour tenter la preuve, on a commencé à faire la distinction de cas pour l'état précédent, en commençant par `PNouvert`, Avert_techn et `Avert_train`. Pour ces trois cas-là, lorsque la partie gauche de l'implication est vérifiée, la partie droite l'est aussi. Par contre, lorsqu'on teste avec `Barrière_fermeture`, et que la partie gauche de l'implication est vérifiée, on obtient que le nouvel état est `Barrière_ouverture`. Or, la partie droite de l'équation dit que les feux rouges doivent être commandés éteints ($CLR = FALSE$), et c'est le cas uniquement si l'état est `PNouvert`. La preuve sur Rodin aboutit à devoir prouver une contradiction.

Pour la vérifier, ce cas est reproduit dans une simulation en ProB, et en effet, on trouve que l'invariant, correspondant à la propriété, est violée.

On peut observer les différents états atteints avec comme hypothèse que le passage à niveau est commandé à l'ouverture et que les barrières sont commandées à l'ouverture à la figure 4.4.

Il est possible que cette propriété soit violée à cause d'un problème dans le modèle. Lors de la modélisation du changement d'état, décrit par le diagramme 3.7, on prend l'hypothèse qu'on ne peut parcourir qu'une seule flèche par cycle. De ce fait, lorsque l'état précédent est `Barrière_fermeture` et que $CLX = FALSE$, on défini le nouvel état comme étant `Barrière_ouverture` même si les barrières sont déjà contrôlées ouvertes ($\prod_{r=1}^{\#r} KOB(r) == TRUE$).

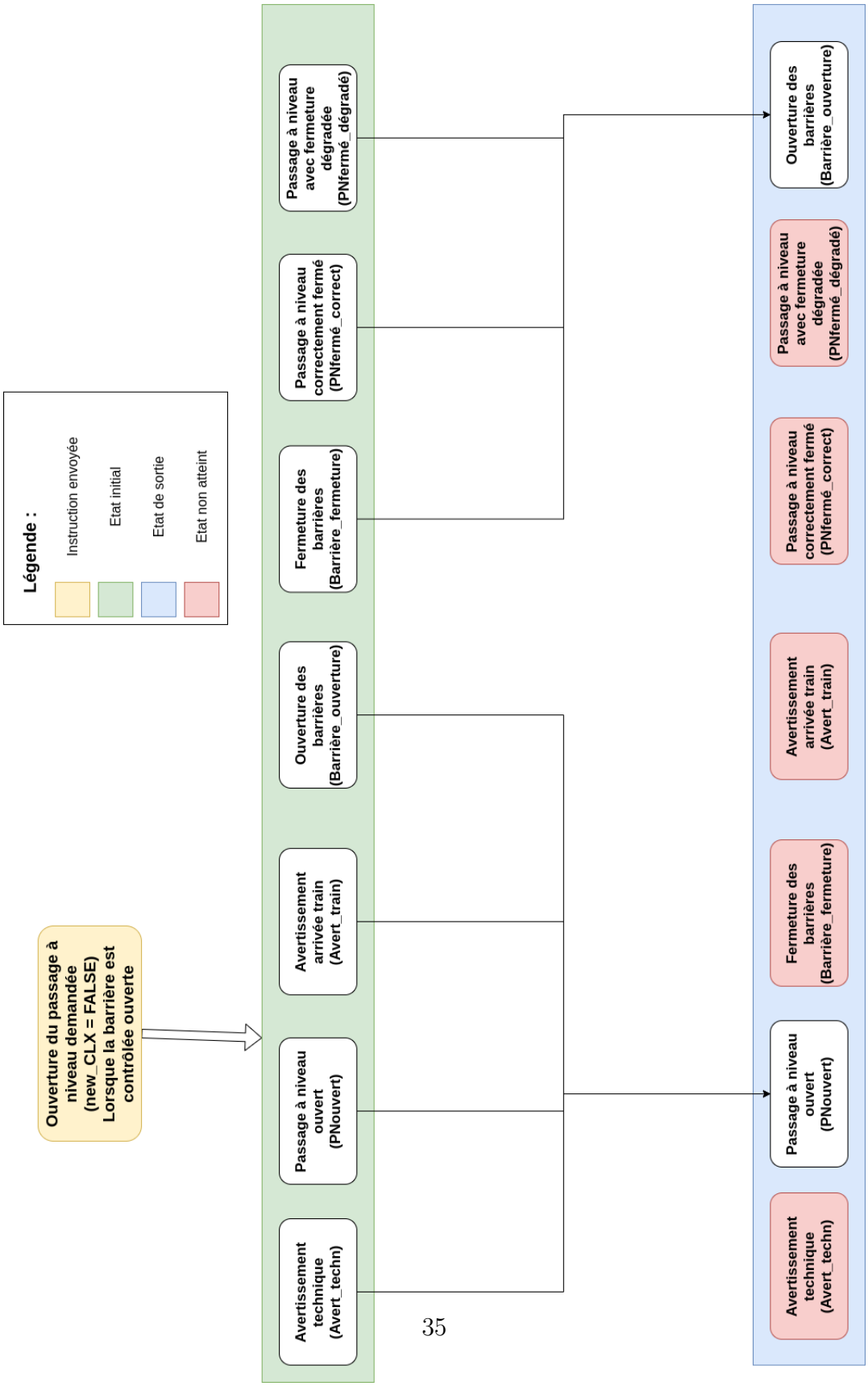

FIGURE 4.4 – Illustration pour les propriétés 7, 8 et 9

Si on peut directement passer de `Barrière_fermeture` à `PNouvert`, la propriété a plus de chance d'être vérifiée. Pour être complètement vérifiée, il faut également que l'on puisse directement passer des états `PNfermé_correct` et `PNfermé_dégradé` vers l'état `PNouvert` s'il est commandé à l'ouverture ($CLX = FALSE$) et que les barrières sont contrôlées ouvertes ($\prod_{r=1}^{\#r} KOB(r) == TRUE$).

Cette propriété est donc fausse pour le modèle. Il y a eu une tentative pour la vérifier grâce à Rodin. Puis un contre-exemple a été trouvé grâce à ProB.

Cependant la propriété n'est violée que lors d'un cycle et est vérifiée juste après. De plus, le cas où la propriété est violée se produit très rarement car il faut qu'au moment où les barrières commencent à se fermer (mais qu'elles soient toujours ouvertes), la fermeture du passage à niveau soit annulée. La violation de la propriété implique juste que les feux rouges ne s'éteignent pas, mais s'éteindront un cycle plus tard. Comme un cycle ne dure qu'un court instant et qu'il y a peu de chance que le cas survienne, la violation de la propriété n'est pas impactante.

Propriété 8

Cette propriété vérifie si les feux rouges sont bien commandés à rester éteints lorsque les conditions suivantes sont réunies :

- passage à niveau commandé ouvert manuellement en mode "gardiennage"
- toutes les barrières contrôlées ouvertes.

$$\overline{HS1}.\overline{HS2}.HS3.\left(\prod_{r=1}^{\#r} KOB(r)\right) = TRUE$$
$$\Rightarrow CLR = FALSE$$

FIGURE 4.5 – Propriété 8

Cette propriété est presque la même que la propriété 7. La seule différence est la façon de rendre le passage à niveau commandé ouvert ($CLX = FALSE$). Une nouvelle simulation sur ProB a été faite pour rendre vraie la partie gauche passant par l'état `Barrière_fermeture` juste avant. L'invariant correspondant à la propriété est en effet violée car l'état est `Barrière_ouverture` et donc la commande des feux rouges est toujours activée. La figure 4.4 est également correcte pour cette propriété.

La propriété est donc fausse avec un contre-exemple trouvé grâce à ProB.

Tout comme pour la propriété 7, la violation ne dure qu'un seul cycle où les feux rouges restent allumés et ne se produit que rarement. La violation de la propriété n'est donc pas impactante.

4.3.3 Allumer les feux blancs

4.3.4 Propriété 9

La propriété vérifie si les feux blancs sont bien commandés actifs lorsque les conditions suivantes sont réunies :

- passage à niveau commandé ouvert par l'SSI (Solid State Interlocking)
- la logique d'annonce locale ne détecte pas de train
- le passage à niveau n'est pas commandé manuellement à la fermeture
- passage à niveau en mode 'automatique'
- tous les feux rouges éteints
- toutes les barrières sont contrôlées ouvertes
- toutes les zones d'annonce contrôlées non hors-service

$$
\begin{aligned}
&(NOTannFX. \\
&.NOTannZAX \\
&.\overline{FT}.HS1.\overline{HS2}.\overline{HS3} \\
&.\left(\prod_{m=1}^{=m}\overline{KLR(m,1)}\right).\left(\prod_{m=1}^{=m}\overline{KLR(m,2)}\right).\left(\prod_{r=1}^{=r}KOB(r)\right) \\
&.\overline{RHS(1)}.\overline{RHS(2)}) = TRUE \\
&\quad \Rightarrow CLP = TRUE
\end{aligned}
$$

FIGURE 4.6 – Propriété 9

La violation de cette propriété a été découverte grâce à ProB. Lors de la simulation en ProB, pour trouver un contre-exemple de la propriété 7, il s'est avéré que la propriété 9 est violée dans le même scénario. Cette propriété vérifie l'activation de la commande pour les feux blancs CLP, et cette commande est $TRUE$ que quand l'état est `PNouvert`. Donc si le précédent état était `Barrière_fermeture`, le nouveau est `Barrière_ouverture`. La figure 4.4 est également correcte pour cette propriété.

La propriété est donc fausse avec un contre-exemple trouvé grâce à ProB.

Tout comme pour les propriétés 7 et 8, la violation ne dure qu'un seul cycle et ne se produit que rarement et implique juste que les feux blancs ne vont s'allumer qu'un cycle plus tard. La violation de la propriété n'est donc pas impactante.

4.3.5 Éteindre les feux blancs

Propriété 10

Cette propriété vérifie que la commande pour allumer les feux blancs est désactivée lorsque les conditions suivantes sont réunies :

- passage à niveau commandé fermé par l'SSI (Solid State Interlocking)
- passage à niveau en mode "automatique"

$$(\overline{preFXHS}.annFX.HS1.\overline{HS2}.\overline{HS3}) = TRUE$$
$$\Rightarrow CLP = FALSE$$

FIGURE 4.7 – Propriété 10

Cette propriété a été prouvé par Rodin. Il faut faire la distinction de cas sur les états pour définir le nouvel état. Pour que les feux blancs soient commandés à l'extinction, il suffit que le nouvel état ne soit pas `PNouvert`.

Cette propriété est donc correcte pour le modèle vérifiée grâce à Rodin avec une aide manuelle.

Propriétés 16 et 17

Ces 2 propriétés servent à vérifier que la commande pour allumer les feux blancs soit désactivée (CLP = FALSE) lorsque les conditions suivantes sont réunies :

- passage à niveau commandé ouvert par l'SSI (Solid State Interlocking)
- la logique d'annonce locale ne détecte pas de train
- le passage à niveau n'est pas commandé manuellement à la fermeture
- le passage à niveau est en mode 'automatique'
- au moins un feu rouge (le feu rouge 1 pour la propriété 16, et le feu rouge 2 pour la propriété 17) d'au moins un signal routier est allumé.

$$NOTannFX$$
$$.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{m=1}^{=m} KLR(m,1)\right) = TRUE$$
$$\Rightarrow CLP = FALSE$$

FIGURE 4.8 – Propriété 16

$$NOTannFX$$
$$.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{m=1}^{=m} KLR(m,2)\right) = TRUE$$
$$\Rightarrow CLP = FALSE$$

FIGURE 4.9 – Propriété 17

La violation de ces propriétés a été découverte grâce à ProB. Ces propriétés sont violées, lorsque le passage à niveau est ouvert, qu'au moins un feu rouge est encore actif, mais que la commande des feux blancs (CLP) est à TRUE. CLP est actif quand l'état est `PNouvert`. Or le passage vers l'état `PNouvert` n'est pas lié à KLR(m,r).

La propriété est donc fausse avec un contre-exemple trouvé grâce à ProB.

La violation de cette propriété implique que lorsque les feux rouges s'allument alors qu'ils ne devraient pas, les feux blancs ne s'éteignent pas. Ce n'est pas très grave car il n'y a pas de risque vu qu'aucun train n'est impliqué. Pour empêcher la violation, il faudrait ajouter une condition pour passer de l'état `PNouvert` à `Avert_techn` qui prend en compte la mesure des feux rouges.

Propriété 18

Cette propriété sert à vérifier que les feux blancs sont commandés à l'extinction, lorsque :

- au moins une zone d'annonce est contrôlée hors service.

$$RHS(1) + RHS(2) = TRUE$$
$$\Rightarrow CLP = FALSE$$

FIGURE 4.10 – Propriété 18

Un model checking de ProB a découvert que cette propriété est violée. Elle est violée lorsque

- le passage à niveau est dans l'état `PNouvert`
- RHS1 RHS2 et RHSintern sont TRUE actuellement mais FALSE dans le cycle précédent
- les feux blancs sont commandés allumés (CLP = TRUE)

Le problème vient cette fois-ci de la modélisation. Dans la définition de CLP (commande des feux blancs), il y a la variable RHSintern (zone d'annonce contrôlée hors service). Dans un cycle, la mise à jour de la variable interne RHSintern se fait juste avant la mise à jour des variables de sortie. Cependant, la mise à jour dans le modèle se fait avec la valeur précédente de RHSintern plutôt que la nouvelle.

Cette propriété est donc violée à cause d'une erreur de modélisation et non une erreur du système. Cette erreur devra être corrigée et la propriété devra être testée à nouveau.

4.3.6 Commander les signaux sonores actifs

Propriété 20

Cette propriété est utilisée pour vérifier si la commande du signal sonore est bien activée lorsque les conditions suivantes sont réunies :

- passage à niveau commandé fermé par la logique d'annonce locale
- passage à niveau en mode "automatique"
- aucun contrôle des deux feux rouges d'au moins un signal routier

$$annZAX\_HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{m=1}^{=m}\overline{KLR(m,1)}.\overline{KLR(m,2)}\right)=TRUE$$
$$\Rightarrow CSB=TRUE$$

FIGURE 4.11 – Propriété 20

À l'aide de Rodin, on a pu prouver cette propriété. Il a fallu aider la preuve en faisant de la distinction de cas pour les états afin de déterminer le nouvel état. La commande pour le signal sonore est en partie lié au nouvel état mais aussi à certaines alarmes si le nouvel état est `Barrière_fermeture`.

La propriété est correcte, prouvée grâce à Rodin avec une aide manuelle.

Propriété 25

Cette propriété sert à vérifier si le signal sonore est commandé actif lorsque les conditions suivantes sont réunies :

- passage à niveau commandé fermé manuellement en mode "travaux"
- passage à niveau en mode "automatique"
- au moins une barrière contrôlée non fermée

$$FT.HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{r=1}^{=r}\overline{KFB(r)}\right) = TRUE$$
$$\Rightarrow CSB = TRUE$$

FIGURE 4.12 – Propriété 25

Cette propriété est vérifiée par le modèle. Rodin l'a validée. Il fallait aider la preuve en faisant de la distinction de cas pour l'état précédent et d'autres variables déterminant le nouvel état.

Cette propriété est donc vérifiée par Rodin

Propriétés 37 et 38

Ces propriétés sont utilisées pour vérifier si la commande du signal sonore est bien active quand il y a un problème de fermeture de barrière, c'est-à-dire quand les conditions suivantes sont réunies :

- passage à niveau commandé ouvert par l'SSI (Solid State Interlocking)
- la logique d'annonce locale ne détecte pas un train
- PN n'est pas commandé manuellement à la fermeture
- PN en mode 'automatique'
- au moins une barrière contrôlée non fermée à temps (pour la propriété 37) ou non fermée à temps dans un cycle précédent (pour la propriété 38)
- au moins une barrière contrôlée non ouverte.

$$NOTannFX$$
$$.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3}$$
$$.\left(\sum_{r=1}^{=r}\left(\overline{KFB(r)}.(Tb > Tb\_\max)\right)\right).\left(\sum_{r=1}^{=r}\overline{KOB(r)}\right) = TRUE$$
$$\Rightarrow CSB = TRUE$$

FIGURE 4.13 – Propriété 37

Les propriétés 37 et 38 sont violées dans le même contexte. Lorsque l'état passe en mode "ouverture de barrière" et que toutes les conditions à gauche de l'implication sont vraies, CSB peut être false. Pour que CSB soit vrai quand les barrières sont en cours d'ouverture, il faut que l'alarme 5 ou l'alarme 9 soit activée. Pour que l'alarme 5 soit active quand le passage à niveau n'est pas ouvert, il faut qu'au moins un des poteau-feux ait ses 2 feux rouges éteints.

$$
\begin{aligned}
&NOTannFX \\
&.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3} \\
&\quad .memoTb\_KFB.\left(\sum_{r=1}^{r=} \overline{KOB(r)}\right) \quad = TRUE \\
&\quad \Rightarrow CSB = TRUE
\end{aligned}
$$

FIGURE 4.14 – Propriété 38

Pour que l'alarme 9 soit active, il faut que le passage à niveau soit en mode fermé et qu'une des barrières ne soit plus contrôlée fermée. Par contre, l'alarme 6 est activée, mais celle-ci ne fait pas partie de la définition de CSB. Cette alarme permet de signaler que les barrières ne se sont pas fermées à temps, lorsqu'elles ont été commandées à la fermeture.

Ces propriétés sont donc violées. Rodin a tenté de prouver la propriété 37. Et un contre-exemple a été trouvé grâce à ProB. Ce même contre-exemple a rendu la propriété 38 fausse également.

La violation de ces propriétés implique que le signal sonore n'est pas actif lors de l'ouverture des barrières, s'il y a eu un soucis précédemment lors de la fermeture de celles-ci. Ce n'est pas très grave car lorsque les barrières s'ouvrent, il n'y a aucun train qui va arriver. Pour résoudre le problème, il suffit de modifier la définition de la commande du signal sonore (CSB) en ajoutant qu'il doit être également activé quand l'état est `Barrière_ouverture` et que l'alarme A6 est activée.

4.3.7 Commander les barrières à la fermeture

Propriété 42

Cette propriété sert à vérifier que la barrière est commandée à la fermeture quand les conditions suivantes sont réunies :

- passage à niveau commandé fermé par la logique d'annonce locale
- passage à niveau en mode 'automatique'
- déroulement délai de pré annonce

$$
\begin{aligned}
&annZAX.HS1.\overline{HS2}.\overline{HS3}.(T > Ttrans) = TRUE \\
&\quad \Rightarrow CB = TRUE
\end{aligned}
$$

FIGURE 4.15 – Propriété 42

T est remplacé par T15 car T n'est pas défini et Ttrans apparaît toujours avec T15 dans les spécifications.

La propriété est violée lorsque T15 vient d'être strictement plus grand que Ttrans, donc au moment de fin de la pré annonce. Dans ce cas-là, l'état est toujours `Avert_train` car la mise à jour de la variable d'état se fait avant la mise à jour de l'horloge T15. Et comme la variable de sortie CB est dirigée par l'état (qui doit être en mode `PNfermé` ou `Barrière_fermeture`), CB est encore à FALSE pendant une itération supplémentaire.

Cette propriété est donc violée. Un contre-exemple à celle-ci a été trouvé grâce à ProB.

Cette propriété n'est violée que lors d'un cycle. Les barrières ne vont commencer à se fermer seulement un cycle plus tard. Comme un cycle d'exécution est très court, ce n'est pas très grave. Pour rendre la propriété vraie, il faut encore se pencher dessus.

Propriété 41, 43, 44

Ces 3 propriétés sont très semblables à la propriété 42. Elles servent à vérifier que la barrière est commandée à la fermeture quand :

- le passage à niveau est commandé à la fermeture
- qu'assez de temps s'est écoulé dans cet état pour pouvoir commencer à baisser les barrières

La différence est la façon dont le passage à niveau est commandé à la fermeture.

- Propriété 41 : passage à niveau commandé fermé par l'SSI (Solid State Interlocking)
- Propriété 42 : passage à niveau commandé fermé par la logique d'annonce locale
- Propriété 43 : passage à niveau commandé fermé manuellement en mode "travaux"
- Propriété 44 : passage à niveau commandé fermé manuellement en mode "gardiennage"

$$annFX.HS1.\overline{HS2}.\overline{HS3}.(T > Ttrans) = TRUE \Rightarrow CB = TRUE$$

FIGURE 4.16 – Propriété 41

$$FT.HS1.\overline{HS2}.\overline{HS3}.(T > Ttrans) = TRUE$$
$$\Rightarrow CB = TRUE$$

FIGURE 4.17 – Propriété 43

$$\overline{HS1}.HS2.\overline{HS3}.(T > Trans) = TRUE$$
$$\Rightarrow CB = TRUE$$

FIGURE 4.18 – Propriété 44

Vu que la raison de la violation de la propriété 42 n'est pas lié à la première partie, mais à la deuxième partie ($T15 > Ttrans$), les propriétés 41, 43 et 44 doivent également être fausses dans certains cas. J'ai donc tenté de reproduire le cas où les propriétés sont violées avec ProB. Un cas a bien été trouvé et les propriétés sont donc violées.

Tout comme pour la propriété 42, ces propriétés ne sont violées que lors d'un cycle. Les barrières ne vont commencer à se fermer seulement un cycle plus tard. Comme un cycle d'exécution est très court, ce n'est pas très grave. Pour rendre les propriétés vraies, il faut encore se pencher dessus.

4.4 Critères de sécurité logique interne et signalisation ferroviaire

4.4.1 Signaux ferroviaires de couverture PN commander à la fermeture

Propriété 56

Cette propriété sert à vérifier qu'une alarme est bien activée quand :

- PN commandé fermé manuellement en mode 'gardiennage'
- au moins une barrière contrôlée non fermée à temps dans un cycle précédent.

$$\overline{HS1}.HS2.\overline{HS3}\,\text{memoTb\_KFB} = TRUE$$
$$\Rightarrow DA(1) = DA(2) = TRUE$$

FIGURE 4.19 – Propriété 56

Cette propriété ne semble pas correcte. En cherchant un contre exemple avec proB, elle pourrait-être vrai si memoTb_KFB => A6. En ajoutant cet invariant, on peut valider cette propriété.

Lors de la preuve de cet invariant, un obstacle est survenu. Dans memoTb_KFB il y a Tb < Tb_max, ce qui parait bizarre. Je me suis permis de remplacer le < par $\leq$.

Cette propriété est donc validée grâce à Rodin.

Propriété 61

Cette propriété sert à vérifier qu'une certaine alarme est bien activée lorsque :

- passage à niveau commandé ouvert par l'SSI (PN ouvert et perdre de KFB)
- la logique d'annonce locale ne détecte pas un train
- le passage à niveau n'est pas commandé manuellement à la fermeture
- le passage à niveau est en mode 'automatique'
- le passage niveau est en état 'ouvert'
- au moins une barrière contrôlée fermée

$$
\begin{aligned}
&NOTannFX \\
&.NOTannZAX.\overline{FT}.HS1.\overline{HS2}.\overline{HS3} \\
&\quad .\overline{CLR}.\left(\sum_{r=1}^{\#r} KFB(r)\right) = TRUE \\
&\quad\quad \Rightarrow DA(1) = DA(2) = TRUE
\end{aligned}
$$

FIGURE 4.20 – Propriété 61

Découverte de la violation de la propriété grâce à ProB. Dans l'état où la propriété est violée, les barrières sont contrôlées à la fois ouvertes et à la fois fermées (KOB et KFB valent TRUE). Les alarmes A15 et A16 sont donc passées à TRUE. A15 indique si une barrière est contrôlée ouverte, alors que toutes les barrières sont contrôlées fermées, et A16 indique si une barrière est contrôlée fermée, alors que toutes les barrières sont contrôlées ouvertes. Cependant, ces alarmes ne font pas partie de DA. Ce qui pourrait faire vérifier la propriété, c'est que par exemple A8 soit TRUE (qui elle fait partie de DA). A8 passe à TRUE quand le passage à niveau passe en Avertissement technique, ce qui n'est possible que si KOB vaut FALSE, or il n'y a aucun contrôle de la variable KFB à cet endroit de la machine d'état.

On a découvert que cette propriété est violée grâce à ProB.

Il s'agit d'un problème d'alarme lorsque les barrières sont contrôlées à la fois ouvertes et fermées (ce qui doit arriver très rarement). Pour solutionner le problème, il suffit d'ajouter les alarmes A16 dans la définition de la variable de sortie DA.

Propriété 62

Cette propriété sert à vérifier l'activation d'une alarme lorsque les conditions suivantes sont réunies :

- passage à niveau en état 'fermé' (autre état que `PNouvert`)
- occupation de longue durée (plus 600s)
- mode 'travaux' non actif.

$$(T > 600s).\ \overline{NT600}.\overline{FT}.HS1.\overline{HS2}.\overline{HS3} = TRUE$$
$$\Rightarrow DA(1) = DA(2) = TRUE$$

FIGURE 4.21 – Propriété 62

Pour vérifier la propriété, il faut remplacer 600s par Alarm_600 ainsi que T par T600. En effet, la variable T600 et le paramètre Alarm_600 sont les éléments qui permettent de vérifier que le passage à niveau ne reste pas trop longtemps hors de l'état `PNouvert`.

Après cette modification, la propriété a été facilement prouvée. En effet, lorsque le passage à niveau reste trop longtemps hors de l'état `PNouvert`, l'alarme A7 est passée à TRUE ce qui rend DA1 et DA2 TRUE.

Cette propriété est donc validée grâce à Rodin.

4.5 Propriétés relatives au RA/Ra

4.5.1 A tester uniquement sur les passages à niveau de deuxième catégorie

Propriété 112

Cette propriété sert à tester l'activation de l'alarme A8 lorsque :

- au moins une barrière n'est pas contrôlée ouverte
- le passage à niveau est ouvert

$$NOTannFX.NOTannZAX.HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{m=1}^{=m}\overline{KOB(m)}\right)$$
$$\Rightarrow \overline{RA(1)}.\overline{RA(2)}.\overline{Ra(1)}.\overline{Ra(2)}$$

FIGURE 4.22 – Propriété 112

La propriété est sensée tester l'alarme A8 (perte d'un contrôle de barrière ouverte, lorsque le passage à niveau est ouvert, donc que l'état doit être `Avert_techn`), or la partie gauche de l'implication est également valable pour le système en mode `Barrière_ouverture`.

Cette propriété est donc fausse et a été trouvé grâce à ProB.

La propriété est sensée tester l'activation d'une alarme dans certaines conditions. Or les variables utilisées dans la propriété ne reproduisent pas celles-ci. La violation de la propriété n'est donc pas grave, et il faut la modifier pour qu'elle vérifie les conditions souhaitées.

4.5.2 A tester sur les passages à niveau de deuxième et de troisième catégorie

Propriété 117 et 118

Ces propriété servent à tester l'activation de certaines alarmes :

- 117 : Test de l'alarme A18 lorsqu'un feu rouge détecté allumé en situation de `PNouvert`
- 118 : Test de l'alarme A1 lorsqu'on détecte l'absence de lampe blanche en cas de `PNouvert`

$$NOTannFX.NOTannZAX.\sum KLR(m,1)+KLR(m,2).\overline{\prod_{u=1}^{=u}\overline{RA(u)}.\prod_{u=1}^{=u}\overline{Ra(u)}}$$
$$\Rightarrow \prod_{u=1}^{=u}\overline{RA(u)}.\prod_{u=1}^{=u}Ra(u)$$

FIGURE 4.23 – Propriété 117

$$NOTannFX.NOTannZAX.HS1.\overline{HS2}.\overline{HS3}.\left(\sum_{m=1}^{m=m}\overline{KLP(m)}\right).\overline{\prod_{u=1}^{u=u}\overline{RA(u)}.\prod_{u=1}^{u=u}\overline{Ra(u)}}$$
$$\Rightarrow \prod_{u=1}^{u=u}\overline{RA(u)}.\prod_{u=1}^{u=u}Ra(u)$$

FIGURE 4.24 – Propriété 118

Un contre exemple à ces propriétés a été découvert grâce au model checking de ProB. Elles ne sont pas vraies dans le cas où la partie gauche de l'implication est vérifiée, mais que l'état n'est pas `PNouvert`. Il y a en effet moyen d'avoir un état `Avert_train` tout en respectant les conditions. Il n'y a pas assez de conditions pour s'assurer que CLX soit FALSE (ce qui est indispensable pour être dans l'état `PNouvert`). Il faudrait rajouter comme condition ($FT = FALSE$), car si on pose $FT = TRUE$, alors CLX est TRUE et on ne peut pas être dans le cas où le passage à niveau est ouvert car il est commandé à la fermeture. Pour la propriété 117, il faut également ajouter des conditions sur HS1 et HS2.

Ces propriétés sont donc violées et ça a été découvert grâce à ProB.

La violation de ces propriétés n'est pas grave car les conditions décrites ne sont pas assez précises pour reproduire celles voulues. Pour corriger les propriétés, il faut donc les modifier en ajoutant une condition sur FT (FT = FALSE) et pour la 117 également des conditions sur HS1 et HS2 (HS1 = TRUE et HS2 = FALSE).

4.6 Synthèse des résultats de la vérification

Un certain nombre de propriétés a donc pu être prouvée grâce à Rodin avec une aide manuelle ou non. Ce sont les propriétés 4, 10, 13, 14, 20, 25, 47, 56, 62, 68, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 88, 89, 90, 92, 100, 101, 102, 103, 104, 109, 110, 111 et 114.

Certaines autres propriétés présentes certains cas où elles sont violées, et ces cas ont été découverts grâce à ProB. Ce sont les propriétés 1, 7, 8, 9, 16, 17, 23, 37, 38, 41, 42, 43, 44, 61, 112, 117 et 118.

Le reste des propriétés n'a pas encore pu être vérifié.

Une partie des violations trouvées sont des cas limites où la propriété ne sera violée que lors d'un cycle et elle redevient correcte le cycle suivant (par exemple pour les propriétés 7, 8, 9, 41, 42, 43 et 44). Cela ne pose donc pas de vrais problèmes de sécurité.

Une autre partie des violations est liée aux alarmes (par exemple, pour les propriétés 37, 38, 61, 112, 117, 118). Cela prouve que :

- soit, il y a une erreur dans les spécifications des alarmes ou des variables les utilisant
- soit, il y a une erreur dans les propriétés vérifiant ces alarmes, c'est-à-dire que les conditions dans les propriétés ne sont pas correctes.

Chapitre 5

Travail futur

Le modèle a été terminé et toutes les propriétés ont été développés. Mais le travail n'est pas encore finalisé.

Dans le modèle, plusieurs hypothèses ont été posées pour le simplifier. Une étape future serait de modéliser les variables et paramètres délaissées dans le modèle existant. Il y a eu le nombre de barrières et de poteaux-feux qui ont chacun été limité à 2, alors qu'ils peuvent aller de 2 à 8. Pour valider complètement le modèle, il faut donc que chaque possibilité soit testée et que les propriétés soient vérifiées.

Le modèle contient certaines erreurs. Il faut donc le corriger afin qu'il reflète bien le système.

Toutes les propriétés n'ont pu être vérifiées. L'étape suivante serait de finaliser la validation des propriétés restantes. Pour se faire, on peut utiliser le modèle checking de ProB et l'exécuter jusqu'à ce qu'il ne trouve plus aucune propriété violée, et à ce moment-là, prouver les propriétés restantes avec Rodin en utilisant la même technique d'aide manuelle que pour les propriétés déjà prouvées dans le chapitre 4.

Comme certaines propriétés sont erronées, on pourrait adapter le système afin d'améliorer la sécurité grâce aux informations données dans le chapitre 4 lors de l'explication des propriétés violées. Il y a également, les valeurs des variables au moment où les propriétés sont violées qui sont disponible sur un lien github.

Une fois les corrections effectuées dans les spécifications du modèle et des propriétés, il faudra corriger le modèle. Et ainsi, pouvoir tenter de le valider à nouveau.

Chapitre 6

Conclusion

Le nouveau système de passage à niveau a été modélisé en Rodin. Ce modèle fonctionne et est capable de vérifier des propriétés de sécurité propres au passage à niveau.

Cependant, un bon nombre de propriétés ne sont pas respectées dans toutes les situations. Mais celles-ci sont en général très spécifiques, ce sont des cas limites ou des cas presque impossibles dans la réalité.

Pour commencer, il y a des erreurs de variables dans les propriétés (certains noms de variables n'étaient pas corrects comparés aux noms de variables décrits dans les spécifications). Ces erreurs ont été corrigées pour la vérification et n'ont donc pas eu d'impact sur le travail de vérification.

De plus, il y a également des propriétés qui ne sont violées que sur un cycle. En effet, les propriétés 41, 42, 43 et 44 sont violées seulement au cycle où le délai de pré-annonce vient d'être dépassé et le changement d'état se fera alors au cycle suivant.

Les propriétés 7, 8 et 9 sont violées sur un cycle, à cause du fait que la modélisation ne permet pas au passage à niveau d'atteindre directement l'état ouvert, lorsqu'il est dans certains états malgré que les conditions le permettent. Dans la réalité, ces propriétés sont donc vérifiées.

Ensuite, il doit y avoir un oubli dans la définition de la variable de la commande du signal sonore (CSB). Lorsque les barrières sont en cours d'ouverture, il faudrait que le signal sonore soit activé lorsque l'alarme A6 est activée pour que les propriétés 37 et 38 soient vérifiées. Cette alarme est activée lorsque les barrières ne se sont pas fermées à temps.

Dans l'état actuel des vérifications, on peut conclure, que le système est plutôt satisfaisant point de vue sécurité. Il y a bien sur encore quelques lacunes, mais celles-ci devraient pouvoir être corrigées afin d'avoir un système de passage à niveau encore plus sécurisée.

Bibliographie

[1] John J. Allan, J. Allan, C. A. Brebbia, A. F. Rumsey, G. Sciutto, and S. Sone. *Computers in Railways X : Computer System Design and Operation in the Railway and Other Transit Systems*. WIT Press, 2006. Google-Books-ID : tWjQCwAAQBAJ.

[2] Thierry Lecomte, David Déharbe, Étienne Prun, and Erwan Mottin. Applying a formal method in industry : A 25-year trajectory. pages 70–87, 11 2017.

[3] Simon Busard, Quentin Cappart, Christophe Limbree, Charles Pecheur, and Pierre Schaus. Verification of railway interlocking systems. *Electronic Proceedings in Theoretical Computer Science*, 184, 06 2015.

[4] Enclenchement. June 2020. Page Version ID : 171530401.

[5] Jean-Raymond Abrial. *Modeling in Event-B, System and Software Engineering*. 2010.

[6] Rodin User's Handbook v.2.8.

[7] Jens Bendisposto, Michael Leuschel, Olivier Ligot, and Mireille Samia. La validation de modèles event-b avec le plug-in prob pour rodin. *Technique et Science Informatiques*, 27 :1065–1084, 10 2008.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**