Introduction aux méthodes formelles
Principaux concepts

Walter SCHÖN

Méthodes formelles

2

- Terme regroupant une famille de méthodes permettant
  - L'expression des propriétés d'un logiciel sous différents formalismes mathématiques éliminant toute ambigüité.
  - En particulier l'expression de propriétés invariantes qui restent vérifiées en permanence (exemple : propriétés de sécurité),
  - La possibilité de passer d'un niveau d'abstraction élevé (spécification décrivant le besoin et non la solution) au concret (implantations où les algorithmes sont décrits) par un processus de raffinement,
  - Un processus de preuve permettant de démontrer mathématiquement la cohérence d'un niveau (respect des propriétés invariantes) et entre niveaux de raffinement.

Méthodes formelles

- Les méthodes formelles sont désormais quasi incontournables pour le développement de logiciels critiques (dont les dysfonctionnements peuvent avoir des conséquences graves sur les personnes, les biens ou l'environnement)
- Domaines d'application : Transport, Nucléaire, Défense, Aéronautique, Spatial…
- Les normes (ex. EN 50128 pour le ferroviaire) exigent pratiquement leur utilisation pour le logiciel critique
- La TVM (signalisation TGV), le SACEM (Paris ligne A), l'automatisation des lignes 14 et 1 de Paris sont des exemples de projets développés en méthodes formelles (et utilisant aussi le processeur codé)
- Ce cours introduit brièvement les principaux concepts.

Méthodes formelles : Preuve vs Test

- Test : repose sur le choix d'un jeu de tests pertinents mais jamais exhaustif => Le test peut révéler une faute de conception. Qu'il n'en révèle pas ne prouve pas qu'il n'y en n'ait pas.
- Preuve : démonstration mathématique faite pour l'ensemble du domaine de définition des variables. Est au test ce que « quel que soit » est à « il existe ».
- Mais :
  - N'est prouvé que ce qui est exprimé,
  - La preuve restreint les types utilisables (plus de réels, que des entiers),
  - Ne pas réussir à prouver un théorème ne prouve pas qu'il soit faux (pour mémoire : le Théorème de Gödel montre que l'indécidabilité est inhérente à tout système formel assez riche pour contenir l'arithmétique des entiers).

Limites de la preuve

En pratique les preuves sont faites automatiquement par un outil (le prouveur) qui contient :

- Des lemmes (axiomes de l'arithmétique, de la théorie des ensembles, mais aussi certains lemmes pouvant en être déduits par les règles d'inférence),
- Des règles d'inférence permettant de combiner les lemmes pour fabriquer des théorèmes.

A partir des propriétés souhaitées exprimées par les développeurs, le prouveur génère des obligations de preuves (futurs théorèmes s'il parvient ensuite à les prouver).

Le prouveur parvient généralement à prouver la plupart (80 à 90%) les obligations de preuve, dans de rares cas il parvient à les réfuter (prouver qu'elles sont fausses ce qui révèle une faute de conception).

Dans 10 à 20% des cas il ne parvient ni à prouver ni à réfuter.

Limites de la preuve

L'existence d'obligations de preuves résistant à la preuve automatique est généralement liée aux limites de l'outil (bien moins performant à l'heure actuelle que l'intelligence humaine dans l'activité de preuve mathématique).

Dans la plupart des cas on parvient à les prouver en guidant l'outil (mode interactif), ou en rajoutant des assertions (lemmes intermédiaires que le prouveur doit commencer par prouver).

Sinon la preuve est à faire par l'intelligence humaine.

L'ajout de nouvelles règles ou de nouveaux lemmes dans la base de l'outil est possible mais à manipuler avec prudence (risque de créer un système formel incohérent.

Il est en effet facile de montrer que si le système formel contient deux axiomes contradictoires, toute propriété est à la fois prouvable (=démontrable dans le langage des mathématiciens) et réfutable (= tout est vrai ainsi que son contraire).

Limites de la preuve

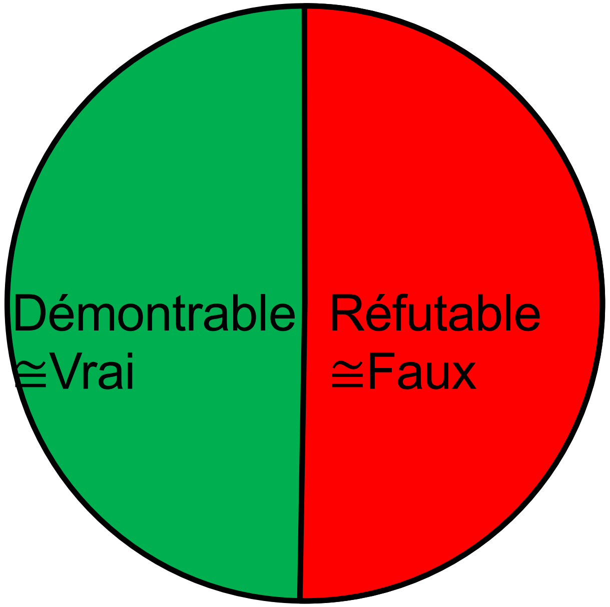
La limite théorique ultime est de toutes façons dans le (premier) théorème d'incomplétude de Gödel (1931) qui prouve que (énoncé approximatif bien que très proche de l'énoncé correct) :

- Tout système formel assez riche pour contenir l'axiomatique des entiers est :
  - Soit incohérent
  - Soit incomplet, à savoir qu'il y existe des propriétés dites « indécidables » qui ne sont ni démontrables ni réfutables

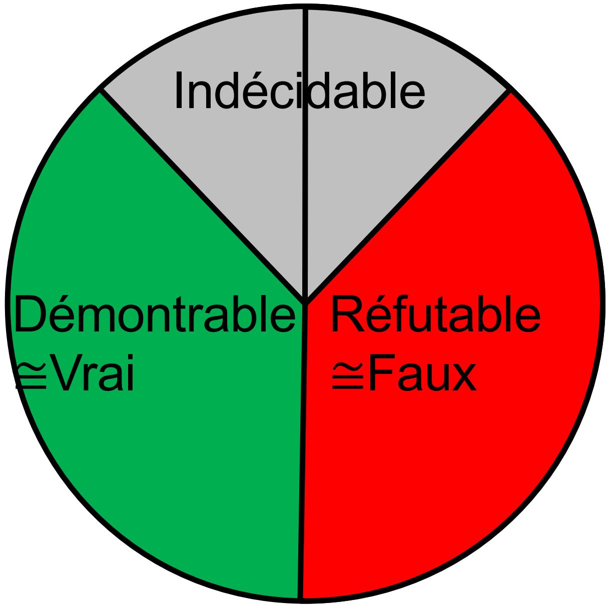
Cette incomplétude est incontournable car ayant exhibé une propriété indécidable (sa négation l'est donc également), si on ajoute un axiome permettant de la démontrer (ou de la réfuter), il sera toujours possible de fabriquer une propriété indécidable dans le système enrichi !

Limites de la preuve

Le théorème de Gödel met à mal le rêve de Hilbert (dans son deuxième problème en 1900) selon lequel l'arithmétique pouvait être entièrement axiomatisée par un système cohérent.



Mathématiques
selon Hilbert



Mathématiques
selon Gödel

Limites de la preuve

9 Quelques exemples de propriétés formelles :

- (Grand) théorème de Fermat : l'équation $x^n+y^n=z^n$ n'a pas de solution en nombre entier pour tout entier n>2 : énoncé au 17ème siècle par Fermat, démontré en 1994 par A. Wiles (avec une axiomatique beaucoup plus puissante que celle de l'époque),
- Conjecture de Goldbach : tout nombre entier pair à partir de 4 est somme de deux nombres premiers : supposée démontrable mais toujours non démontrée,
- Hypothèse du continu (premier problème de Hilbert) : il n'existe pas d'infini de taille intermédiaire entre celle des entiers (dénombrable) et celle des réels (continu) : énoncé par Cantor (1874) prouvée indécidable dans l'axiomatique actuelle (Cohen 1963),
- Et pour finir en apothéose : si un système formel est cohérent, alors la preuve de cette cohérence est une propriété indécidable dans ce système (second théorème d'incomplétude de Gödel).

Ambiguïtés et pièges du langage courant

Ambiguïtés liées à l'écrit ou à l'absence de ponctuation

- Dernière minute : plus de postes à l'UTC : plus aucun ou davantage ?
- L'étudiant dit le professeur est un âne : qui est l'âne ?

Mots à double sens

- Vous êtes mon *hôte* : qui reçoit qui ?
- Pour moissonner on utilise *toujours* la faux : forcément ou encore maintenant ?

Ambiguïtés dues aux tournures

- Il poursuit la jeune fille à vélo : qui pédale ?
- I know a man with a wooden leg named Smith. What is the name of the other leg? (Mary Poppins)

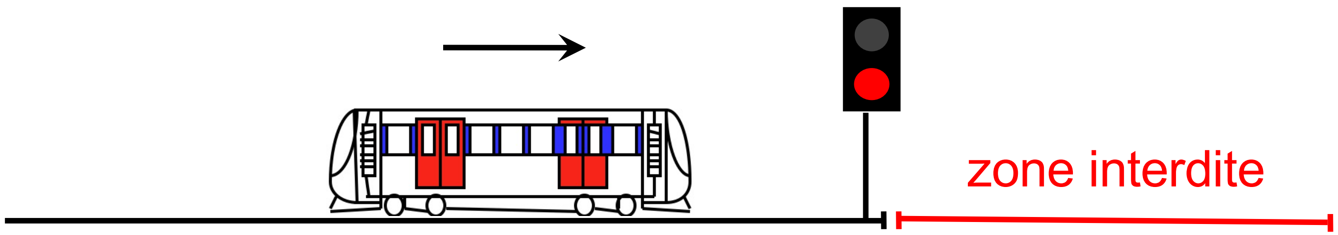
Ambiguïtés et pièges du langage courant

P «Tu sais»

- Opérateur not : ne… pas. Donc not(P) : «Tu ne sais pas»
- Alternative : opérateur not : être sans. Donc not(P) : «Tu es sans savoir»
- Mais par ailleurs not(«Tu sais») ⇔ «Tu ignores»
- •Donc «Tu n'ignores pas» ⇔ not(not(P)) ⇔ P «Tu sais»
- •«Tu n'es pas sans savoir» ⇔ not(not(P)) ⇔ P «Tu sais»
- •«Tu es sans ignorer» ⇔ not(not(P))⇔P «Tu sais»
- •«Tu n'es pas sans ignorer» ⇔ not(not(not(P)))⇔not(P) «Tu ne sais pas»
- •«Tu n'es pas sans ne pas ignorer» ⇔not(not(not(not(P)))) ⇔ P «Tu sais»

Informel vs formel

- Informel : Si le train pénètre dans la zone interdite alors un freinage d’urgence doit être déclenché

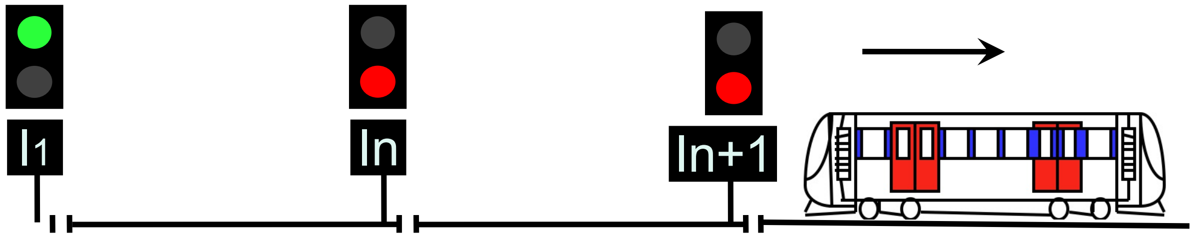


- Formel (basé sur la théorie des ensembles et la logique mathématique) : $Position\_train \subseteq POSITIONS \land$ $Zone\ interdite \subseteq POSITIONS \land$

  $Position\_train \cap Zone\_interdite \neq \emptyset \Rightarrow Frein\_urgence = TRUE$

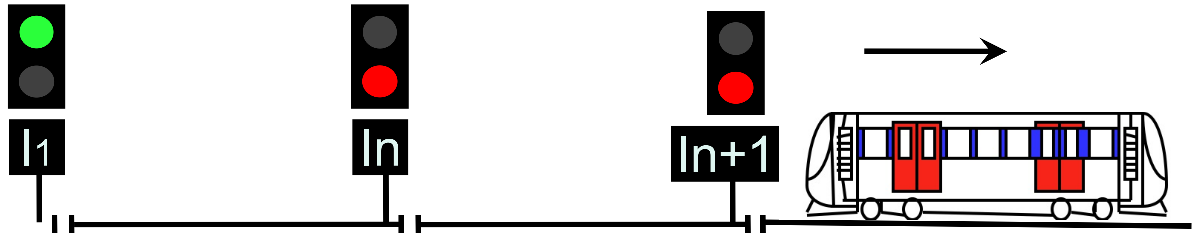
Formel vs informel

- Informel : La ligne est découpée en zones élémentaires appelées cantons. Ces cantons peuvent être occupés par au plus un train.



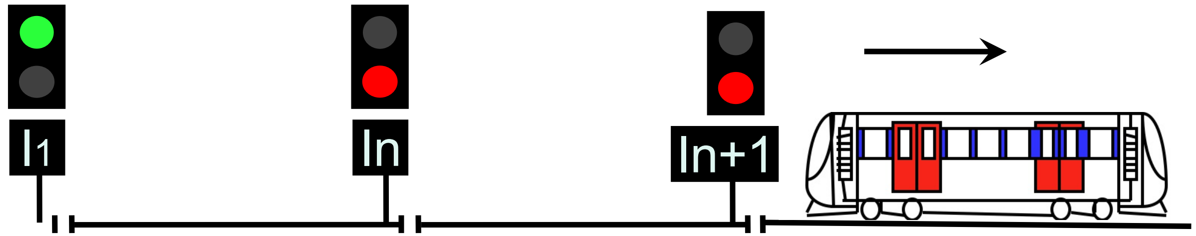
- Formel : Est_occupe_par ∈ CANTONS +-> TRAINS
- Lire : Est_occupe_par est une fonction partielle de l'ensemble des CANTONS dans l'ensemble des TRAINS
  - C'est une fonction (tout CANTON a au plus une image TRAIN)
  - Elle est partielle (un CANTON peut ne pas avoir d'image)

Développement formel : INVARIANTS



- De telles propriétés, qui doivent rester toujours respectées sont appelées INVARIANTS.
- Les propriétés de sécurité peuvent toujours être traduites sous formes d'INVARIANTS exprimées sous forme d'affirmations logiques appelées prédicats.
- Autres exemples :
  - A tout canton est associé un signal d'entrée : formalisable par une bijection de CANTONS vers SIGNAUX
  - Si un canton est occupé alors son signal d'entrée et le signal d'entrée du canton précédent est rouge : formalisable par une implication logique liant les occupations et les apparences de signaux

Développement formel : INVARIANTS

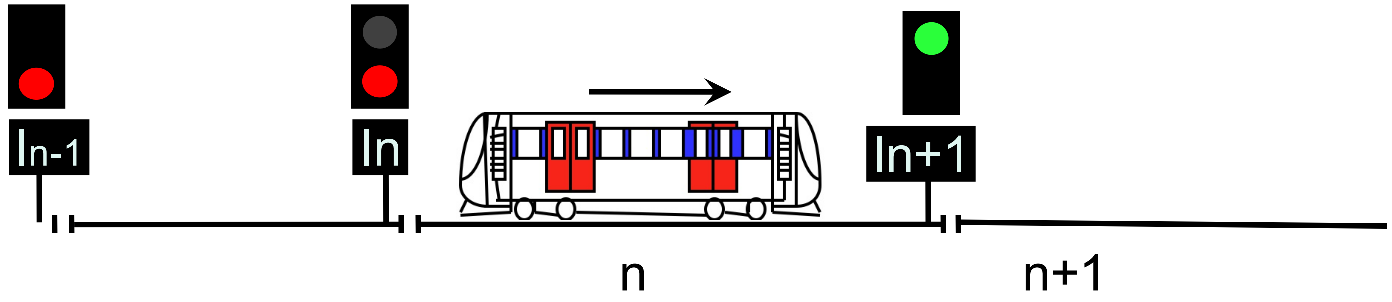


- La première étape (cruciale) du formel est donc la traduction des exigences (informelles) initiales (issues du cahier des charges etc.) en INVARIANTS formels.
- Cette étape est cruciale car tout oubli ou erreur à ce stade n'est évidemment par couvert par la suite du processus formel.
- En revanche toute propriété exprimée correctement entre dans un processus de preuve permettant de démontrer (mathématiquement) que tout le développement (jusqu'au code) la respecte…

Développement formel : OPERATIONS

- L'état du système est décrit par des VARIABLES qui peuvent donc évoluer.
- L'INVARIANT fait intervenir les VARIABLES mais aussi éventuellement des constantes (CONSTANTS) et des ensembles (SETS) qui peuvent rester abstraits au niveau de la spécification formelle.
- Les évolutions sont liées aux services rendus par le système, formalisées par des OPERATIONS.
- Ces OPERATIONS sont également décrites dans un langage formel appelé langage des substitutions (« substituant » les nouvelles valeurs des VARIABLES aux anciennes).
- Le processus de preuve consiste à montrer que toute OPERATION respecte l'INVARIANT.

Développement formel : OPERATIONS



OPERATIONS

Avance_train =

n := n+1 || I_{n+1} := ROUGE || I_{n-1} := VERT

noter la substitution en parallèle

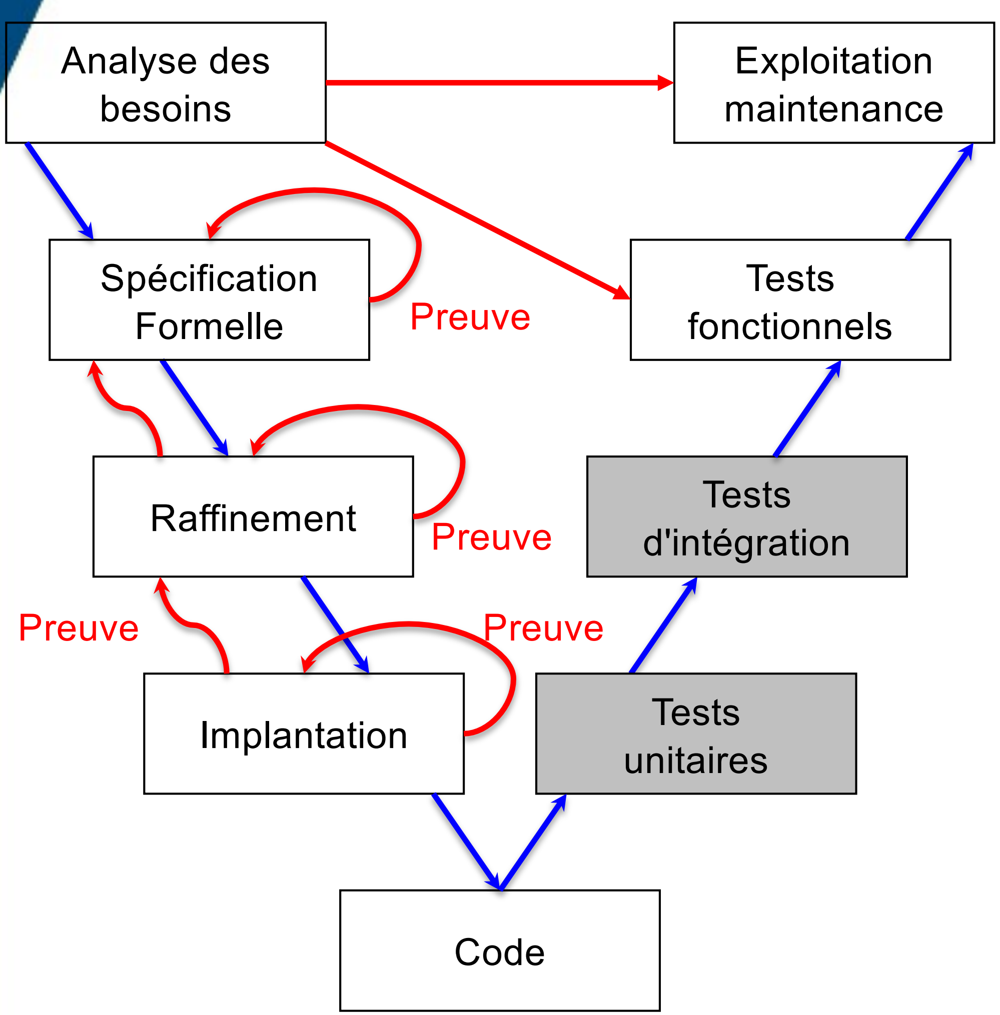
Obligations de preuve

- Chaque OPERATION génère une Obligation de Preuve (Proof Obligation : PO) dont il faudra montrer qu'elle est vraie, à la manière de la démonstration mathématique d'un théorème.
- Une PO d'opération traduit mathématiquement l'affirmation suivante :
  - Si l'INVARIANT est VRAI avant l'OPERATION alors il sera VRAI après l'OPERATION

Raffinement, implantation

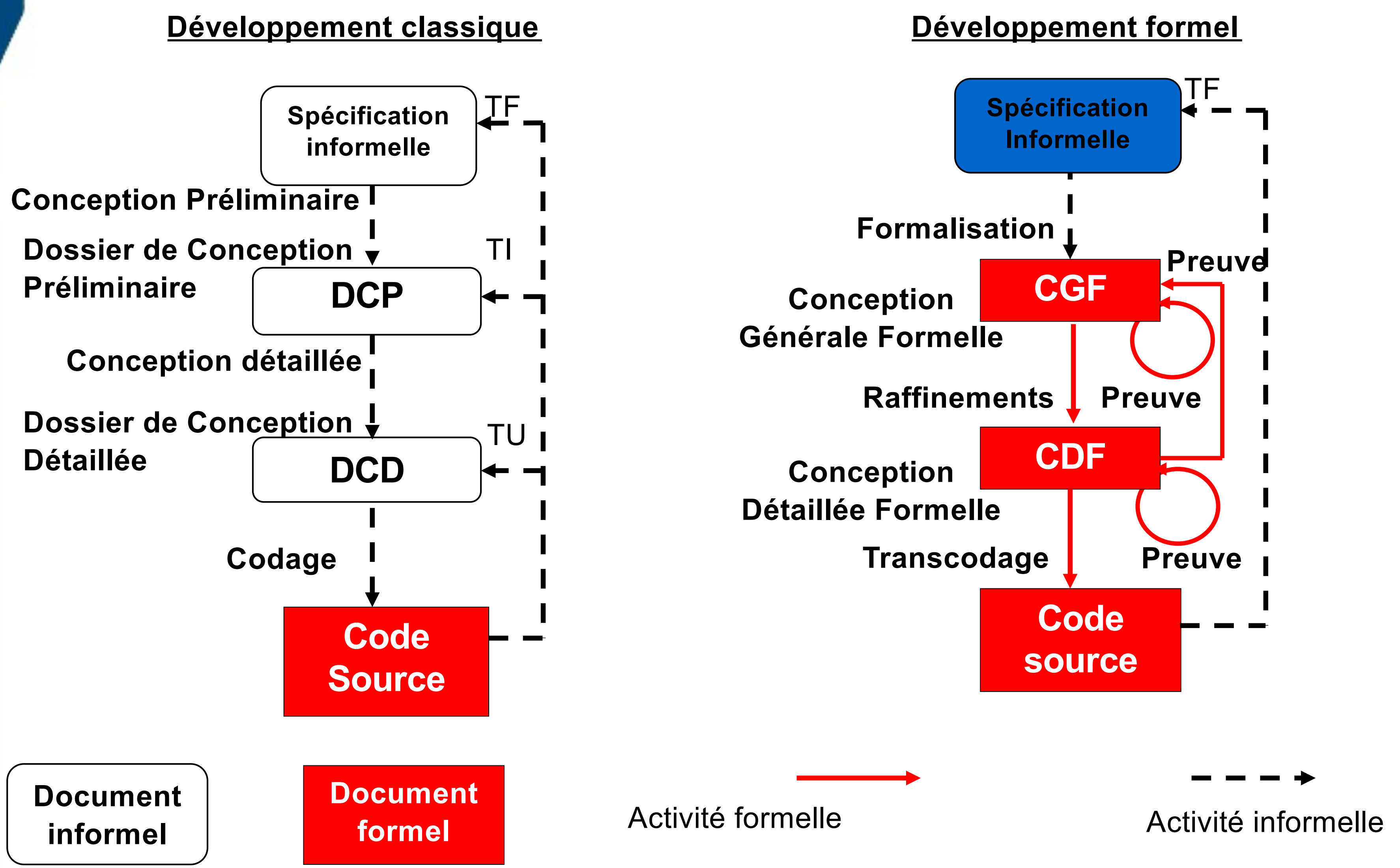
- Au niveau des spécifications, les OPERATIONS doivent rester au niveau du besoin (le « quoi ») sans préciser le « comment » (algorithmes etc.).
- A ce stade on peut donc avoir des opérations très abstraites et non déterministes (choix…).
- Le processus de raffinement (REFINEMENT) vise à préciser le « comment » jusqu'au raffinement ultime : l'implantation (IMPLEMENTATION).
- L'implantation est (pratiquement) le code final (transcodage automatique vers des langages type ADA ou C++).
- Chaque raffinement génère une PO qui traduit mathématiquement l'affirmation suivante :
  - L'opération raffinée est compatible (ne fait pas ce que ne ferait pas) avec l'opération que l'on raffine.

Processus formel

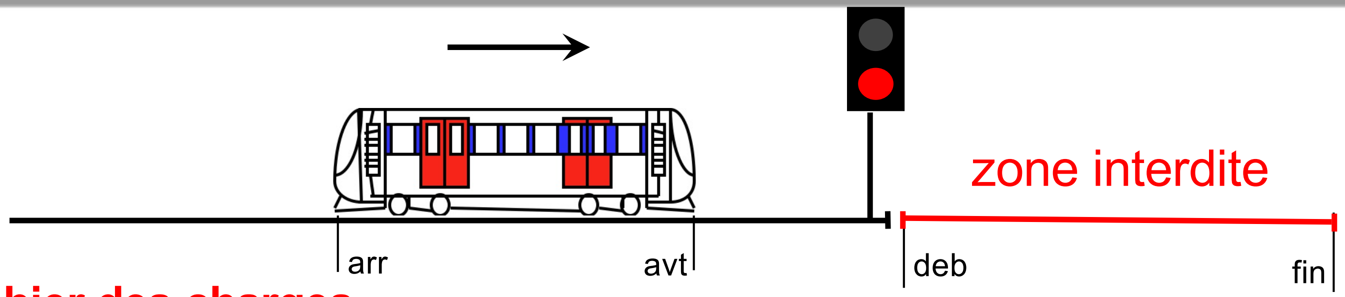


- Au final il est donc prouvé que le code respecte les invariants.
- Une grande partie des activités de vérification classique (tests unitaires et d'intégration) n'est donc plus utile.
- En revanche l'étape finale de validation d'un processus classique (tests fonctionnels) est maintenue, car prend en entrée des documents avant formalisation.

Développement classique vs formel



Exemple simplifié



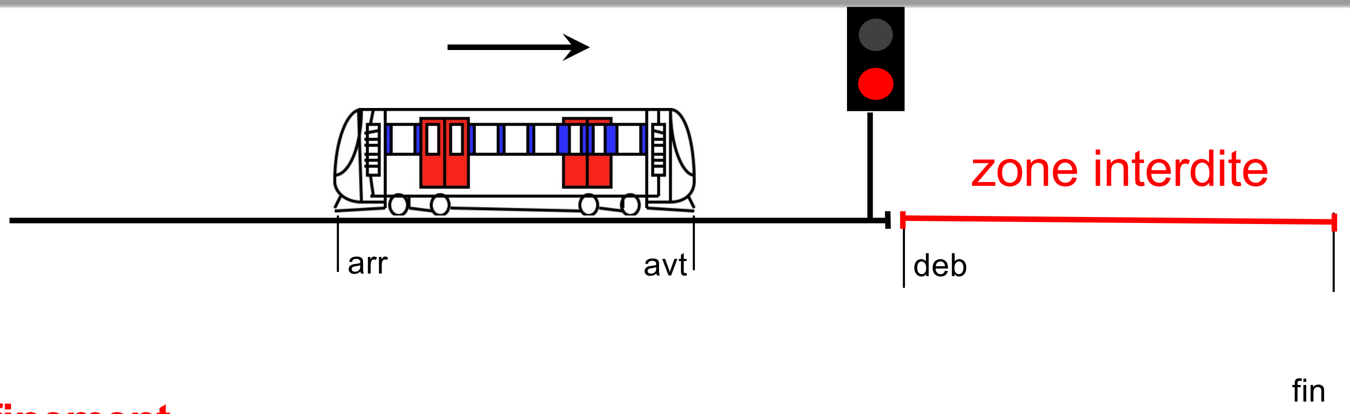
Cahier des charges

« Si le train pénètre dans la zone interdite
alors un freinage d'urgence doit être déclenché »

Spécification formelle

```
MACHINE ZI
SETS POS
VARIABLES position_train, zone_interdite, freinage_urgence
INVARIANT position_train <: POS & zone_interdite <: POS & freinage_urgence
: BOOL &
(zone_interdite /\ position_train /= {} => freinage_urgence = TRUE)
INITIALISATION position_train, zone_interdite, freinage_urgence
:(position_train <: POS & zone_interdite <: POS & freinage_urgence : BOOL &
(zone_interdite /\ position_train /={} => freinage_urgence=TRUE))
END
```

Exemple simplifié

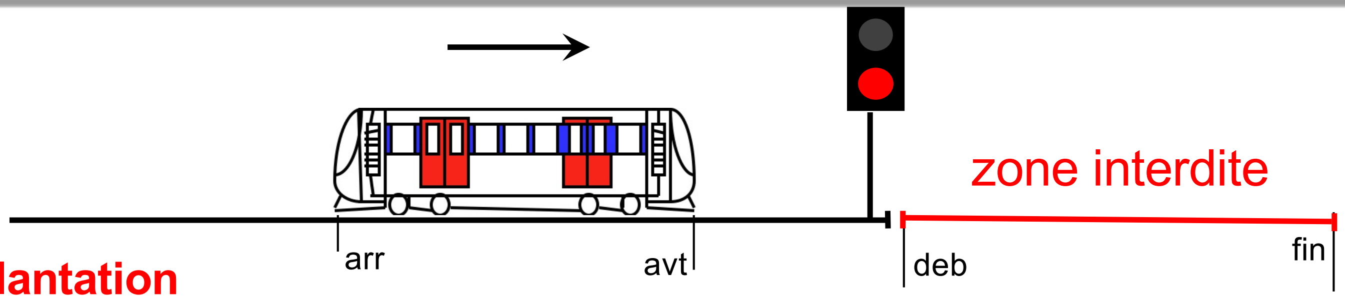


Raffinement

```
REFINEMENT   ZI_r1
REFINES   ZI
ABSTRACT_VARIABLES   position_train ,   zone_interdite ,   freinage_urgence
INITIALISATION
position_train, zone_interdite :(position_train<:POS & zone_interdite<:POS)
|| freinage_urgence := bool(position_train /\ zone_interdite /= {})
END
```

/* La machine abstraite se bornait à affirmer que l'initialisation respectait l'invariant. Le raffinement précise un peu les choses en donnant une formule explicite (mais non implantable) de la variable freinage_urgence*/

Exemple simplifié



Implantation

```
IMPLEMENTATION   ZI_r1_i
REFINES   ZI_r1
VALUES POS=0..100
CONCRETE_VARIABLES   arr, avt, deb, fin, freinage_urgence
INVARIANT arr : NAT & avt : NAT & deb : NAT & fin : NAT &
              position_train=arr..avt & zone_interdite=deb..fin
INITIALISATION arr:=0; avt:=10; deb:=50; fin:=80;
          IF deb>fin THEN freinage_urgence:=FALSE
          ELSIF avt<deb THEN freinage_urgence:=FALSE
          ELSIF arr>fin THEN freinage_urgence:=FALSE
          ELSE freinage_urgence:=TRUE
          END
END
```