



Logiques temporelles pour la spécification

UE « Modélisation et spécification » — 2021-2022

I

Modélisation -> Arnaud Sangnier
Spécification -> François Laroussinie

Page web du cours:

<https://www.irif.fr/~sangnier/enseignement/modspec.html>

Spécifier un système

système = programme, un algorithme, un protocole, ...

spécifier =

- ▶ énoncer la « correction » du système
- ▶ énoncer les propriétés attendues pour le système.
- ▶ énoncer l'absence de bug

Spécifier un système exemple

- des boutons pour choisir la boisson
 - des boutons pour fixer le niveau de sucre
 - un bouton annulation
 - un moyen de paiement
-
- un voyant pour chaque boisson
 - 4 voyants pour indiquer le niveau de sucre
 - un voyant pour indiquer que la boisson est en préparation et un bip pour signaler qu'elle est prête.
 - un voyant pour inviter à payer



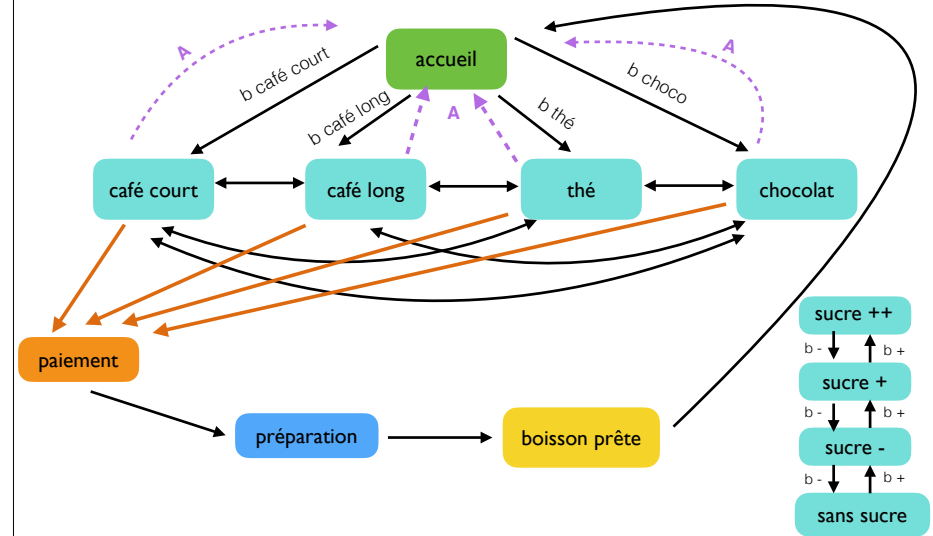
Spécifier un système exemple

- Initialement tous les voyants sont éteints.
- On sélectionne une boisson en appuyant sur le bouton correspondant.
- Dès qu'une boisson est sélectionnée, le voyant pour payer est allumé. Il s'éteint dès que la transaction a eu lieu.
- Après le paiement, la préparation commence (avec allumage du voyant correspondant) et un bip retentit à la fin de la préparation.
- Appuyer sur le bouton annulation avant le paiement revient dans l'état d'accueil.



Spécifier un système exemple

Une machine à café



Voir les cours d'Arnaud Sangnier
pour la construction de modèles.

C'est une étape cruciale !
Un modèle faux ne sert pas à grand chose...

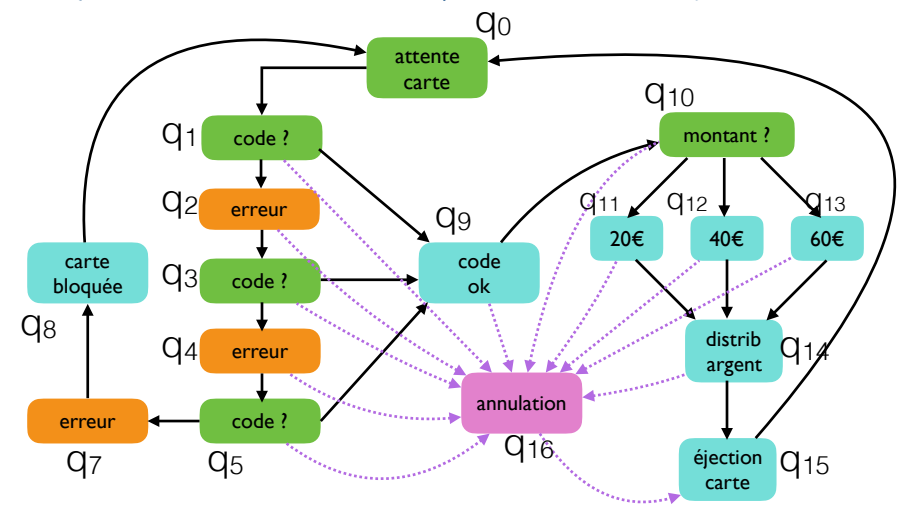
Ici on supposera toujours que l'on dispose d'un
« STE »: un système de transitions étiquetés.

$$S = (Q, Act, \rightarrow, q_0, AP, L)$$

Système de transitions étiquetés

$$S = (Q, Act, \rightarrow, q_0, AP, L)$$

$AP = \{att. carte, code?, c. bloquée, code ok, \dots\}$



NB: on ne construit pas un STE « à la main » !

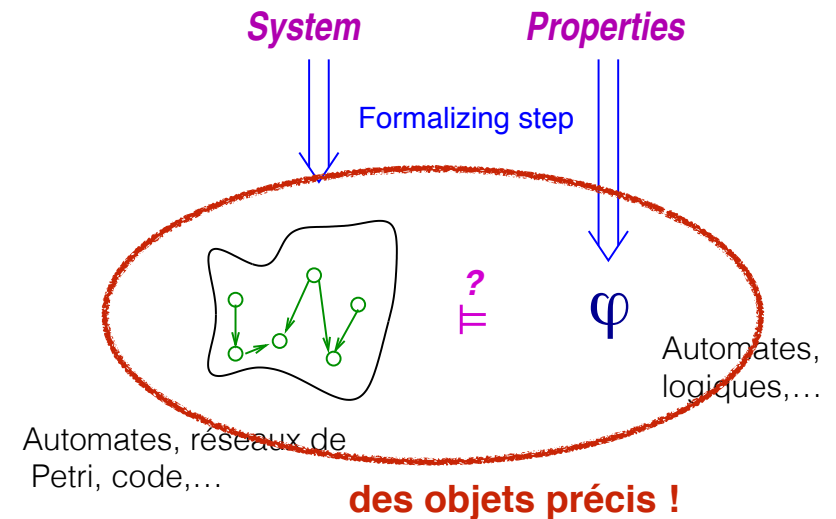
On utilise des langages de **haut niveau** pour décrire le comportement du système étudié (produit synchrone, réseau de Pétri, programmes, etc.).

Et ces langages ont une **sémantique définie sous la forme d'un STE**.

Même idée pour la spécification:

On utilise des langages de spécification avec une **sémantique précise** !

Model-checking



Exemple

algorithme d'exclusion mutuelle

```
boolean D1 := False
boolean D2 := False
```

```
Processus P1:
loop forever:
p1: Section NC
p2: D1 := True
p3: await (not D2)
p4: section critique
p5: D1 := False
```

```
Processus P2:
loop forever:
p1: Section NC
p2: D2 := True
p3: await (not D1)
p4: section critique
p5: D2 := False
```

+ hypothèse d'atomicité

→ un STE (voir la construction précise avec Arnaud Sangnier !)

Exemple

algorithme d'exclusion mutuelle

→ un STE avec $AP = \{D_1, D_2, SC_1, SC_2\}$

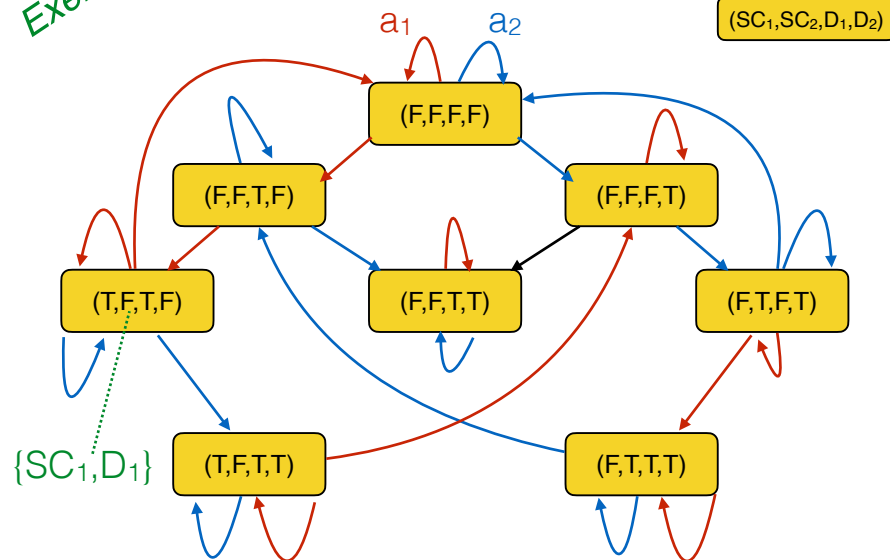
- ▶ D_1 étiquette les états où la variable D_1 est vraie,
- ▶ D_2 étiquette les états où la variable D_2 est vraie,
- ▶ SC_1 étiquette les états où le processus 1 est en section critique,
- ▶ SC_2 étiquette les états où processus 2 est en section critique.

Act = { a_1 , a_2 }

Et Q contient 8 états.

Exemple

algorithme d'exclusion mutuelle



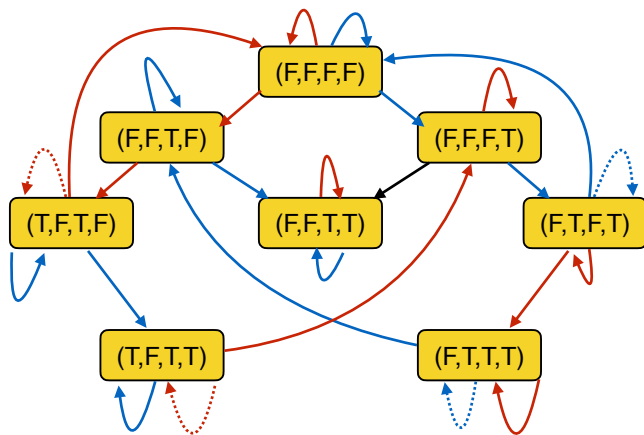
algorithme d'exclusion mutuelle

propriétés attendues

- **Exclusion mutuelle**: Jamais les deux processus ne peuvent se trouver en SC au même moment.
- Il n'y a jamais pas de **blocage**.
- **Absence de famine**: Si un processus demande l'accès à la SC, il y arrivera un jour.
- **Attente bornée**: Si un processus demande l'accès à la SC, l'autre processus ne peut pas passer avant lui plus d'une fois.

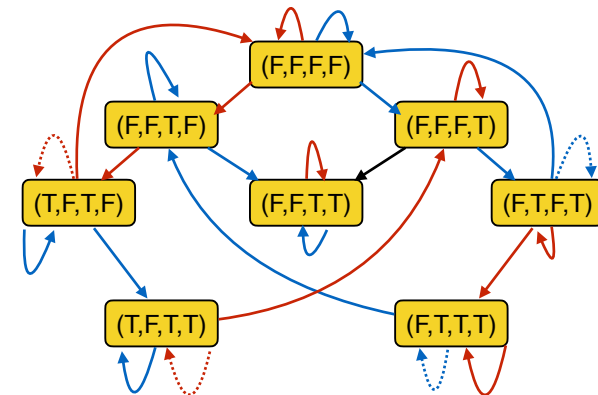
En général, on vérifie ces propriétés sous hypothèses d'équité entre processus et en supposant que chaque section critique se termine.

algorithme d'exclusion mutuelle



Équité = infinité de transitions bleues *et* de rouges.
 SC termine = infinité de transitions bleues *et* de rouges **pleines**.

algorithme d'exclusion mutuelle



+ équité
 + SC finies

→ un ensemble d'exécutions possibles du système modélisé.

Vérifient-elles les propriétés attendues ?

algorithme d'exclusion mutuelle

propriétés attendues

- ▶ **Exclusion mutuelle:** Jamais les deux processus ne peuvent se trouver en SC au même moment. ✓
- ▶ Il n'y a jamais pas de **blocage**. ?
- ▶ **Absence de famine:** Si un processus demande l'accès à la SC, il y arrivera un jour. ✗
- ▶ **Attente bornée:** Si un processus demande l'accès à la SC, l'autre processus ne peut pas passer avant lui plus d'une fois. ?

(→ l'algorithme n'est pas bon !)

Comment énoncer précisément ce genre de propriétés ?

- ▶ **Exclusion mutuelle:** Jamais les deux processus ne peuvent se trouver en SC au même moment.
- ▶ Il n'y a jamais pas de **blocage**.
- ▶ **Absence de famine:** Si un processus demande l'accès à la SC, il y arrivera un jour.
- ▶ **Attente bornée:** Si un processus demande l'accès à la SC, l'autre processus ne peut pas passer avant lui plus d'une fois.

Spécifier un distributeur de café

- **Initialement** tous les voyants sont éteints.
- On sélectionne une boisson en appuyant sur le bouton correspondant.
- **Dès qu'**une boisson est sélectionné, le voyant pour payer est allumé. Il s'éteint dès que la transaction a eu lieu.
- **Après** le paiement, la préparation commence (avec allumage du voyant correspondant) et un bip retentit à la fin de la préparation.
- Appuyer sur le bouton annulation **avant** le paiement revient dans l'état d'accueil.



Spécifier un distributeur de billets

- ▶ Si le bon code est donné, on peut choisir une somme et on obtiendra de l'argent.
- ▶ Après avoir fait trois erreurs de code, la carte est bloquée.
- ▶ Après avoir obtenu l'argent, la carte est éjectée.
- ▶ A tout moment, si on appuie sur annulation, la carte est éjectée sans donner d'argent sauf si l'utilisateur a fait trois erreurs avant.

Spécifier un système réactif

système réactif = système qui interagit avec un environnement.

- ▶ Il ne calcule pas un résultat en un temps fini.
- ▶ Il maintient une interaction avec son environnement.
- ▶ Sa correction se base sur l'ordre des actions/événements tout au long de son exécution.

avant/après/jusqu'à/depuis

+

si alors (\Rightarrow), et (\wedge), ou (\vee)

→ la logique temporelle

Logique propositionnelle

Syntaxe:

$P \in AP$

$\phi, \psi ::= P \mid \neg \phi \mid \phi \vee \psi \mid \phi \wedge \psi$

carte_éjectée \wedge argent_distribué

erreur \vee annulation

$\neg (CS_1 \wedge CS_2)$

$\neg CS_1 \vee \neg CS_2$

$(\neg CS_1) \vee (\neg CS_2)$

$D_1 \wedge \neg D_2$

L'évaluation d'une formule dépend de la valeur de vérité de chaque proposition atomique dans un état.

Logiques temporelles

Les logiques temporelles étendent la logique propositionnelle avec:

- des « modalités temporelles » (ou des « opérateurs temporels »)
- Elles sont interprétées dans des modèles munis d'une **notion de temps**.

Il y a de nombreuses logiques temporelles !
(et encore plus que ça !)

- LTL « Linear-time Temporal Logic »
- CTL « Computation Tree Logic »

LTL « Linear-time Temporal Logic »

Logique de temps linéaire.

Le comportement d'un système est vu comme l'ensemble de ses exécutions prises séparément.

La notion de temps (avant/après/jusqu'à/depuis) s'interprète donc le long d'une exécution d'un STE.

LTL- (premier fragment)

Syntaxe:

$P \in AP$

$\phi, \psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{X}^{-1}\phi \mid \mathbf{F}^{-1}\phi$

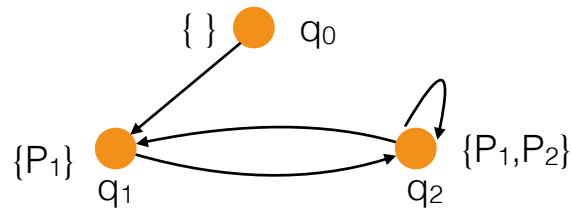
$\mathbf{X}\phi$: demain ϕ

$\mathbf{F}\phi$: un jour dans le futur ϕ

$\mathbf{X}^{-1}\phi$: hier ϕ

$\mathbf{F}^{-1}\phi$: un jour dans le passé ϕ

Exemple



$\text{Exec}(q_1) = \{ (q_1 q_2 q_1 q_2 \dots, \dots, q_1 q_2 q_2 q_2 \dots, \dots) \}$

ie $\text{Exec}(q_1) = (q_1 q_2^+)^+ q_2^\omega \cup (q_1 q_2^+)^{\omega}$

STE

$S = (Q, \text{Act}, \rightarrow, q_0, AP, L)$

NB: ici on n'utilisera pas les actions sur les transitions (Act).

NB: on supposera que tout état a au moins un succ. par \rightarrow .

Exécutions:

séquence infinie $\rho = s_0 s_1 s_2 s_3 \dots$ telle que:

$\triangleright s_i \in Q \quad \forall i = 0, 1, 2, \dots$

$\triangleright (s_i, s_{i+1}) \in \rightarrow \quad (\text{ou } s_i \rightarrow s_{i+1})$

Notation: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots \rightarrow s_k \rightarrow$

et $\rho(i) = s_i$, $\rho^i = s_i s_{i+1} s_{i+2} \dots$ $\rho|_i = s_0 s_1 \dots s_i$

$\text{Exec}(q) =$ ensemble des exécutions issues de q .

LTL- (premier fragment)

Syntaxe:

$P \in AP$

$\phi, \psi ::= P \mid \neg\phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X}\phi \mid \mathbf{F}\phi \mid \mathbf{X}^{-1}\phi \mid \mathbf{F}^{-1}\phi$

$\mathbf{X}\phi$: demain ϕ

$\mathbf{F}\phi$: un jour dans le futur ϕ

$\mathbf{X}^{-1}\phi$: hier ϕ

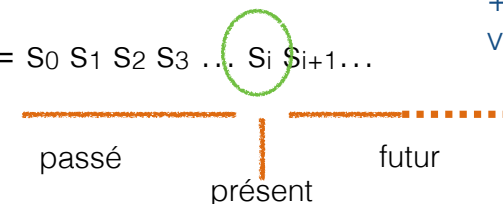
$\mathbf{F}^{-1}\phi$: un jour dans le passé ϕ

\rightarrow on interprète les formules de LTL- sur une position

i le long d'une exécution ρ d'un STE.

+L pour les
val. des AP.

$\rho = s_0 s_1 s_2 s_3 \dots s_i s_{i+1} \dots$



LTL- (premier fragment)

Syntaxe:

$P \in AP$

$\phi, \psi ::= P \mid \neg \phi \mid \phi \vee \psi \mid \phi \wedge \psi \mid \mathbf{X} \phi \mid \mathbf{F} \phi \mid \mathbf{X}^{-1} \phi \mid \mathbf{F}^{-1} \phi$

Sémantique:

soit ρ une exécution d'un STE $\mathbf{S} = (Q, \text{Act}, \rightarrow, q_0, AP, L)$

soit i un entier ≥ 0

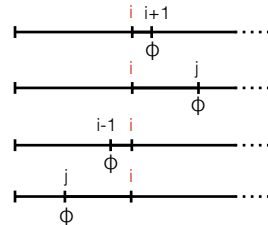
$\rho, i \models P$ ssi $P \in L(\rho(i))$

$\rho, i \models \mathbf{X} \phi$ ssi $\rho, i+1 \models \phi$

$\rho, i \models \mathbf{F} \phi$ ssi $(\exists j \geq i. \rho, j \models \phi)$

$\rho, i \models \mathbf{X}^{-1} \phi$ ssi $(i > 0 \wedge \rho, i-1 \models \phi)$

$\rho, i \models \mathbf{F}^{-1} \phi$ ssi $(\exists j \leq i. \rho, j \models \phi)$



LTL-

$\mathbf{S} = (Q, \text{Act}, \rightarrow, q_0, AP, L)$

$\rho, i \models \phi$ est désormais défini !

Et $\mathbf{S} \models \phi$?

Rappel:

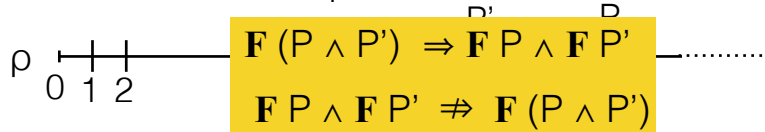
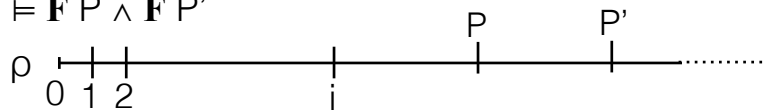
Avec les logiques de temps linéaire, le comportement d'un système est vu comme l'ensemble de ses exécutions prises séparément.

$\mathbf{S} \models \phi$ si et seulement si $\rho, 0 \models \phi \quad \forall \rho \in \text{Exec}(q_0)$

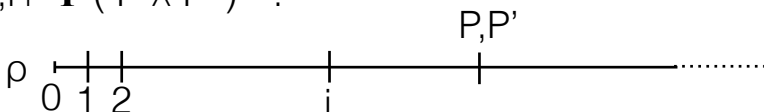
Exemples de formules

Comparer $\mathbf{F} P \wedge \mathbf{F} P'$ et $\mathbf{F} (P \wedge P')$

$\rho, i \models \mathbf{F} P \wedge \mathbf{F} P'$

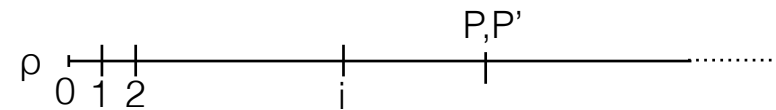
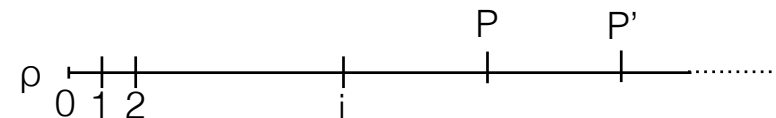


$\rho, i \models \mathbf{F} (P \wedge P')$?

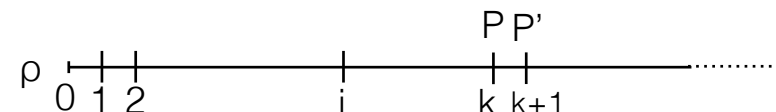


Exemples de formules

$\rho, i \models \mathbf{F} (P \wedge \mathbf{F} P')$?

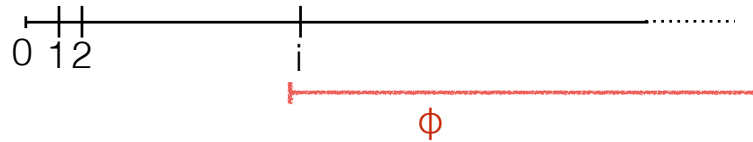


$\rho, i \models \mathbf{F} (P \wedge \mathbf{X} P')$?



Exemples de formules

$\rho, i \models \neg \mathbf{F} \neg \phi$??

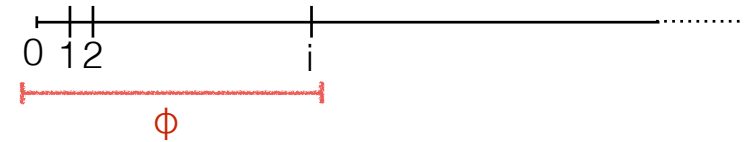


→ ϕ est vrai pour tous les états $i, i+1, i+2, \dots$
 « toujours dans le futur »

On le note: $\mathbf{G} \phi = \neg \mathbf{F} \neg \phi$

Exemples de formules

$\rho, i \models \neg \mathbf{F}^{-1} \neg \phi$??



→ ϕ est vrai pour tous les états du passé: $i, i-1, i-2, \dots$

« toujours dans le passé »

On le note: $\mathbf{G}^{-1} \phi = \neg \mathbf{F}^{-1} \neg \phi$

Exemples de formules

$\rho, i \models \neg \mathbf{X} \phi$

$\iff \rho, i \models \mathbf{X} \neg \phi$ (car exécutions infinies)

(sans cette hypothèse, on aurait:

$\rho, i \models \mathbf{X} \neg \phi \Rightarrow \rho, i \models \neg \mathbf{X} \phi$)

Exemples de formules

\mathbf{G} (problème $\Rightarrow \mathbf{F}$ alarme)

\mathbf{G} (alarme $\Rightarrow \mathbf{F}^{-1}$ problème)

\mathbf{G} (request $\Rightarrow \mathbf{F}$ service)

$\mathbf{G} (\neg \text{bug})$

Exemples de formules

G F accueil

F G ok

GF request \Rightarrow **GF** service

GF (a \wedge b) implique **GF** a \wedge **GF** b

GF a \wedge **GF** b n'implique pas **GF** (a \wedge b)

G a \Rightarrow **FG** a \Rightarrow **GF** a \Rightarrow **F** a