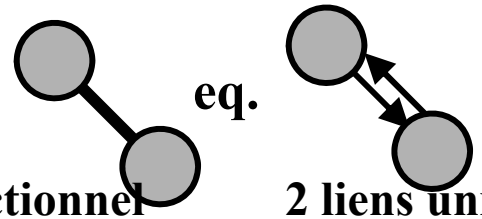
Algorithmique Distribuée
Modèle

Janna Burman
janna.burman@lisn.fr

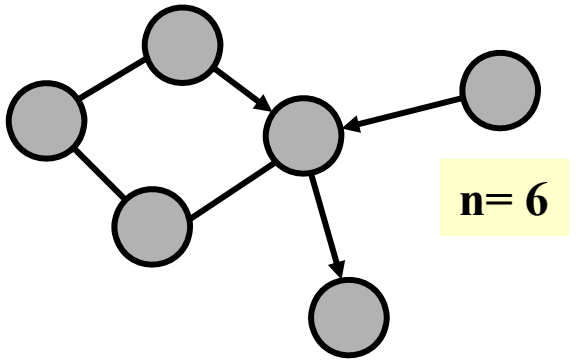
Graphe/topologie de réseau de communication (1)

Représenté par un graphe dirigé ***G(V,E)*** :

- l'ensemble ***V*** de nœuds du réseau représente les entités calculant et communicant, appelés ***processus***
- l'ensemble ***E*** = {(i,j) : i,j ∈ V} d'arêtes dirigés représente les **liens/canaux de communication** unidirectionnels (qui peuvent transporter des messages)
- Par défaut, le graphe de communication sera non-dirigé (avec des liens bidirectionnels uniquement)
- |V| = ***n***



lien bidirectionnel **eq.** **2 liens unidirectionnel**



Rappel : notion de la théorie de graphes

- Le ***degré d'un nœud*** est le nombre d'arêtes incidentes (pour des graphes dirigés, c'est la somme de degrés (d'arêtes) entrant(es) et sortant(es)).
- Les ***voisins*** d'un nœud p sont tous les nœuds q avec qui il existe une arête entre p et q ou q et p. Les voisins connectés seulement par une arête unidirectionnel pq appelés *successeurs* et ceux par qp sont appelés *prédécesseurs*.
- Un ***chemin de p à q*** est une séquence p = s0, s1, …, sk = q des nœuds t.q. pour tout i entre 0 et k il existe une arête (dirigé) de si à si+1.
- Un ***cycle*** (simple) est un chemin de p à p (sans répétition des nœuds, à part de p).
- Le ***graphe est dit connexe*** (fortement) si pour tous nœuds p et q il existe un chemin (dirigé) entre ces deux. Par défaut, on ne considère que des graphes/réseaux connexes.
- Une ***distance*** entre p et q, noté **d(p,q)** ou **dist(p,q)**, est la longueur de chemin le plus court entre p et q.
- Un ***diamètre*** du graphe/réseau, noté par **Diam** ou **diamètre**, est la distance maximum entre toute paire de nœuds.
- Topologies d'exemple : anneau, arbre, étoile, réseau complet, grille …

Liens

- Chaque lien de communication représenté par une arête dirigée dans *E* peut contenir soit des messages d'un ensemble définit (de type défini) soit être vide

- **Un canal/lien a un *état***, qui est le multi-ensemble des messages en transit dans le canal (ceux qui étaient émis, mais pas encore reçus).
  - Si les canaux conservent l'ordre des messages (**canaux *FIFO***) l'état est une suite de messages.

Processus

- entité indépendante de communication et de calcul (avec sa mémoire dédiée)
- composé de :
  - Son **programme (algorithme) local** (déterministe ou non-déterministe, mais pas probabiliste, dans ce cours)
    - donnés sous forme d'un ***pseudo-code***, pas forcement liés à un langage particulier, mais, de façon très libre.
    - comprend toutes les structures d'un langage habituel : affectation (:=), branchements conditionnels, … et aussi des primitives de communication : ***envoyer*** et ***recevoir***.
    - a des variables locales, qu'il est le seul à pouvoir les accéder. Nous appellerons ***état*** d'un processus la suite des valeurs de ses variables à un moment donné.
  - l'équipement pour communiquer avec les autres processus (i.e. pour réaliser *envoyer* et *recevoir*) et ***l'interface avec la couche de communication*** (e.g. avec liaison de données)
  - ***l'interface avec la couche d'application extérieure /environnement***, qui est juste au-dessus de couche algorithmique

Evénements et actions

- Le calcul qui fait le processus dépend de l'algorithme décrit par son programme, de son état courant, et des ***événements* qui sont détectables ou réalisés par le processus** comme :
  - *les* ***événements d'entrées/sorties*** (de/à la couche d'application extérieure /environnement; e.g., réveil de processus, passage de données, tic d'horloge)
  - *les* ***événements internes*** définis dans le programme (e.g. satisfaction d'un prédicat)
  - *les* ***événements d'envoie et de réception*** **de messages.**

- On peut aussi distinguer entre les ***actions de processus décrits dans le programme*** par :
  - les ***actions internes*** (calcul interne modifiant son *état* interne/local – de valeurs de variables locales),
  - *les* ***actions de communication*** (envoie et réception de messages, qui modifient l'état d'un canal ainsi que l'état d'un processus - pour recevoir),
  - *les* ***actions d'entrée/sortie*** (lecture/écriture des paramètres/données de/à l'environnement/application).

Événement de *Réveil*

- *Réveil* de processus peut être : ***spontané*** ou ***provoqué*** et *simultané* ou pas.
- **Par défaut**, on va supposer que **au moins un processus se réveille *spontanément*** dans le réseau, pour commencer le calcul distribué (l'exécution).
  - On peut imaginer que c'est l'environnement qu'envoie un signal pour réveiller certaines processus, ou tout simplement que ces processus se réveillent *spontanément*.
- Les autres processus qui ne se réveillent pas de cette façon, se réveillent sur la réception d'un premier message, dit de façon ***provoquée***.
- Dans certain cas, on va aussi supposer que tous les processus se réveillent ***simultanément*** (de façon *spontanée*).

Primitives *Envoyer* et *Recevoir*

- ***envoyer (m, destinataire)*** ou *envoyer(m)* à *destinataire*
  - *m* est une variable locale, de type message défini, contenant le message à envoyer
  - *destinataire* est un identificateur d'un processus *voisin* ou d'un canal sortant
  - On peut imaginer que l'entité message contenant m envoyée au temps t, se déplace le long du canal de communication jusqu'à arriver à destination au temps t + D. Nous appelons D le ***délai d'acheminement/transmission du message***.

- ***recevoir (y, canal)*** ou *recevoir(y)* sur *canal*
  - *y* est une variable locale, de type message défini
  - *canal* est un paramètre optionnel et désigne un identificateur d'un processus voisin ou d'un canal entrant
  - Si un message est arrivé de/sur *canal*, alors l'information contenue dans le message va dans la variable y
  - Sinon, et on dit que la réception est ***bloquante***, le processus est mis en attente, jusqu'à ce que l'interruption signalant l'arrivée du message le rende de nouveau actif. Il se débloque et reçoit le message comme précédemment.

Transmission des messages et systèmes *synchrone* et *asynchrone*

- Si le **délai de transmission** des messages **admet une borne supérieure fixe et connue** (ou si tout message arrive exactement en un temps fixe), on dit que la ***transmission des messages ou le système elle-même est synchrone***, et ***asynchrone* si ce n'est pas le cas**.
  - Transmission asynchrone : le délai de transmission est fini, mais pas borné
  - Dans le cas où le délai de transmission des messages admet une borne supérieure fixe et inconnue, la transmission dite ***asynchrone de délai borné***.

- **Des systèmes synchrones sont sauvent modélisés de fonctionner par round**:
  - les processus ont des horloges locales sans dérive et parfaitement synchronisées et
  - le délai de transmission des messages admet une borne supérieure fixe,
  - → tel que on peut organiser les exécutions en séquence infinie de ***round synchrones :***
    - à chaque round (au même temps pour tout processus), chaque processus d'abord envoie des messages à ces voisins (si il y en a),
    - puis reçoit des messages venant de ses voisins et exécute les actions de l'algorithme.

- Dans la vie réelle, la transmission des messages est-elle synchrone ou asynchrone ?

Règles gardées

- Pour spécifier le pseudo-code d'un processus nous utilisons sauvent le format des règles gardées.
- Une règle gardée est constituée d'une condition **G** appelée ***garde*** et un ensemble d'***actions*** **A** et notée **G → A**.
- Cala est pour décrire les actions conditionnées et des réactions aux événements, dans le programme de processus.
- **Les actions A sont exécutées si G est satisfaite** (et quand le processus est activé).
- **Si plusieurs gardes (de plusieurs règles gardées) sont satisfaites**, une de règles est choisie de façon non-déterministe (ou suivant des priorités, si données) pour être activée.
- **Une garde** peut décrire un événement comme [lors de réception d'un message m] ou [lors de réveil] ou un événement de satisfaction de prédicat comme [b = true et m = 'ack'] …

Pas d'un processus

- **Chaque pas** est **une réaction à un événement et décrit par les actions** du programme (algorithme).
  - Il est **supposé d'exécuter de façon atomique/instantanée** (sans interruption).
  - Correspond aux actions d'une règle gardée
- **Le prochain *pas*** d'un processus **dépend de l'état de processus et de ses événements (et l'information qui vient avec**, comme celle de messages reçus ou d'instructions envoyées par l'environnement).
- Le **résultat d'un pas** :
  - par les actions internes, peuvent **modifier** seulement **l'état du processus**
  - par les actions de communication (envoyer et recevoir), **modifient l'état d'un canal** ainsi que l'état du processus (pour recevoir)
  - par les actions d'entrées, peuvent modifier l'état du processus

Configuration et *Exécution*

- Pour les analyses d'un algorithme, il est pratique de considérer le vecteur des états de tous les processus et de tous les canaux, qu'on appelle ***configuration***.

- On peut voir le système évoluer de configuration en configuration sous l'effet des actions internes et de communication (résultants des pas de processus). Une telle evolution est appelée ***exécution***.

Problème et son algorithme correct

Problème est un prédicat P sur les exécutions (qui définit l'ensemble d'exécution admissibles)

Algorithme/Système A résout un problème P (dit aussi **correct**) si toute exécution de A satisfaite le prédicat P (ou est dans l'ensemble P)

Terminaison

- Une ***configuration terminale*** est une configuration qui ne change plus, i.e. à partir de quelle, aucun message ne circule / ne pas en transit, et aucune action (n'est applicable) ne peut changer l'état d'aucun processus.

- La ***terminaison est atteinte*** quand (dans l'exécution) la configuration terminal est atteinte.
  - A priori la terminaison est ***implicite***, car dépend de la configuration (de l'état global) du système.
  - Les processus peuvent détecter la terminaison, i.e. passer à l'*état* particulier appelé *terminal* et/ou terminer son programme. Dans ce cas on dit que la terminaison est ***explicite*** pour les nœuds qui la détectent.

Mesures de complexité (1)

- **Les exécutions d'algorithme/système distribué consomment des ressources** du réseau (liens de communication, temps de calcul, mémoire, energie …). L'analyse de complexité d'un algorithme détermine la quantité utilisée par une ressource particulière pendant les exécutions.

- **Par défaut**, on s'intéresse à l'analyse de complexité **de cas pire** →
  - **le maximum de ressources consommées sur toutes les exécutions possibles**
  - **jusqu'à la terminaison, si atteinte**
  - **ou jusqu'à qu'un autre ensemble de configurations,** (qui est sauvent défini via la définition du problème), **est atteint**.

Mesures de complexité (2)

Les complexités d'algorithme sont généralement exprimées :

- **en termes (fonction) des paramètres qui décrivent la taille du problème** : n, |E|, diamètre…

- **en ordre de grandeur**

  Soit f et g deux fonctions des entiers aux entiers.

  - **f = O(g)**, s'il existe un constant c>0 t.q. pour tout x à partir d'un x>x1, f(x) <= c g(x).
  - **f = Ω(g),** s'il existe un constant c>0 t.q. pour tout x à partir d'un x>x1, f(x) >= c g(x).
  - **f = Θ (g),** si f est simultanément O(g) et Ω(g).

Mesures de complexité de communication et de mémoire

- ***Complexité en communication*** :
  - ***Complexité en messages*** : mesurée en nombre de messages maximum (possible) envoyés dans une exécution
  - ***Complexité en bits*** : mesurée en nombre de bits total maximum envoyés (inclus dans des messages envoyés) dans une exécution
    (Chaque identifiant d'un nœud est sauvent supposé être représenté par O(log n) bits.)

- ***Complexité en espace/mémoire*** : en quantité de mémoire (en bits ou nombre d'états) d'un processus, nécessaire pour allouer de la place pour toute valeur de toute variable utilisée par l'algorithme.

Complexité en temps

Exprimée en quantité de temps maximum (possible) jusqu'à la terminaison (ou autres configurations) dans une exécution

- Dans les systèmes **synchrones** : en termes de rounds (**le temps d'acheminement d'un message vaut *exactement* un round / 1 unité de temps**; un pas de calcul vaux 0 temps).

- Dans les systèmes **asynchrones** : en termes de « rounds/unités de temps *asynchrones* ». Ici, il n'y pas de notion de temps immédiate naturelle, mais il est calculée sous les hypothèses suivantes : le calcul local (un pas de calcul) vaut 0 temps et **le temps d'acheminement d'un message vaut *au plus* 1 unité de temps**.

Questions ?