

SÉANCE 1

MODÈLES DE DÉVELOPPEMENT DE LOGICIEL



uOttawa

L'Université canadienne
Canada's university

SUJETS

Cycle de vie d'un projet logiciel

Processus «boîte noire» et «boîte blanche»

Modèle cascade

Modèle itératif

Modèles agile

- Scrum



uOttawa

L'Université canadienne
Canada's university

AU DÉBUT...



Est-ce qu'il y a des problèmes avec cette approche?



uOttawa

L'Université canadienne
Canada's university

CYCLE DE VIE

La cycle de vie d'un produit

- depuis la création d'une idée pour un produit à travers
 - analyse du domaine
 - collecte et modélisation des exigences
 - conception de l'architecture et spécification
 - programmation et vérification
 - livraison et le déploiement
 - maintenance and évolution
 - retraite



uOttawa

L'Université canadienne
Canada's university

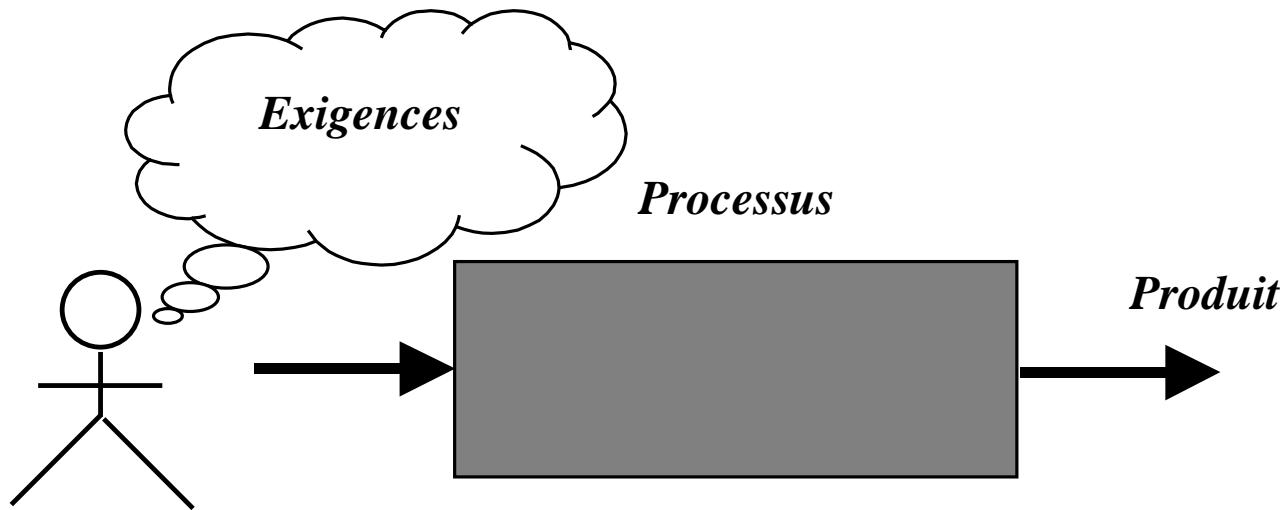
BESOIN POUR DES MODÈLES POUR GÉNIE LOGICIEL

Les symptômes d'insuffisance : la crise de logiciel

- temps prévu et coût dépassé
- attentes des utilisateurs n'est pas atteintes
- mauvaise qualité

La taille et valeur économique des applications logicielles nécessite des "modèles de processus"

PROCESSUS «BOÎTE NOIRE»





uOttawa

L'Université canadienne
Canada's university

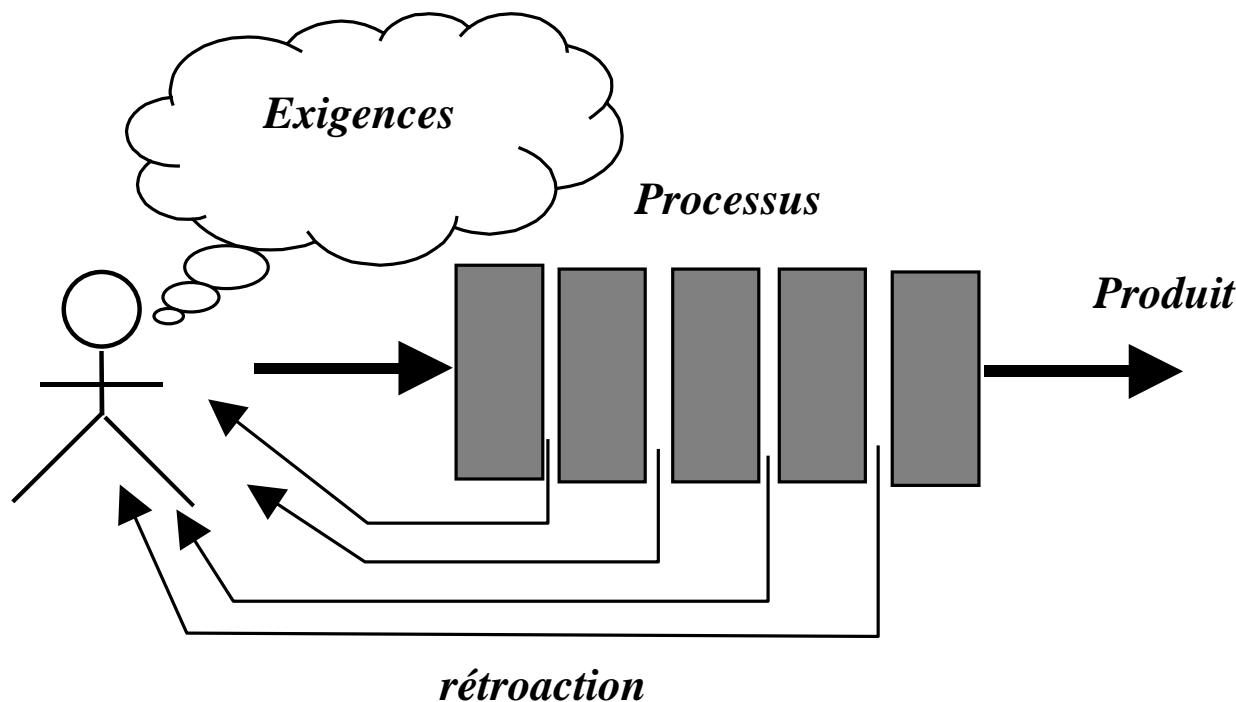
PROBLÈMES

L'hypothèse est que les exigences peuvent être pleinement compris avant le développement

Malheureusement l'hypothèse presque jamais valide

Interaction avec le client ne se produit qu'au début (exigences) et la fin (après la livraison)

PROCESSUS «BOÎTE BLANCHE»





uOttawa

L'Université canadienne
Canada's university

AVANTAGES

Réduire les risques en améliorant la visibilité

Autoriser les changements de projet pendant que le projet avance

- basés sur la réaction du client

LES ACTIVITÉS PRINCIPALES

Elles doivent être réalisées indépendamment du modèle

Le modèle affecte simplement le flux entre les activités



uOttawa

L'Université canadienne
Canada's university

MODÈLE CASCADE

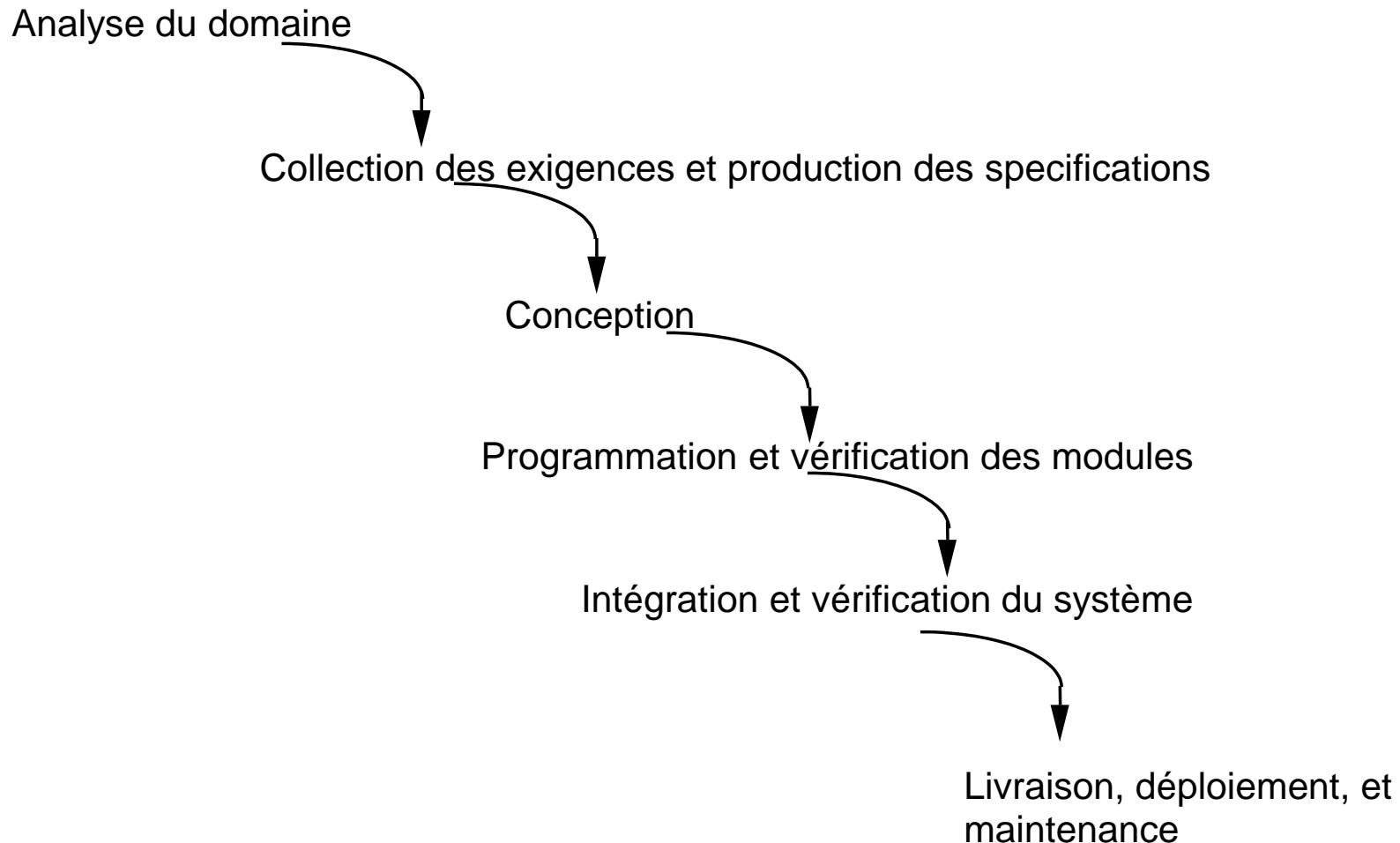
Inventé dans les années 1950 pour les grands systèmes de défense aérienne, popularisé dans les années 1970

Organise les activités dans un flux séquentiel

- On standardise les sorties des différentes activités

Existe en de nombreuses variantes, tous partageant le style de flux séquentiel

MODÈLE CASCADE





uOttawa

L'Université canadienne
Canada's university

POINTS FORTS DU CASCADE

Facile à comprendre, facile à utiliser

Fournit la structure pour les personnels inexpérimenté

Les étapes sont bien comprises

Crée des exigences stables



uOttawa

L'Université canadienne
Canada's university

FAIBLESSES DU CASCADE

Toutes les exigences doivent être connus d'avance

Livrables de chaque phase sont considérés statique- inhibe la flexibilité

Peut donner une fausse impression de progrès

Ne reflète pas la vrais nature de développent de logiciel:
résolution des problèmes d'une façon itératif

L'intégration est un *big bang* à la fin

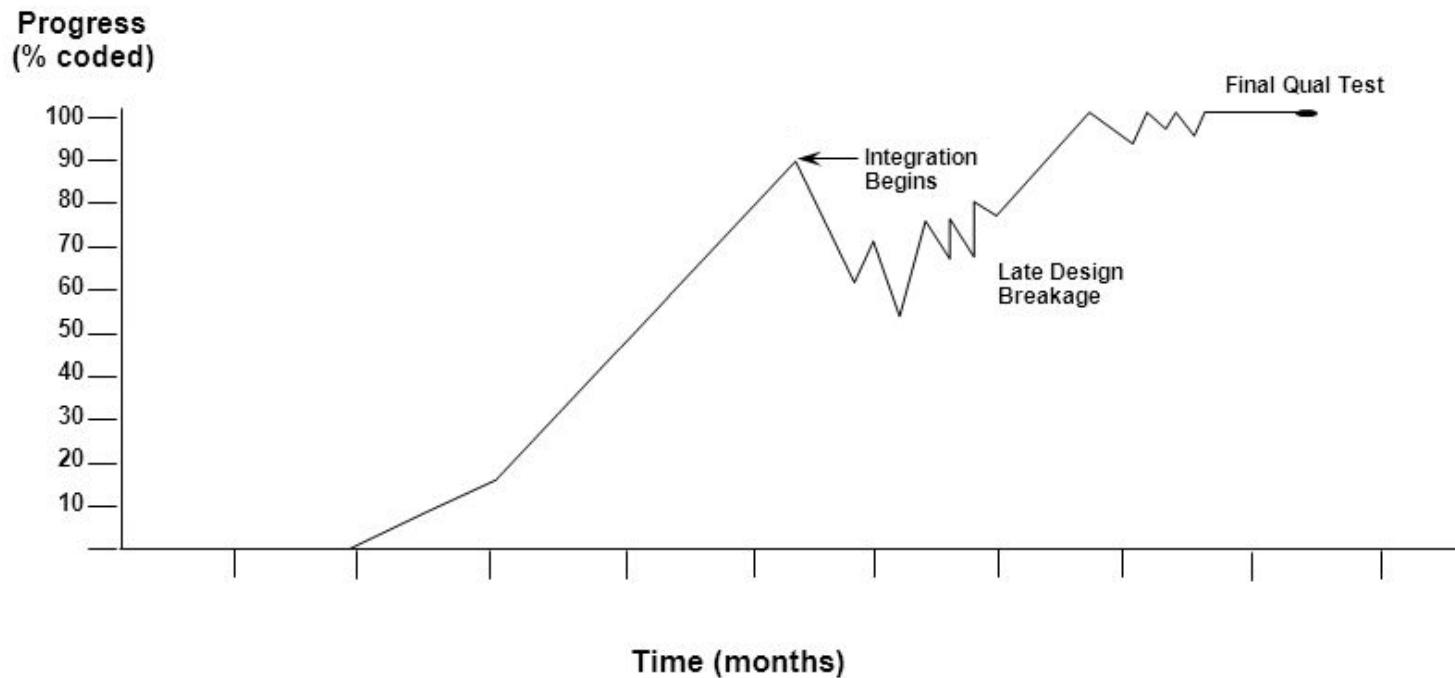
Peu de chances pour que le client voie et vérifie le système
(avant qu'il soit trop tard)



uOttawa

L'Université canadienne
Canada's university

RUPTURE DE CONCEPTION TARDIVE





uOttawa

L'Université canadienne
Canada's university

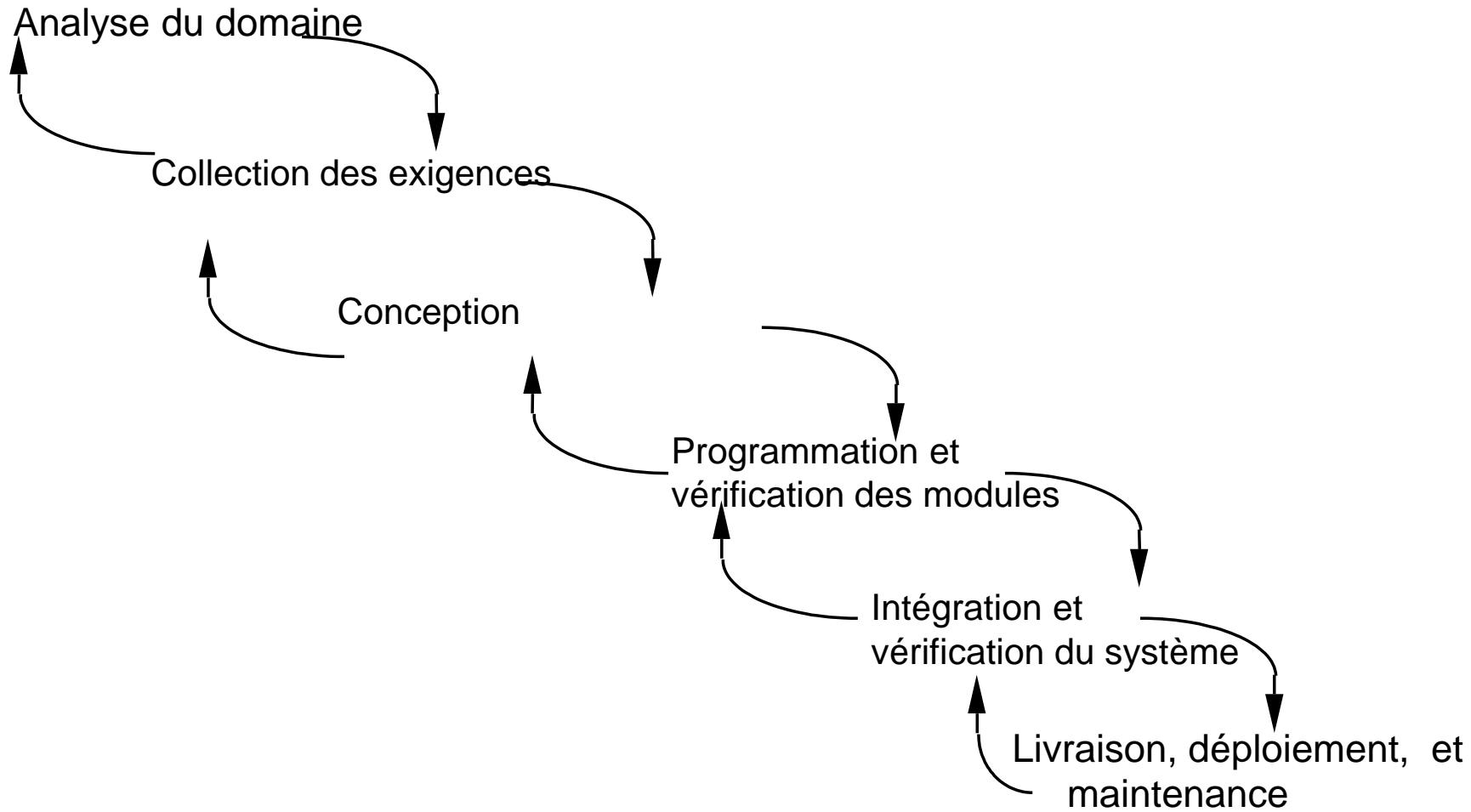
QUAND UTILISER CASCADE

Aujourd'hui, presque jamais!!

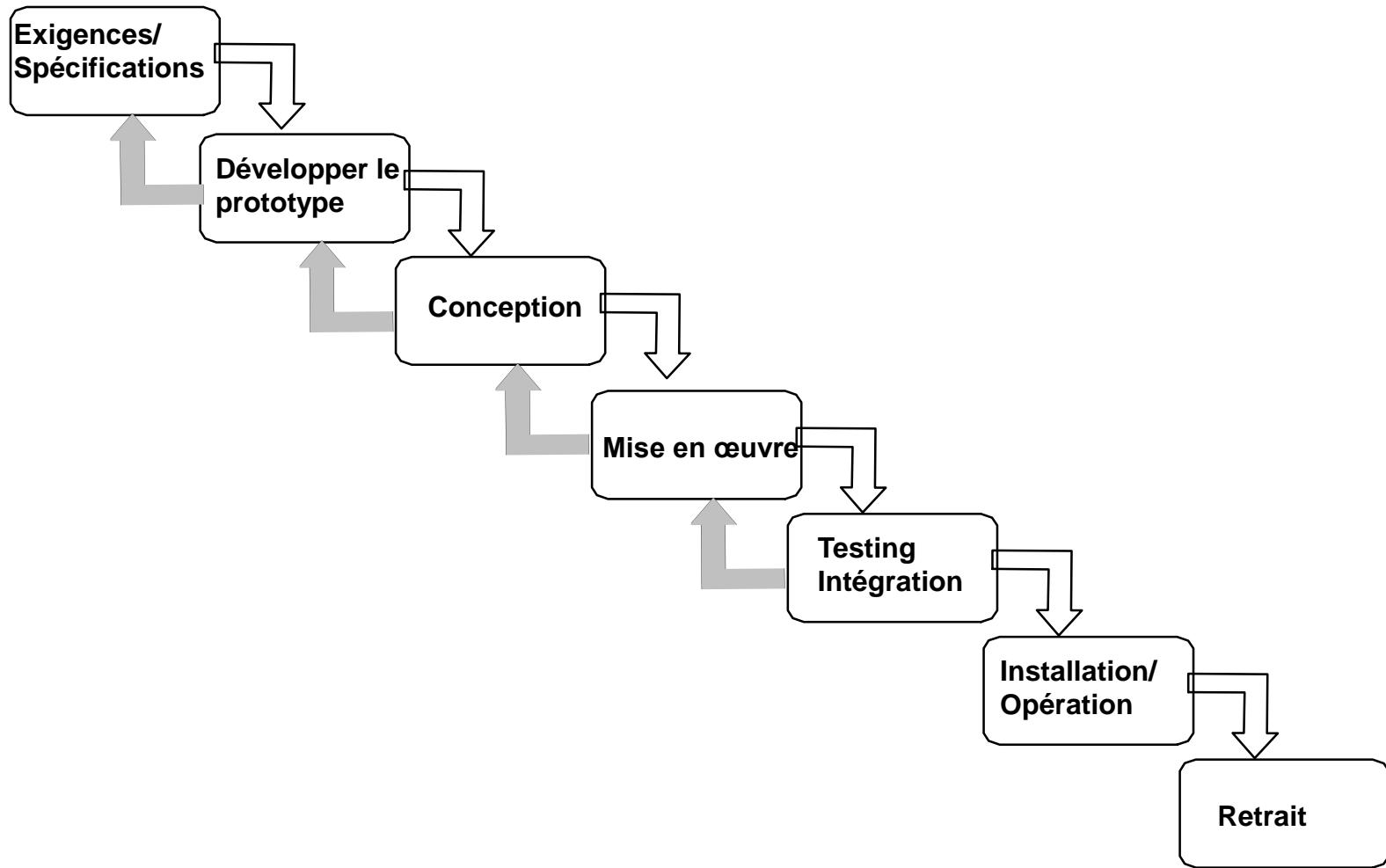
Mais, rarement, lorsque:

- Les exigences sont très bien connus
- Définition du produit est stable
- La technologie est *très bien* comprise
- Nouvelle version d'un produit existant (*peut-être!*)
- Portage d'un produit existant à une nouvelle plate-forme

CASCADE – AVEC RÉACTION



CASCADE AVEC PROTOTYPAGE RAPIDE



MODÈLE DE DÉVELOPPEMENT ITÉRATIF

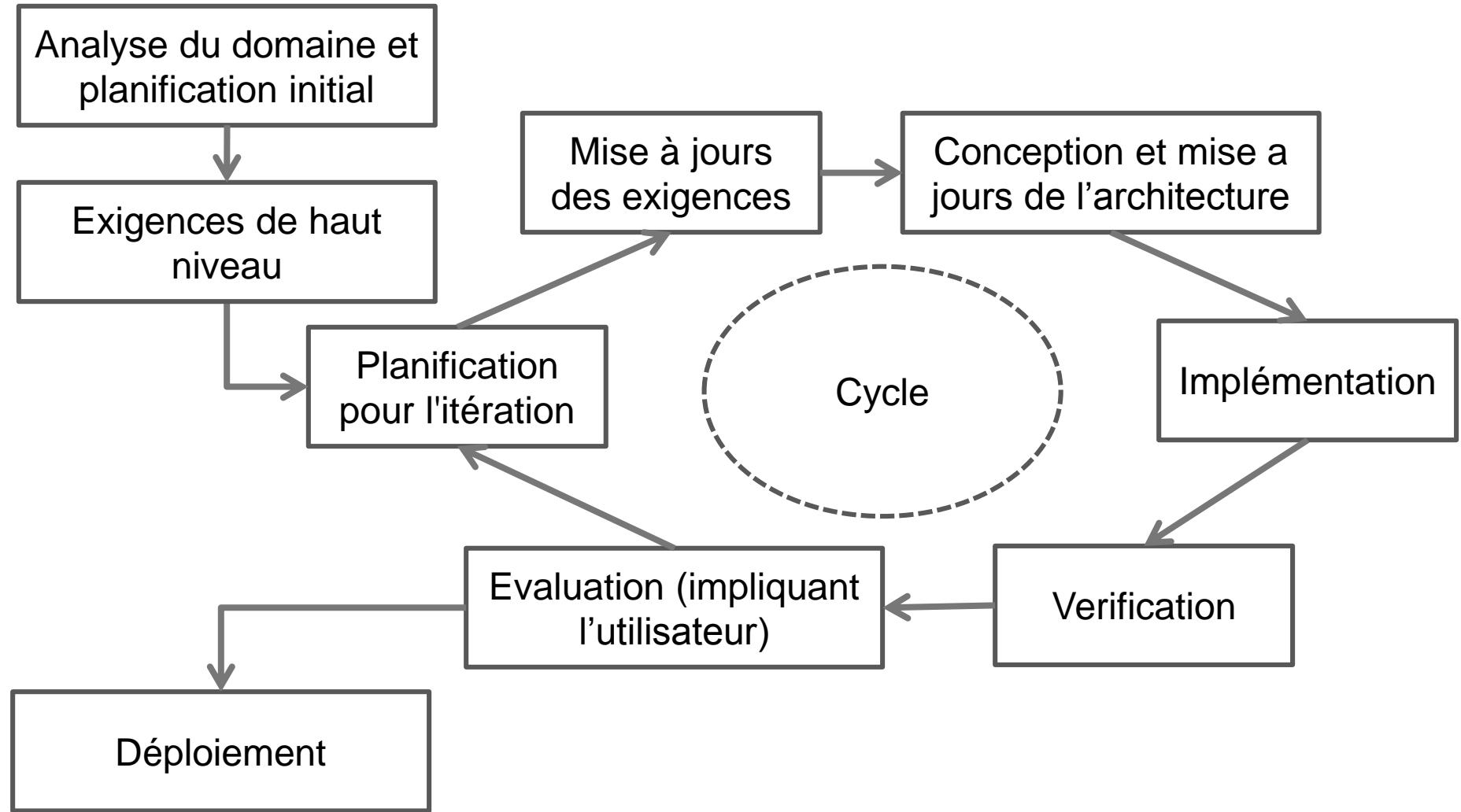
Développer le système par cycle répété (itérations)

- Chaque cycle est responsable du développement d'une petite portion de la solution (tranche de fonctionnalité)

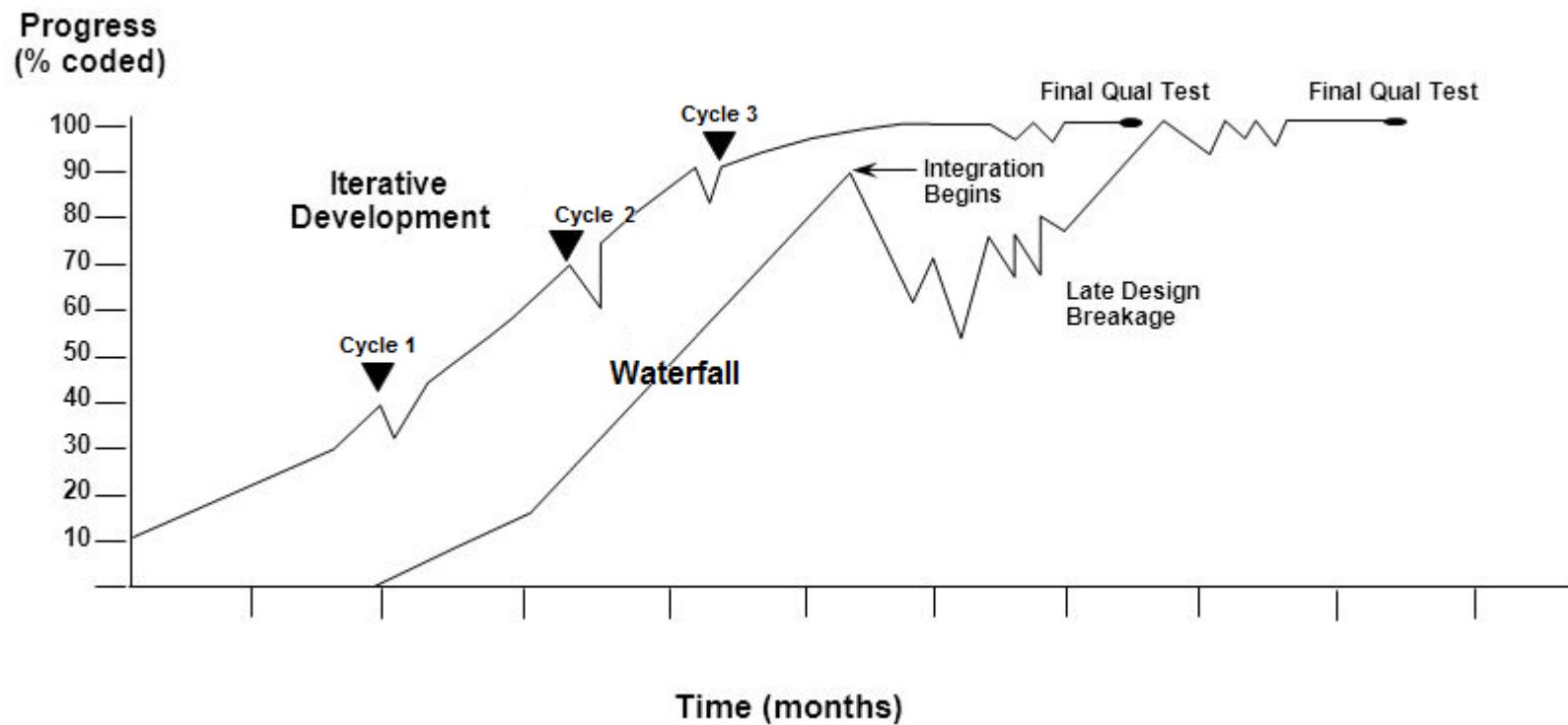
Contraste avec cascade:

- Cascade est un processus itératif particulier avec un seul cycle

PROCESSUS DE DÉVELOPPEMENT ITÉRATIF



INTÉGRATION CONTINUE





uOttawa

L'Université canadienne
Canada's university

PROCESSUS AGILE

Insatisfaction avec les processus de développement des années 1980 et 1990 (après des faillites successive) a résulté dans la création de méthodes agiles

Ces méthodes:

- Concentre sur le code plutôt que la conception
- Sont basés sur une approche itérative pour le développement de logiciels
- Sont supposé d'aider à produire des logiciels rapidement

L'objectif de ces méthodes est de réduire les frais généraux (overhead) dans le processus de développement (par exemple limiter la documentation) et à être en mesure de répondre rapidement à l'évolution des exigences



uOttawa

L'Université canadienne
Canada's university

MANIFESTE AGILE

Nous découvrons des meilleures façons pour développement de logiciels. Grâce à ce travail, nous sommes rendu-compte que:

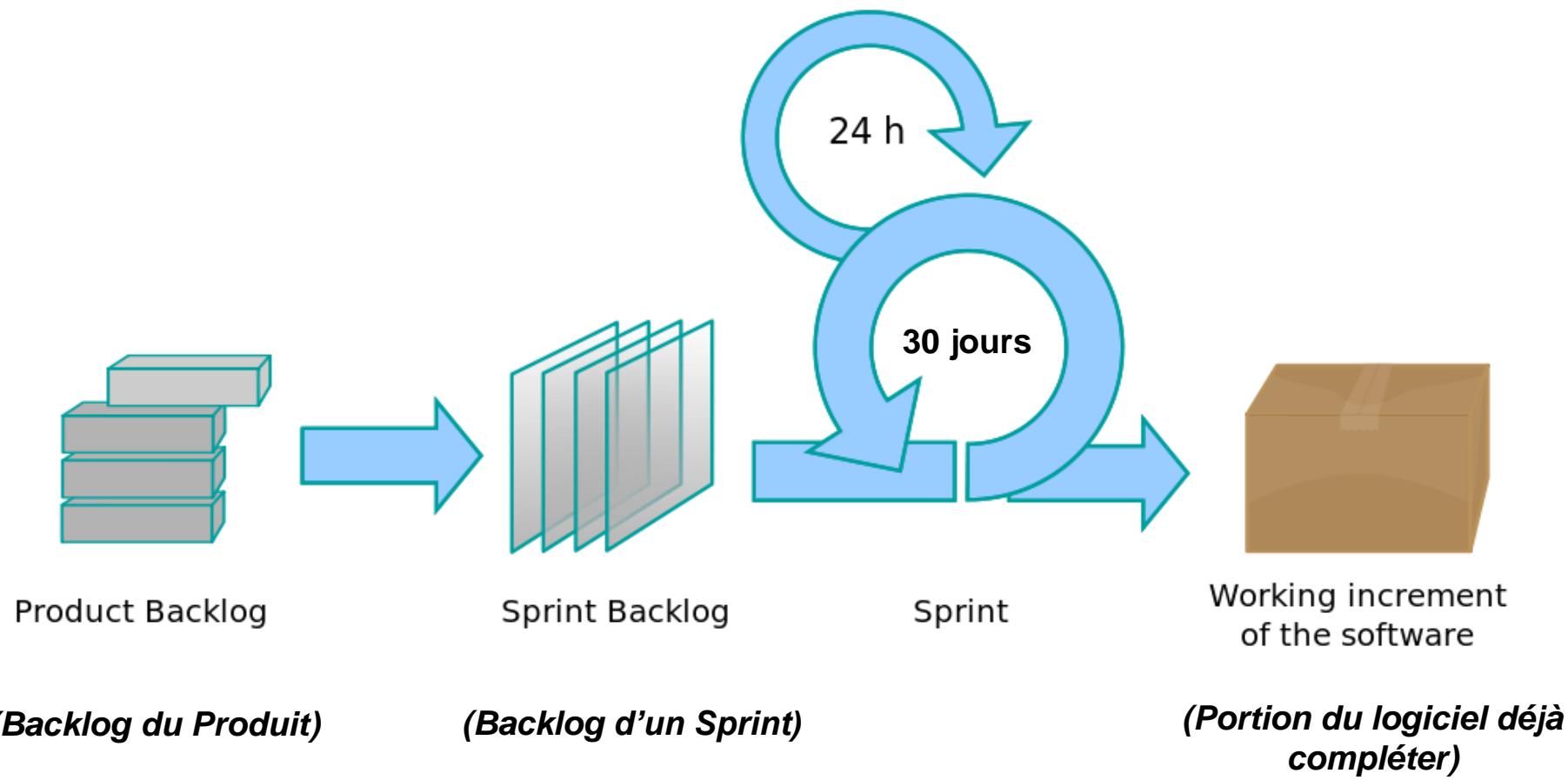
Plus utile	Moins utile
Interaction entre les individus	Processus et outils
Logiciel qui fonctionne	Documentation compréhensive
Collaboration avec le client	Négociations prolongées sur les contrats
Réagir au changement	Suivre un plan

Cependant, les items sur la droite sont utiles, main nous accordant plus d'importance sur les items à gauche

LES PRINCIPES DES MÉTHODES AGILES

Règle	Description
Participation du client	Les clients devraient être impliqués tout au long du processus de développement. Leur rôle est de fournir et de prioriser les exigences et d'évaluer les itérations déjà complétées.
Livraison incrémentielle	Le logiciel est développé par incréments avec le client spécifiant les exigences à inclure dans chaque incrément.
Individus et non processus	Les compétences de l'équipe de développement doivent être reconnus et exploités. Membres de l'équipe avoir la liberté de développer leurs propres méthodes de travail sans processus normatifs.
Acceptez les changement	Supposez que les exigences du système vont changer et concevez alors le système pour accommoder ces changements futurs.
Maintenez la simplicité	Concentrer sur la simplicité dans l'architecture du logiciel que vous développez et dans le processus de développement. Lorsque c'est possible, éliminer les complexités de votre système.

PROCESSUS SCRUM





uOttawa

L'Université canadienne
Canada's university

PROBLÈMES AVEC LES MÉTHODES AGILES

Il peut être difficile de maintenir l'intérêt des clients qui sont impliqués dans le processus

Membres de l'équipe peuvent être inadaptées à l'intensité qui caractérise les méthodes agiles

Priorisation des changements peut être difficile lorsqu'il y a plusieurs parties prenantes

Le minimisation de documentation: presque rien n'est capturé, le code est la seule autorité

MERCI!

QUESTIONS?

SEANCE 2

**ANALYSE DU DOMAIN ET
MODÉLISATION DES
EXIGENCES**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Analyse du domaine

- Modélisation de domaine

Exigences

- Exigences fonctionnelles
- Exigences non-fonctionnelles
- Cas d'utilisation
- Diagrammes de cas d'utilisation
- Relations entre les cas d'utilisation
- Exemples de cas d'utilisation

Histoires d'utilisateurs des processus agiles



uOttawa

L'Université canadienne
Canada's university

ANALYSE DU DOMAINE

L'analyse de domaine nous permet de comprendre le domaine du problème

On est concerné par le domaine dans lequel les clients s'attendent à utiliser le logiciel

Analyse de domaine: recueillir l'information sur le domaine des experts, de livres, de logiciels existants et de sa documentation ...

- L'objectif est de comprendre le domaine du problème indépendamment du système particulier que nous développeront
- Nous ne cherchons pas à délimiter le système du restant de l'environnement
- Nous nous concentrons sur les concepts et la terminologie du domaine d'application avec une portée plus large que le futur système.



uOttawa

L'Université canadienne
Canada's university

ANALYSE DU DOMAINE

Vous n'êtes *pas* censé devenir un expert de domaine

- Mais vous devez rassembler suffisamment d'informations pour comprendre le problème

Avantages:

- Développement plus rapide: communiquer plus efficacement avec les intervenants
- Une meilleure compréhension des besoins des utilisateurs
- Anticipation des extensions



uOttawa

L'Université canadienne
Canada's university

ANALYSE DE DOMAINE

- 1. Glossaire des termes définissant la terminologie et les concepts communs du domaine du problème**
- 2. Connaissance générale du domaine (information de base bien connus par les experts)**
- 3. Les clients et les utilisateurs (qui achèteront ou pourraient acheter le logiciel)**
- 4. Environnement (le nouveau système interagira-t-il avec les systèmes existants?)**
- 5. Modèle de domaine**
 - Diagramme de classes UML pour modéliser la relation entre les entités
 - Modèle de relation d'entité
- 6. Tâches et procédures actuellement exécutées**
- 7. Les applications des compétiteurs**



uOttawa

L'Université canadienne
Canada's university

MODÈLE DE DOMAINE

Il n'existe pas de vue unifiée en ingénierie logicielle concernant ce qu'est un modèle de domaine

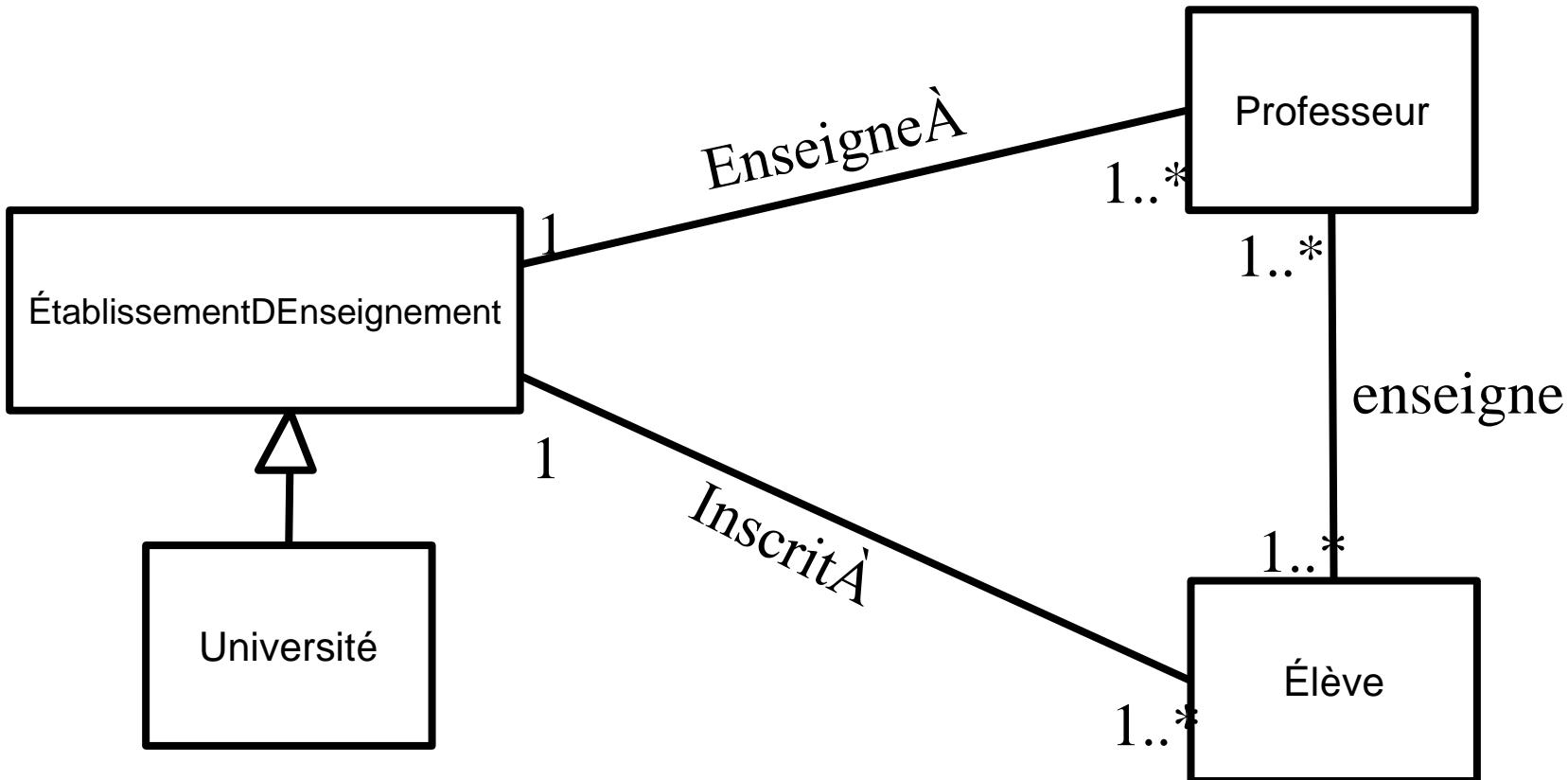
Cependant, en général, la modélisation du domaine est la décomposition d'un domaine en ses entités individuelles

C'est une façon de décrire et de modéliser les entités de domaine et les relations entre elles

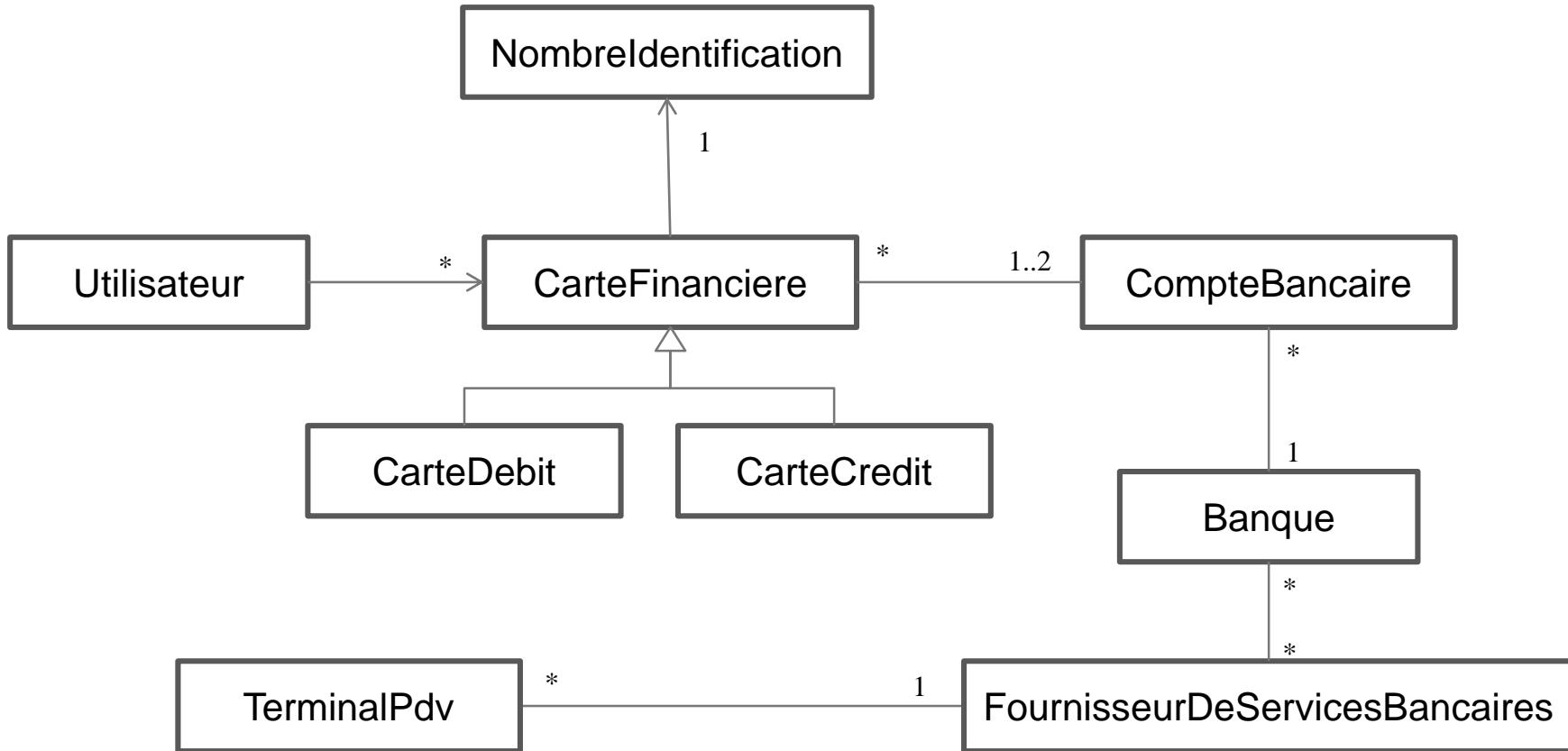
- Ces entités décrivent collectivement l'espace de domaine

En générale, le modèle de domaine pour toutes les entreprises du même domaine doit être presque similaire

EXEMPLE - MODÈLE DE DOMAINE - DIAGRAMME DE CLASSE UML



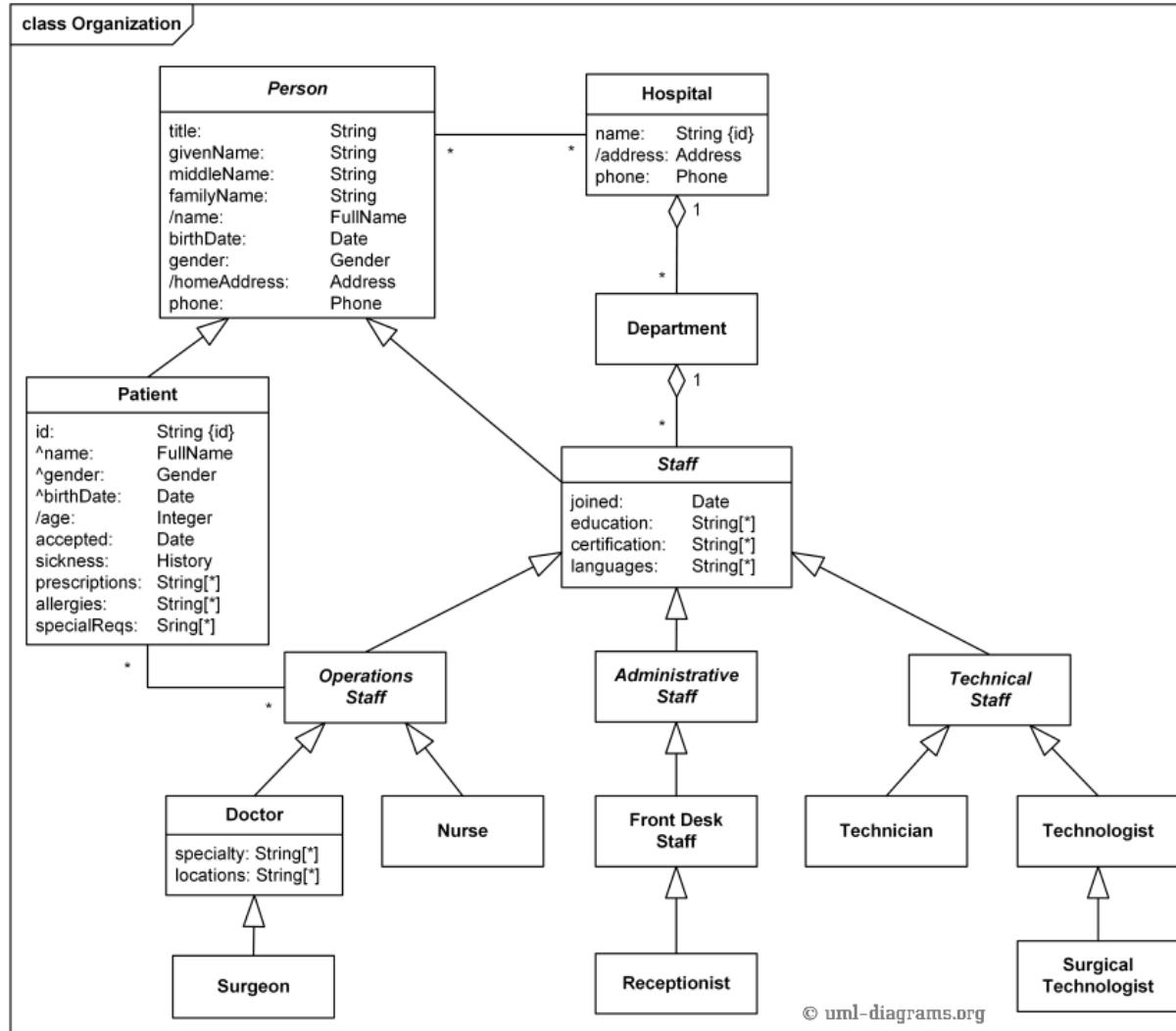
EXEMPLE - MODÈLE DE DOMAINE - DIAGRAMME DE CLASSE UML



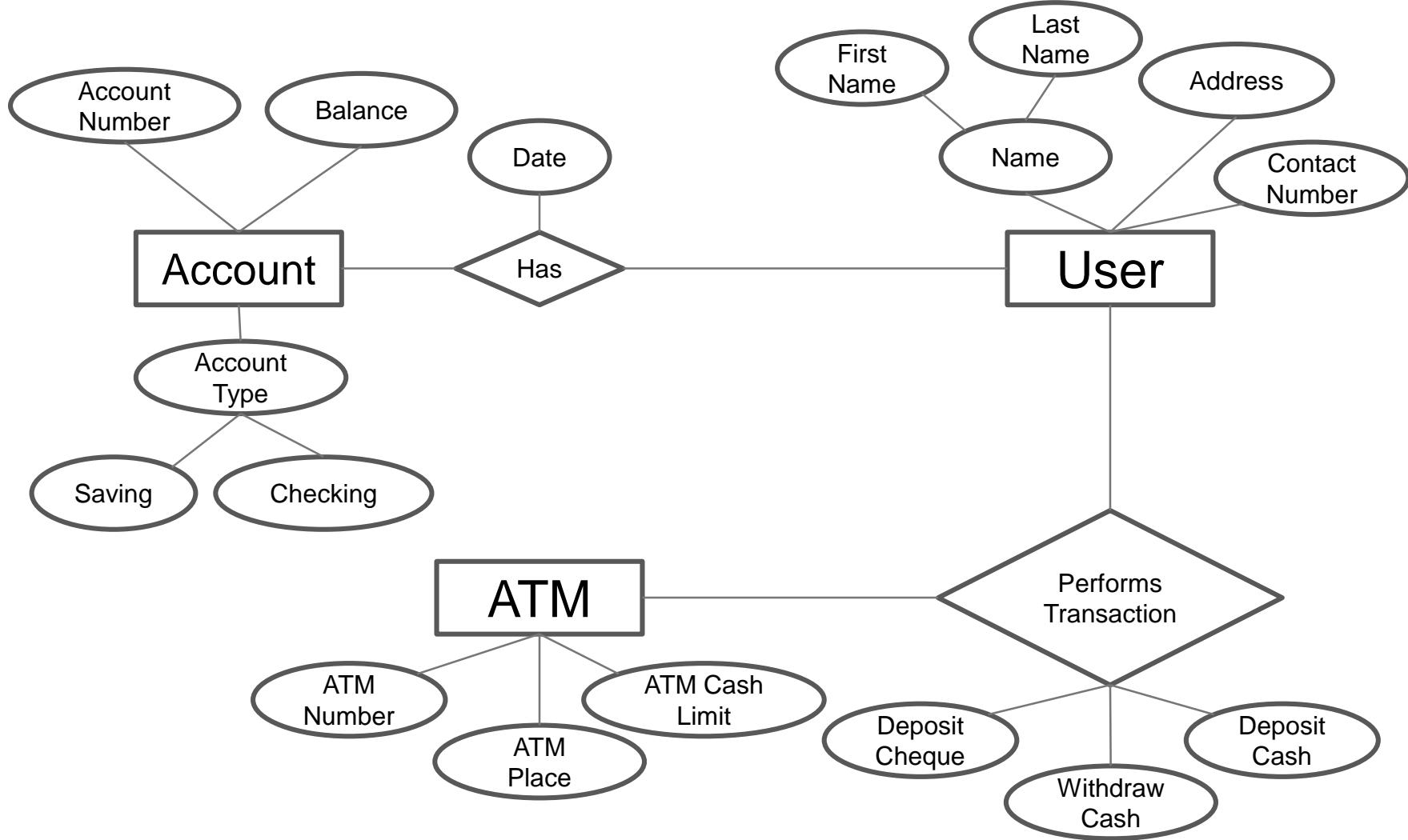
EXEMPLE - MODÈLE DE DOMAINE - DIAGRAMME DE CLASSE UML



uOttawa
L'Université canadienne
Canada's university



EXEMPLE - MODÈLE DE DOMAINE - MODÈLE DE RELATION D'ENTITÉ





uOttawa

L'Université canadienne
Canada's university

EXIGENCES

Nous devons savoir exactement ce que le client a besoin:

Recueil des exigences

- C'est là que les attentes du client sont capturées
- Réalisé à partir des entrevues avec les parties prenantes et l'analyse d'un système existant (qu'il soit manuel ou automatisé)
- Capturez des informations en utilisant généralement des déclarations non techniques de haut niveau

Exemple

- Exigence 1: « Nous devons développer un portal de client en ligne »
- Exigence 2: « Le portal doit lister tous nos produits »
- ...

Évitez d'ajouter des détails de conception et d'implémentation

Tenez compte uniquement des fonctions que l'application doit supporter



uOttawa

L'Université canadienne
Canada's university

EXIGENCES



EXIGENCES FONCTIONNELLES

Capturer le comportement prévu du système

- Peut être exprimer comme services, tâches ou fonctions que le système effectue

Les cas d'utilisation sont rapidement devenus une pratique répandue pour capturer les exigences

- Ceci est surtout vrai dans la communauté objet-orientée d'où ces notions origines
- Leur application n'est pas limitée aux systèmes objet-orientés



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION

Un cas d'utilisation définit un ensemble d'interactions but-orienté entre les acteurs externes et le système en considération

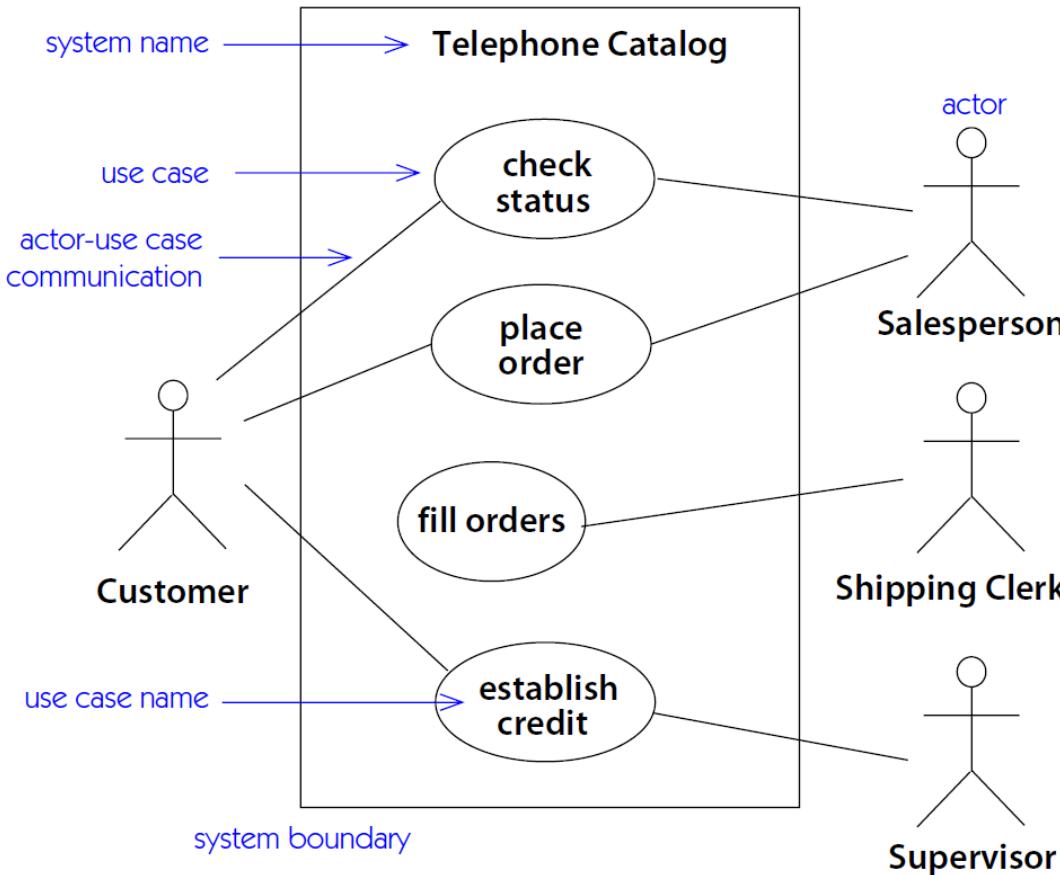
Les acteurs sont des composantes à l'extérieur du système qui interagissent avec le système

- Un acteur peut être une classe d'utilisateurs ou d'autres systèmes

Un cas d'utilisation est initié par un utilisateur qui possède un but particulier, et est complété lorsque le but est atteint

Il décrit la séquence d'interactions entre les acteurs et le système qui sont nécessaires pour livrer le service qui satisfait au but

DIAGRAMMES DE CAS D'UTILISATION



DIAGRAMMES DE CAS D'UTILISATION

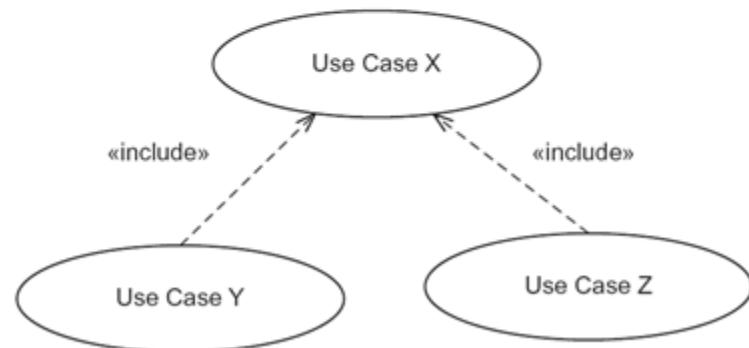
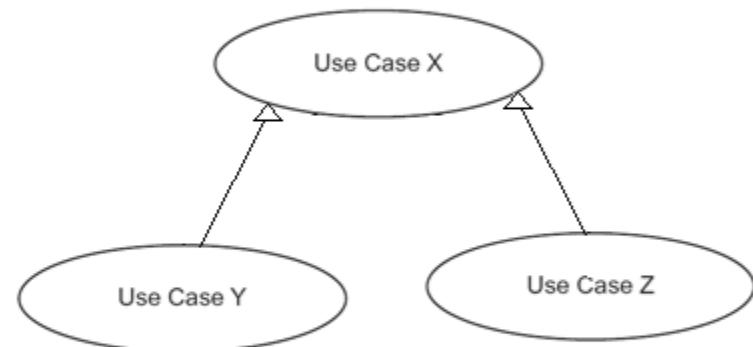
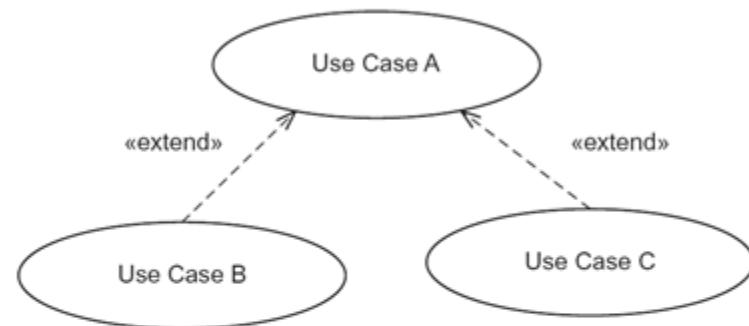
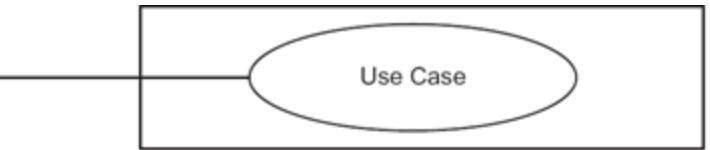


uOttawa

L'Université canadienne
Canada's university



Actor



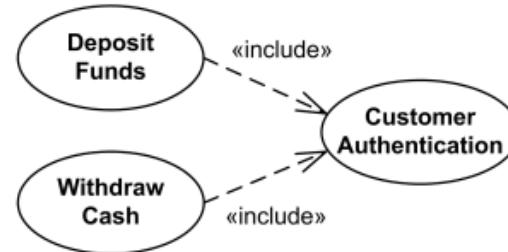


uOttawa

L'Université canadienne
Canada's university

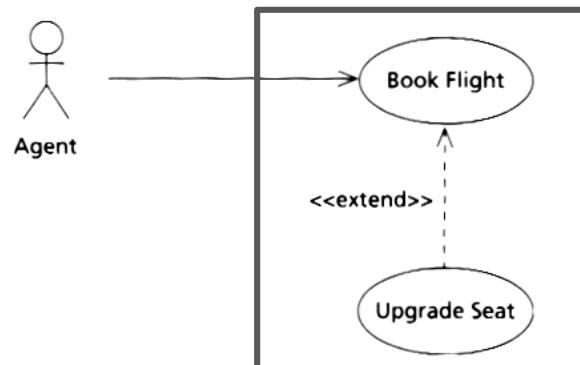
RELATIONS ENTRE LES CAS D'UTILISATION

Relation d'inclusion: un fragment de cas d'utilisation qui est dupliqué en multiples cas d'utilisation



Relation d'extension: cas d'utilisation qui ajoute conditionnellement des étapes à un cas d'utilisation de première classe

- Exemple:



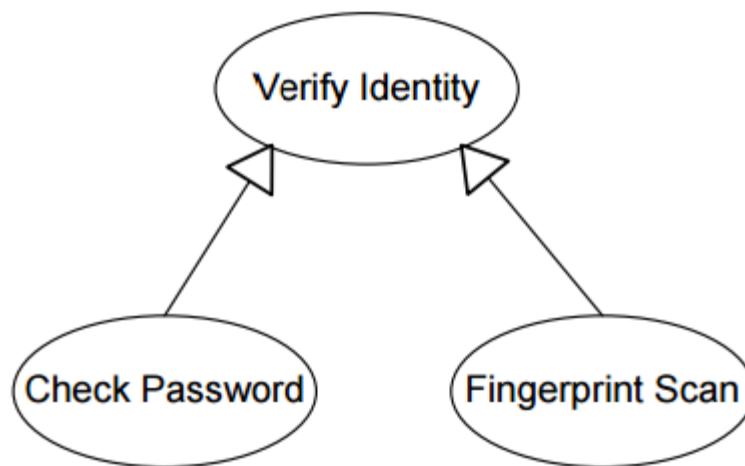


uOttawa

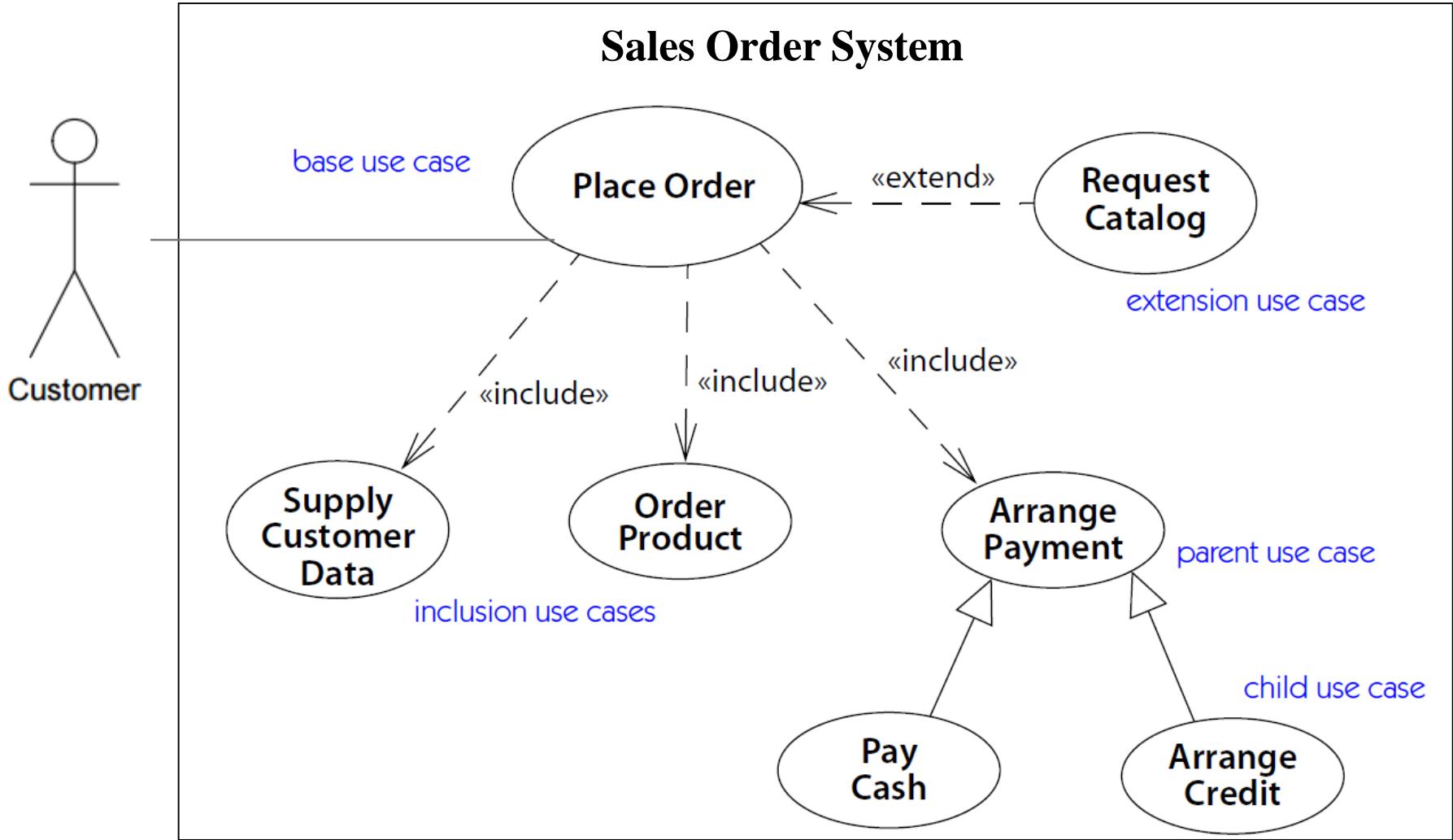
L'Université canadienne
Canada's university

RELATIONS ENTRE LES CAS D'UTILISATION

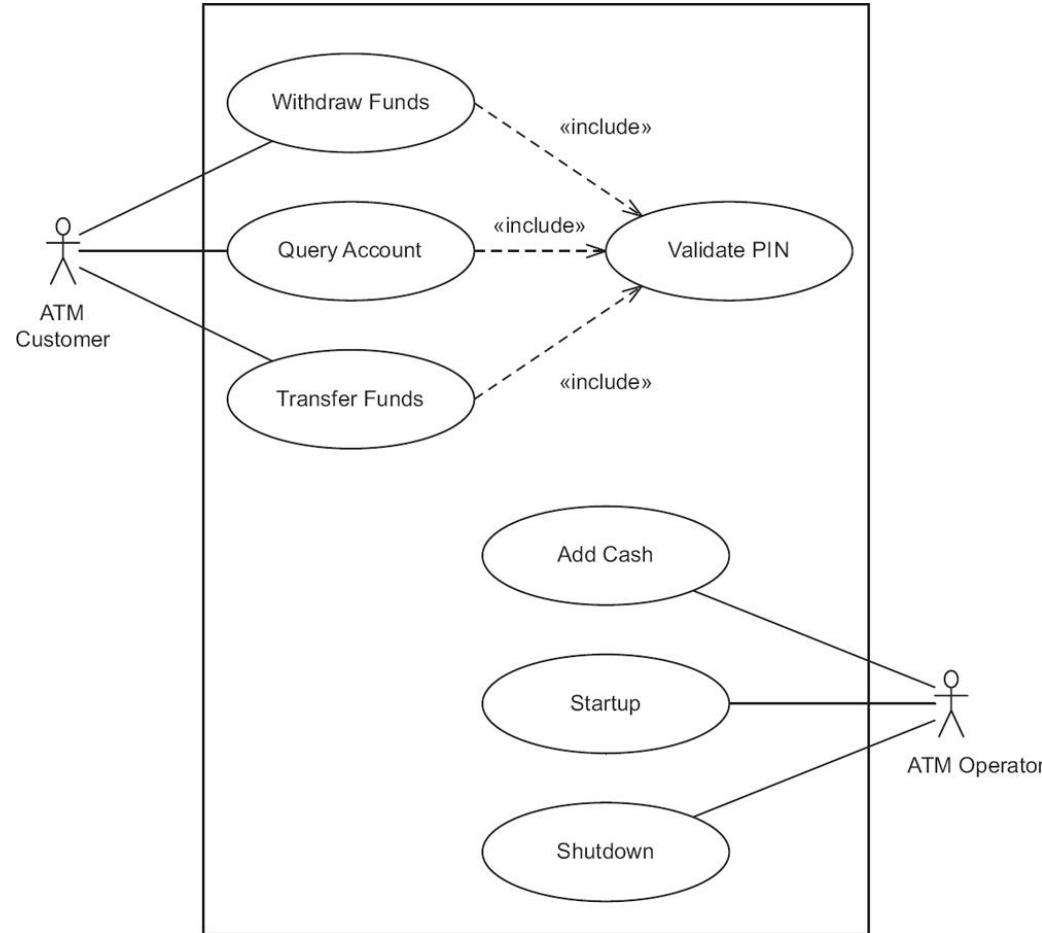
Relation Généralisation: un des cas d'utilisation enfant sera exécuté



RELATIONS ENTRE LES CAS D'UTILISATION



CAS D'UTILISATION – EXEMPLE DE L'ATM



CAS D'UTILISATION – EXEMPLE DE GUICHET AUTOMATIQUE BANCAIRE (ATM)

Acteurs:

- Client de l'ATM
- Opérateur de l'ATM

Cas d'utilisation:

- Le client peut:
 - Retirer des fonds d'un compte chèque ou épargne
 - Voir la balance du compte
 - Transférer des fonds d'un compte à un autre
- L'opérateur de l'ATM peut:
 - Éteindre la machine ATM
 - Remplir le distributeur de monnaie de la machine
 - Démarrer la machine ATM



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – EXEMPLE DE L'ATM

Valider le NIP est une cas d'utilisation *d'inclusion*

- Il ne peut pas être exécuté tout seul
- Doit être exécuté en faisant partie du *cas d'utilisation concret*

Cependant, un *cas d'utilisation concret* peut être exécuté



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM – VALIDER LE NIP (1)

Nom du cas d'utilisation: Valider le NIP

Sommaire: Le système valide le NIP du client

Acteur: Client de l'ATM

Précondition: L'ATM est inoccupé, affichant le message de bienvenue.



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM – VALIDER LE NIP (2)

Séquence principale:

1. Le client insert sa carte ATM dans le lecteur.
2. Si le système reconnaît la carte, il lit le numéro de la carte.
3. Le système demande au client son NIP.
4. Le client rentre son NIP.
5. Le système vérifie la date d'expiration de la carte et si la carte a été déclarée perdue ou volée.
6. Si la carte est valide, le système vérifie si le NIP rentré correspond au NIP de la carte maintenu par le système.
7. Si les NIP matchent, le système vérifie quels comptes sont accessibles avec la carte ATM.
8. Le système affiche les comptes du client et demande au client quel type de transaction il veut: retirer, voir ou transférer.



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM – VALIDER LE NIP (3)

Séquences alternatives:

- Étape 2: Si le système ne reconnaît pas la carte, il éjecte la carte
- Étape 5: Si le système détermine que la carte est expirée, le système confisque la carte.
- Étape 5: Si le système détermine que la carte a été déclarée perdue ou volée, le système confisque la carte.
- Étape 7: Si le NIP rentré par le client ne matche pas le NIP de cette carte, le système redemande le NIP.
- Étape 7: Si le client rentre un NIP incorrect trois fois, le système confisque la carte.
- Étapes 4-8: Si le client rentre « Annuler », le système annule la transaction et éjecte la carte.

Poste-condition: Le NIP du client a été validé.



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM - RETIKER DES FONDS (1)

Nom du cas d'utilisation: Retirer des fonds

Sommaire : Le client retire un montant spécifique de fonds d'un compte bancaire valide .

Acteur : Le client de l'ATM

Dépendance: Inclure le cas d'utilisation pour valider le NIP.

Condition: ATM est inactif , l'affichage d'un message de bienvenue.



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM - RETIKER DES FONDS (2)

Séquence principale :

1. Inclure le cas d'utilisation pour valider le NIP.
2. Le client sélectionne « retrait », rentre le montant, et sélectionne le numéro de compte .
3. Le système vérifie si le client dispose de suffisamment de fonds dans le compte et si la limite quotidienne ne sera pas dépassée .
4. Si toutes les conditions sont vérifiées, le système autorise la distribution de l'argent.
5. Le système distribue le montant d'argent requis .
6. Le système imprime un reçu avec le numéro de transaction , type de transaction, le montant retiré , et le solde du compte .
7. Le système éjecte la carte .
8. Le système affiche le message de bienvenue.



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION – ATM - RETIKER DES FONDS (3)

Séquences alternatives :

- Étape 3: Si le système détermine que le numéro de compte n'est pas valide , il affiche un message d'erreur et éjecte la carte.
- Étape 3: Si le système détermine que les fonds sont insuffisants dans le compte du client , il affiche une excuse et éjecte la carte .
- Étape 3: Si le système détermine que le montant de retrait quotidien maximal autorisé a été dépassé , il affiche une excuse et éjecte la carte .
- Étape 5: Si les fonds de l' ATM sont épuisées, le système affiche une excuse , éjecte la carte , et arrête le guichet automatique.

Poste-condition : les fonds du client ont été retirés .



uOttawa

L'Université canadienne
Canada's university

EXIGENCES NON FONCTIONNELLES

Les exigences fonctionnelles définissent ce que le système doit faire

Les exigences non fonctionnelles définissent comment le système est supposé être.

- Généralement, décrit les attributs du système tels que sécurité, fiabilité, maintenance, d'évolutivité, utilisabilité...

EXIGENCES NON FONCTIONNELLES

Les exigences non-fonctionnelles peuvent être spécifiées dans une section distincte de la description de cas d'utilisation

- Dans l'exemple précédent, pour le cas d'utilisation « valider le NIP », il peut y avoir une exigence de sécurité que le numéro de la carte et le NIP doivent être encryptés.

Les exigences non fonctionnelles peuvent être spécifiées pour un groupe de cas d'utilisation ou pour le système entier.

- **Exigence de sécurité:** Le système doit encrypté le numéro de carte ATM et le NIP.
- **Exigence de performance:** Le système doit répondre aux sélections de l'acteur dans un délai de 5 secondes.



uOttawa

L'Université canadienne
Canada's university

DOCUMENT DES EXIGENCES LOGICIELLES

Combinez les exigences fonctionnelles et non fonctionnelles dans un document

- Vous pouvez également intégrer la sortie de l'analyse de domaine dans le document

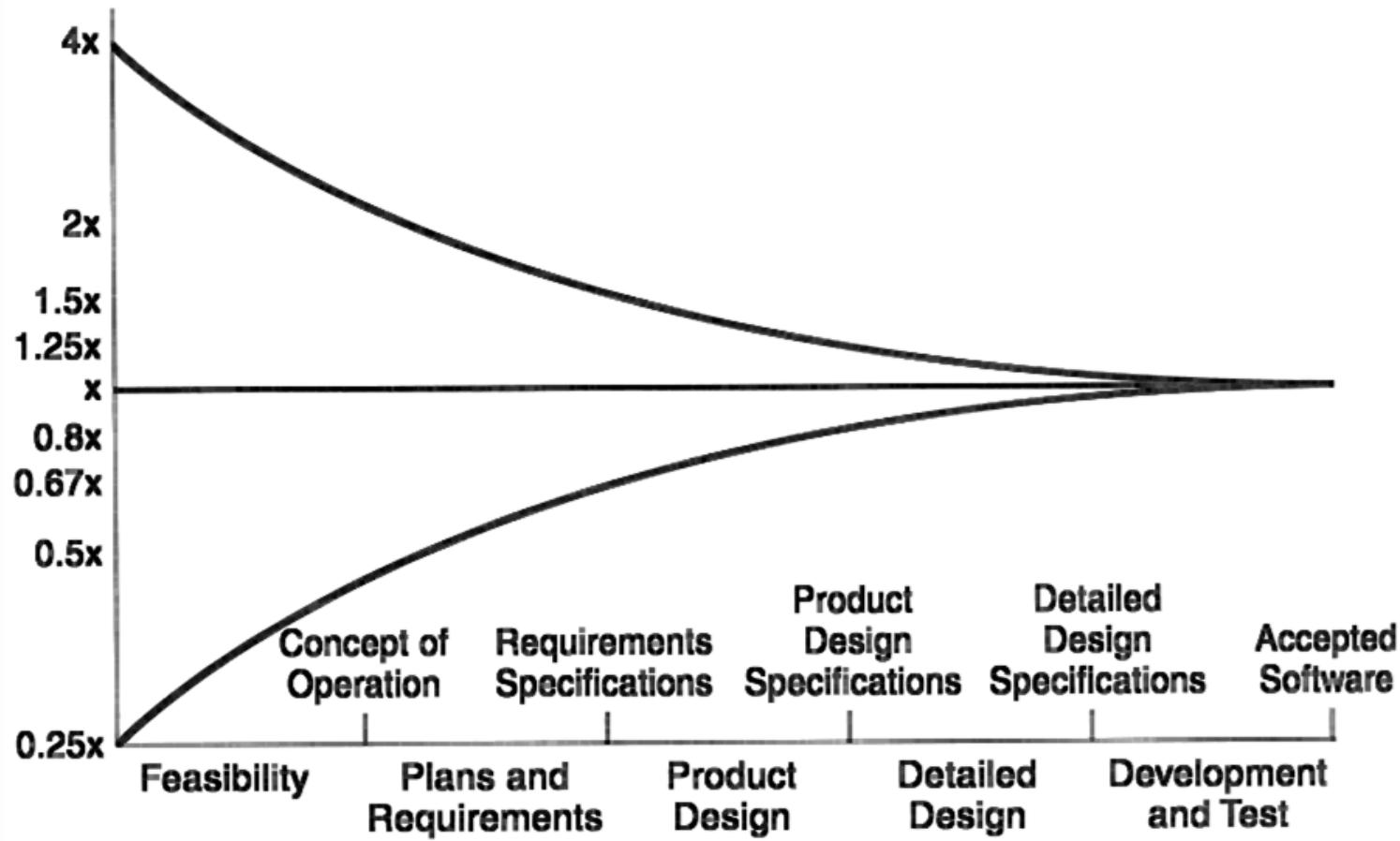
Ce document établit la base de l'accord entre développeur et client

- **Ne continuez pas** tant que toutes les parties prenantes ne sont pas satisfaites avec ce document

Ce document fournit une base pour estimer les coûts des produits, les risques, et le calendrier

- *Mais ne devriez-vous pas déjà avoir estimé le coût (voir diapositive suivante)?*

PRÉCISION D'ESTIMATION DE LA TAILLE DU LOGICIEL





uOttawa

L'Université canadienne
Canada's university

EXIGENCES DANS LES PROCESSUS AGILES

Exigences dans SCRUM et programmation extrême sont décrites en utilisant des histoires d'utilisateurs

Une histoire d'utilisateur est une définition de haut niveau d'une exigence

- Descriptions simples d'une fonction dit du point de vue de la personne qui désire la nouvelle capacité
- Au cours de la population du backlog, ils contiennent juste assez d' informations pour que les développeurs peuvent produire une estimation raisonnable de l'effort
- Plus tard, plus de détails peuvent être ajoutés en consultation avec le client

Ils suivent généralement un format simple:

En tant que <type d'utilisateur>, je veux <un but> afin que <une raison quelconque>



uOttawa

L'Université canadienne
Canada's university

EXIGENCES DANS LES PROCESSUS AGILES

Les histoires d'utilisateurs peuvent être écrits à différents niveaux de détail

- Une histoire d'utilisateur qui couvre une grande quantité de fonctionnalités est généralement connu comme une histoire épique
- Une histoire épique ne peut pas être complétée en une seule itération
- Elle est décomposée en histoires d'utilisateurs qui peuvent être implémentées en une seule itération

Exemple

- Histoire épique: **En tant qu'utilisateur, je veux** sauvegarder tout mon disque dur.
- Historique des utilisateurs:
 - **En tant qu'utilisateur, je veux** spécifier des fichiers ou des dossiers à sauvegarder selon la taille du fichier, la date de création et date de modification.
 - **En tant qu'utilisateur, je veux** indiquer les dossiers à ne pas sauvegarder **afin que** mon lecteur de sauvegarde ne soit pas rempli de choses que je n'ai pas besoin de sauvegarder

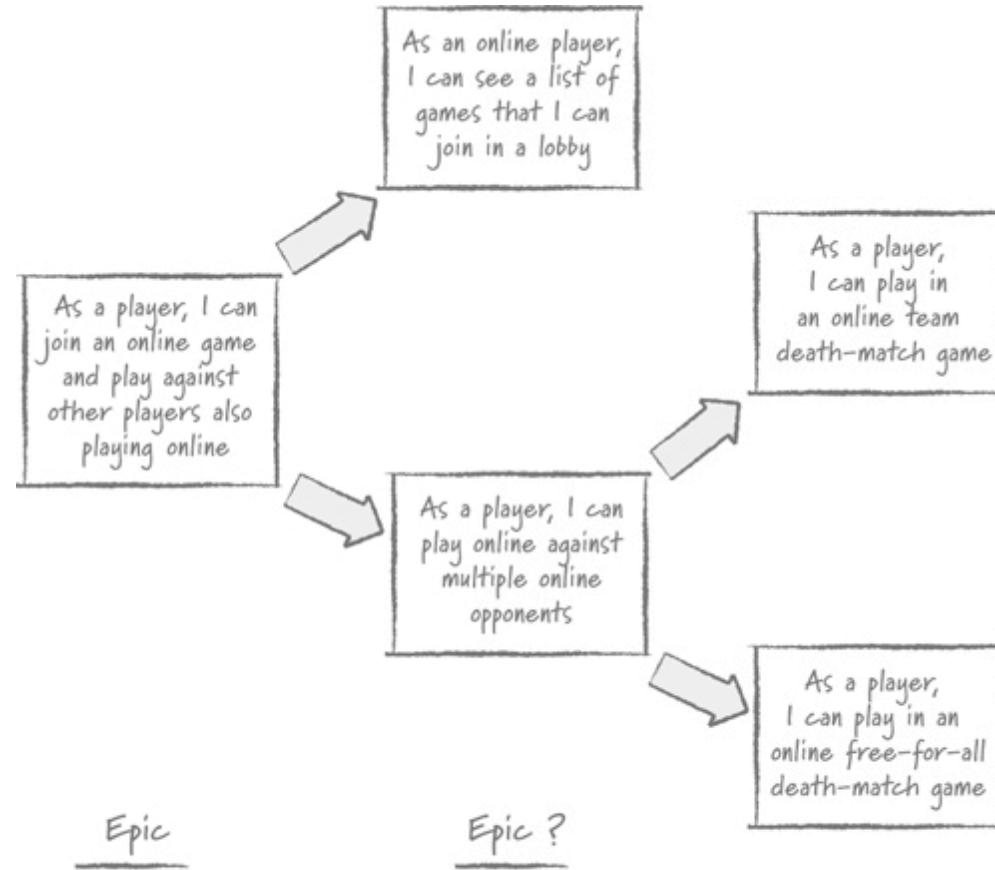
EXEMPLE: HISTOIRES D'UTILISATEUR POUR LE DÉVELOPPEMENT DE JEUX



uOttawa

L'Université canadienne
Canada's university

Décomposition des histoires épiques



EXEMPLE: HISTOIRES D'UTILISATEUR POUR LE DÉVELOPPEMENT DE JEUX

Ajoutez des conditions de satisfaction à une user story si vous le jugez nécessaire

Exemple:

User Story:

*As a player, I want to
shoot an enemy
character and see it
react so that I know
when it is hit.*

Conditions of Satisfaction

*When the enemy is shot in the head,
they stumble back*

*When the enemy is shot in the left
side, they twist left*

*When the enemy is shot in the right
side, they twist right*

MERCI!

QUESTIONS?

SÉANCE 3

DIAGRAMMES D'ACTIVITÉS UML



uOttawa

L'Université canadienne
Canada's university

SUJETS

Révision de modélisation de logiciel

Diagrammes d'activités UML

- Activités
- Actions
- Flux de contrôle
- Flux d'objets
- Décision et nœuds de fusion
- Fork et join
- Fil d'exécution conditionnel
- Partition
- Signal
- Région Interruptible
- Région d'Expansion

Modélisation de domaine et processus de logiciel en utilisant les diagrammes d'activités

Génération d'un diagramme d'activité à partir d'une histoire d'utilisateur

MODÉLISATION DE LOGICIEL

UML définit treize types de diagrammes de base, divisées en deux catégories générales:

- Modélisation Structurelle
- Modélisation Comportementale

Modèles structurels définissent l'architecture statique d'un logiciel

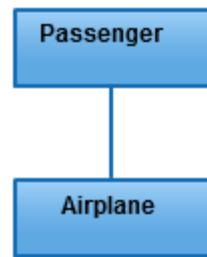
- Ils sont utilisés pour modéliser les «choses» qui composent un modèle - les classes, les objets, les interfaces et les composants physiques
- En outre, ils sont utilisés pour modéliser les relations et dépendances entre les éléments

RÉVISION DES ASSOCIATIONS UML

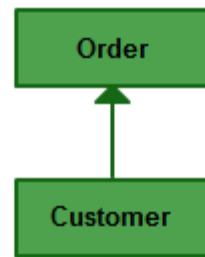


uOttawa

L'Université canadienne
Canada's university



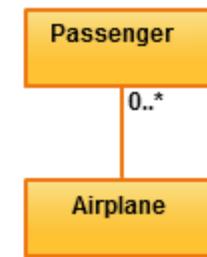
Association



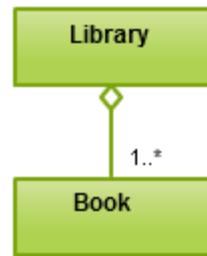
Directed Association



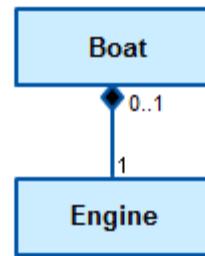
Reflexive Association



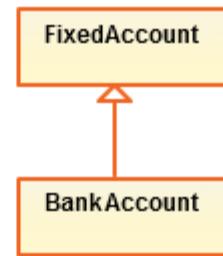
Multiplicity



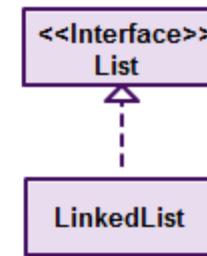
Aggregation



Composition



Inheritance



Realization



uOttawa

L'Université canadienne
Canada's university

ÉVÉNEMENTS DANS UML

Dans UML, on peut modéliser quatre types d'événements:

- Signaux: objet envoyé **d'une façon asynchrone** par un objet et reçu par un autre
- Appels: appels de méthode entre objets (souvent d'une façon synchronisée)
- Passage du temps (une minuterie)
- Changement d'état

Les événements peuvent être externes ou internes

- Les événements externes sont ceux qui passent entre le système et ses acteurs (ex. peser un bouton de GUI)
- Les événements internes sont ceux qui sont passés à travers les objets qui résident dans le système (ex. exception IO)

DIAGRAMMES D'ACTIVITÉS UML

Diagrammes d'activités montrent le flux de travail dès le début jusqu'à la fin

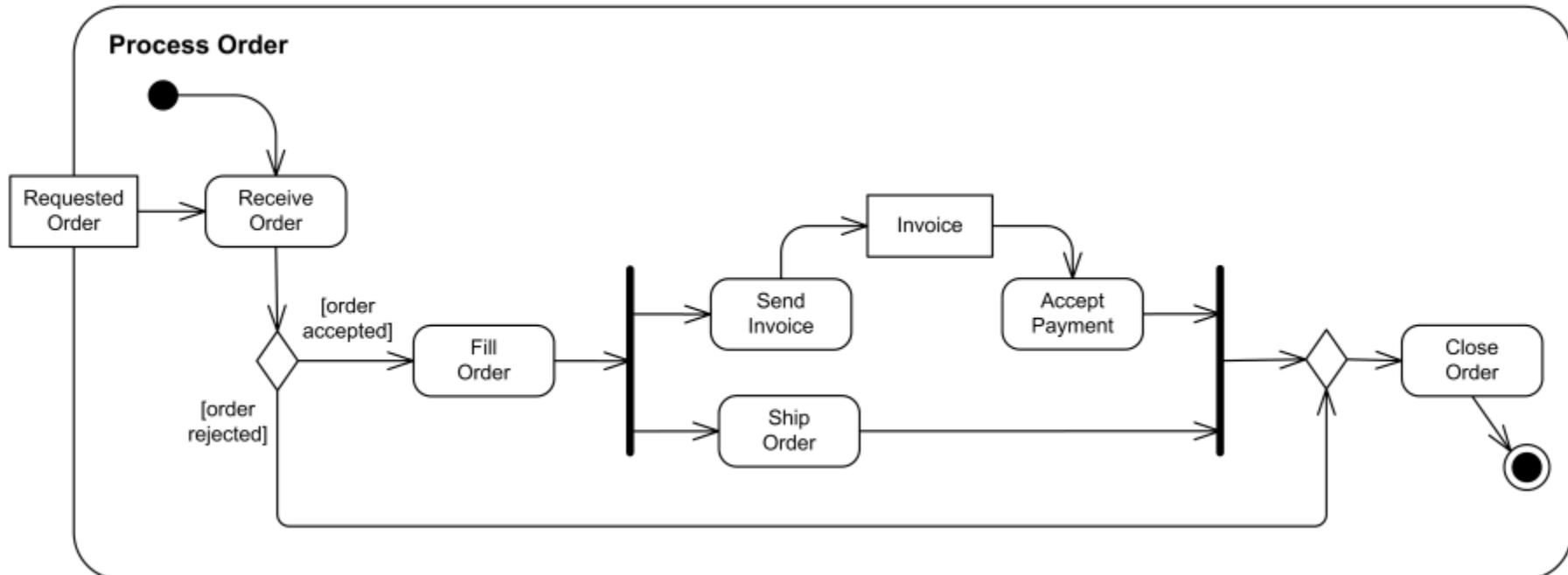
- Détail les nombreux trajets de décision qui existent dans la progression des événements contenus dans l'activité

Très utile pour modéliser le comportement de concurrence

DIAGRAMMES D'ACTIVITÉS UML

Un exemple d'un diagramme d'activité est illustrée ci-dessous

(Nous reviendrons sur ce diagramme)





uOttawa

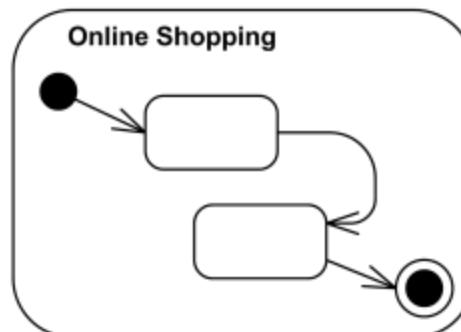
L'Université canadienne
Canada's university

ACTIVITÉ

Une activité est la spécification d'une séquence paramétrée de comportement

- Ça prend du temps
- Semblable à un état, où le critère pour quitter l'état est l'achèvement de l'activité

Représenté par un rectangle aux coins arrondis renfermant toutes les actions et les flux de contrôle





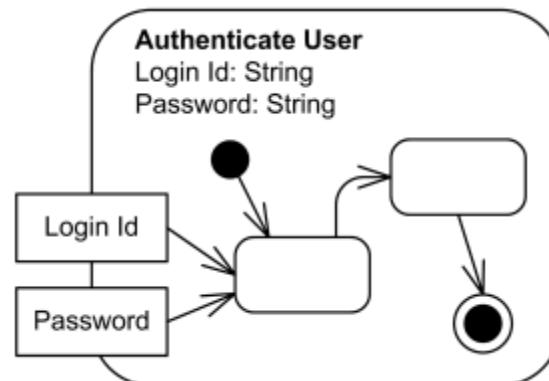
uOttawa

L'Université canadienne
Canada's university

PARAMÈTRES D'ACTIVITÉS

Les paramètres d'activité sont affichés sur le bord et listés en-dessous du nom d'activités tel que:

nom du paramètre: type du paramètre





ACTIONS

Une action représente une étape unique dans une activité

Perform
Action

L'action pourrait être exprimée dans un langage d'action dépendant de l'application

```
for (Account a: accounts)
    a.verifyBalance();
end_for
```

Il existe quatre façons desquelles une action peut être déclenchée

1. Aussitôt qu'une activité commence
2. Durant la durée de vie de l'activité
3. En réponse à un événement
4. Juste avant la complétion de l'activité



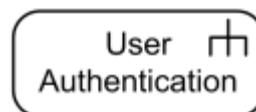
uOttawa

L'Université canadienne
Canada's university

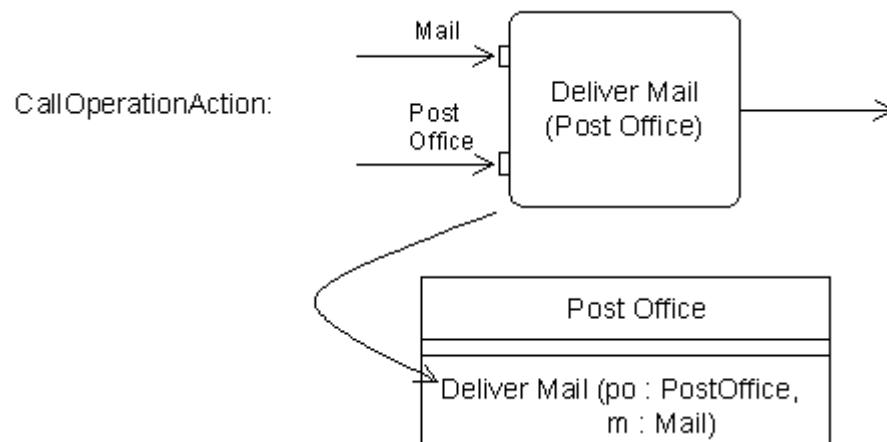
ACTIONS D'APPEL

Action d'appel d'activité : permet d'appeler une autre activité

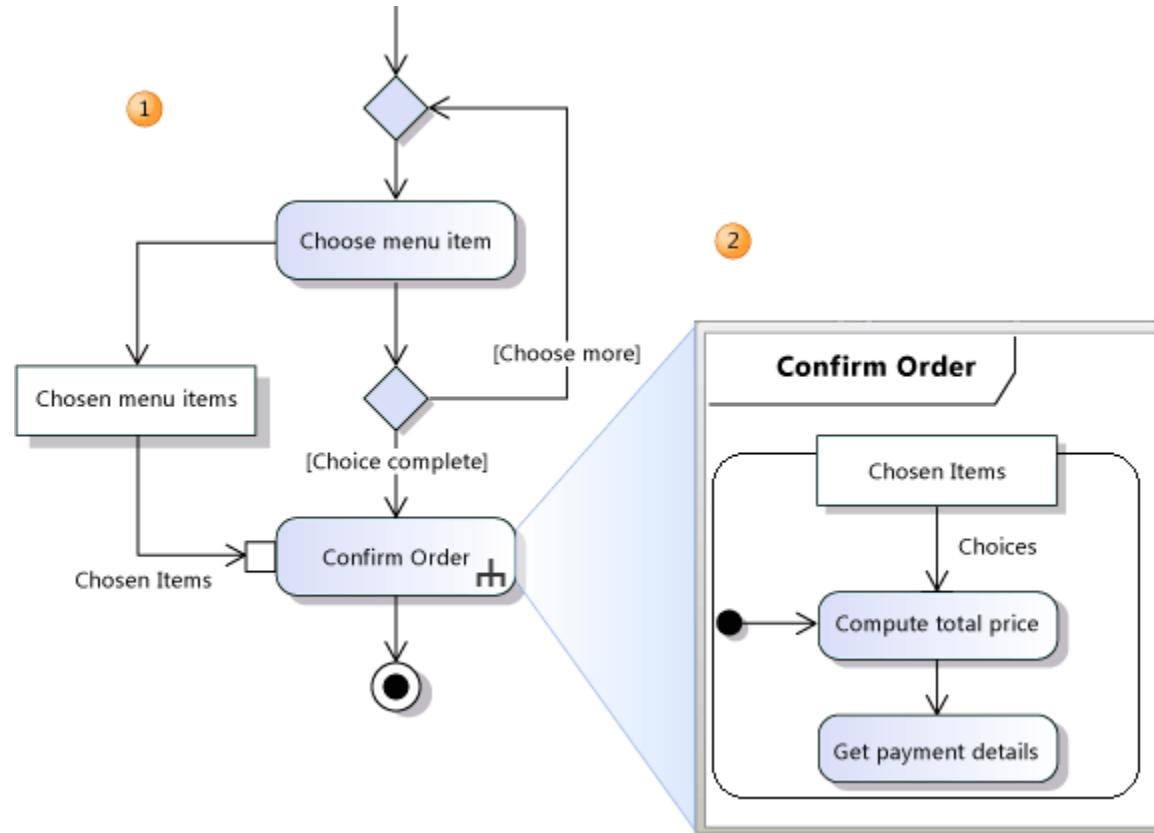
- Ceci évite les définitions redondantes d'activités



Action d'appel d'opération : appelle le comportement d'un élément de structure (opération d'une classe)



ACTIONS D'APPEL D'ACTIVITÉ



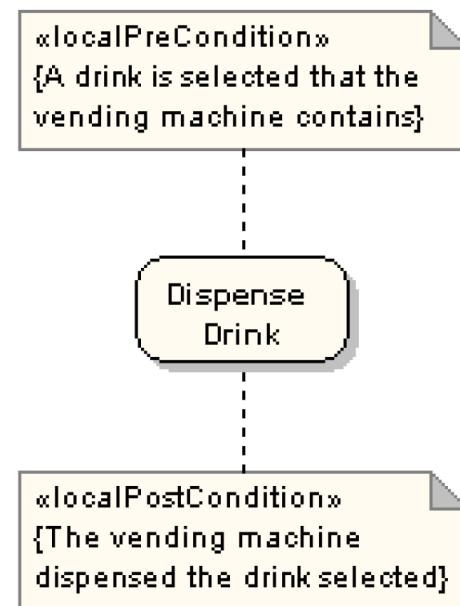


uOttawa

L'Université canadienne
Canada's university

ACTIONS ET CONTRAINTES

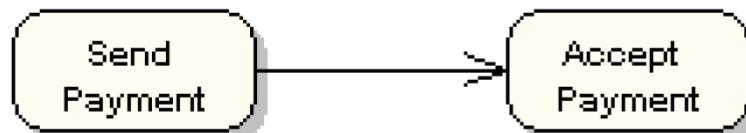
Les contraintes peuvent être attachés aux actions



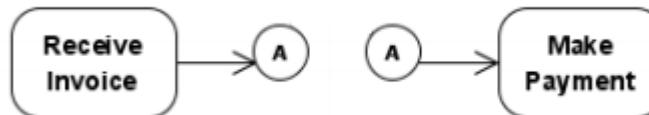
FLUX DE CONTRÔLE

Indique le flux de contrôle d'une action à la prochaine

- Sa notation est une flèche



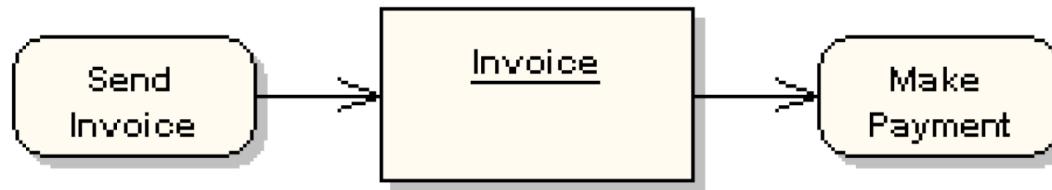
- Utilisation de connecteurs:



FLUX D'OBJETS

Un flux d'objet est un trajet le long duquel des objets peuvent passer

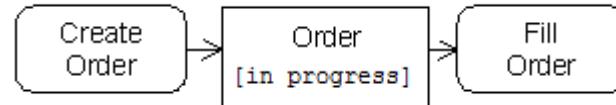
- Un objet est représentée par un rectangle



Un autre façon pour montrer la même chose:



On peut aussi montrer l'état de l'objet qui passe (montré en-dessous du nom de l'objet, entre parenthèses)





uOttawa

L'Université canadienne
Canada's university

NŒUDS INITIAUX ET FINAUX

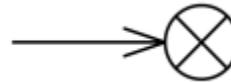
Un nœud initial est un nœud de contrôle auquel le flux commence lorsque l'activité est invoquée



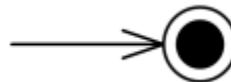
Les activités peuvent avoir plus qu'un seul nœud initial

- Dans ce cas, lorsqu'on invoque une activité, plusieurs flux commencent, un flux à chaque nœud initial

Un nœud final de flux est un nœud final de contrôle qui termine un flux



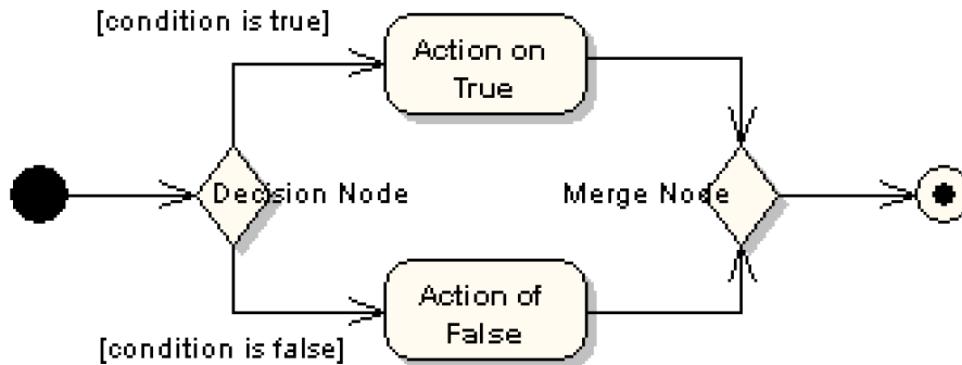
Un nœud final d'activité est un nœud final de contrôle qui arrête tous les flux de l'activité



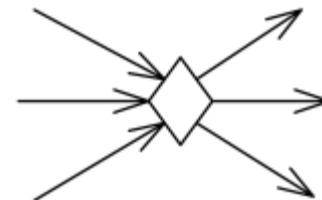
DÉCISION ET NŒUDS DE FUSION

Les nœuds de décision et fusion ont la même notation: une forme de diamant

Le flux de contrôle qui sort d'un nœud de décision sont associés avec des conditions



Nœuds de décision et fusion combinés





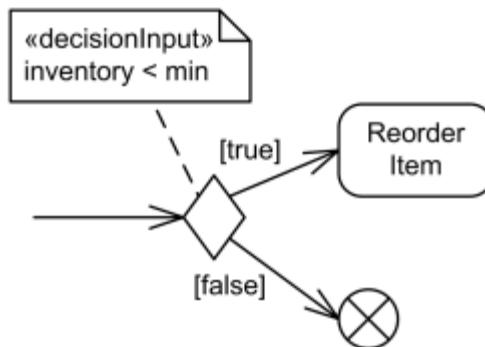
uOttawa

L'Université canadienne
Canada's university

NŒUDS DE DÉCISION

Une décision peut avoir un comportement de l'entrée de décision spécifié

- Les comportements de l'entrée de décision fut introduits dans UML afin d'éviter les re-calculs redondants dans les gardes
- Ils sont spécifiés par le mot-clé «decisionInput» et une condition est placée dans un symbole de note

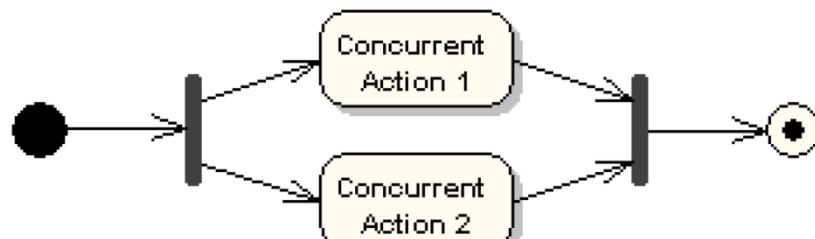


NŒUDS DE FORK ET JONCTION (JOIN)

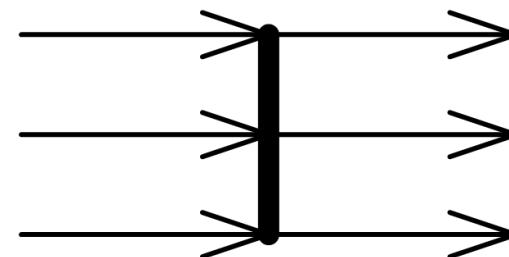
Les nœuds de fork et join ont la même notation: soit une barre horizontale ou verticale

Ils indiquent le début et la fin des threads (fils) de contrôle simultanés

- « Join » synchronise deux entrées et produit une seule sortie
- Le flux de sortie de « join » ne peut pas exécuter jusqu'à ce que toutes les flux d'entrées ont été reçues

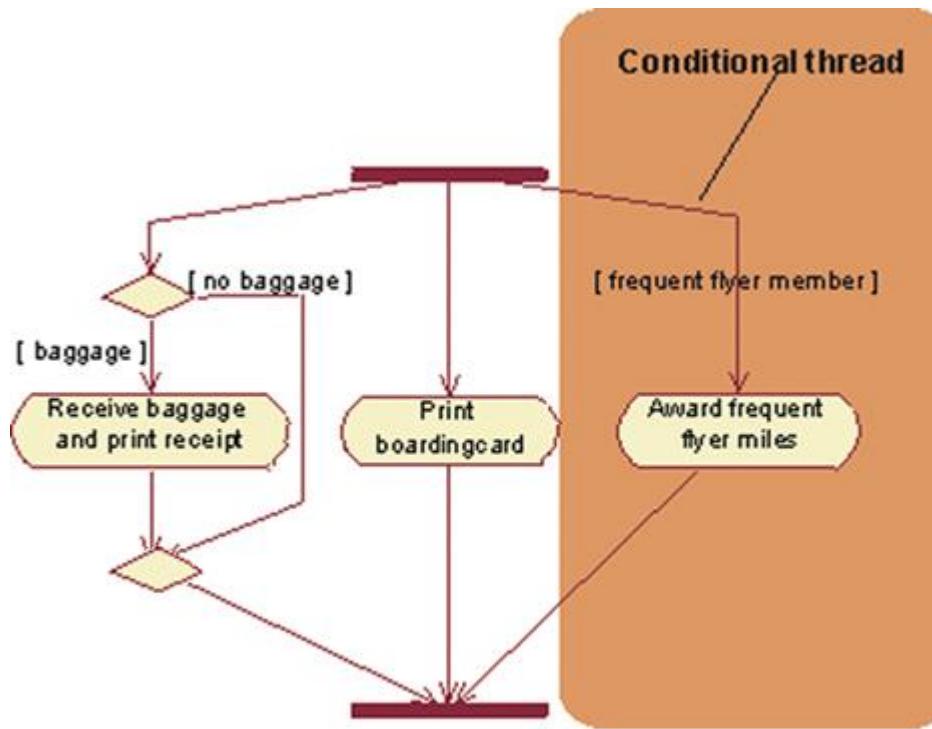


Nœuds fork et join combinés

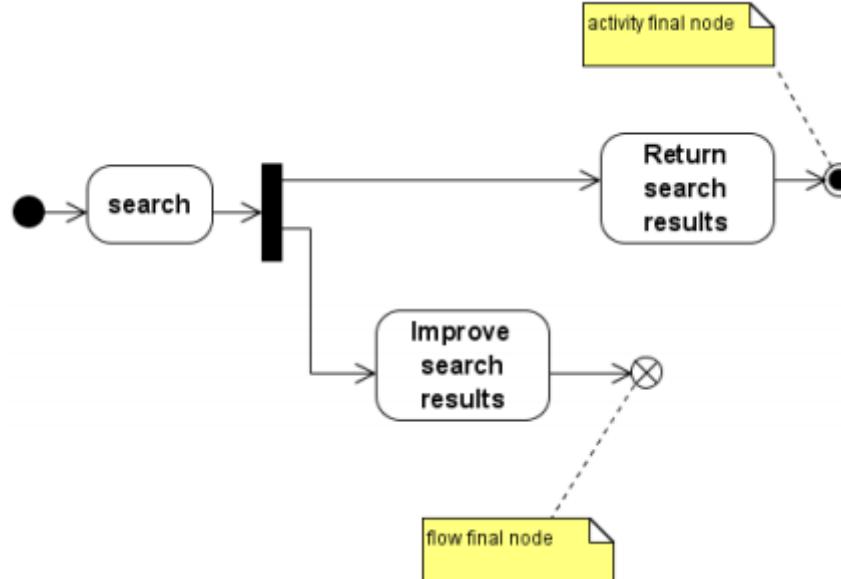


FIL D'EXÉCUTION CONDITIONNEL

Les conditions de garde peuvent être utilisées pour montrer qu'un d'un ensemble de threads simultanés est conditionnel



NŒUDS INITIAUX ET FINAUX

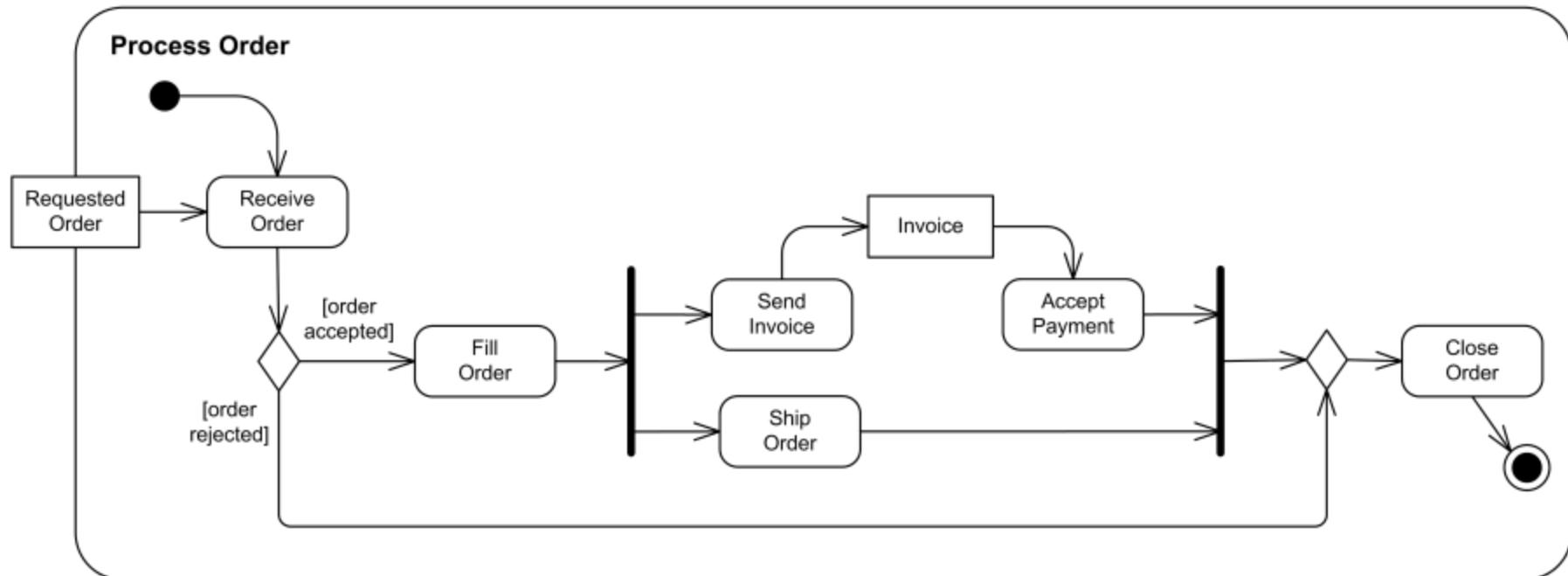


Attention: soyez prudents lorsque vous utilisez un nœud final de flux après un fork

- Dès que le nœud final de l'activité est atteint, toutes les autres actions dans l'activité (incluant ceux avant le nœud final de flux) se terminent

DIAGRAMMES D'ACTIVITÉS UML

Revenant à notre exemple initial

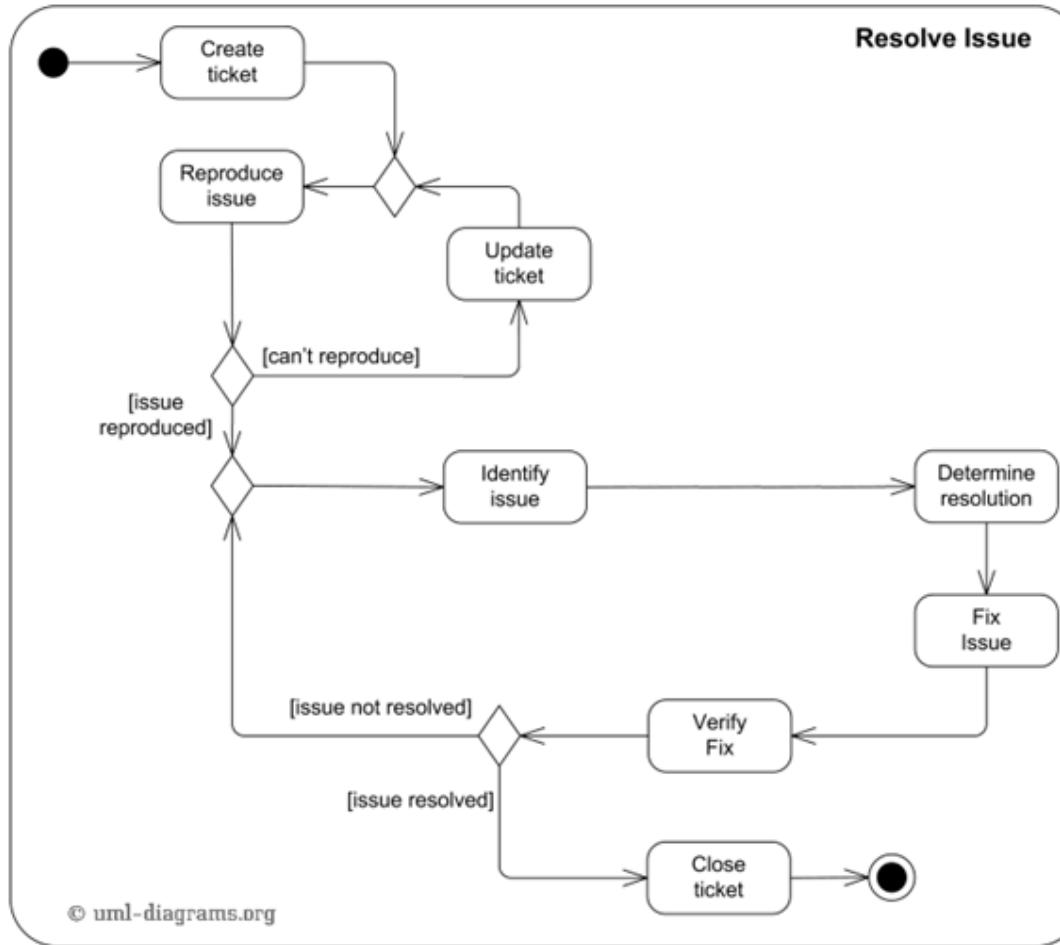




uOttawa

L'Université canadienne
Canada's university

LA GESTION DES PROBLÈMES DANS LES PROJETS DE LOGICIEL

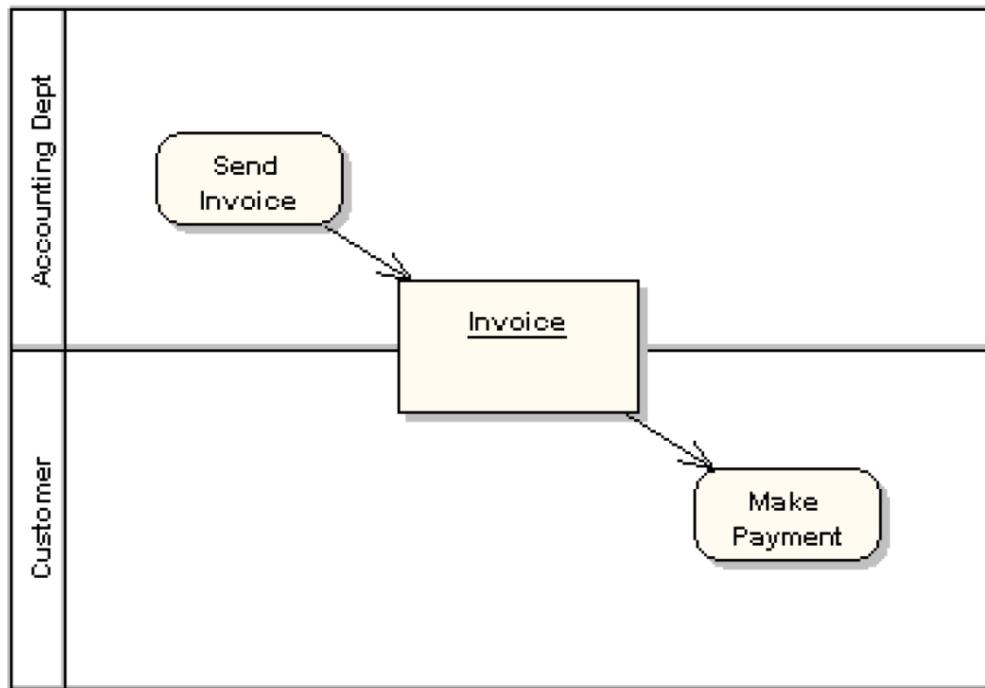


Courtesy of uml-diagrams.org

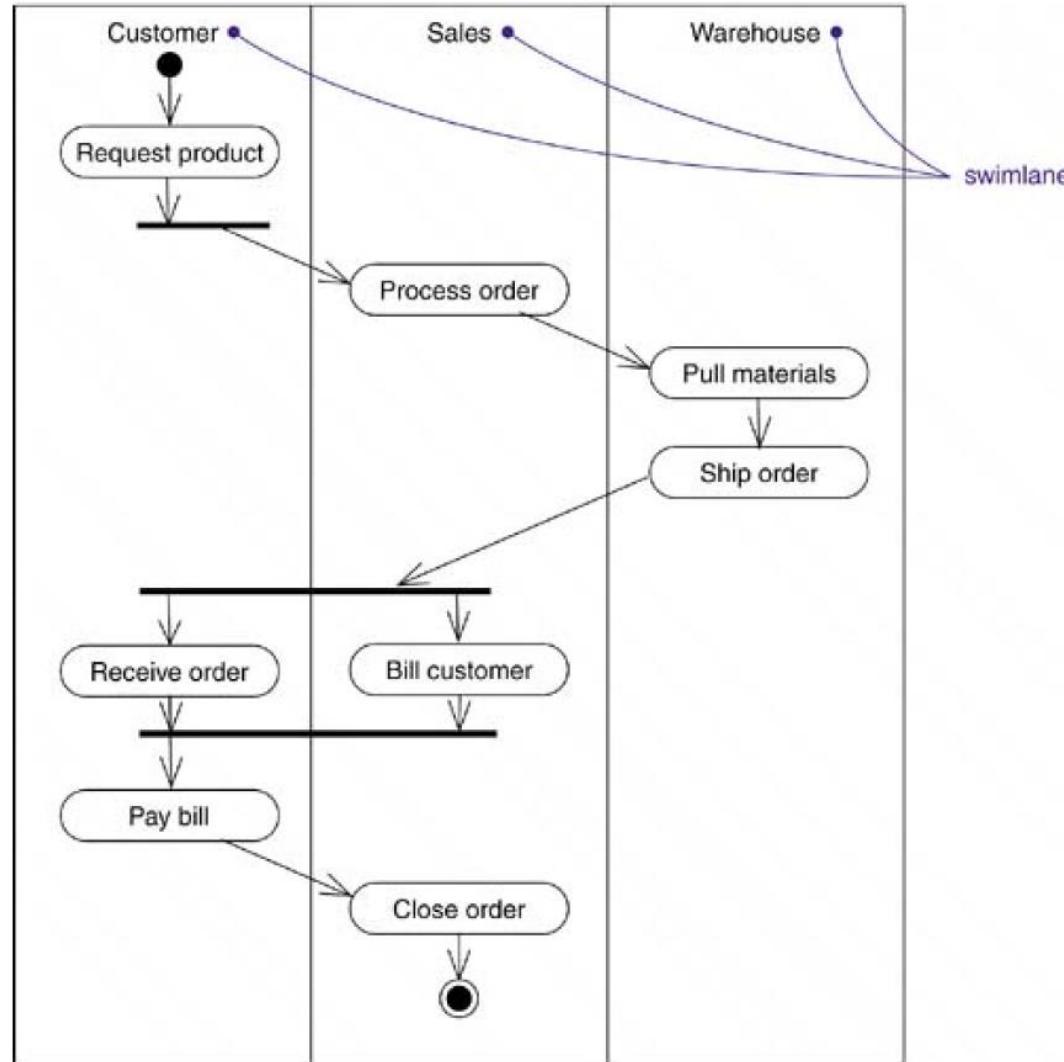
PARTITION

Représenté comme couloirs de natation horizontal ou vertical

- Représente un groupe d'actions qui ont des caractéristiques communes



PARTITION

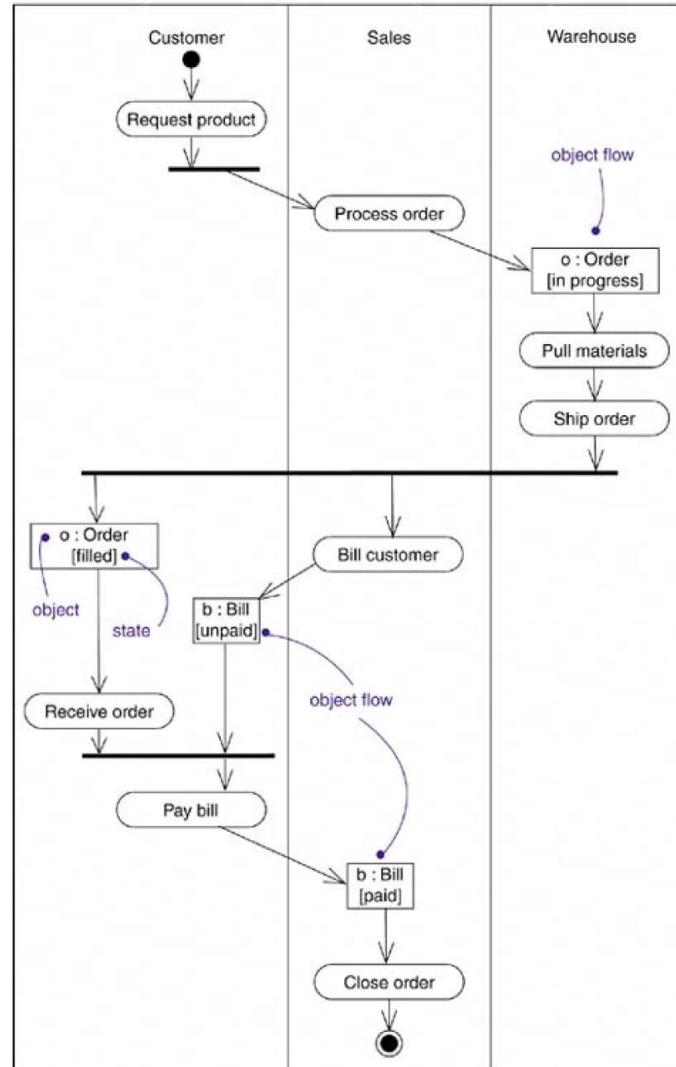




uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE PARTITION AVEC FLUX D'OBJET



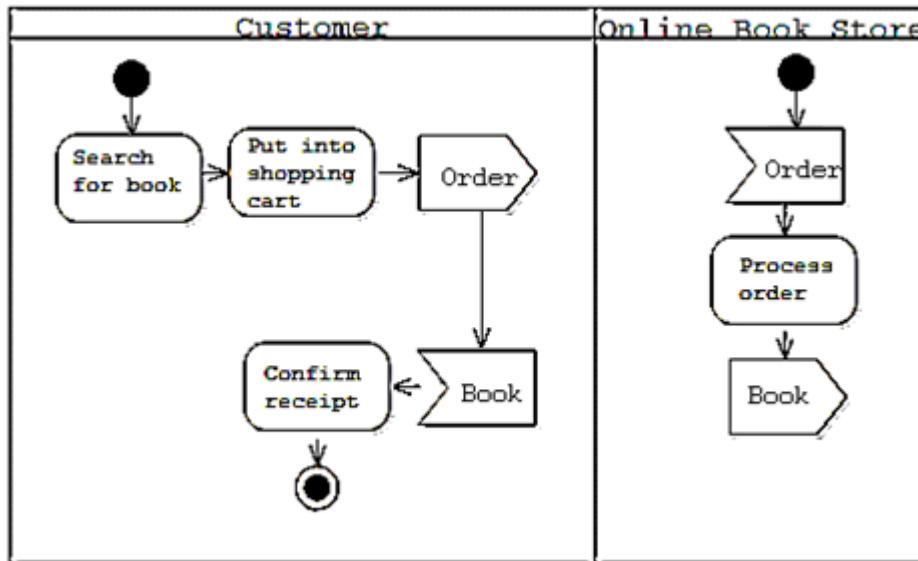
ENVOYER ET RECEVOIR DES SIGNAUX ET DES ÉVÉNEMENTS DE TEMPS

Les flux de contrôle ou les flux d'objets connectent des actions

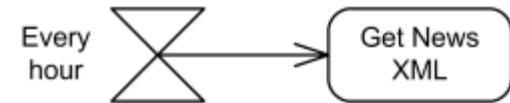
- Ils définissent des processus "synchrones" où le flux est déterminé par une séquence ordonnée d'étapes

Grâce à l'utilisation de signaux, les processus peuvent être désaccouplés

- Nous pouvons atteindre une communication asynchrone



Action d'événement de temps produit une sortie chaque heure

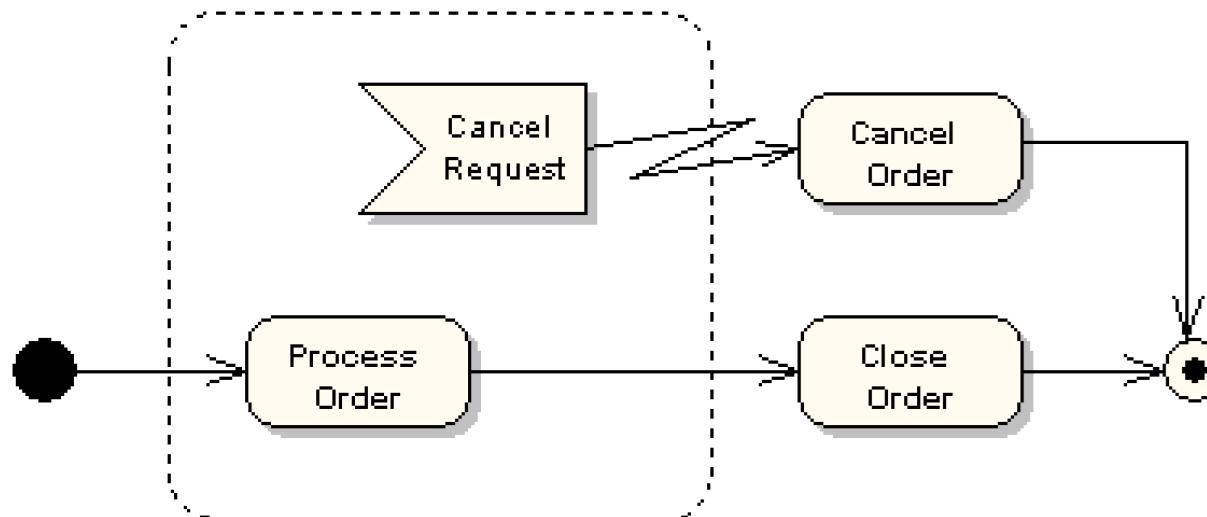


RÉGIONS D'ACTIVITÉ INTERRUPTIBLE

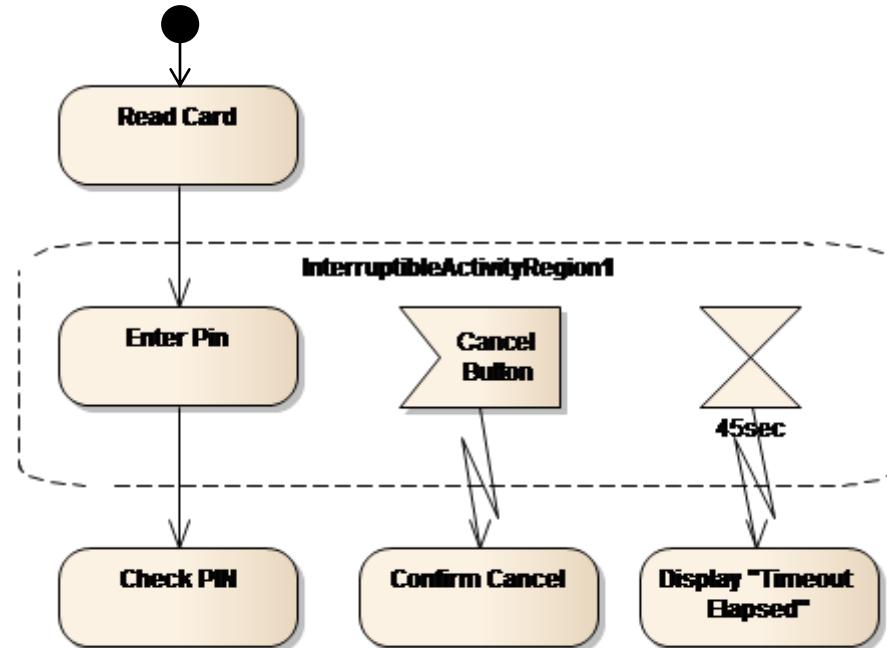
Entoure un ensemble d'actions qui peut être interrompu

Exemple ci-dessous:

- L'action « Process Order » vas exécuter jusqu'à la fin où le contrôle vas être passé vers l'action « Close Order », à moins que l' Interruption « Cancel Request » est reçu, ce qui vas causer le contrôle d'être passé vers l'action « Cancel Order ».



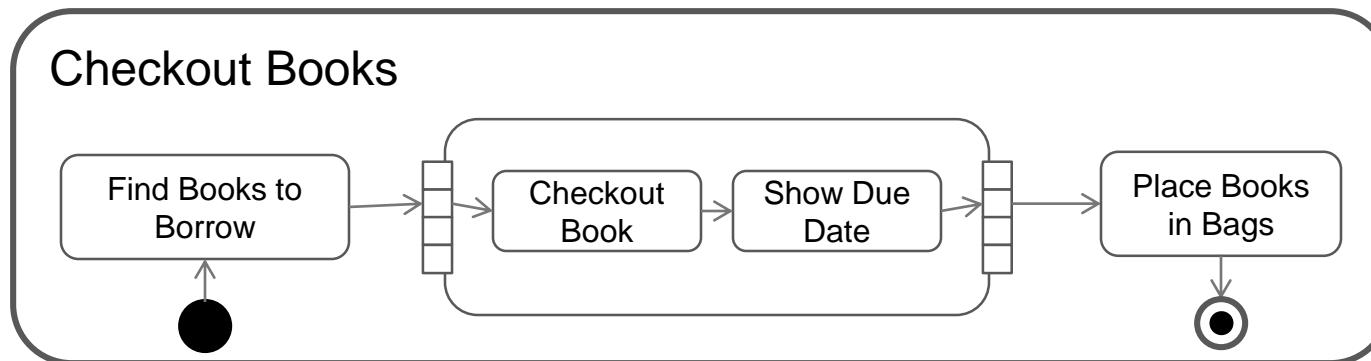
RÉGIONS D'ACTIVITÉ INTERRUPTIBLE



RÉGIONS D'EXTENSION

Une région d'extension est une zone d'activité qui exécute à plusieurs reprises pour consommer tous les éléments d'une collection d'entrée

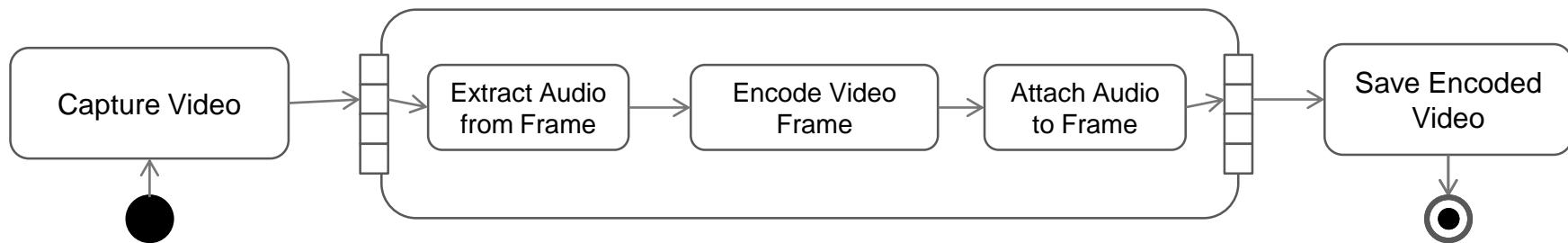
Exemple d'un « checkout » des livres dans une bibliothèque modélisée à l'aide d'une zone d'expansion



RÉGIONS D'EXTENSION

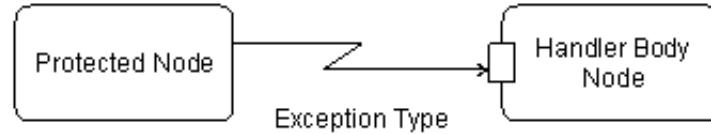
Un autre exemple: Encodage de vidéo

Encode Video



GESTIONNAIRES D'EXCEPTIONS

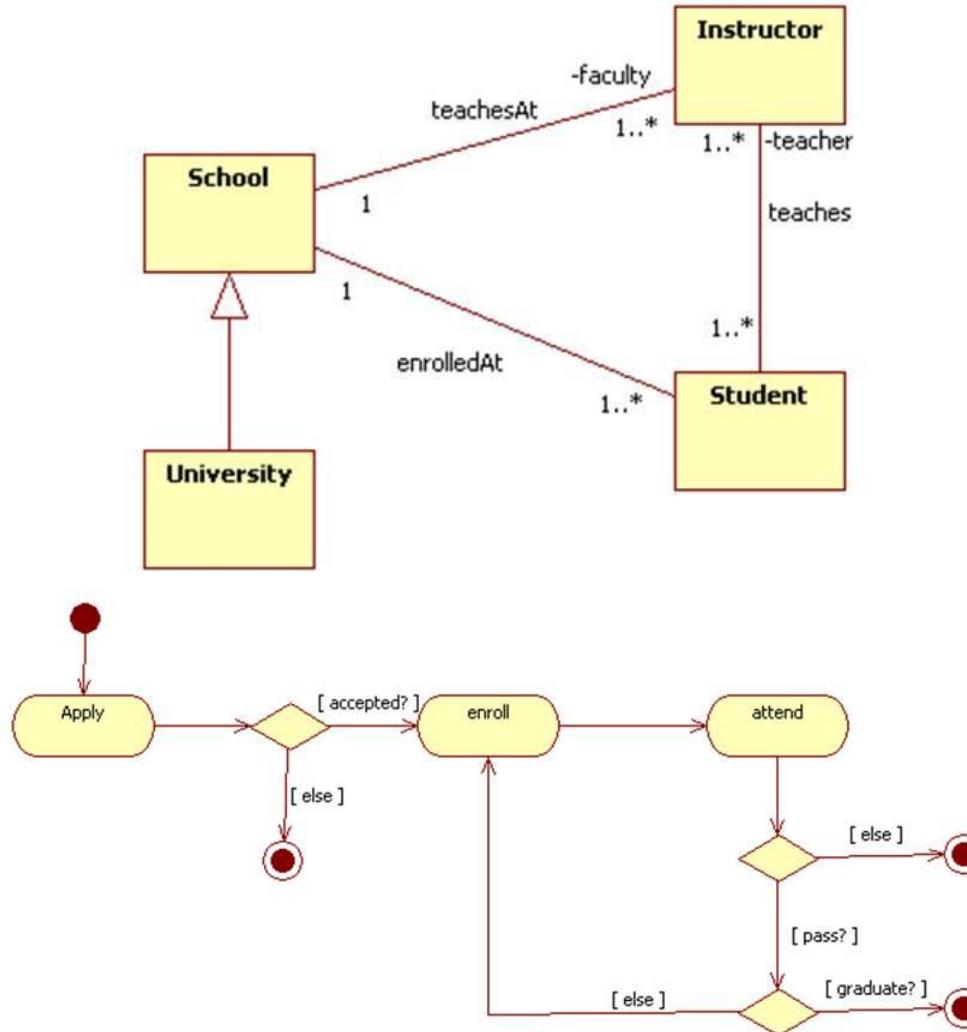
Un gestionnaire d'exception est un élément qui spécifie quoi exécuter dans le cas où l'exception spécifiée se produit pendant l'exécution du nœud protégé



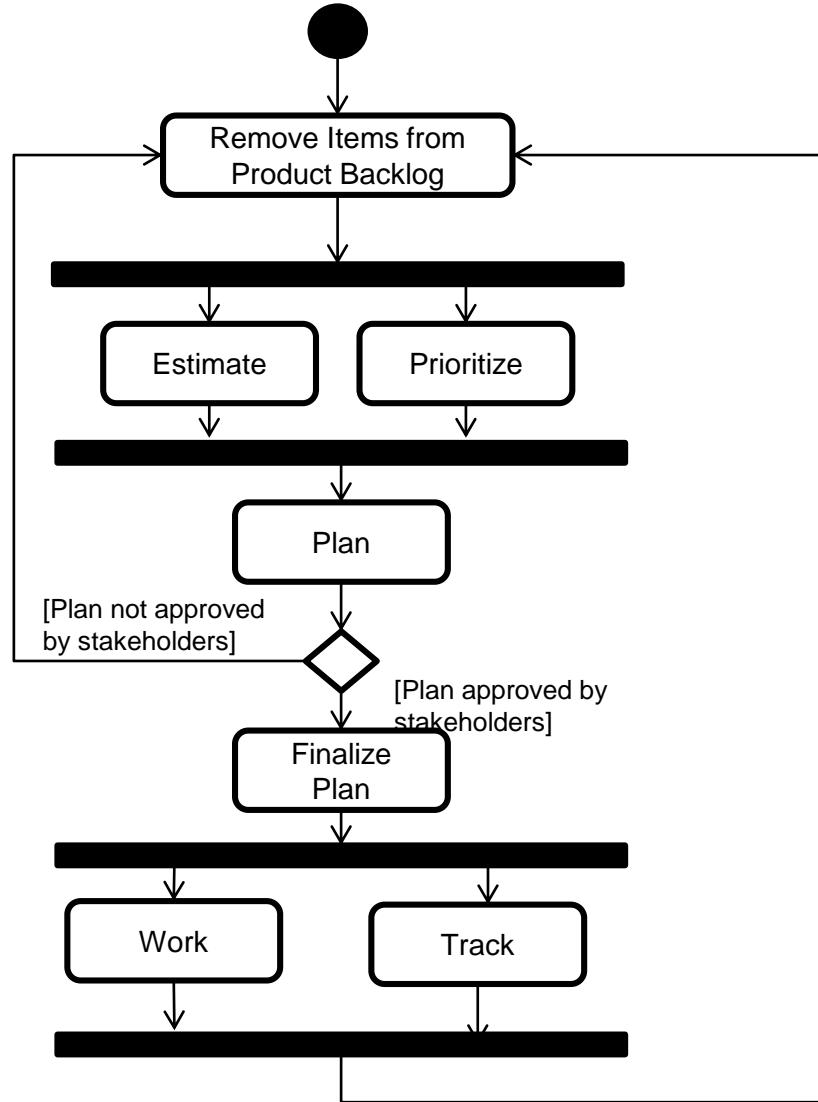
En Java

- “Try block” correspond à un “Protected Node”
- “Catch block” correspond à un “Handler Body Node”

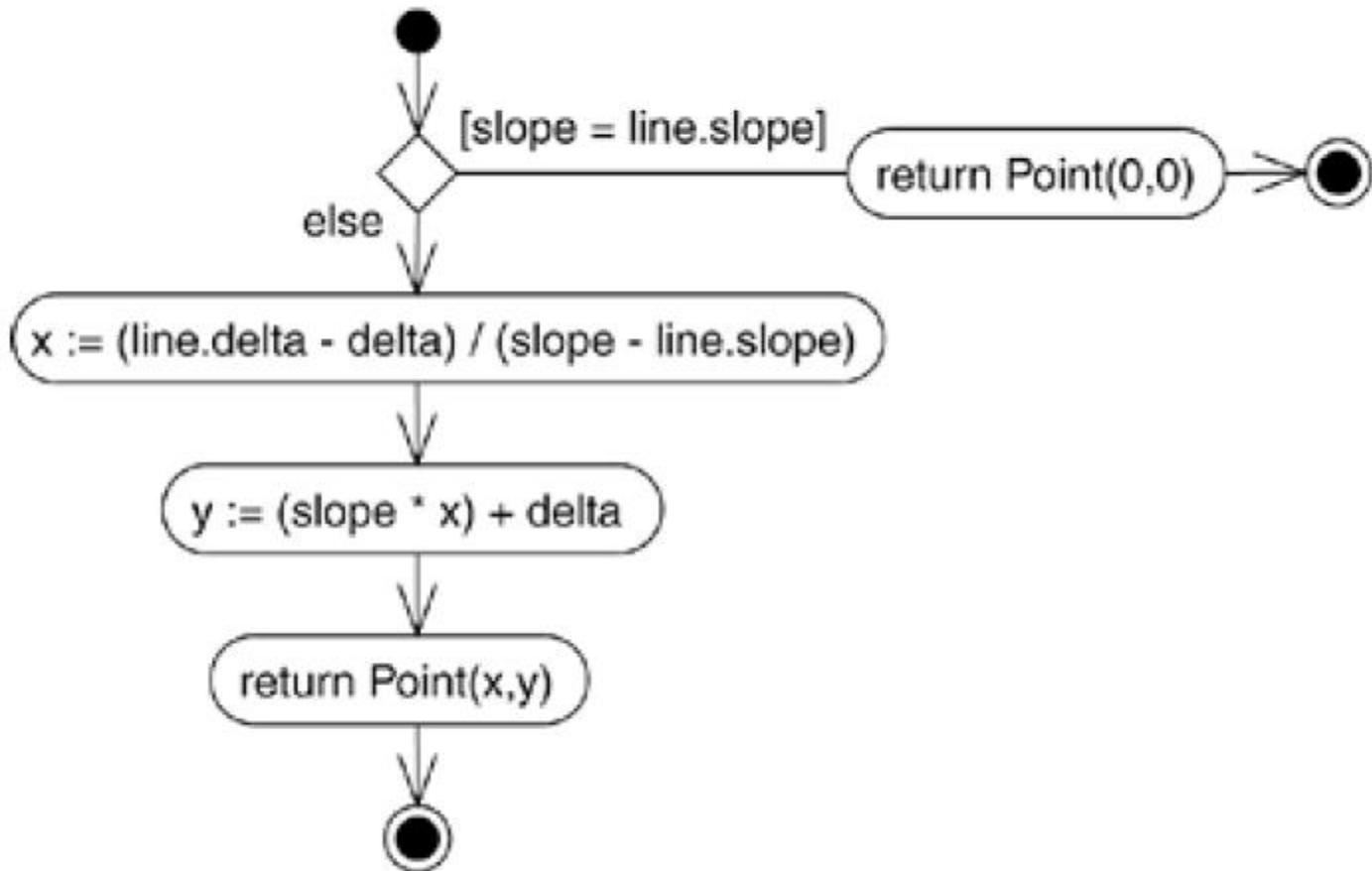
DIAGRAMMES D'ACTIVITÉ UTILISÉS DANS LA MODÉLISATION DE DOMAINES



MODÉLISATION D'UN PROCESSUS LOGICIEL À L'AIDE D'UN DIAGRAMME D'ACTIVITÉ



DIAGRAMMES D'ACTIVITÉ POUR MODÉLISER UNE OPÉRATION



GÉNÉRATION D'UN DIAGRAMME D'ACTIVITÉ UML À PARTIR D'UNE HISTOIRE D'UTILISATEUR

Un diagramme d'activité UML peut fournir une représentation visuelle d'une histoire d'utilisateur pour toutes les parties prenantes

- Des histoires d'utilisateurs complexes (à haut risque) peuvent être développées avec un diagramme d'activité UML
- Nous permet d'identifier tout malentendu entre les parties prenantes
- Rappelez-vous: plus tôt ces malentendus sont réglés, moins ils sont coûteux

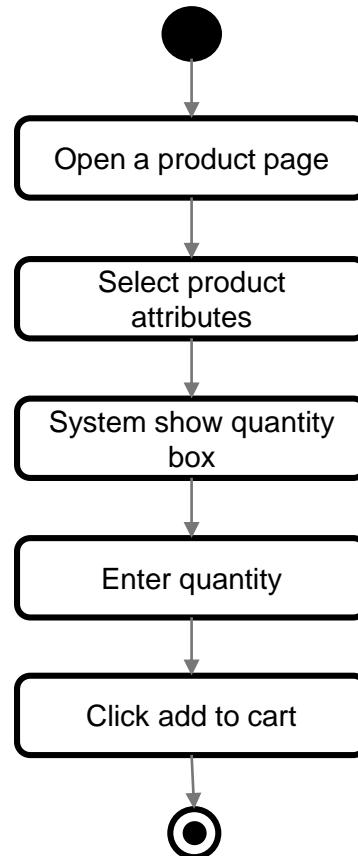
GÉNÉRATION D'UN DIAGRAMME D'ACTIVITÉ UML À PARTIR D'UNE HISTOIRE D'UTILISATEUR

Examinons cet exemple d'une histoire d'utilisateur pour un système de commerce électronique typique:

En tant qu'utilisateur, je veux ajouter un article au panier afin de pouvoir effectuer un achat

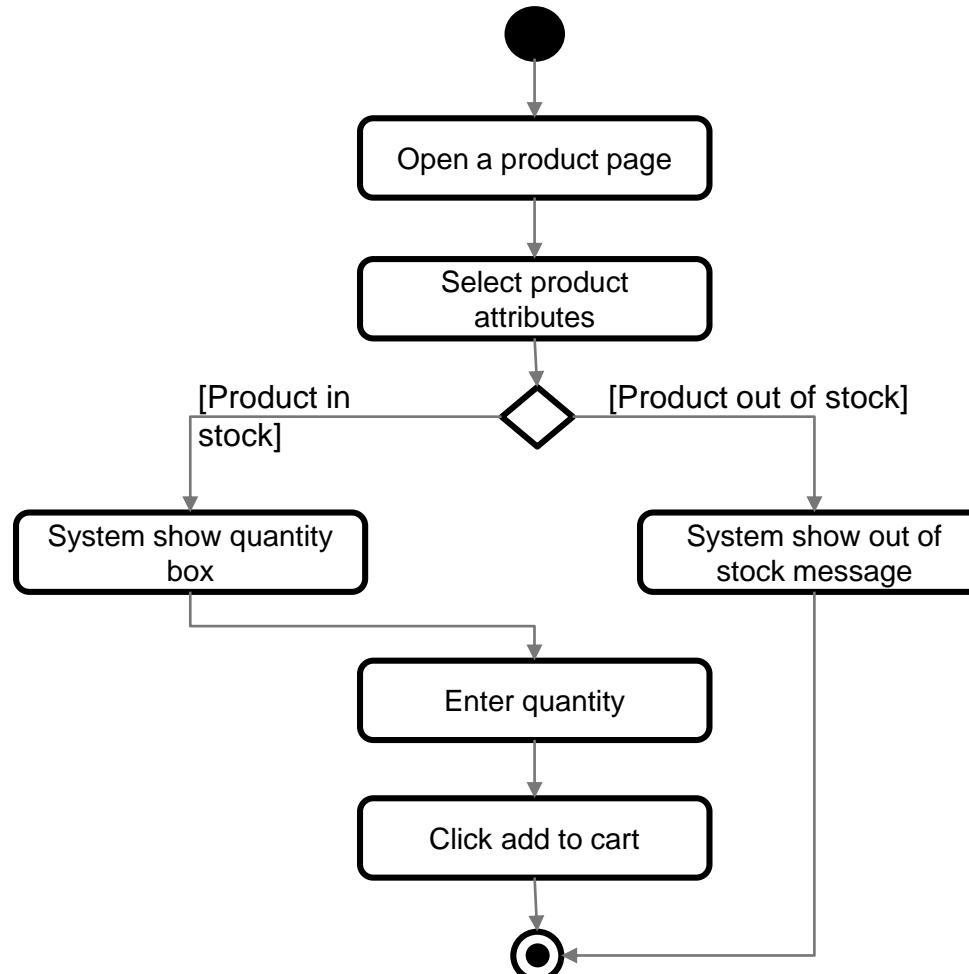
GÉNÉRATION D'UN DIAGRAMME D'ACTIVITÉ UML À PARTIR D'UNE HISTOIRE D'UTILISATEUR

Commencez avec un diagramme simple qui énumère les étapes nécessaires



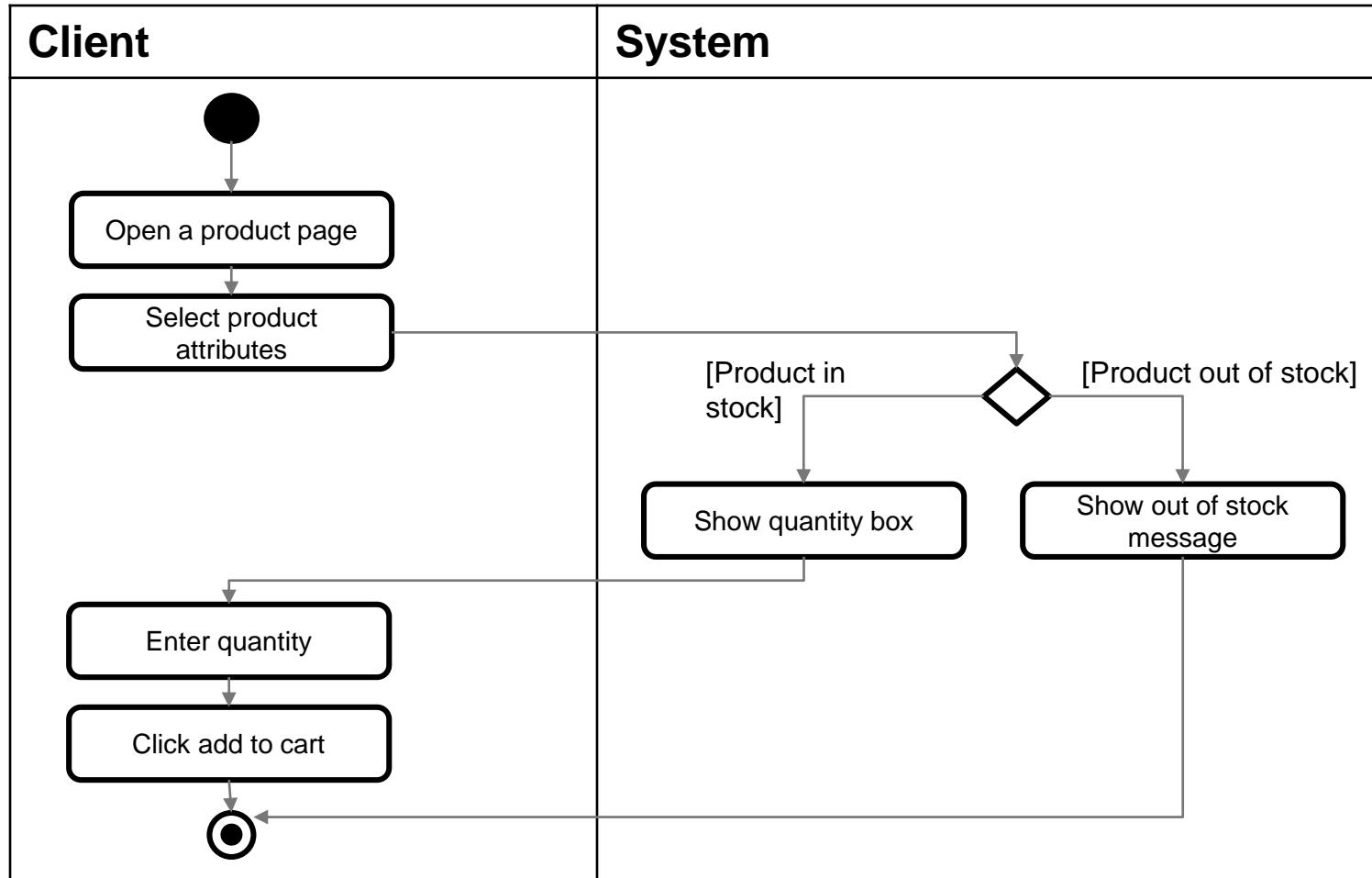
GÉNÉRATION D'UN DIAGRAMME D'ACTIVITÉ UML À PARTIR D'UNE HISTOIRE D'UTILISATEUR

Affiner le modèle pour ajouter des séquences alternatives



GÉNÉRATION D'UN DIAGRAMME D'ACTIVITÉ UML À PARTIR D'UNE HISTOIRE D'UTILISATEUR

Ajoutez des partitions (couloirs de natation) pour clarifier qui fait quoi...



MERCI

QUESTIONS?

SÉANCE 4

MACHINES D'ÉTATS UML



uOttawa

L'Université canadienne
Canada's university

SUJETS

États

Transitions

Gardes

Effets

Actions d'État

Décisions

États composés

Entrée et sortie alternatives

États d'histoire

Étude de cas



DIAGRAMME D'ACTIVITÉ VS MACHINES D'ÉTATS

Pour les diagrammes activités

- Sommet représente les Actions
- Arrêt (flèche) représente une transition qui va avoir lieu après la complétion d'une action et avant le début d'une autre (flux de contrôle)

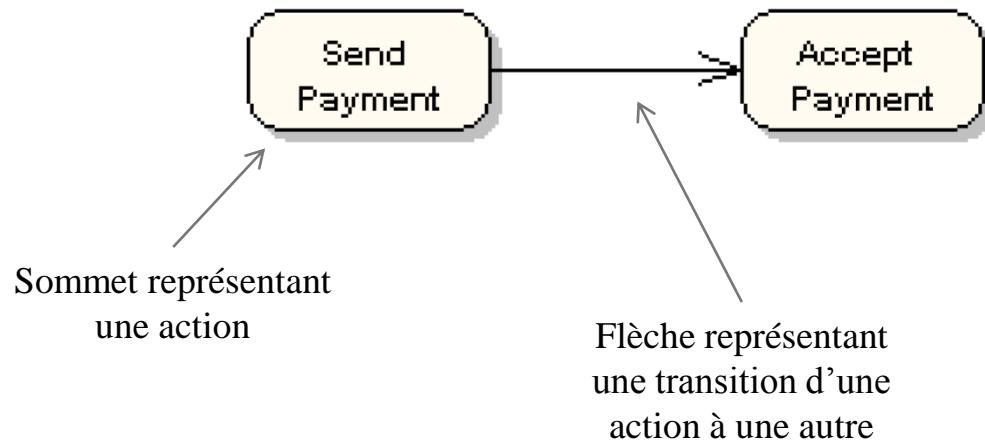
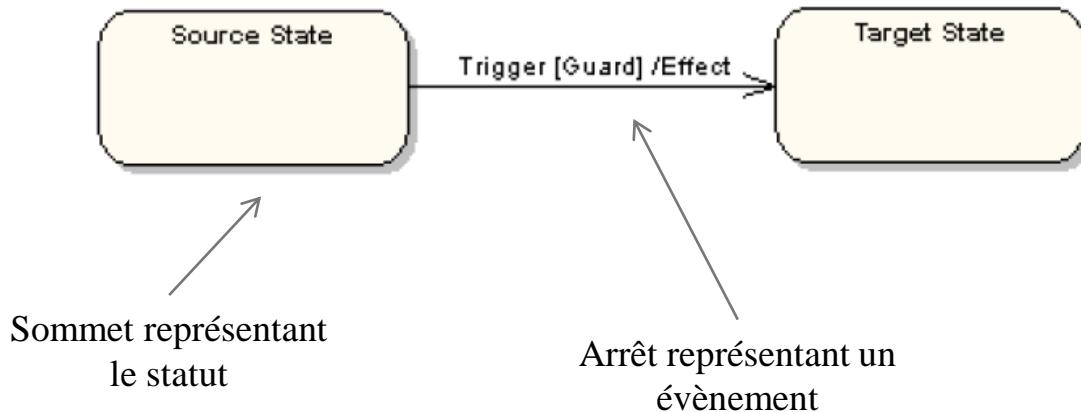


DIAGRAMME D'ACTIVITÉ VS MACHINES D'ÉTATS

Pour les machines d'états

- Sommet représente le statut (état) d'un processus
- Arrêt (flèche) représente un évènement





uOttawa

L'Université canadienne
Canada's university

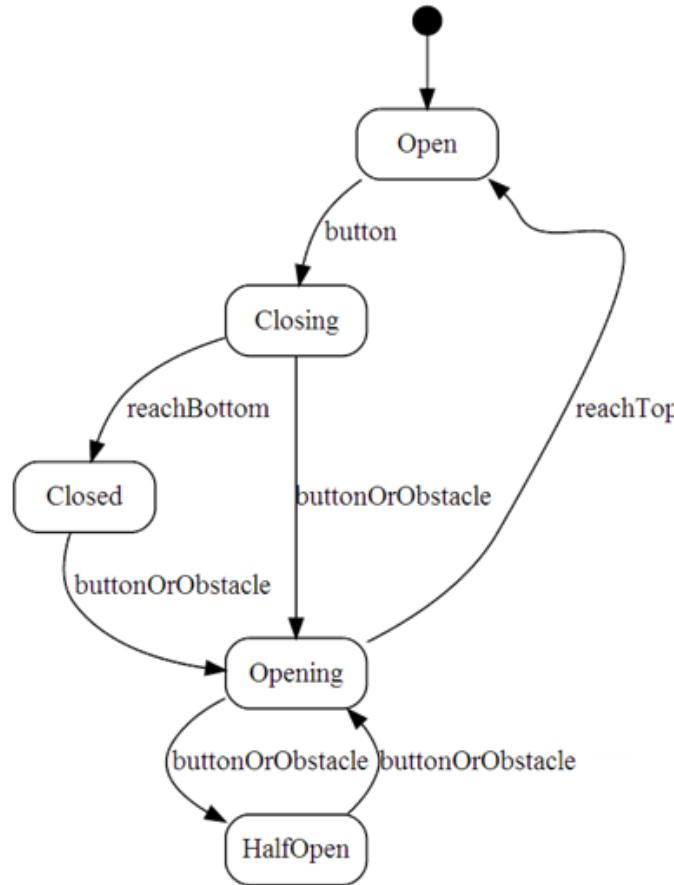
MACHINES D'ÉTATS UML

Utilisé pour modéliser le comportement dynamique d'un processus

- Peut être utilisé pour modéliser de haut niveau le comportement général d'un système entier
- Peut être utilisé pour modéliser le comportement détaillé d'un seul objet
- Tout autre niveau de détail entre ces deux extrêmes est possible

EXEMPLE D'UNE MACHINE D'ÉTATS

Exemple Pour Modéliser les états d'une porte de garage





uOttawa

L'Université canadienne
Canada's university

ÉTATS

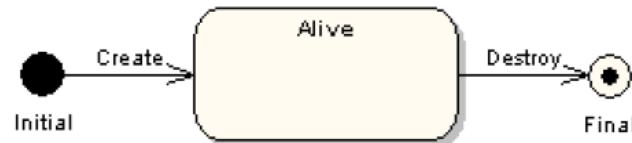
Symbol pour un état



Un système dans un état va rester dans cet état jusqu'à l'occurrence d'un évènement qui lui cause de changer d'état

- Être dans un état veut dire que le système va se comporter d'une façon spécifique en réponse à un évènement qui se produit

Symboles pour les états initial et final



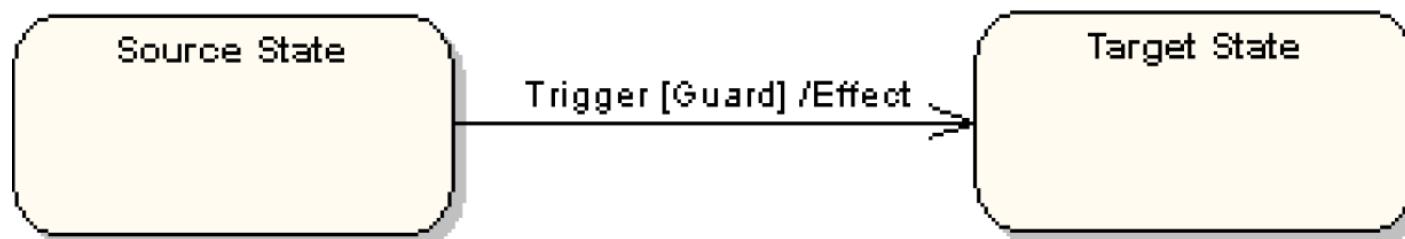


uOttawa

L'Université canadienne
Canada's university

TRANSITIONS

Transitions sont représentées par des flèches





uOttawa

L'Université canadienne
Canada's university

TRANSITIONS

Une transition représente un changement d'état en réponse à un évènement

- It est supposé se produire d'une façon instantanée (ça ne prend pas du temps)

Une transition peut avoir

- **Un Déclencheur (Trigger):** la cause de la transition qui peut être un évènement
- **Une Garde (Guard):** une condition qui doit être vraie pour que le déclencheur cause la transition
- **Un Effet (Effect):** une action qui va être invoquée directement sur le système ou l'objet modélisé lorsque la transition se produit



uOttawa

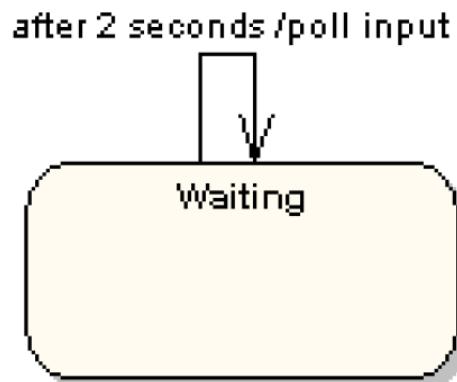
L'Université canadienne
Canada's university

AUTO TRANSITION

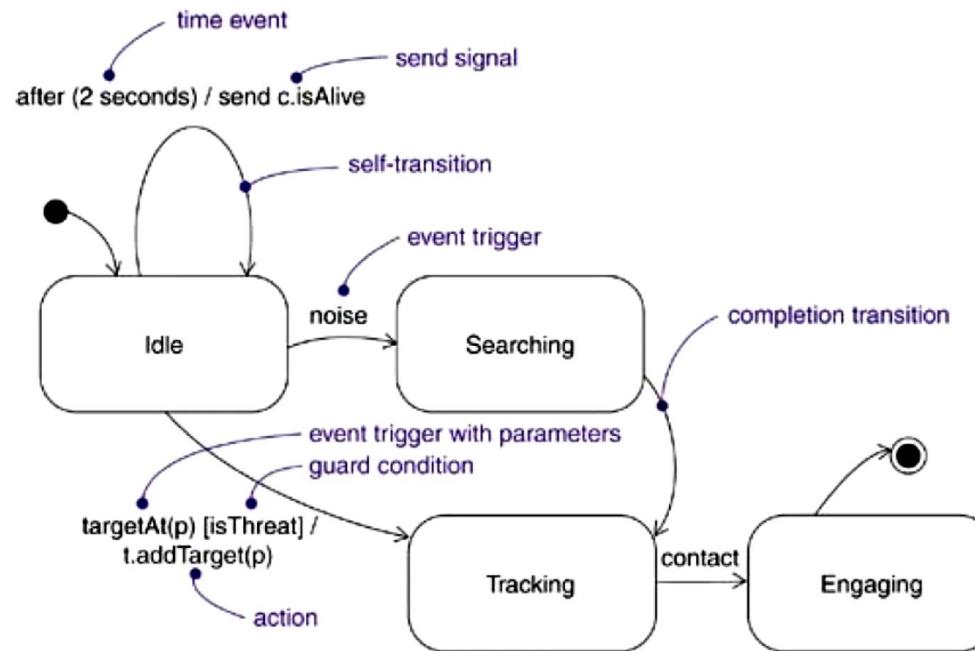
Un état peut avoir une transition qui est dirigée vers lui-même: auto transition

- Ces auto-transitions sont plus utiles quand elles sont associées avec un effet

Ceci est plus utile quand un effet est associé avec la transition



TRANSITIONS D'ACHÈVEMENT



Transition d'achèvement: transition sans déclencheur initié implicitement lorsque l'état source a complété son comportement



uOttawa

L'Université canadienne
Canada's university

CONDITION DE GARDE

Une condition de garde est une expression booléenne placée entre crochets et spécifiée après l'événement déclencheur

La condition de garde n'est évaluée qu'après l'événement déclencheur de sa transition se produit

- Il est possible d'avoir plusieurs transitions à partir du même état source et avec le même déclencheur d'événement, tant que les conditions de garde ne se chevauchent pas



uOttawa

L'Université canadienne
Canada's university

EFFET

Un effet est un comportement qui s'exécute lorsqu'une transition se déclenche

Les effets peuvent inclure

- Calcul en ligne (inline computation)
- Appels d'opération
- Création et destruction d'un autre objet
- Envoi d'un signal à un objet

Pour indiquer l'envoi d'un signal, vous pouvez préfixer le nom du signal avec le mot clé *send*



uOttawa

L'Université canadienne
Canada's university

ACTIONS D'ÉTATS

Un effet peut être associé avec une transition

Si l'état de destination est associé avec plusieurs transitions incidentes (*plusieurs transitions arrive a cet état*), et chaque transition possède le même effet:

- Ce serait mieux d'associer l'effet avec les états (au lieu des transitions)
- On peut réaliser cela en utilisant un effet d'entrée
- On peut aussi ajouter un effet de sortie





uOttawa

L'Université canadienne
Canada's university

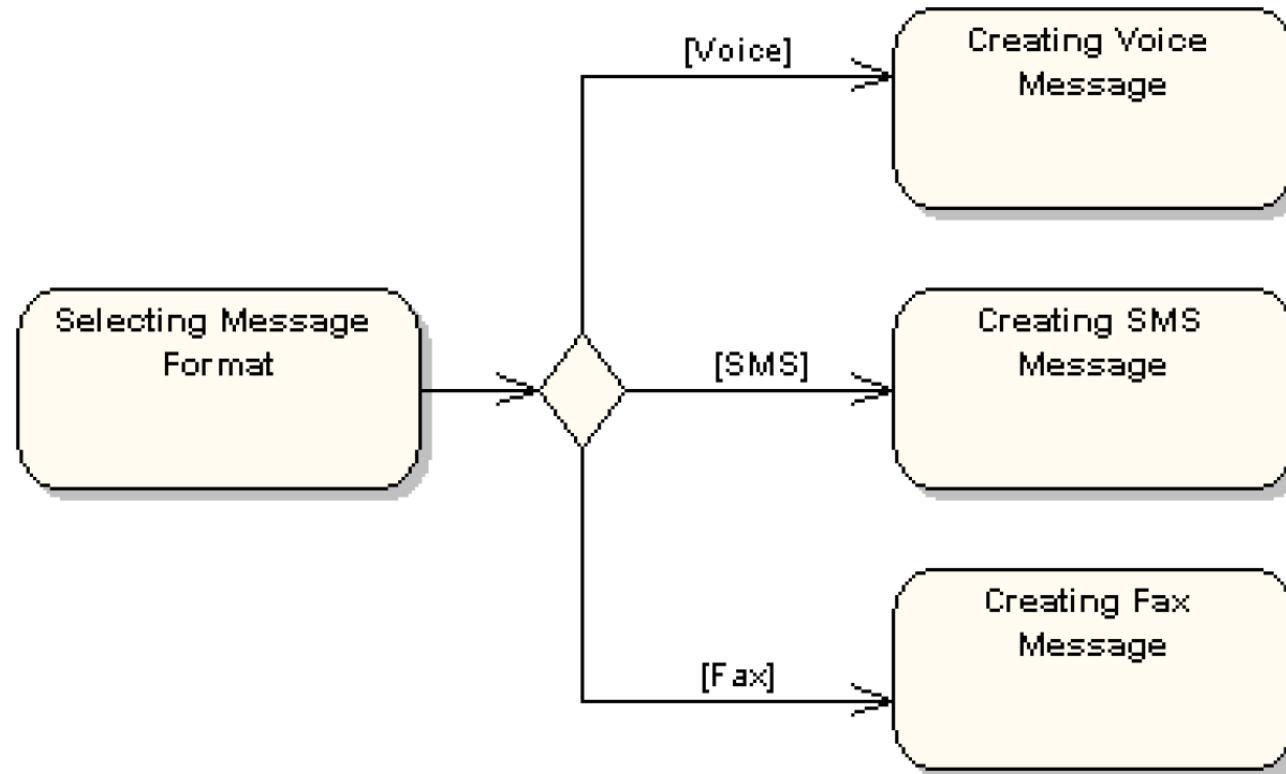
DÉCISIONS

Comme dans les diagrammes d'activité, on peut utiliser des nœuds de décisions (ont les appelle pseudo-états)

Les nœuds de décisions sont représentés par des symboles de diamants

- On a toujours une transition d'arrivée et deux ou plus transitions de départ
- La branche d'exécution est décidée par la condition (garde) associée avec les transitions qui sortent du nœud de décision

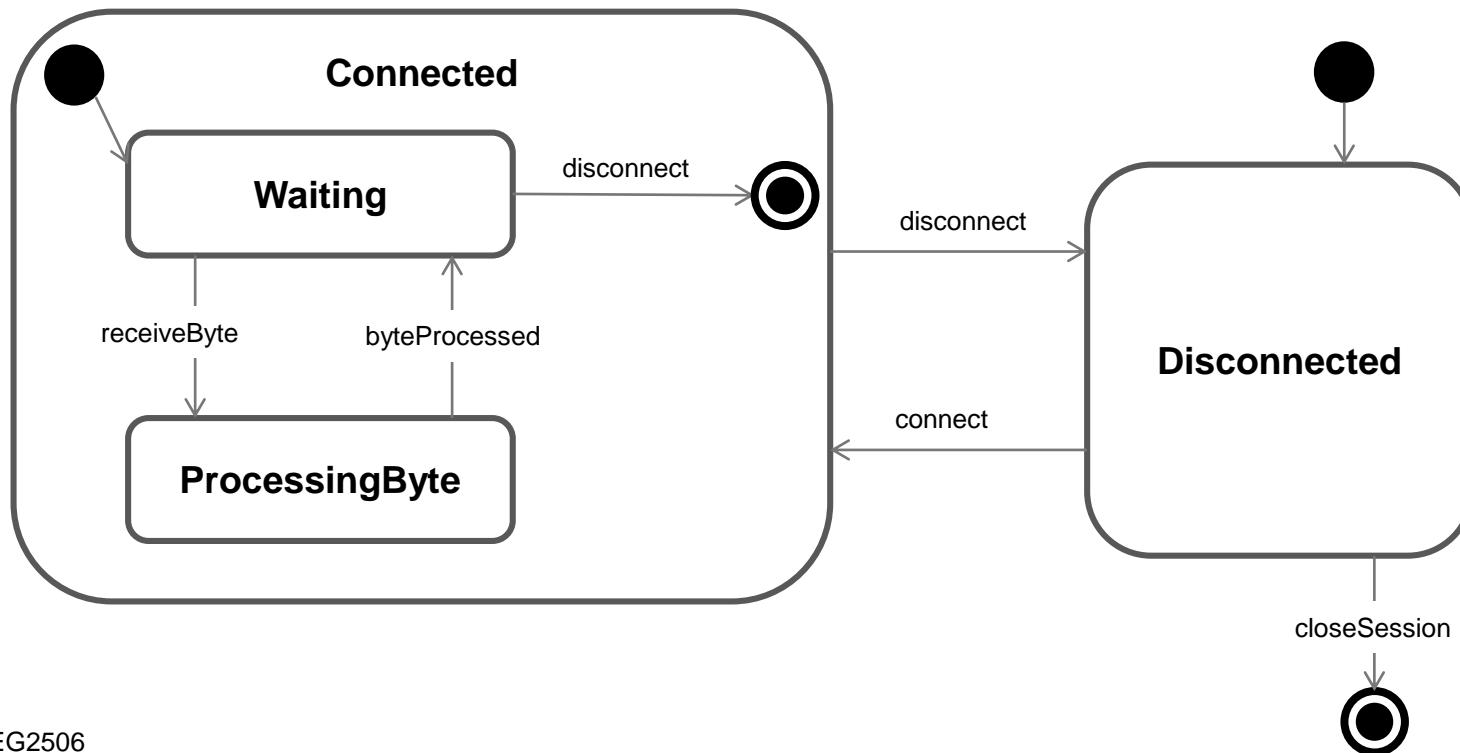
DÉCISIONS



ÉTATS COMPOSÉS

Une machine d'état peut inclure des diagrammes de sous-machine

Exemple d'une application de communication simple:

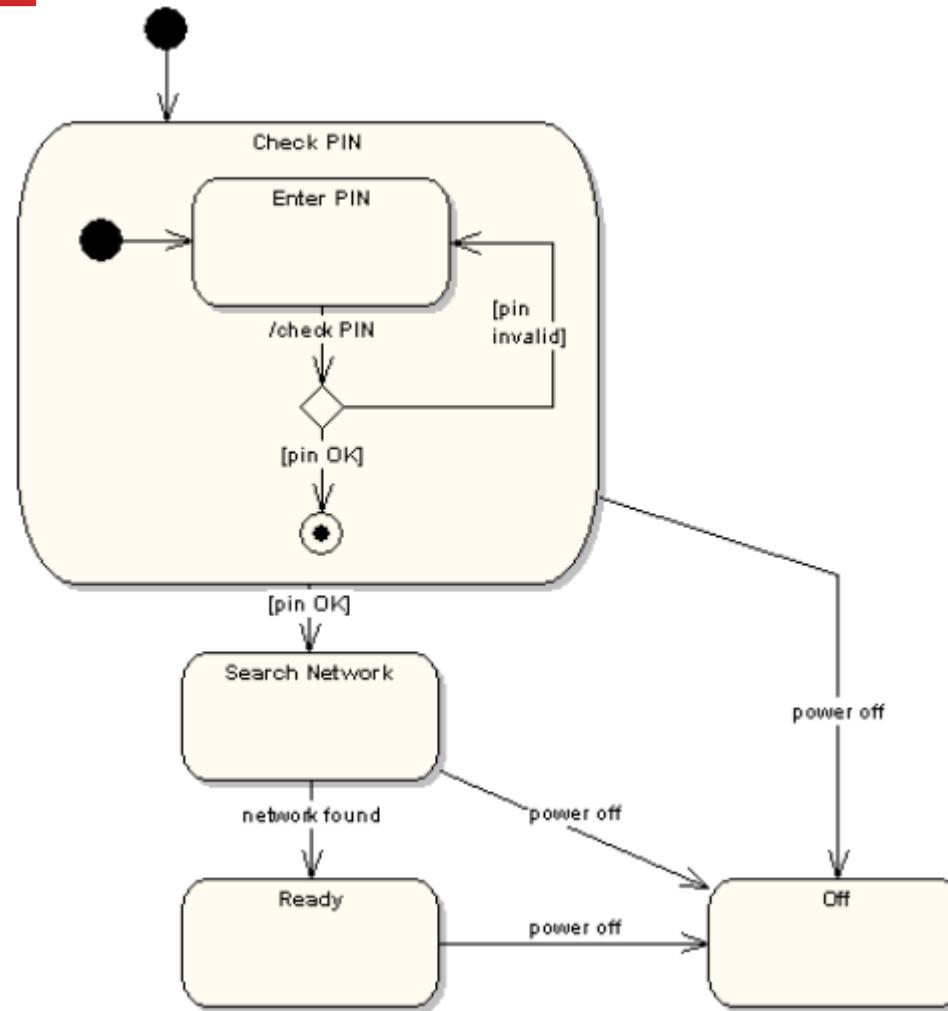


ÉTATS COMPOSÉS - EXEMPLE



uOttawa

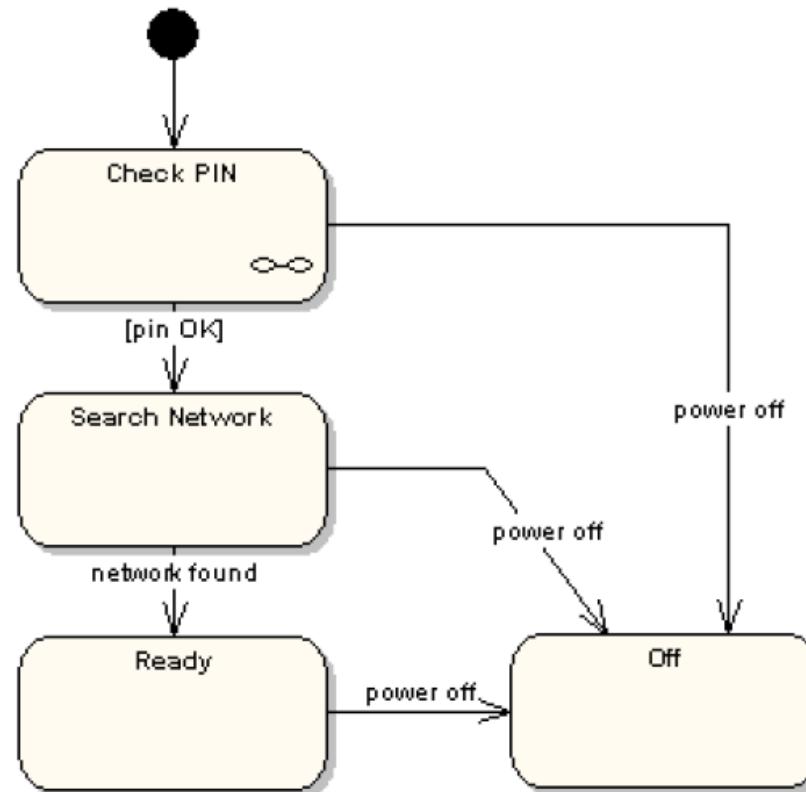
L'Université canadienne
Canada's university



ÉTATS COMPOSÉS - EXEMPLE

Même exemple, mais avec une représentation alternative

- Le symbole dans l'état “Check Pin” indique que les détails de la sous-machine sont spécifiés dans une autre machine d'état





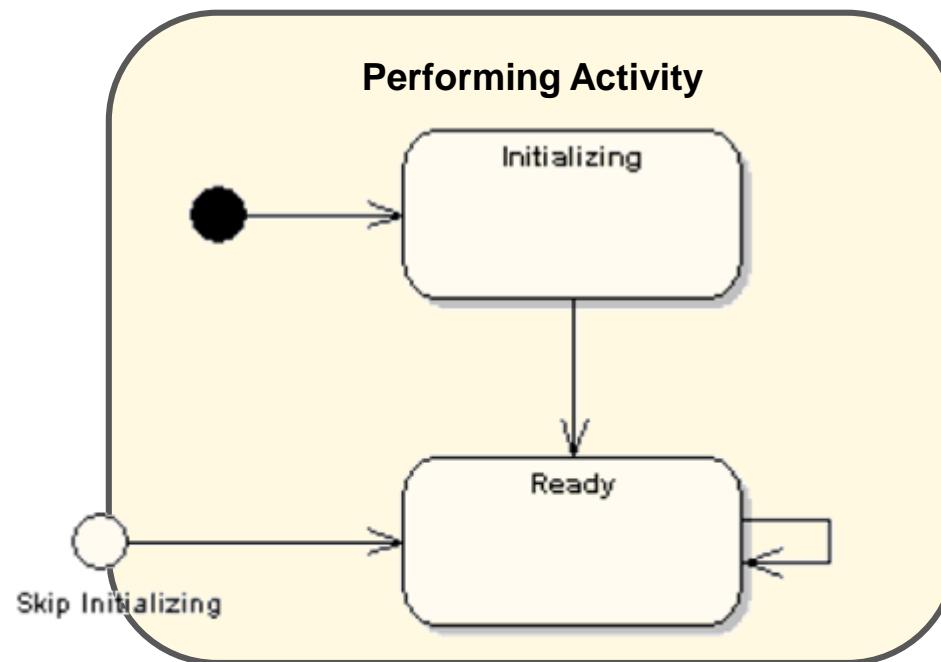
uOttawa

L'Université canadienne
Canada's university

POINTS D'ENTRÉE ALTERNATIVE

Parfois, on n'a pas besoin de commencer l'exécution par l'état initial

- Il arrive des fois qu'on veut commencer l'exécution à partir d'un point alternatif



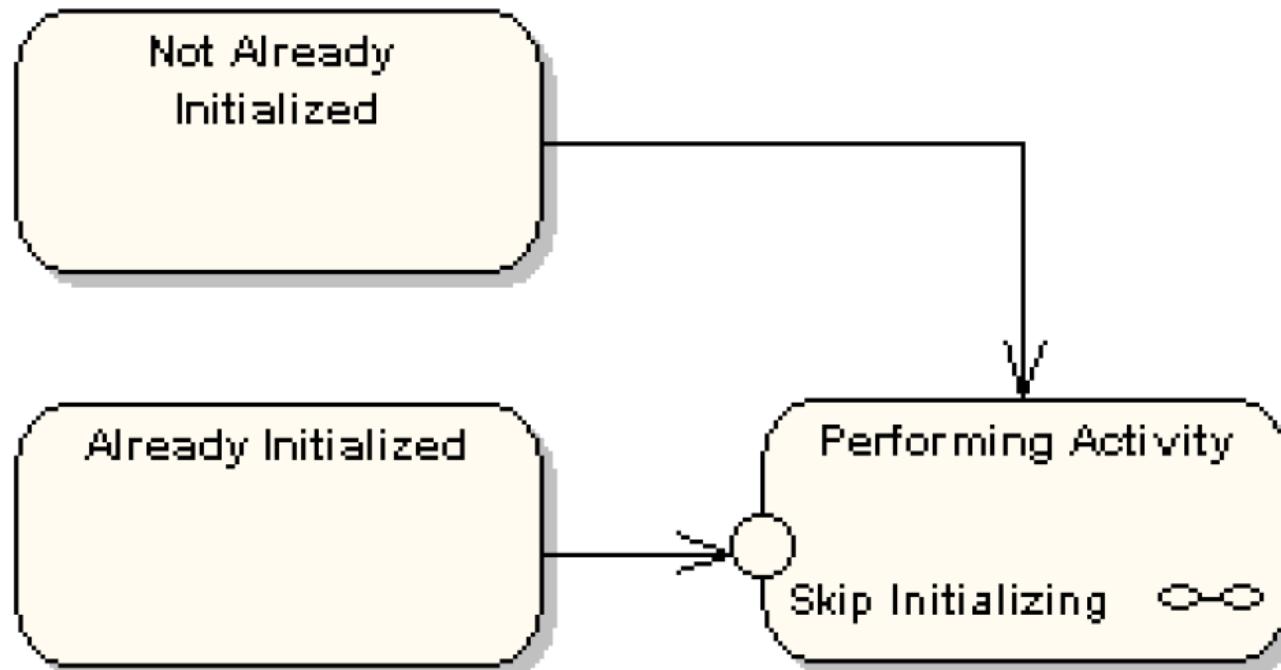
POINTS D'ENTRÉE ALTERNATIVE



uOttawa

L'Université canadienne
Canada's university

Voici le même système qu'on vient de voir, d'un niveau plus haut



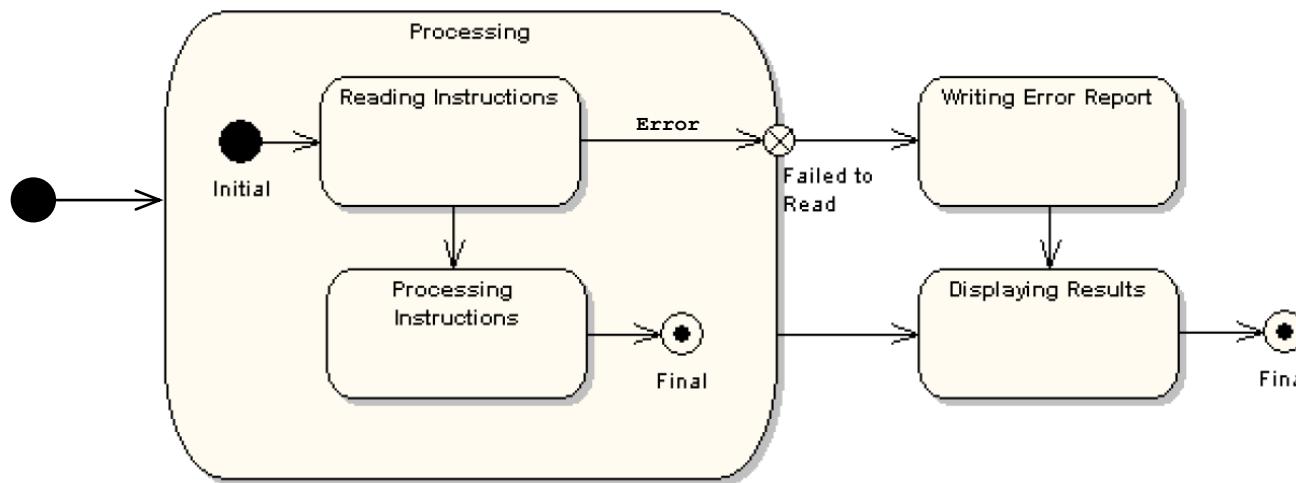
POINTS DE SORTIE ALTERNATIVE



uOttawa

L'Université canadienne
Canada's university

Il est aussi possible d'avoir des points de sortie alternative d'un état composé





uOttawa

L'Université canadienne
Canada's university

ÉTATS D'HISTOIRE

Une machine d'état décrit les aspects dynamiques d'un processus dont le comportement actuel dépend de son passé

Une machine d'état en vigueur spécifie l'ordre légal des états dont un processus peut passer au cours de sa vie

Quand une transition entre dans un état composé, l'action de la sous machine d'état recommence à son état initial

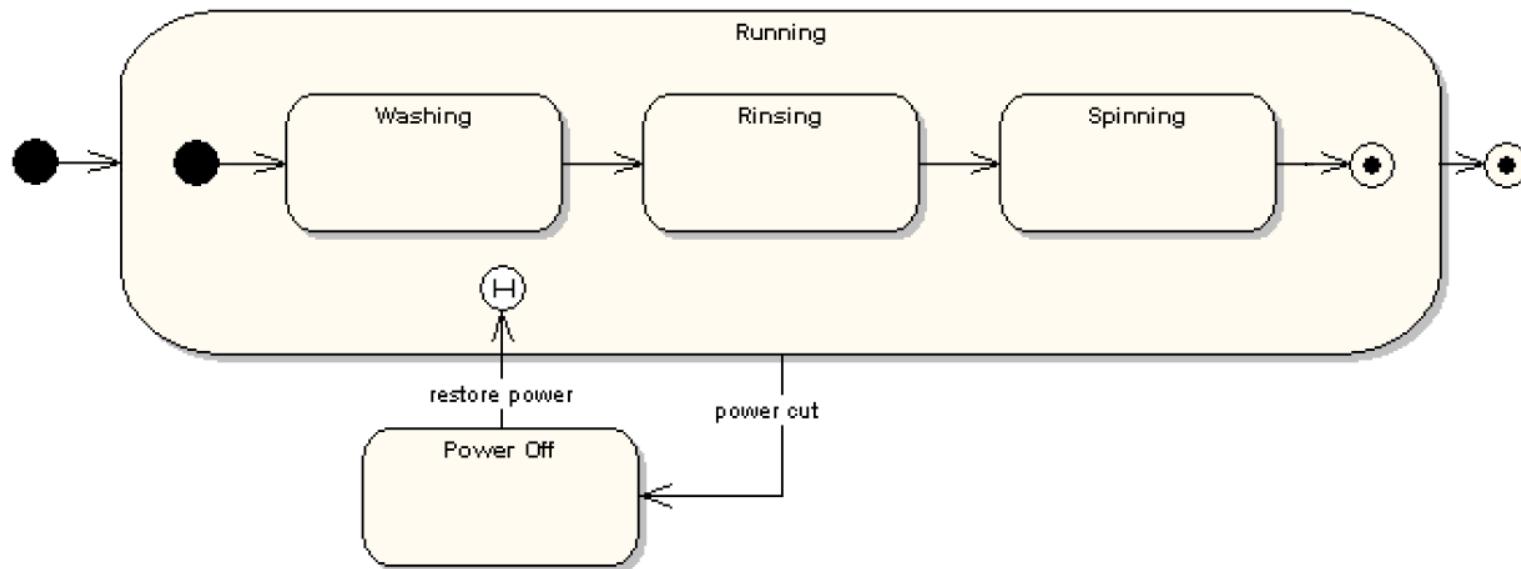
- Sauf si un point d'entrée alternatif est spécifié

Il y a des moments où vous souhaitez modéliser un processus de sorte qu'il se souvient du dernier sous-état qui était actif avant de quitter l'état composé

ÉTATS D'HISTOIRE

Diagramme d'état simple d'une machine à laver:

- Événement coupure du courant électrique: transition à l'état “Power Off”
- Événement courant restauré: transition à l'état actif avant la coupure du courant pour continuer le cycle

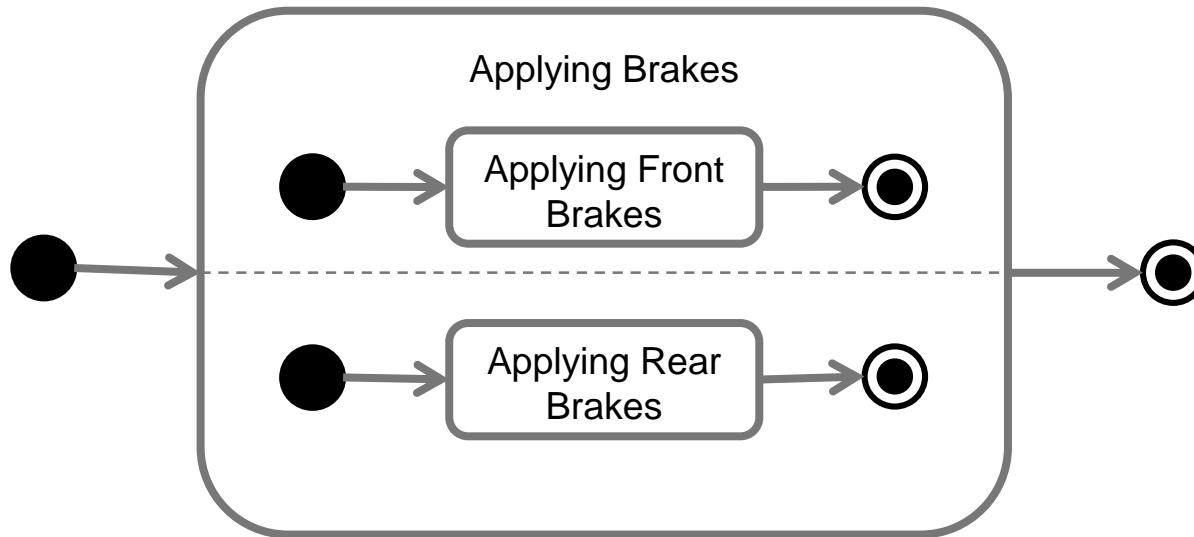


RÉGIONS CONCURRENTIELLES

Les machines de sous-état séquentielles sont le genre de sous-machines les plus connues

- Dans certaines situations de modélisation, on fait appel à des sous machines concurrentielles (deux machines de sous-état ou plus qui travaillent simultanément)

Exemple:



RÉGIONS CONCURRENTIELLES

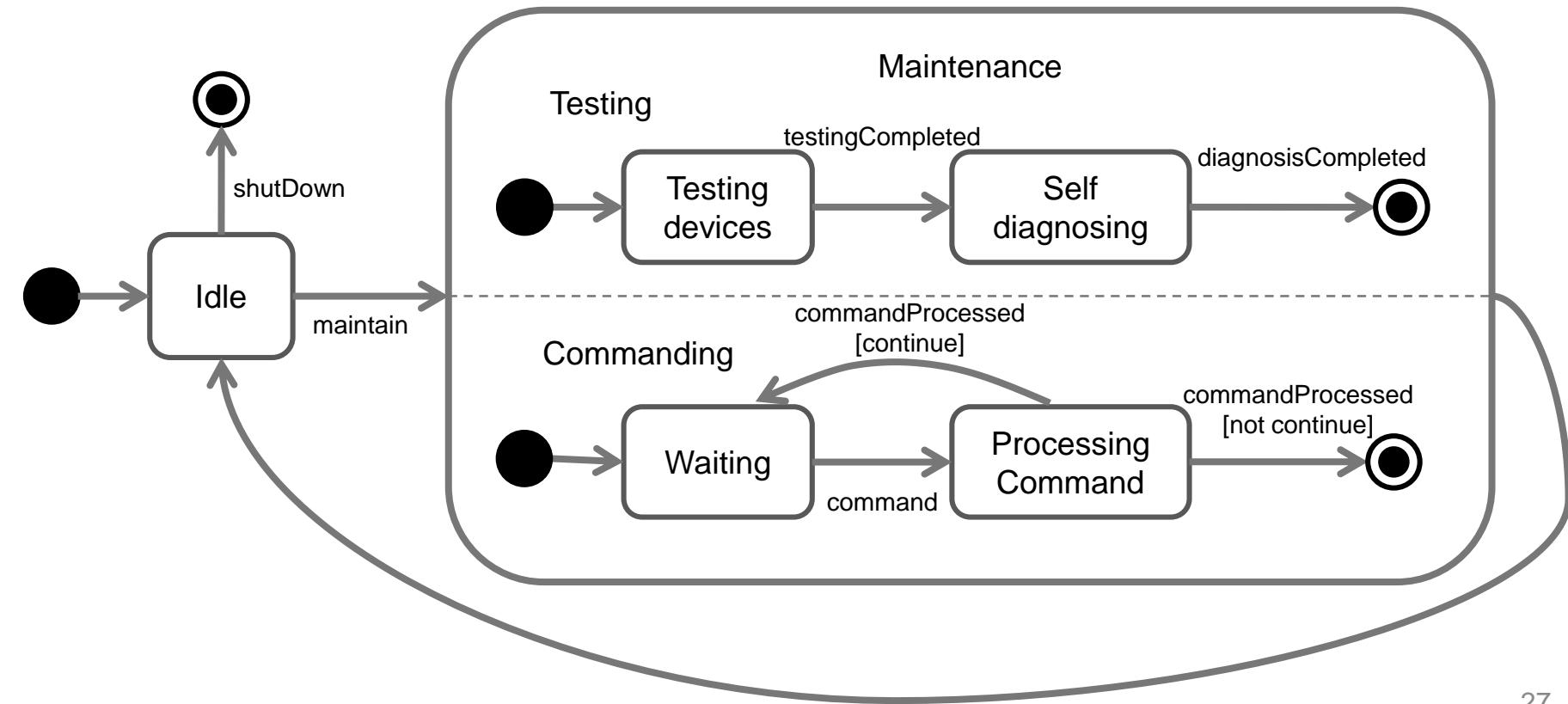
Les régions concurrentielles sont également appelées régions orthogonales

Ces régions nous permettent de modéliser une relation de «et» entre les États (par opposition à la relation par défaut «ou»)

- Cela signifie que dans une machine à sous-états, le système peut être dans plusieurs états simultanément

RÉGIONS CONCURRENTIELLES

Exemple de modélisation d'un système de maintenance à l'aide de régions concurrentielles

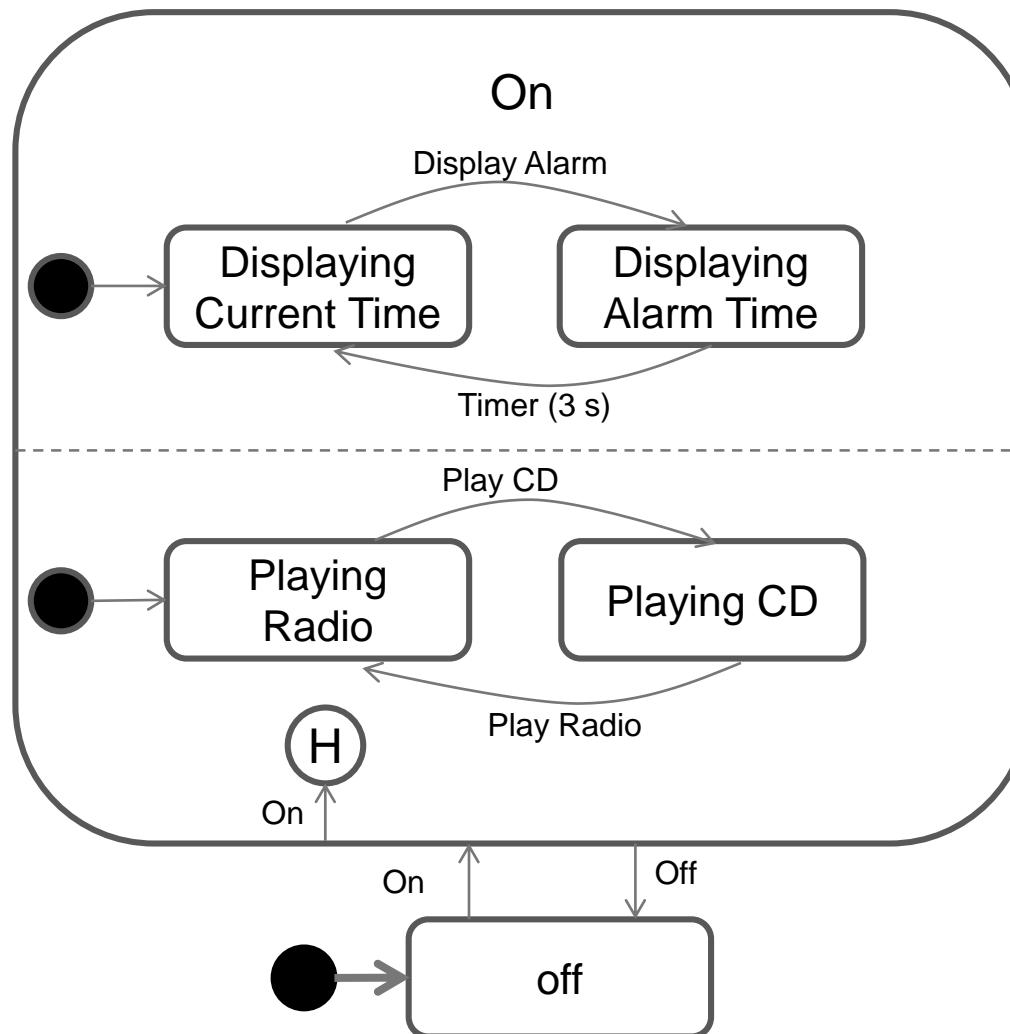




uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UN LECTEUR CD AVEC UNE RADIO



PORTE DU GARAGE – ÉTUDE DE CAS

Contextes

- La compagnie DOORS inc. fabrique des composantes de portes de garage
- Cependant, ils ont des problèmes avec le logiciel intégré (embedded) qui contrôle le moteur (motor unit) de l'ouvre-porte automatique du garage
 - Ils ont développé ce logiciel eux-mêmes
 - Ceci leur cause la perte de clients
- Ils ont décidé de jeter le logiciel existant et embaucher une compagnie de logiciel professionnelle pour livrer un logiciel sans problèmes (bugs)



uOttawa

L'Université canadienne
Canada's university

EXIGENCES DE HAUT-NIVEAU

Exigence 1: Quand la porte du garage est fermée, elle doit s'ouvrir lorsque l'utilisateur pèse le bouton sur le système de contrôle mural ou sur la télécommande

Exigence 2: Quand la porte du garage est ouverte, elle doit se fermer lorsque l'utilisateur pèse le bouton sur le système de contrôle mural ou sur la télécommande

Exigence 3: La porte du garage ne doit pas se fermer sur un obstacle

Exigence 4: Il doit y avoir un moyen pour garder la porte du garage semi-ouverte

Exigence 5: Le système doit accomplir un test auto-diagnostic avant d'effectuer une commande (ouvrir ou fermer) afin de s'assurer que toutes ses composantes sont fonctionnelles

EXIGENCES DU CLIENT

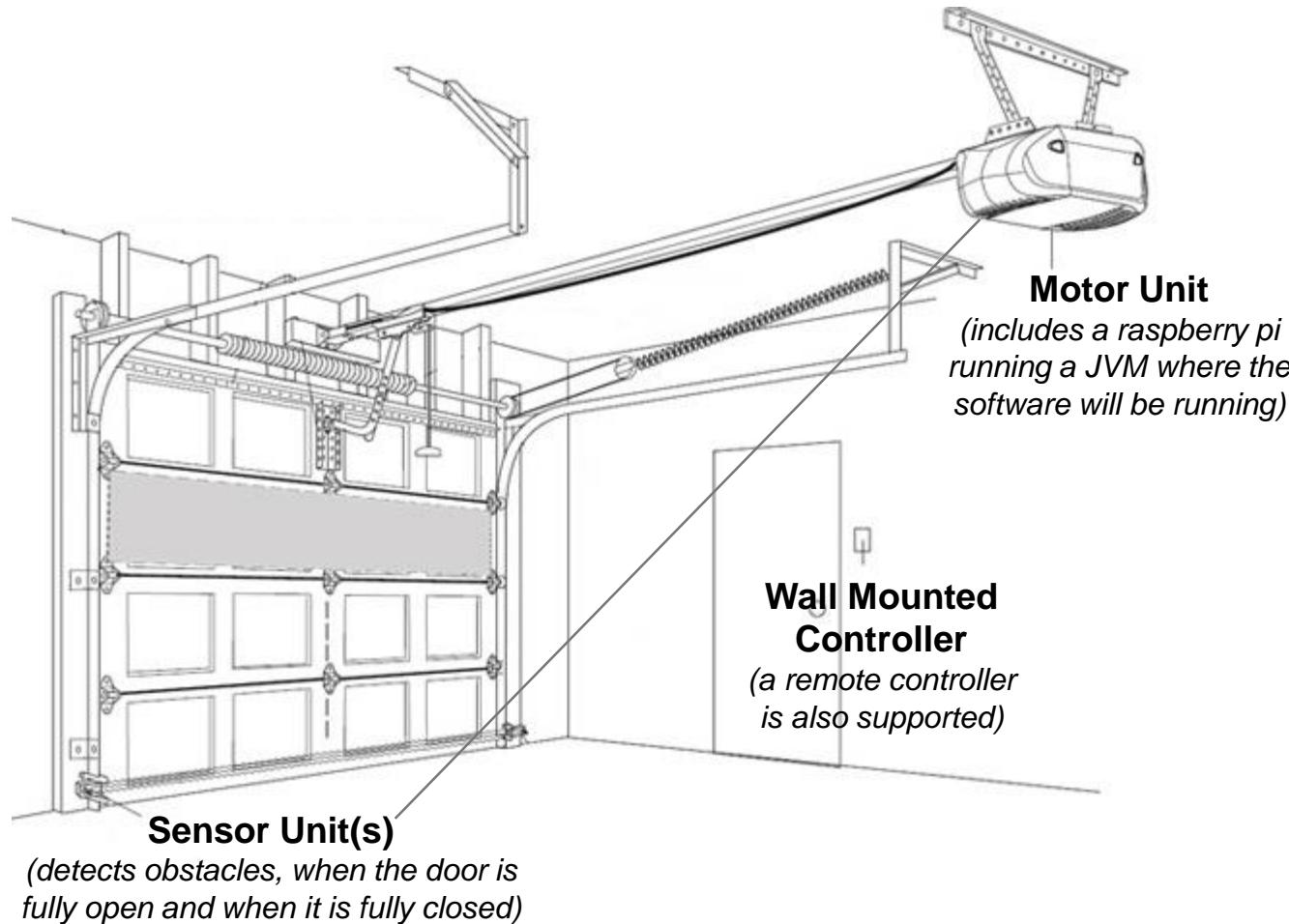
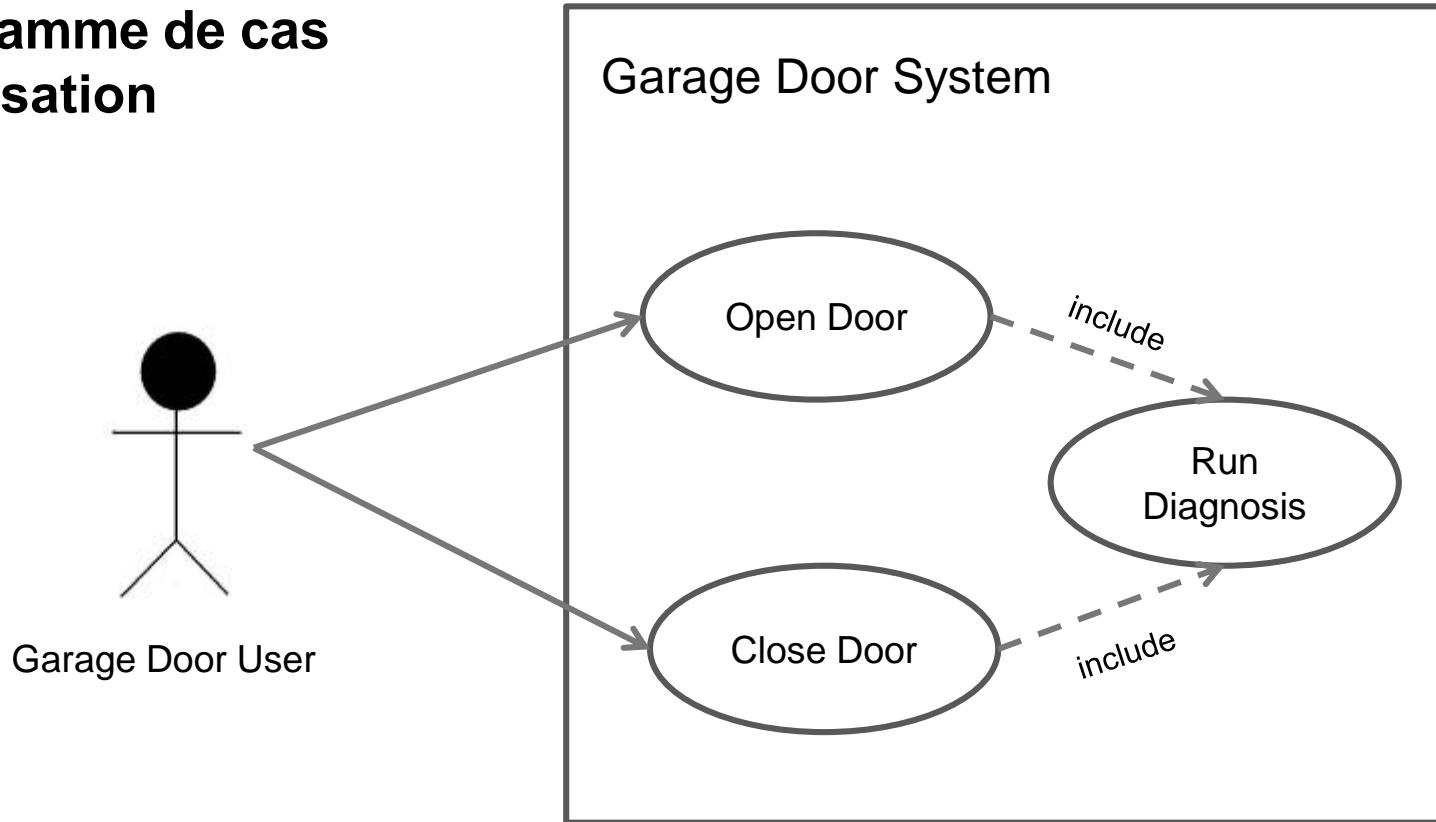


DIAGRAMME DE CAS D'UTILISATION

Diagramme de cas d'utilisation





uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION: EFFECTUER UN DIAGNOSTIC

Nom du cas d'utilisation: Effectuer un test autodiagnostic

Sommaire: Le système effectue un test autodiagnostic

Acteur: L'utilisateur de la porte de garage

Précondition: L'utilisateur a pesé sur la télécommande ou sur le bouton du système de contrôle mural

Séquence:

1. Vérifier si le détecteur fonctionne correctement
2. Vérifier si l'unité moteur fonctionne correctement
3. Si tout les composantes sont vérifiés avec succès, le système autorise l'exécution de la commande

Séquence alternative:

Étape 3: L'un des contrôles échoue et le système attend 3 minutes et déclenche une vérification de nouveau

Poste condition: L'autodiagnostic détermine que le système est opérationnel



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION: OUVRIR LA PORTE

Nom du cas d'utilisation: Ouvrir la porte

Sommaire: Ouvrir la porte du garage

Acteur: L'utilisateur de la porte du garage

Dépendance: Inclure le cas d'utilisation d'effectuer un test autodiagnostic

Précondition: Le système de la porte du garage est opérationnel et prêt à accepter une commande

Séquence:

1. L'utilisateur pèse la télécommande ou le bouton du système de contrôle mural
2. Inclure le cas d'utilisation d'effectuer un test autodiagnostic
3. Si la porte est actuellement en train de se fermer ou elle est déjà fermée, le système ouvre la porte

Séquence alternative:

Étape 3: Si la porte est ouverte, le système ferme la porte

Étape 3: Si la porte est en train de s'ouvrir, le système arrête la porte (la gardant semi-ouverte)

Poste-condition: La porte du garage est ouverte



uOttawa

L'Université canadienne
Canada's university

CAS D'UTILISATION: FERMER LA PORTE

Nom du cas d'utilisation: Fermer la porte

Sommaire: Fermer la porte du garage

Acteur: L'utilisateur de la porte du garage

Dépendance: Inclure le cas d'utilisation d'effectuer un test autodiagnostic

Précondition: Le système de la porte du garage est opérationnel et prêt à accepter une commande

Séquence:

1. L'utilisateur pèse la télécommande ou le bouton du système de contrôle mural
2. Inclure le cas d'utilisation d'effectuer un test autodiagnostic
3. Si la porte est actuellement en train de s'ouvrir ou elle est déjà ouverte, le système ferme la porte

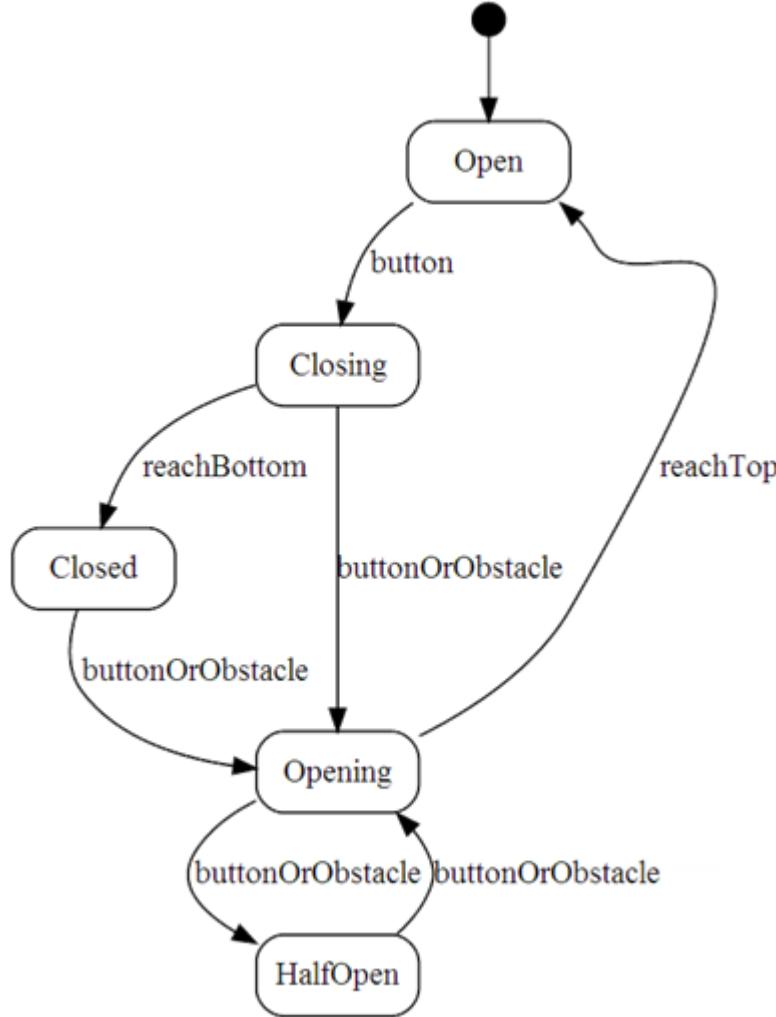
Séquence alternative:

Étape 3: Si la porte est en train de se fermer ou elle est déjà fermée, le système ouvre la porte

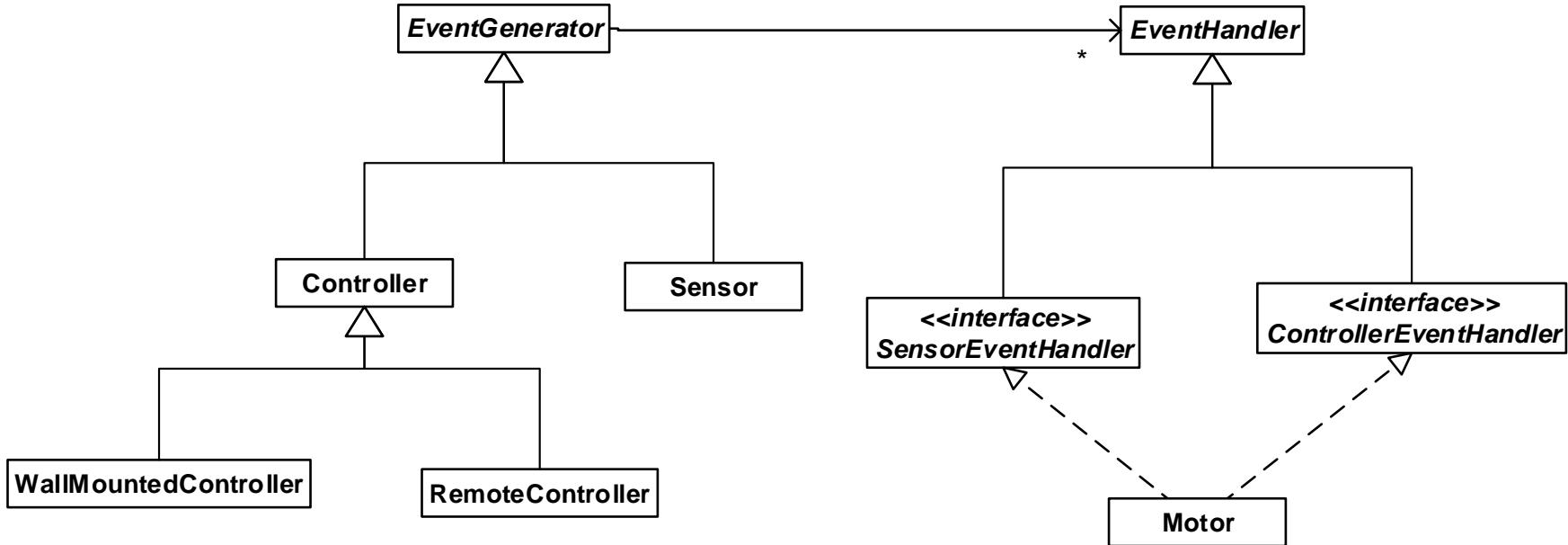
Étape 3: Si la porte est en train de s'ouvrir, le système arrête la porte (la gardant semi-ouverte)

Poste-condition: La porte du garage est fermée

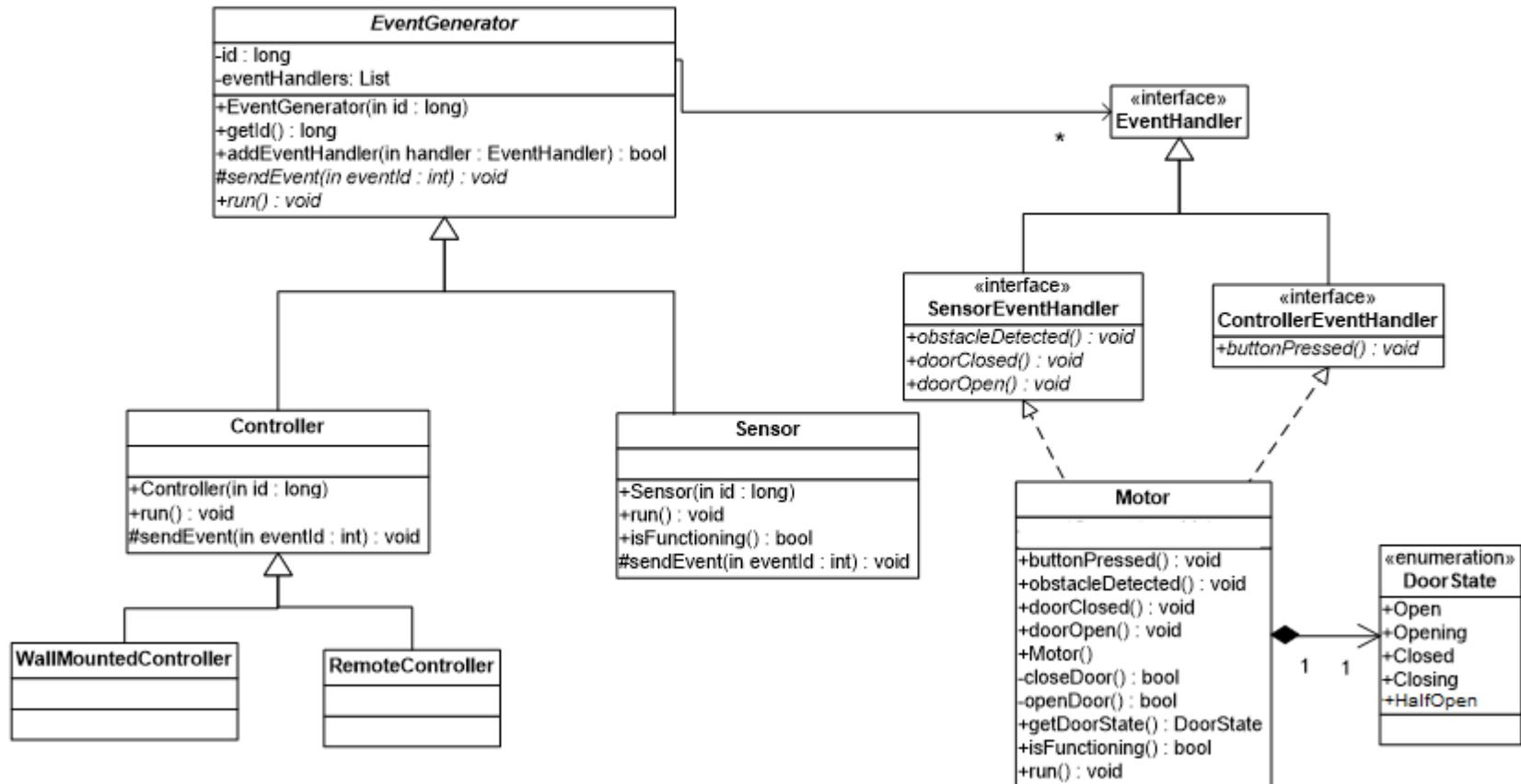
MODÉLISATION COMPORTEMENTALE DE HAUT NIVEAU



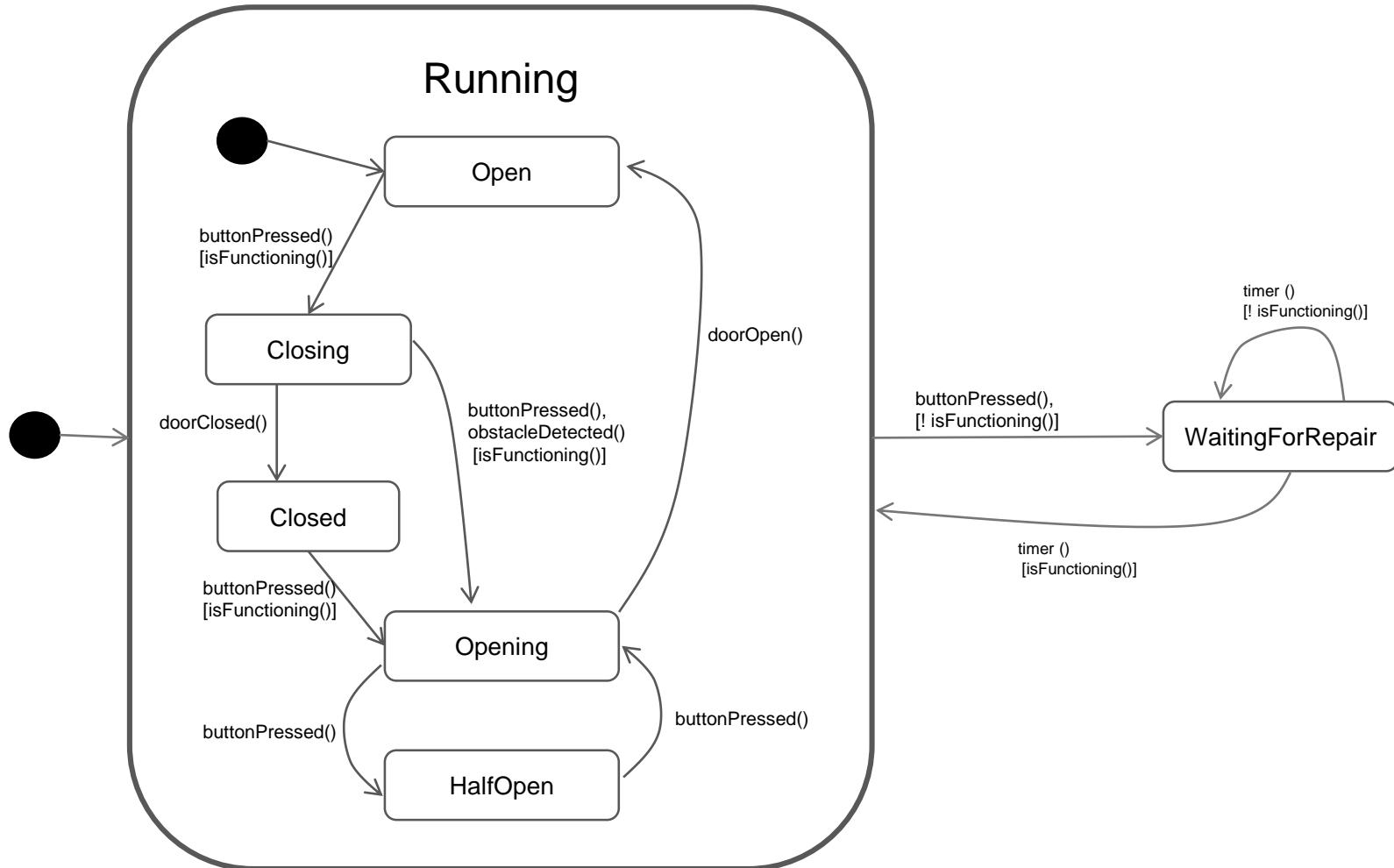
MODÈLE STRUCTURAL DE HAUT NIVEAU



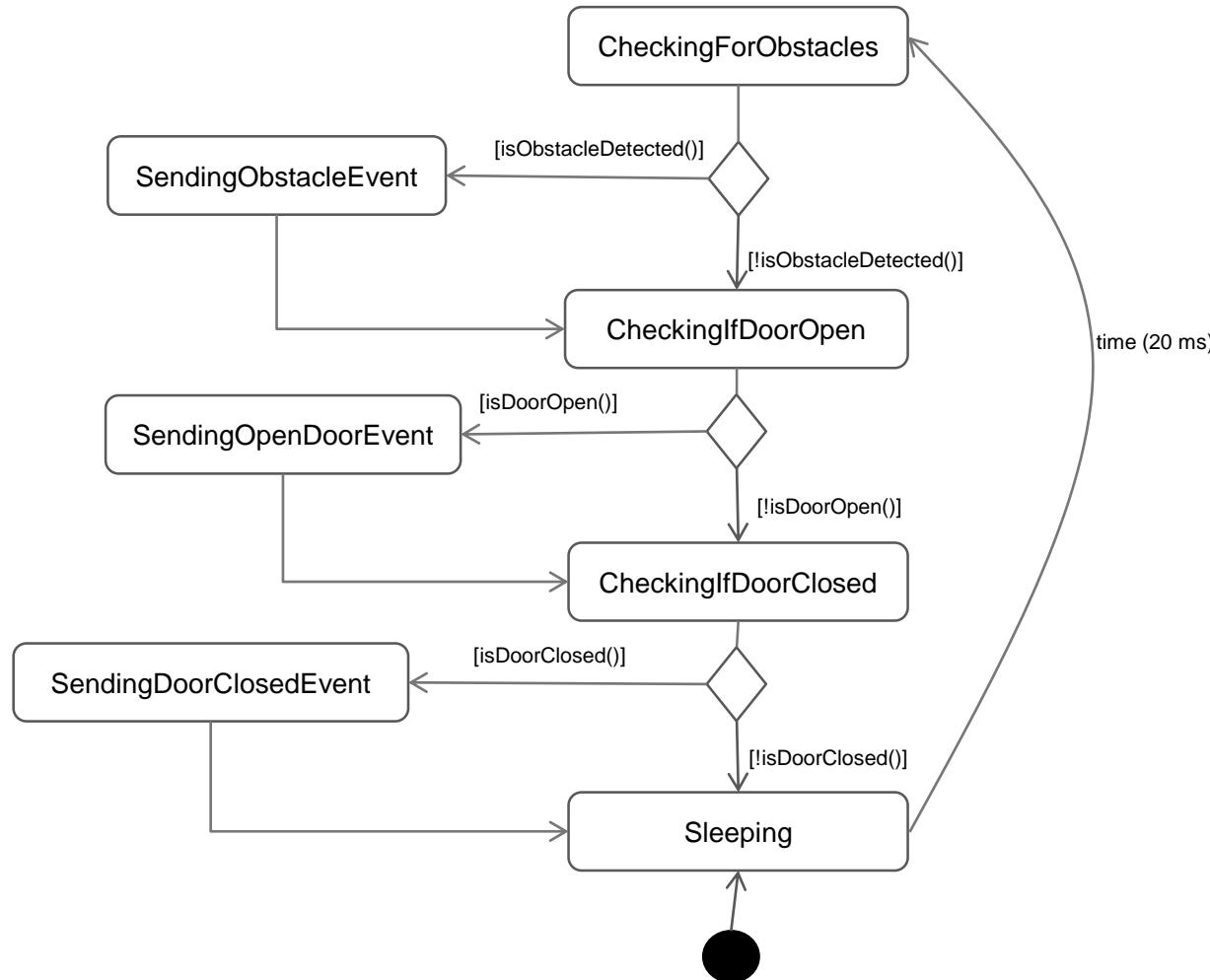
MODÈLE STRUCTURAL RAFFINÉ



MODÈLE COMPORTEMENTAL RAFFINÉ – UNITÉ MOTEUR



MODÈLE COMPORTEMENTAL RAFFINÉ – UNITÉ DÉTECTEUR





uOttawa

L'Université canadienne
Canada's university

PROGRAMMATION

Lorsque nous sommes satisfaits du niveau de détail dans notre modèle comportemental, nous sommes prêts à programmer

Quelques parties du code peuvent être générées directement par des outils du modèle comportemental

- Quelques modifications peuvent être nécessaires (bien que généralement découragés par les fournisseurs d'outils)

CLASSE EVENT GENERATOR



uOttawa

L'Université canadienne
Canada's university

```
public abstract class EventGenerator {

    // Declaring some constants
    protected static final int EVENT_HANDLERS_LIMIT = 100;
    protected static final int BUTTON_EVENT_ID = 1;
    protected static final int DOOR_CLOSED_EVENT_ID = 2;
    protected static final int DOOR_OPEN_EVENT_ID = 3;
    protected static final int OBSTACLE_EVENT_ID = 4;

    protected long id;
    protected List<EventHandler> eventHandlers;

    public EventGenerator (long id){
        this.id = id;

        eventHandlers = new LinkedList<EventHandler>();
    }

    public long getId(){
        return id;
    }

    public boolean addEventHandler(EventHandler handler){
        if (eventHandlers.size() < EVENT_HANDLERS_LIMIT){
            eventHandlers.add(handler);
            return true;
        }
        else {
            return false;
        }
    }

    protected abstract void sendEvent(int eventId);

    public abstract void run();
}
```



uOttawa

L'Université canadienne
Canada's university

CLASSE SENSOR

```
public class Sensor extends EventGenerator implements Runnable{
    private static final int POLLING_INTERVAL = 20; // 20 milliseconds
    private boolean running;
    private Thread pollingThread;

    public Sensor(long id) {
        super (id);
        running = true;

        pollingThread = new Thread(this);

        pollingThread.start();
    }

    @Override
    protected void sendEvent(int eventId) {
        for (EventHandler handler: eventHandlers){

            if (handler instanceof SensorEventHandler){
                SensorEventHandler sensorHandler = (SensorEventHandler)handler;

                if (eventId == OBSTACLE_EVENT_ID){
                    sensorHandler.obstacleDetected();
                }
                else if (eventId == DOOR_CLOSED_EVENT_ID){
                    sensorHandler.doorClosed();
                }
                else if (eventId == DOOR_OPEN_EVENT_ID){
                    sensorHandler.doorOpen();
                }
            }
        }
    }
}
```



uOttawa

L'Université canadienne
Canada's university

CLASSE SENSOR

```
@Override
public void run() {
    // poll sensor hardware continuously in order to check
    // whether there is an obstacle, the door has reached the top (is open)
    // or the door has reached the bottom (is closed)
    while (running) {
        try {
            Thread.sleep(POLLING_INTERVAL);
            // Check for an obstacle
            boolean obstacleDetected = HardwareSensorInterface.isObstacleDetected();
            if (obstacleDetected) {
                sendEvent(OBSTACLE_EVENT_ID);
            }
            // Check if door is open
            //(this call returns true only the first time it is called after the door is open)
            boolean doorOpen = HardwareSensorInterface.isDoorOpen();
            if (doorOpen){
                sendEvent(DOOR_OPEN_EVENT_ID);
            }
            // Check if door is closed
            //(this call returns true only the first time it is called after the door is closed)
            boolean doorClosed = HardwareSensorInterface.isDoorClosed();

            if (doorClosed){
                sendEvent(DOOR_CLOSED_EVENT_ID);
            }

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Sensor
State machine
Implementation



uOttawa

L'Université canadienne
Canada's university

DÉMO UMPLÉ

UMPLE est un outil de modélisation pour permettre ce qu'on appelle la programmation modèle-orientée

- C'est ce qu'on fait dans ce cours

Vous pouvez l'utiliser pour créer des diagrammes de classe (modèles structuraux) et des machines d'état (modèles comportementaux)

Cet outil a été développé à l'Université d'Ottawa

- Une version en ligne existe sur ce lien:

<http://cruise.eecs.uottawa.ca/umpleonline/>

- Il existe aussi un eclipse plugin pour cet outil

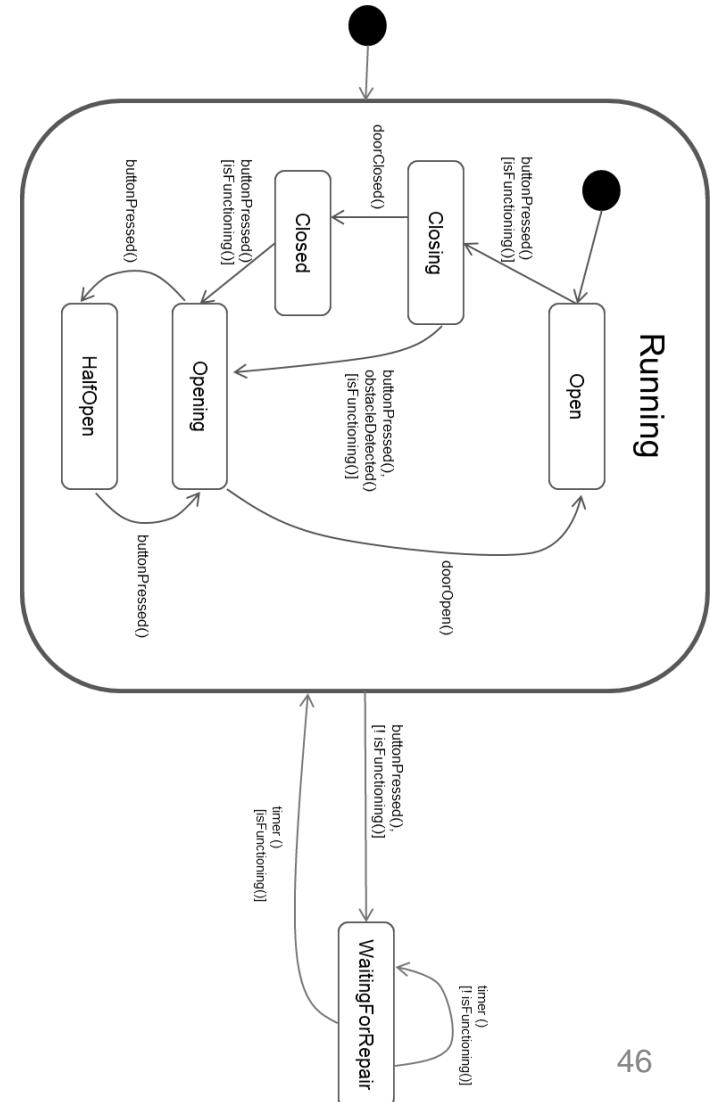
CODE UMPLE POUR MACHINE D'ÉTAT À UNITÉ MOTEUR

```

class Motor {
    status {
        Running {
            Open {buttonPressed()[isFunctioning()]->Closing; }
            Closing { buttonPressed()[isFunctioning()]->Opening;
                ObstacleDetected()[isFunctioning()]->Opening;
                doorClosed()->Closed; }
            Closed { buttonPressed()[isFunctioning()]->Opening; }
            Opening { buttonPressed()->HalfOpen;
                doorOpen()->Open; }
            HalfOpen{buttonPressed()->Opening; }

            buttonPressed()[!isFunctioning()]->WaitingForRepair;
        }
        WaitingForRepair{
            timer()[!isFunctioning()]->WaitingForRepair;
            timer()[isFunctioning()]->Running;
        }
    }
}

```



FRAGMENTS DE CODE DE LA CLASSE MOTEUR



uOttawa

L'Université canadienne
Canada's university

```
public boolean buttonPressed(){
    boolean wasEventProcessed = false;
    Status aStatus = status;
    StatusRunning aStatusRunning = statusRunning;
    switch (aStatus){
        case Running:
            if (!isFunctioning()){
                exitStatus();
                setStatus(Status.WaitingForRepair);
                wasEventProcessed = true;
                break;
            }
            break;
        default: //Other states do respond to this event
    }
    switch (aStatusRunning){
        case Open:
            if (isFunctioning())
            {
                setStatusRunning(StatusRunning.Closing);
                wasEventProcessed = true;
                break;
            }
            break;
        case Closing:
            if (isFunctioning()){
                setStatusRunning(StatusRunning.Opening);
                wasEventProcessed = true;
                break;
            }
            break;
        case Closed:
            if (isFunctioning()){
                setStatusRunning(StatusRunning.Opening);
                wasEventProcessed = true;
                break;
            }
            break;
        case Opening:
            setStatusRunning(StatusRunning.HalfOpen);
            wasEventProcessed = true;
            break;
        case HalfOpen:
            setStatusRunning(StatusRunning.Opening);
            wasEventProcessed = true;
            break;
        default:// Other states do respond to this event
    }
    return wasEventProcessed;
}
```

Switching between high level states

Switching between nest states inside the Running compound state

MERCI!

QUESTIONS?

SÉANCE 5

RÉSEAUX DE PETRI

*CES DIAPOS SONT BASÉES SUR LES NOTES DE:
DR. CHRIS LING ([HTTP://WWW.CSSE.MONASH.EDU/~SLING/](http://WWW.CSSE.MONASH.EDU/~SLING/))*



uOttawa

L'Université canadienne
Canada's university

SUJETS

Réseau de Petri

Exemples

- Terminal PDV
- Distributeur Automatique
- Restaurant
- Producteur-Consommateur

Structures de Réseaux Petri

Propriétés de Réseaux Petri

- Délimitation
- Vivacité
- Accessibilité

OK, NOUS COMMENÇONS...



uOttawa

L'Université canadienne
Canada's university

© MARK ANDERSON, ALL RIGHTS RESERVED WWW.ANDERTOONS.COM



"OK, I'm now going to read out loud every single slide
to you, word for word, until you all wish you'd just die."



uOttawa

L'Université canadienne
Canada's university

INTRODUCTION

Introduit initialement par Carl Adam Petri en 1962.

Un outil de diagramme pour modéliser la concurrence et la synchronisation dans les systèmes distribués

- Ils nous permettent de simuler rapidement un comportement complexe concurrentiel (plus rapide que de faire des prototypes!)

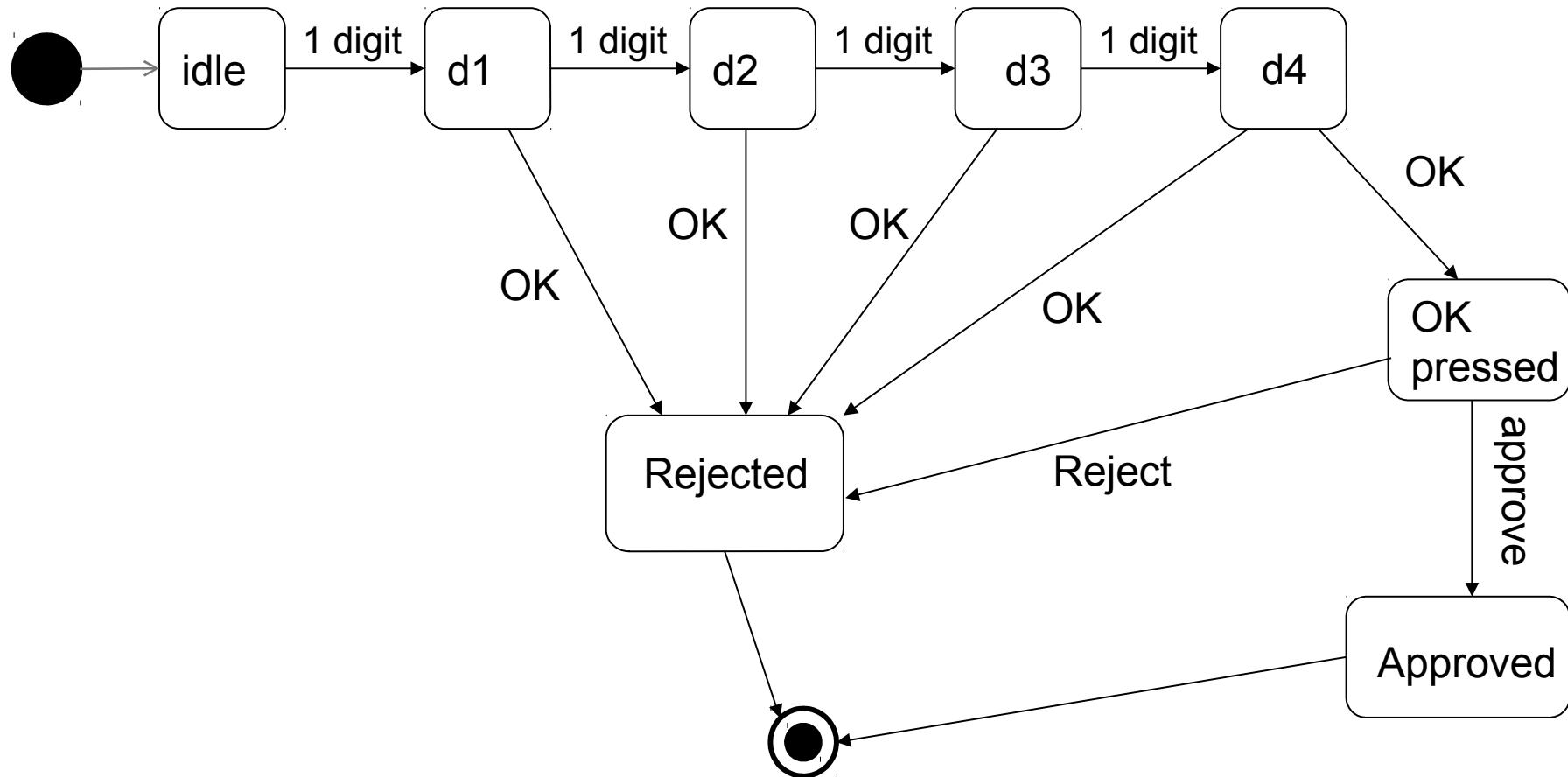
Similaire aux machines d'état UML déjà vues:

- Utilisé comme aide visuel de communication pour modéliser le comportement du système

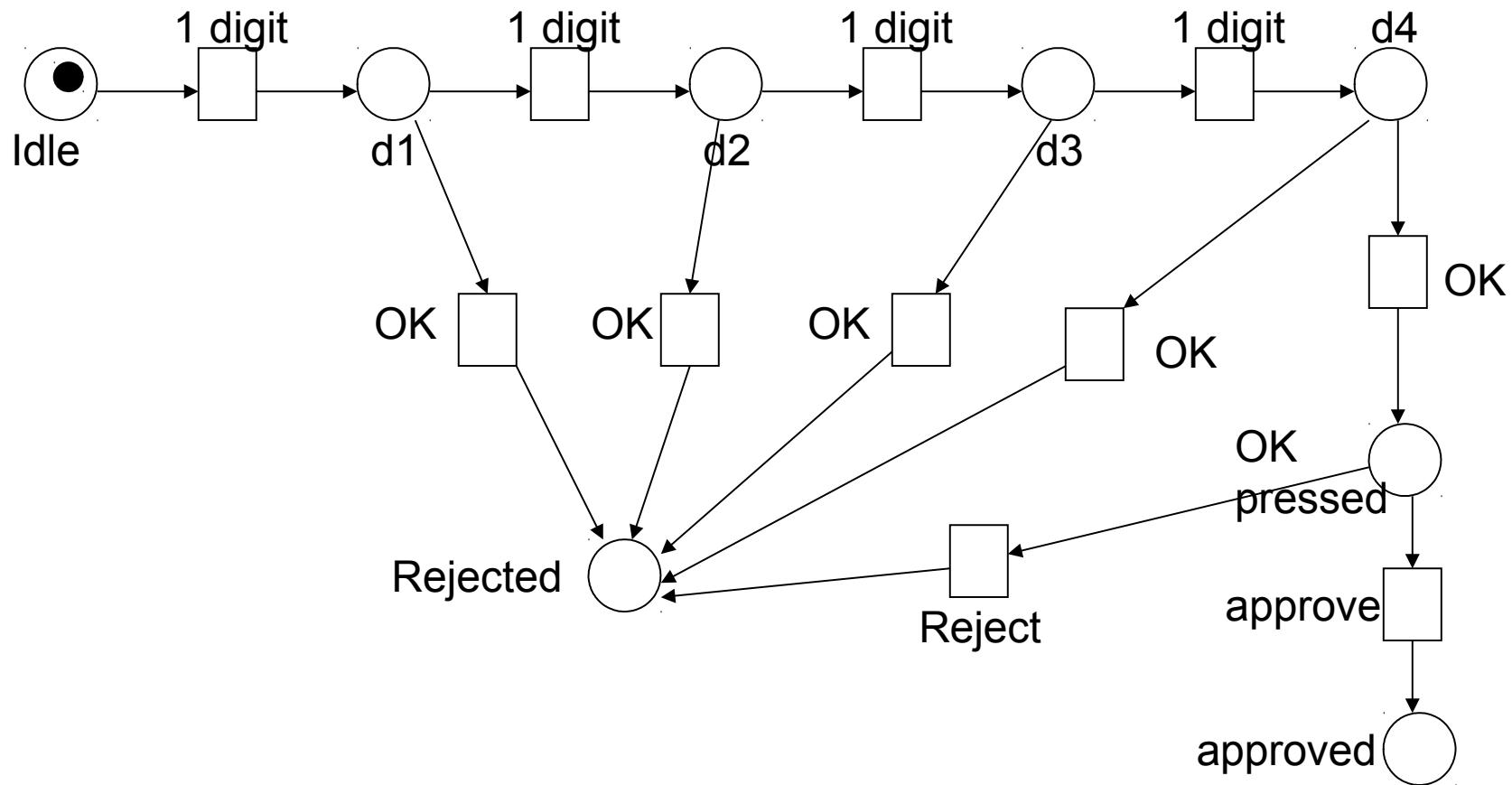
Basé sur une forte fondation en mathématiques

EXAMPLE: TERMINAL PDV (MACHINE D'ÉTAT UML)

(PDV = Point de Vente; POS = Point of Sale)



EXEMPLE: TERMINAL PDV (RÉSEAU DE PETRI)





uOttawa

L'Université canadienne
Canada's university

TERMINAL PDV

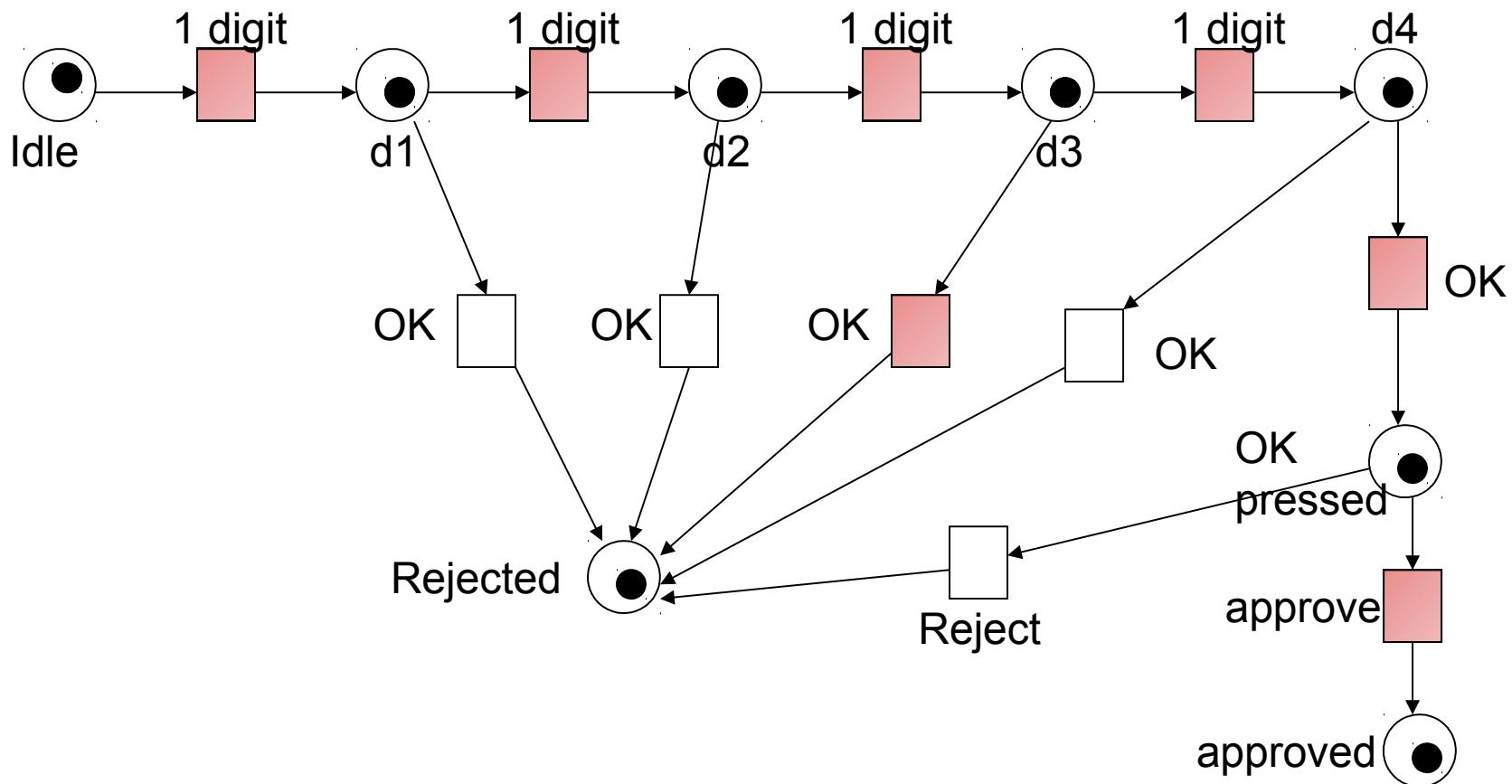
Scénario 1: Normal

- Entrer les 4 chiffres et peser OK.

Scénario 2: Exceptionnel

- Entrer seulement 3 chiffres et peser OK.

EXEMPLE: SYSTÈME PDV (JEU DE JETONS)





uOttawa

L'Université canadienne
Canada's university

LES COMPOSANTES D'UN RÉSEAU DE PETRI

Les termes sont un peu différents que les machines d'état UML

Les réseaux de Petri sont formés de trois types de composantes: *places* (cercles), *transitions* (rectangles) et *arcs* (flèches):

- Les Places représentent les états possibles du système
- Les Transitions sont des événements ou actions qui causent le changement d'état (attention, les transitions ne sont plus des flèches ici)
- Chaque Arc connecte simplement une place à une transition ou une transition à une place.



uOttawa

L'Université canadienne
Canada's university

CHANGEMENT D'ÉTAT

Un changement d'état est dénoté par un mouvement de jeton(s) (points noirs) d'une place(s) à une autre place(s)

- Est causé par l'excitation d'une transition.

L'excitation représente l'occurrence d'un événement ou une prise d'action

L'excitation dépend des conditions d'entrée, dénotée par la disponibilité d'un jeton



uOttawa

L'Université canadienne
Canada's university

CHANGEMENT D'ÉTAT

Une transition est excitable ou permis lorsqu'il existe assez de jetons dans les places d'entrée.

Après l'excitation, les jetons sont transférés des places d'entrée (état précédent) aux places de sortie, dénotant le nouveau état



uOttawa

L'Université canadienne
Canada's university

EXAMPLE: DISTRIBUTEUR AUTOMATIQUE

La machine distribue deux genres de collation – 20c et 15c

Seulement deux types de monnaie peuvent être utilisés

- Pièces de 10c et pièces de 5c (ah les vieux temps!!!)

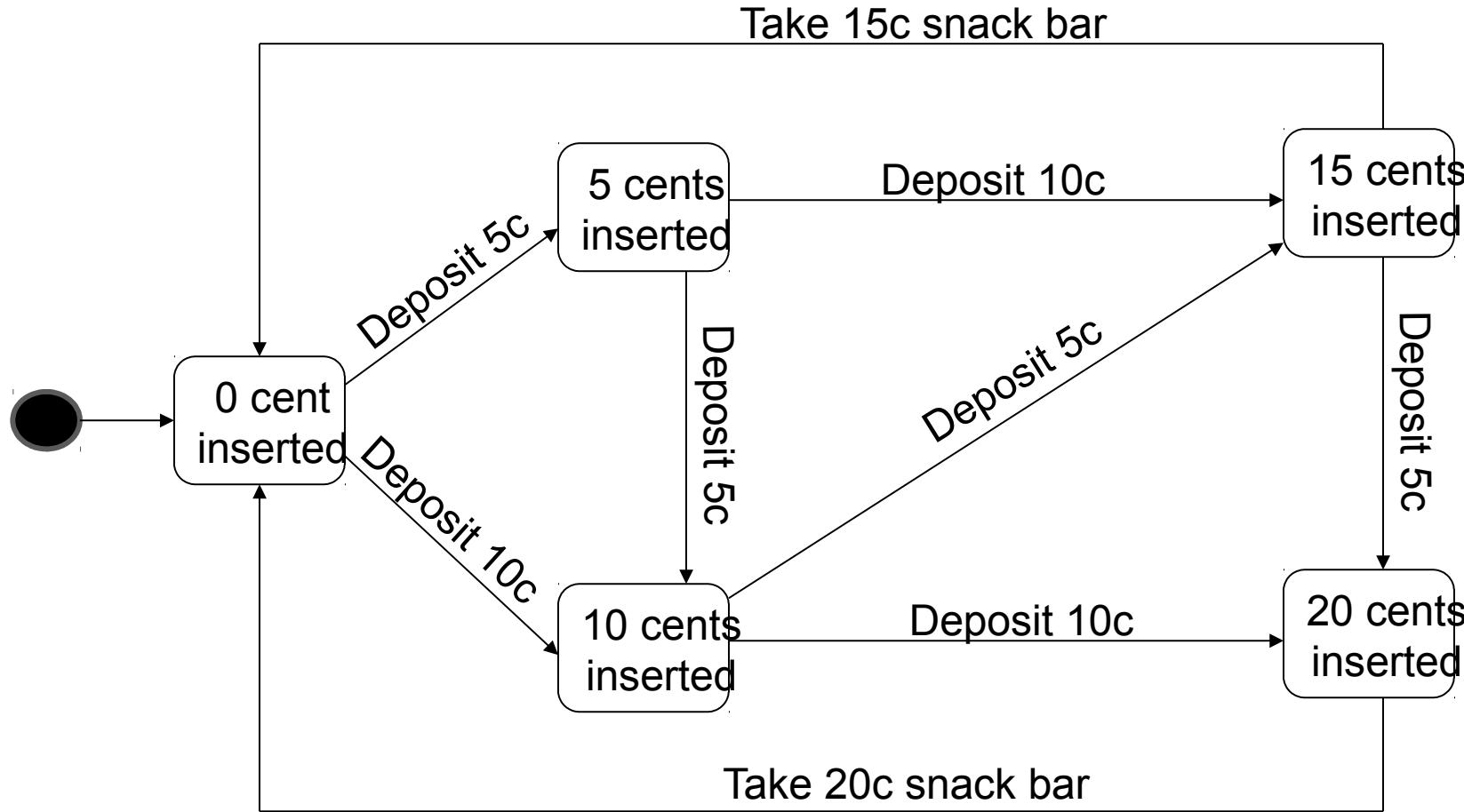
La machine ne retourne pas de monnaie



EXEMPLE: DISTRIBUTEUR AUTOMATIQUE (MACHINE D'ÉTAT UML)



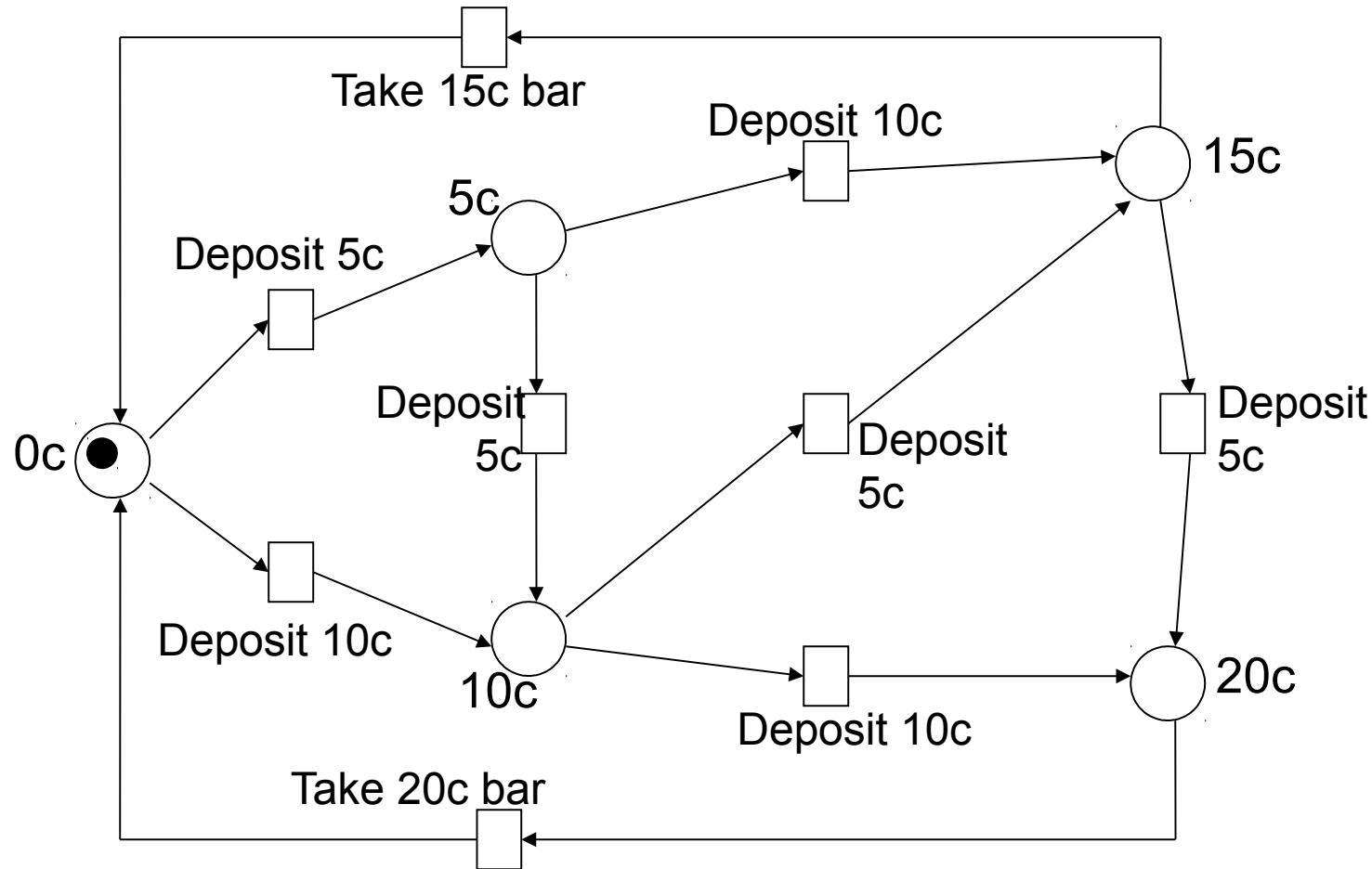
uOttawa
L'Université canadienne
Canada's university



EXEMPLE: DISTRIBUTEUR AUTOMATIQUE (UN RÉSEAU DE PETRI)



uOttawa
L'Université canadienne
Canada's university





uOttawa

L'Université canadienne
Canada's university

EXEMPLE: DISTRIBUTEUR AUTOMATIQUE (3 SCÉNARIOS)

Scénario 1:

- Déposer 5c, déposer 5c, déposer 5c, déposer 5c, prendre la collation de 20c.

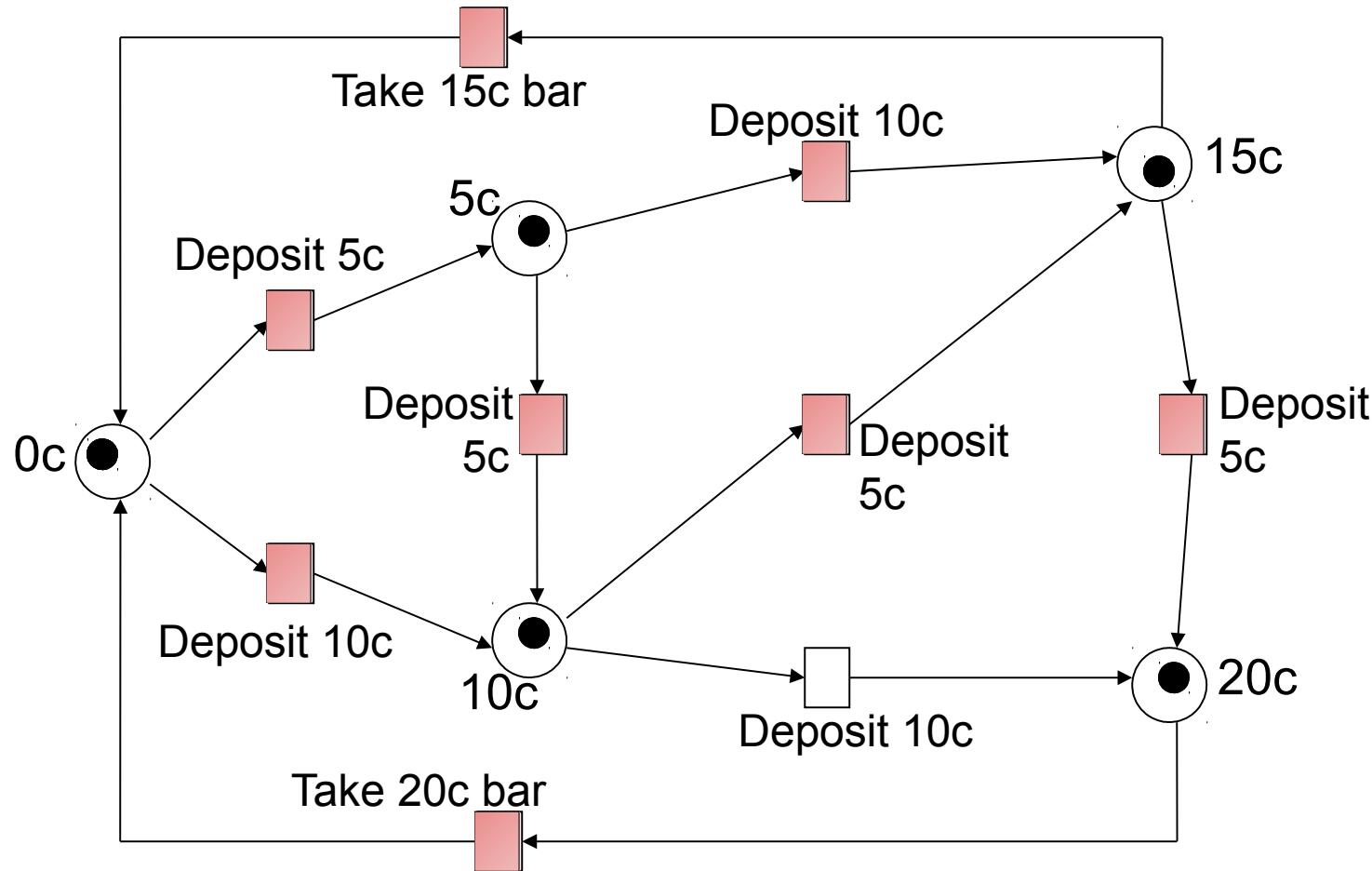
Scénario 2:

- Déposer 10c, déposer 5c, prendre la collation de 15c.

Scénario 3:

- Déposer 5c, déposer 10c, déposer 5c, prendre la collation de 20c.

EXEMPLE: DISTRIBUTEUR AUTOMATIQUE (JEUX DE JETONS)



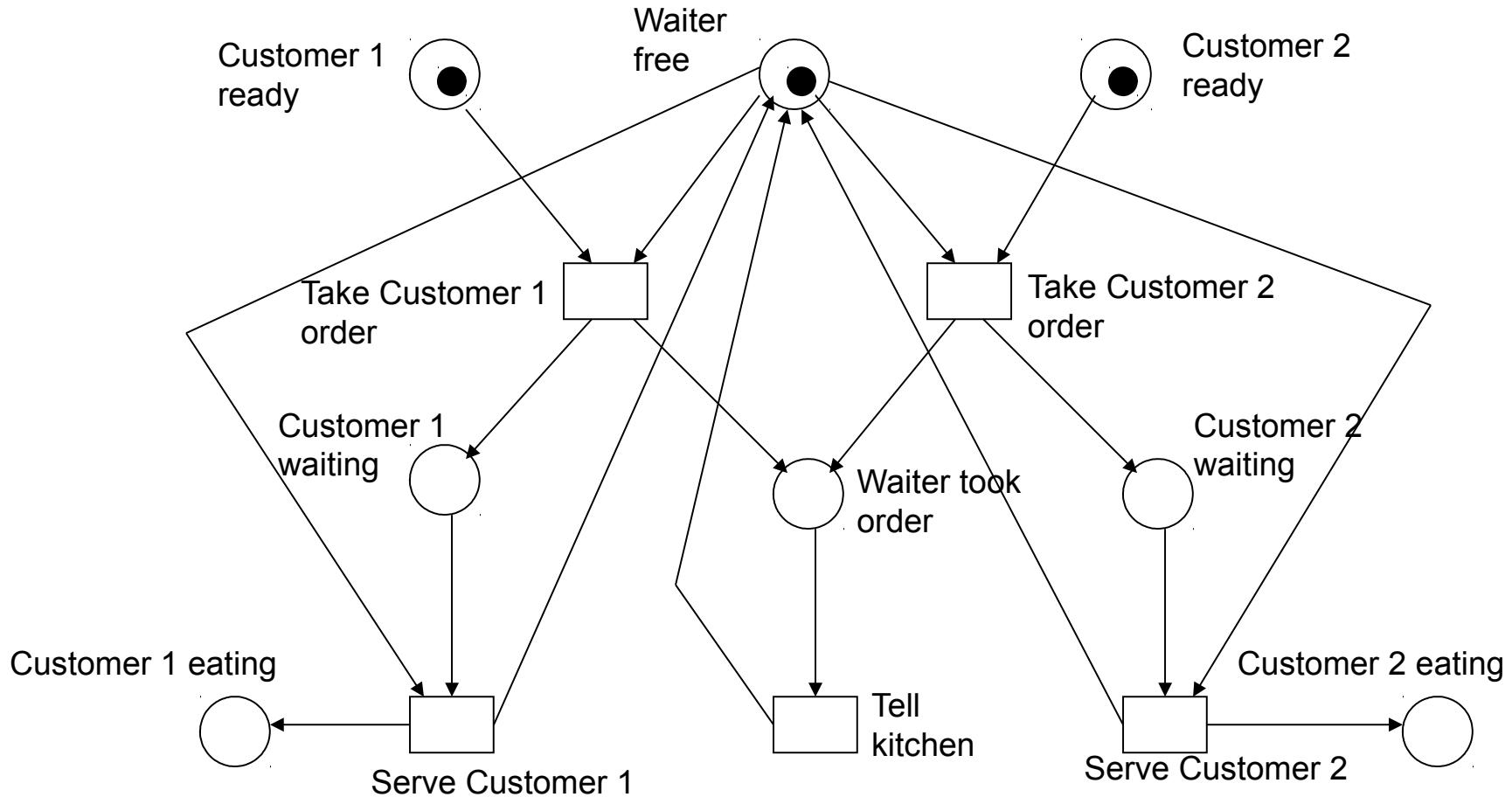
ÉTATS LOCAUX MULTIPLES

Dans le vrai monde, les événements se passent en même temps

Un système peut avoir plusieurs états locaux pour former un état global

Il est nécessaire de modéliser la concurrence et la synchronisation

EXEMPLE: DANS UN RESTAURANT (UN RÉSEAU DE PETRI)



EXEMPLE: DANS UN RESTAURANT (DEUX SCÉNARIOS)

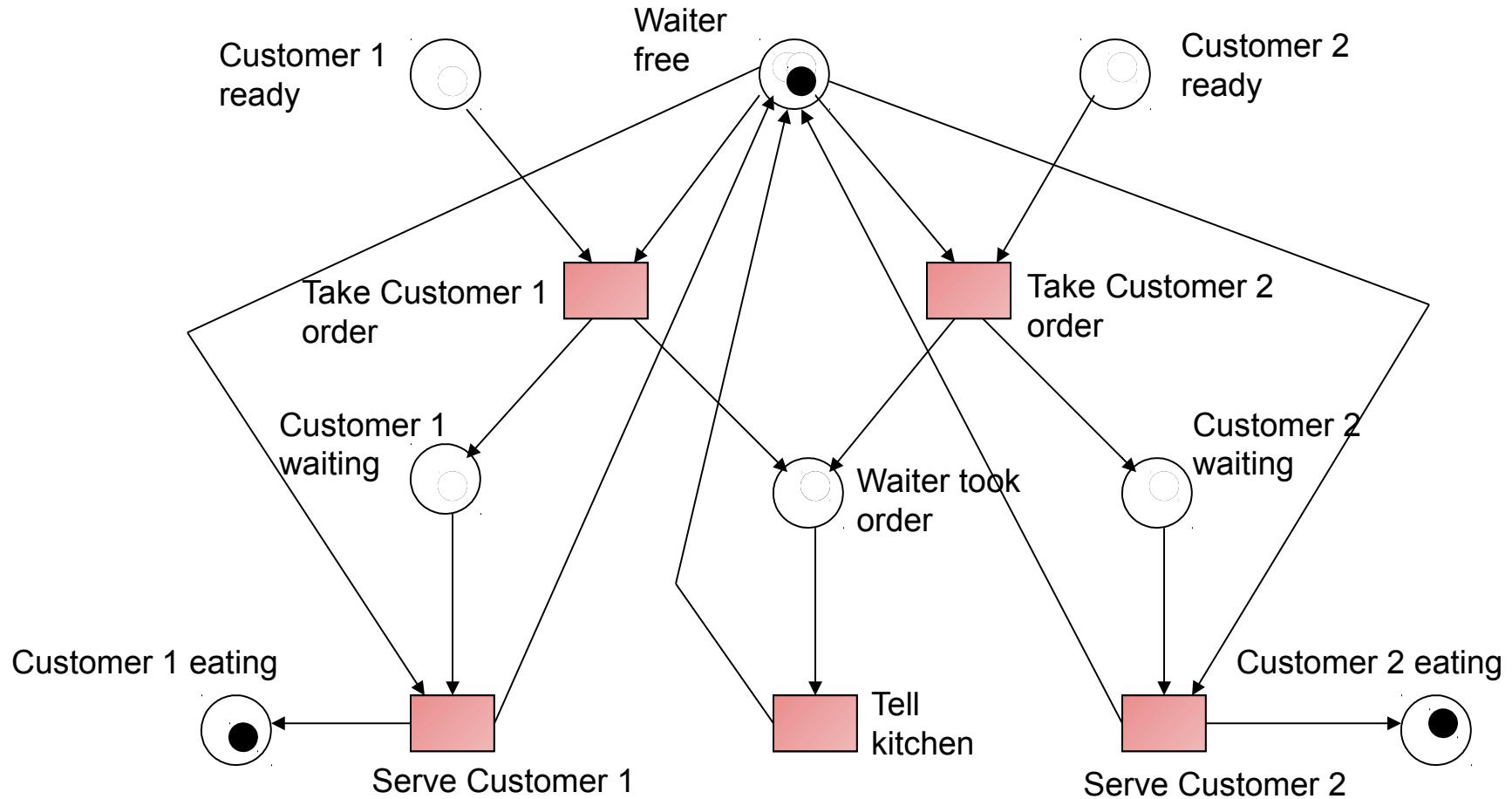
Scénario 1:

1. Take Customer 1 Order
2. Tell Kitchen
3. Serve Customer 1
4. Take Customer 2 Order
5. Tell Kitchen
6. Serve Customer 2

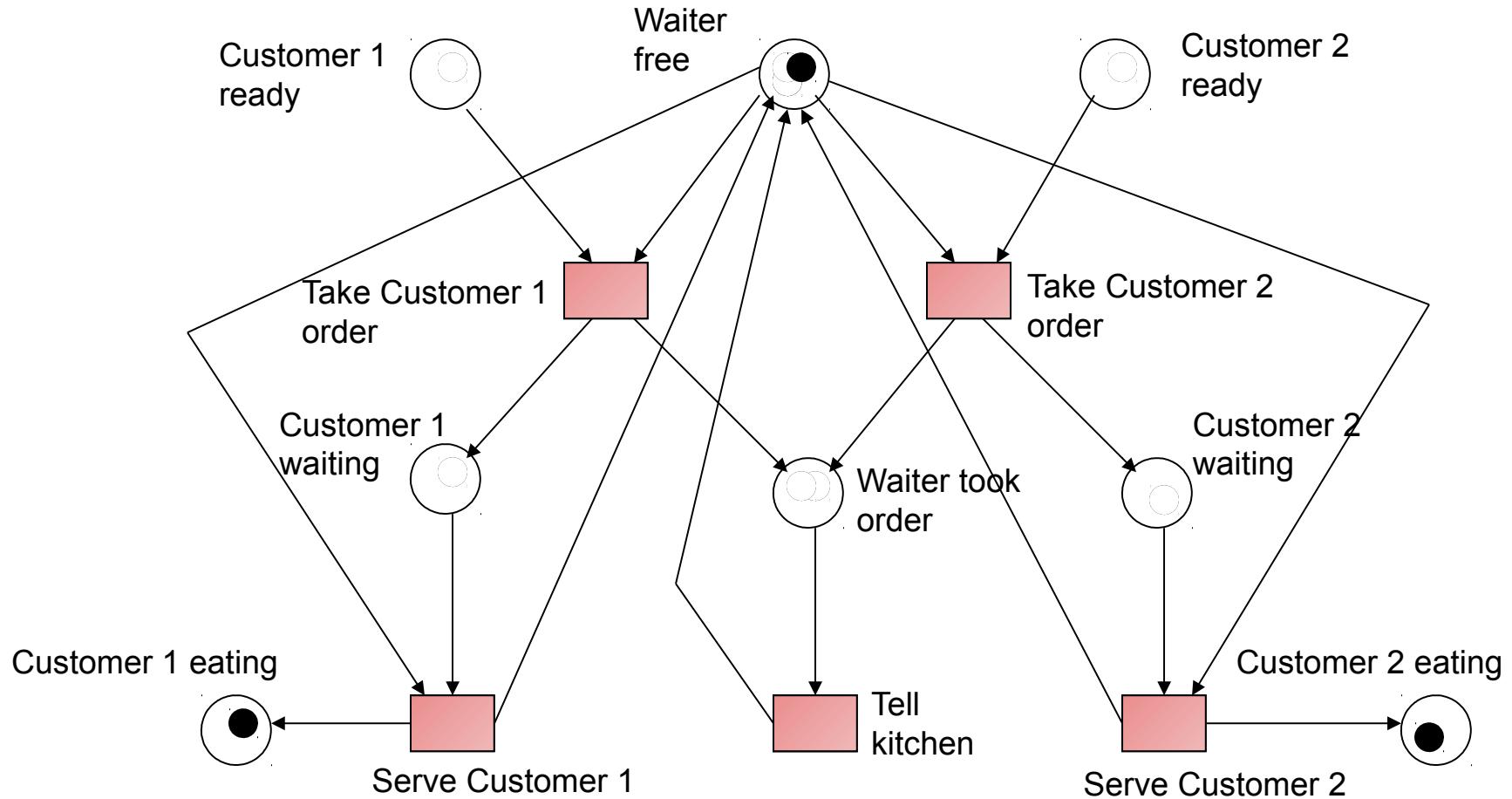
Scénario 2:

7. Take Customer 1 Order
8. Tell Kitchen
9. Take Customer 2 Order
10. Tell Kitchen
11. Serve Customer 2
12. Serve Customer 1

EXEMPLE: RESTAURANT (SCÉNARIO 1)

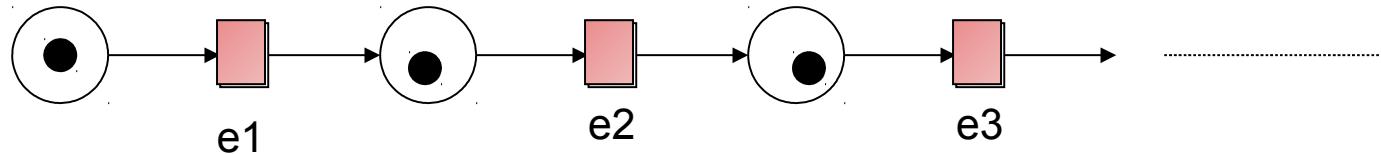


EXEMPLE: RESTAURANT (SCÉNARIO 2)

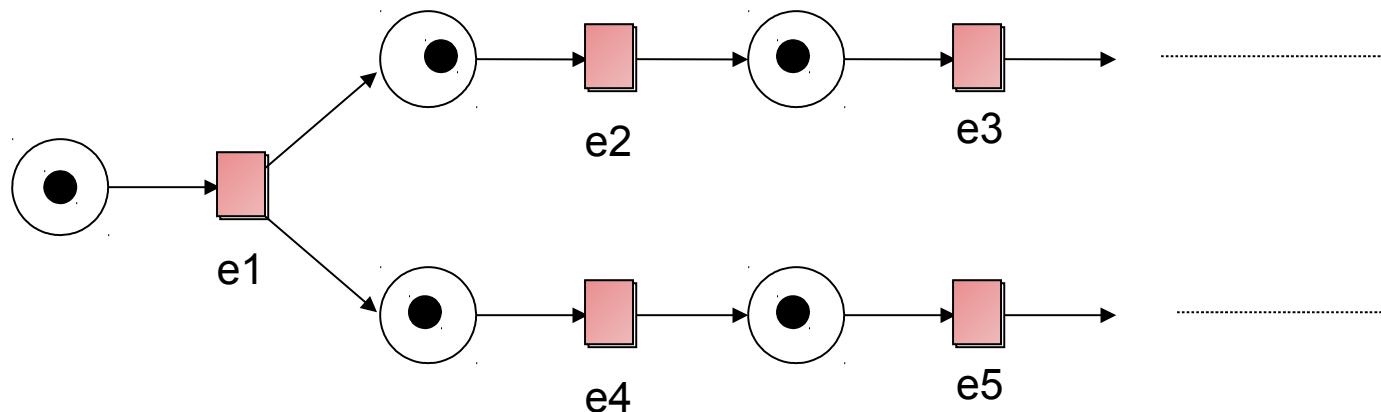


STRUCTURE DE RÉSEAU

Une séquence d'événements/actions:

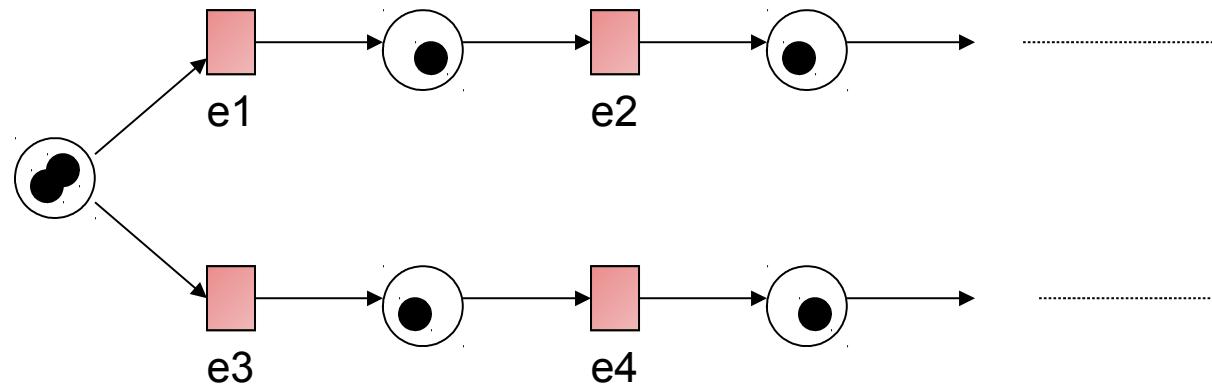


Exécutions concurrentielles:



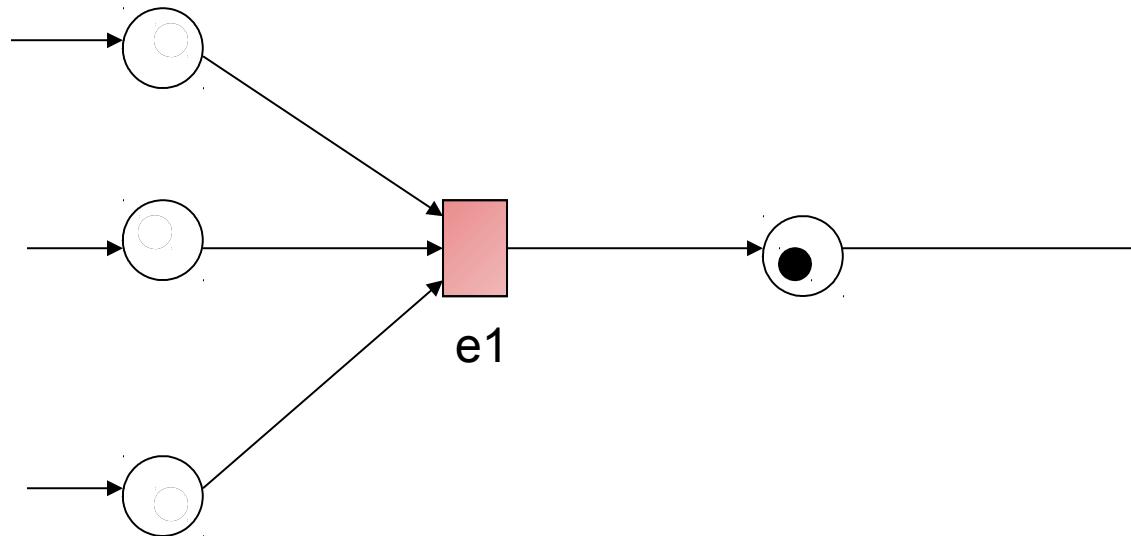
STRUCTURE DE RÉSEAU

**Événements non-déterminants - conflit,
choix ou décision: un choix entre e1, e2 ...
ou e3, e4 ...**



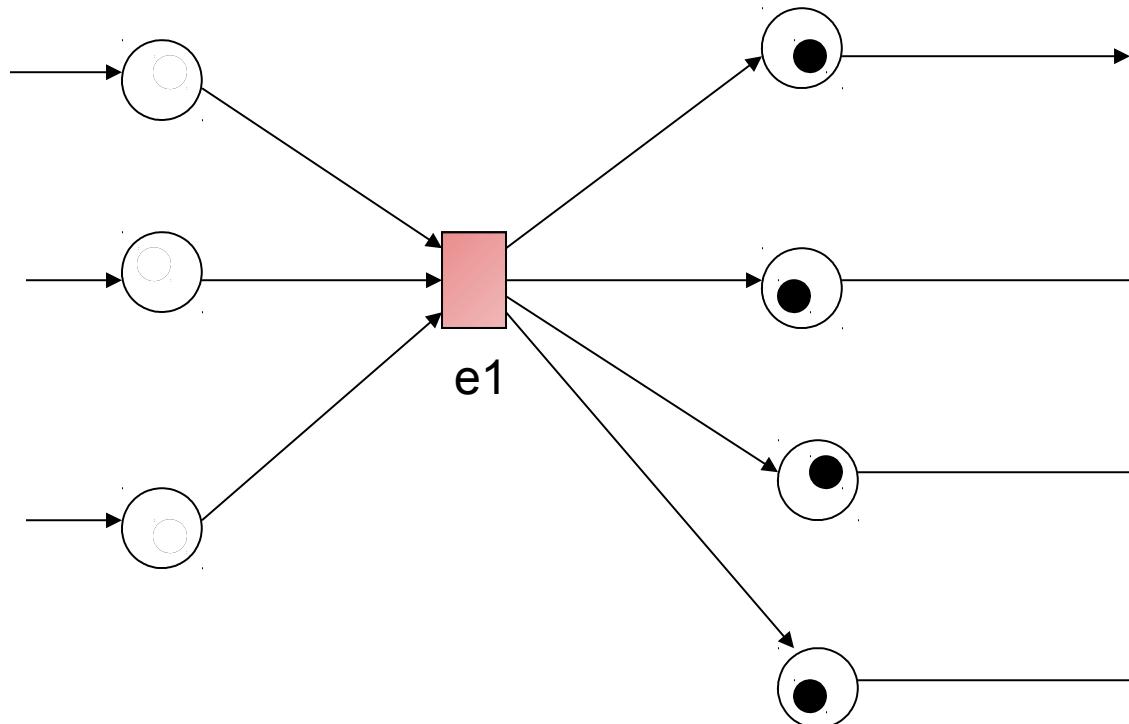
STRUCTURE DE RÉSEAU

Synchronisation



STRUCTURE DE RÉSEAU

Synchronisation et Concurrence





uOttawa

L'Université canadienne
Canada's university

UN AUTRE EXEMPLE

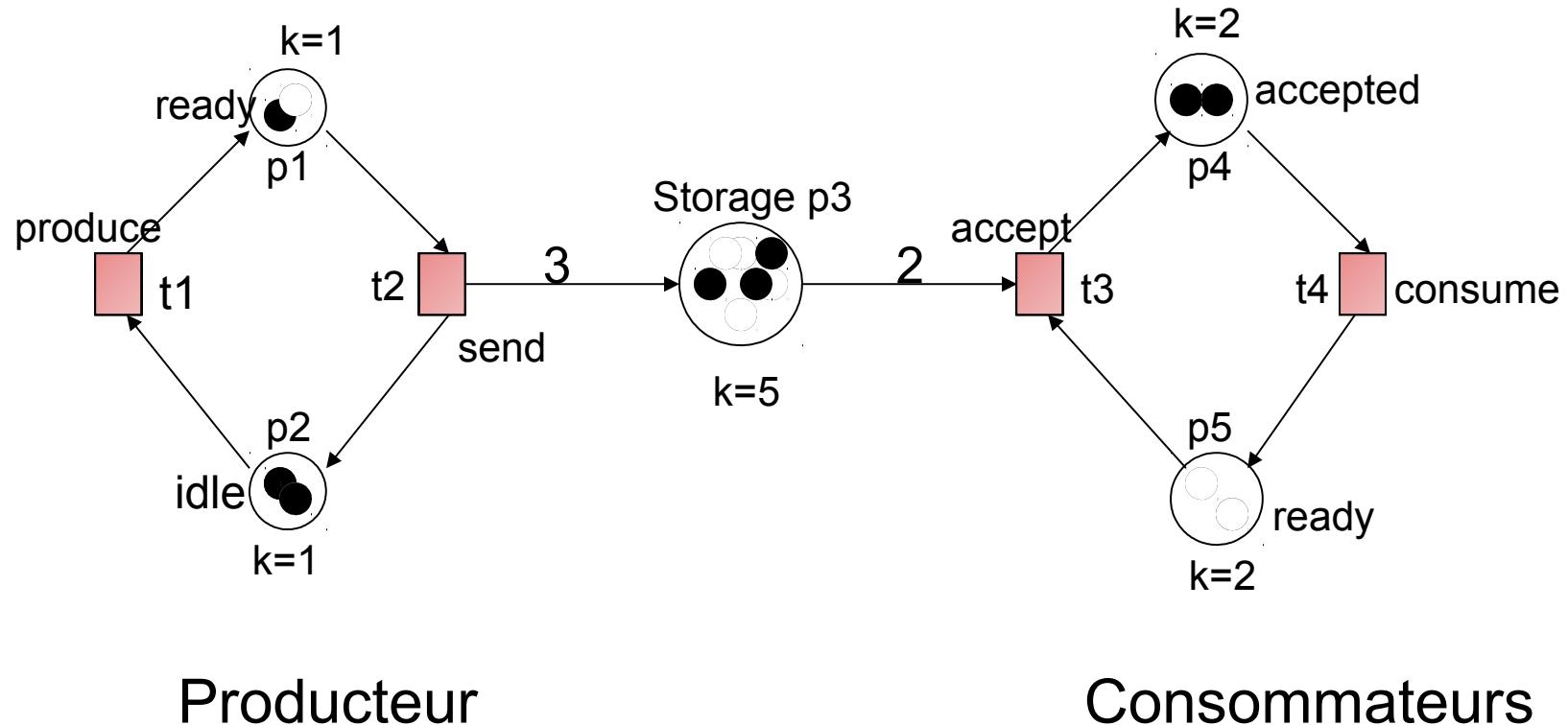
Un système producteur-consommateur est formé de:

- Un producteur
- Deux consommateurs
- Un tampon de stockage

Avec les conditions suivantes:

- Le tampon de stockage peut contenir 5 items au maximum;
- Le producteur envoie 3 items dans chaque production;
- Au maximum, un seul consommateur est capable d'accéder le tampon de stockage à la fois;
- Chaque consommateur enlève deux items lorsqu'il accède le tampon de stockage

UN SYSTÈME PRODUCTEUR- CONSOMMATEUR



Producteur

Consommateurs



uOttawa

L'Université canadienne
Canada's university

UN EXEMPLE DE PRODUCTEUR- CONSOMMATEUR

Dans ce réseau de Petri, chaque place possède une capacité et chaque arc possède un poids

Ceci permet à plusieurs jetons de résider dans une place afin de modéliser un comportement plus complexe



uOttawa

L'Université canadienne
Canada's university

PROPRIÉTÉS COMPORTEMENTALES

Accessibilité

- “Peut-on atteindre un état particulier d'un autre?”

Délimitation

- “Est-ce que le nombre de jetons dans une place peut augmenter infiniment?”

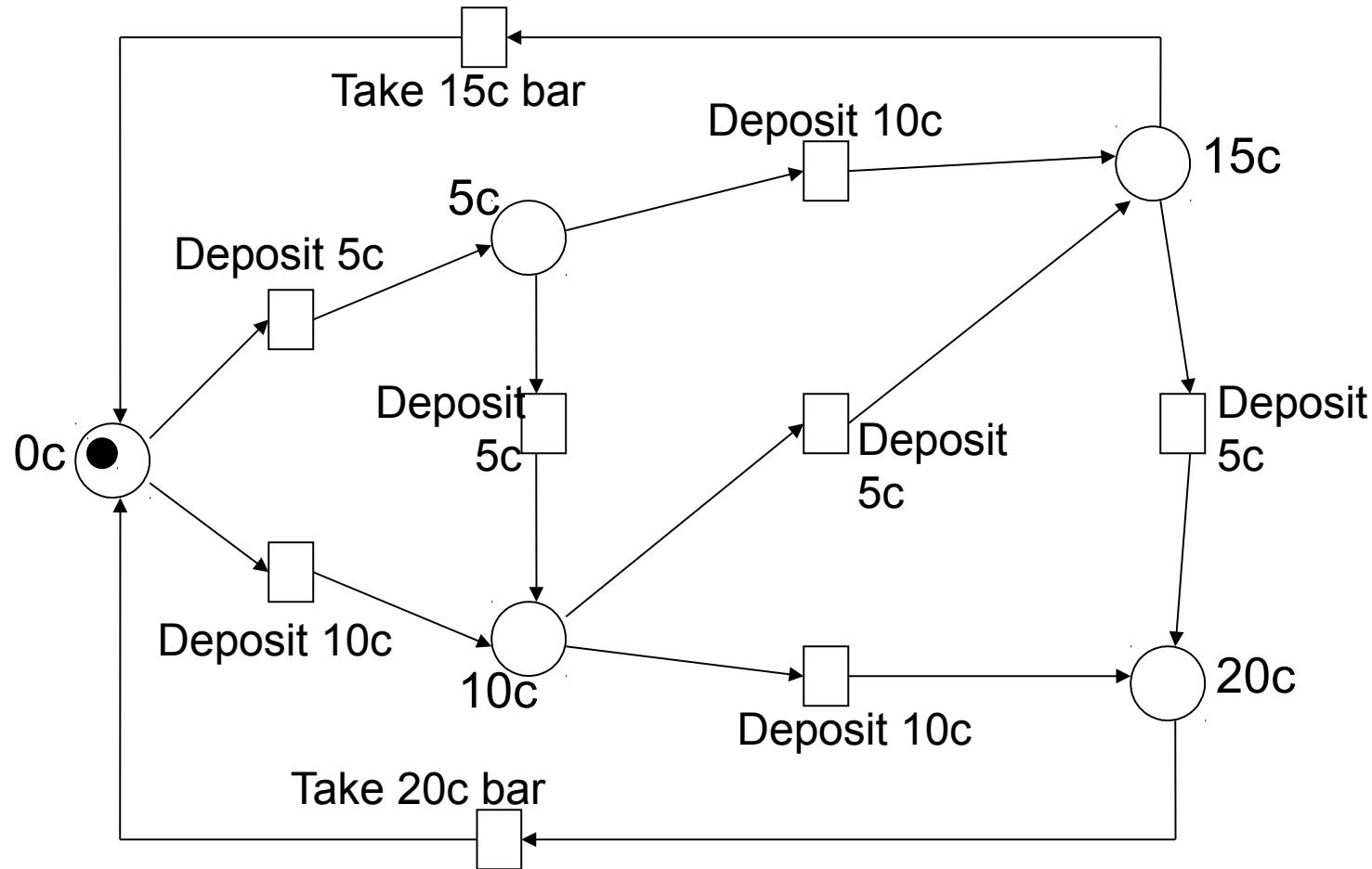
Vivacité

- “Est-ce que le système meurt dans un état particulier?”

SOUVENONS-NOUS DU DISTRIBUTEUR AUTOMATIQUE (JEU DE JETON)



uOttawa
L'Université canadienne
Canada's university

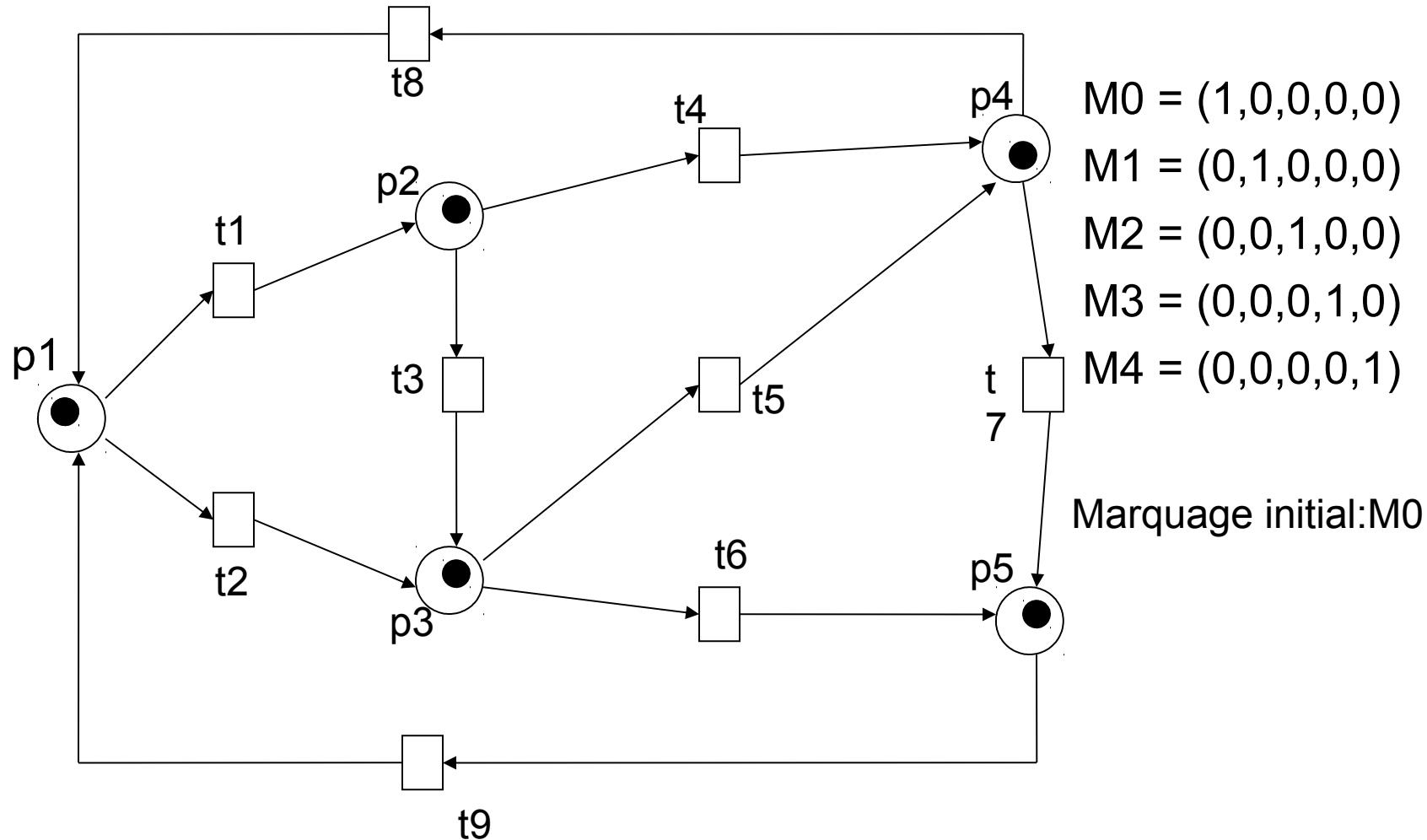


LE MARQUAGE EST UN ÉTAT ...

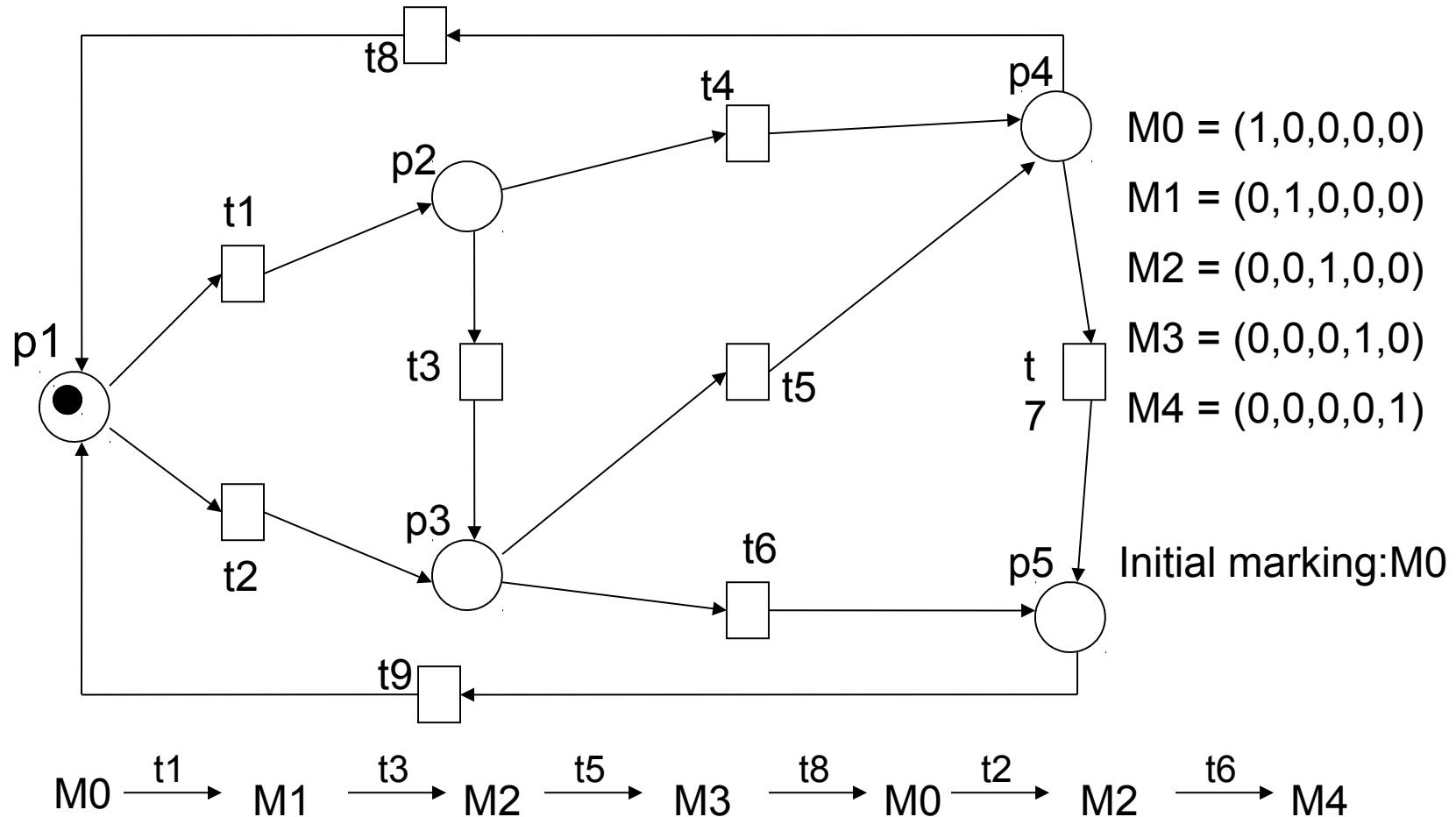


uOttawa

L'Université canadienne
Canada's university



ACCESSIBILITÉ





uOttawa

L'Université canadienne
Canada's university

ACCESSIBILITÉ

Une séquence d'excitation:



“M2 est atteignable de M1 et M4 est atteignable de M0.”

En fait, dans l'exemple du distributeur automatique, tous les marquages sont atteignables de chaque marquage.



uOttawa

L'Université canadienne
Canada's university

DÉLIMITATION

Un réseau de Petri est dit délimité par k (ou simplement délimité) si le nombre de jetons dans chaque place n'excède pas un nombre fini k pour n'importe quel marquage atteignable de M₀.

Le réseau de Petri pour le distributeur automatique est délimité par 1.



uOttawa

L'Université canadienne
Canada's university

VIVACITÉ

Un réseau de Petri est *vivant* si, quel que soit le marquage atteint, il est toujours possible d'effectuer *au moins une transition*

Un réseau de Petri vivant garantit un fonctionnement sans inter-blocage, quelle que soit la séquence de transitions choisie.



uOttawa

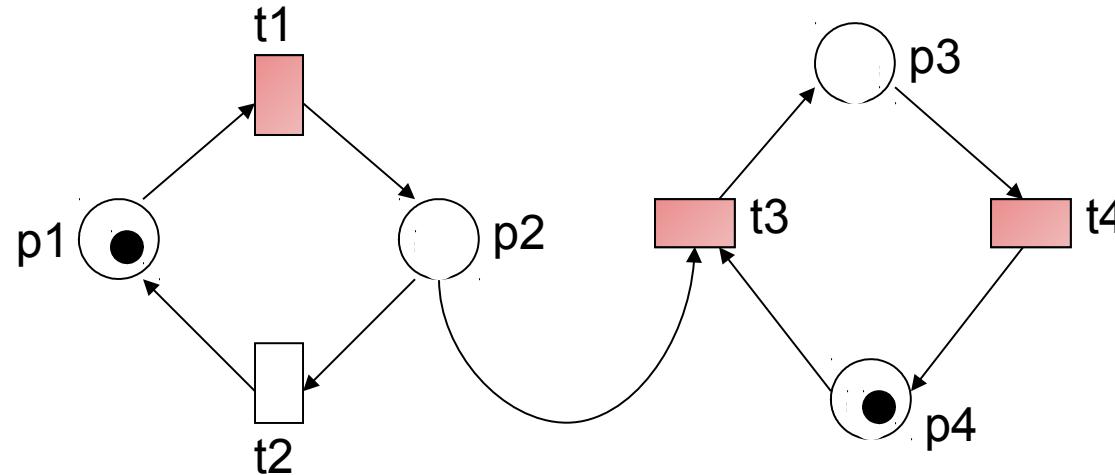
L'Université canadienne
Canada's university

VIVACITÉ

Le distributeur automatique est vivant et le système producteur-consommateur est également vivant.

Une transition est *morte* si elle ne peut jamais être excitée dans n'importe quel séquence de transitions.

UN EXEMPLE



$$M_0 = (1, 0, 0, 1)$$

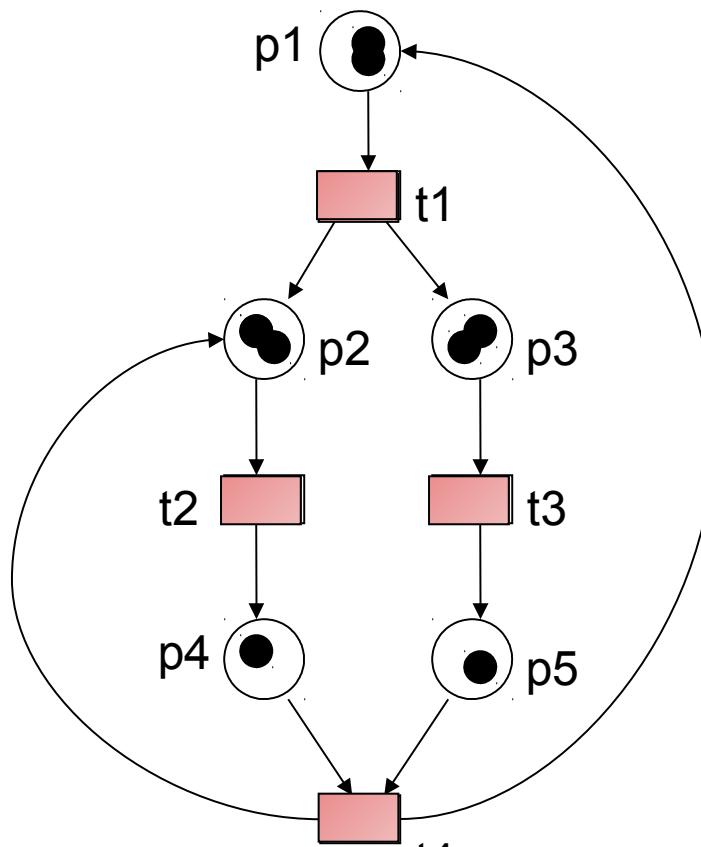
$$M_1 = (0, 1, 0, 1)$$

$$M_2 = (0, 0, 1, 0)$$

$$M_3 = (0, 0, 0, 1)$$

Réseau de Petri délimité mais pas-vivant

AUTRE EXEMPLE



$$M_0 = (1, 0, 0, 0, 0)$$

$$M_1 = (0, 1, 1, 0, 0)$$

$$M_2 = (0, 0, 0, 1, 1)$$

$$M_3 = (1, 1, 0, 0, 0)$$

$$M_4 = (0, 2, 1, 0, 0)$$

⋮

Le système est vivant mais pas délimité

MERCI!

QUESTIONS?

SÉANCE 6

**LANGAGE DE
SPÉCIFICATION ET DE
DESCRIPTION(SDL)**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Les objectifs de SDL

Structure de SDL

- Système
- Bloc

Comportement de SDL

- Processus

Signaux de SDL

- Adressage de processus implicite et explicite

Exemple de définition de système SDL



uOttawa

L'Université canadienne
Canada's university

QU'EST-CE QUE LE SDL

SDL est une langue formelle développée et standardisée par le ITU-T

Elle est prévue pour la spécification d'applications complexes, distribué, événementielles, à temps réel, et interactives

Ces applications impliquent typiquement plusieurs activités concurrentes qui communiquent en utilisant des signaux discrets



uOttawa

L'Université canadienne
Canada's university

QU'EST-CE QUE LE SDL

SDL décrit l'*architecture*, le comportement et les données de systèmes distribués dans des environnements à temps réel

Il est utile pour spécifier la spécification du comportement du système de télécommunication

SDL couvre différents niveaux d'abstraction, depuis une vue d'ensemble étendue jusqu'au niveau de conception détaillé



uOttawa

L'Université canadienne
Canada's university

DOMAINE D'APPLICATION

Type de systèmes:

- Temps réel
- Interactif
- Distribué
- Hétérogène

Type d'information:

- Comportement, structure

Niveau d'abstraction:

- Vue d'ensemble jusqu'aux détails



uOttawa

L'Université canadienne
Canada's university

UTILISATION DU SDL

SDL peut être utilisé pour des étapes variées du développement:

- Modélisation formelle de spécification
- Conception
- *Implémentation?*
 - *SDL n'était pas originalement prévu pour l'implémentation, mais la traduction au code est possible*



uOttawa

L'Université canadienne
Canada's university

OBJECTIFS

L'utilisation d'outils afin de créer, maintenir, et analyser les spécifications

- Descriptions formelles de haute qualité produites d'une façon rapide et moins chère
- Analyse de spécifications pour la compléction et la justesse
- Support de la communication humaine et compréhension des exigences

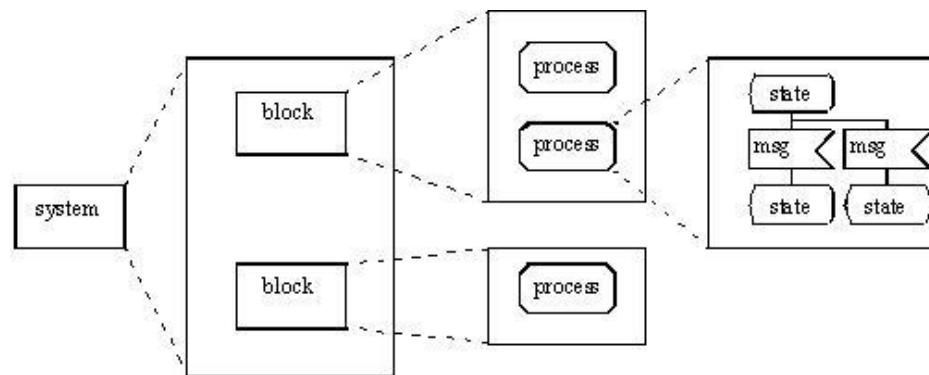
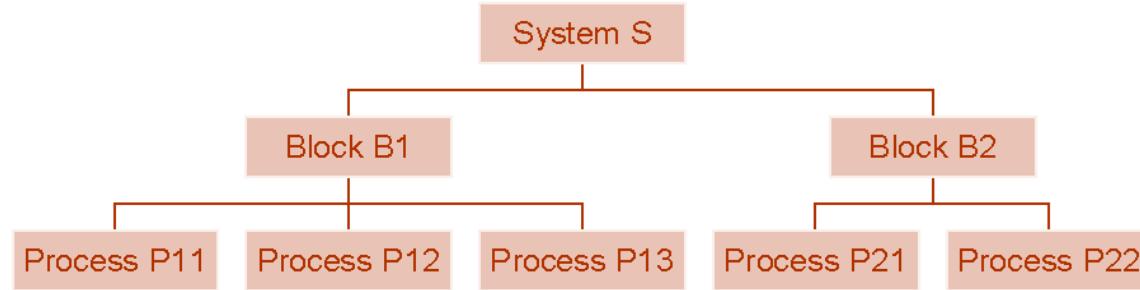
Peut être utilisée pour l'optimisation de la conception

- Fournit les programmeurs d'un moyen facile pour effectuer les vérifications et validations de la conception

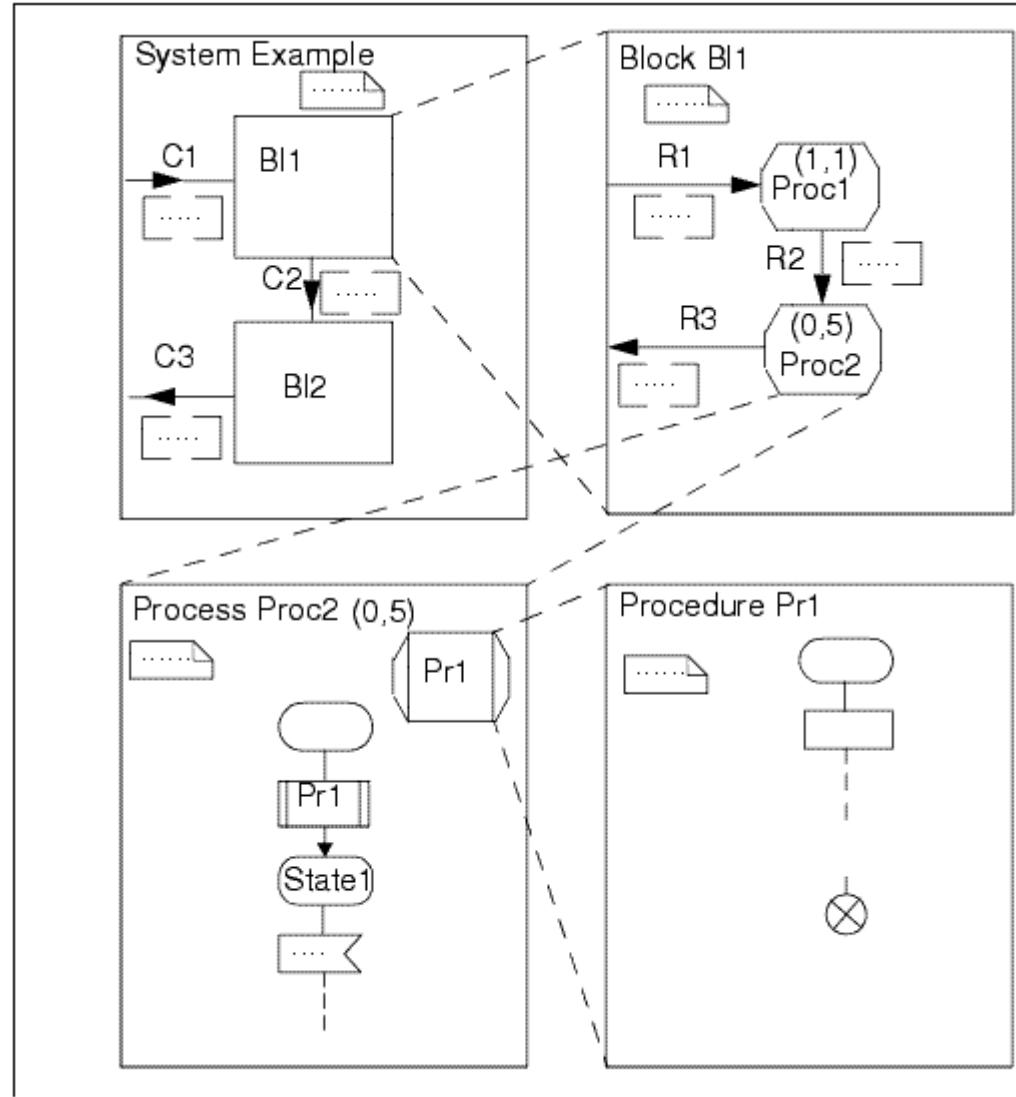
STRUCTURE SDL

Comprend trois niveaux principaux hiérarchiques:

- Système
- Bloc
- Processus



STRUCTURE SDL





uOttawa

L'Université canadienne
Canada's university

STRUCTURE SDL

Un système contient un ou plusieurs blocs, interconnectés ensemble et avec les limites du système par des canaux

- Les processus communiquent ensemble en utilisant des signaux envoyés par des canaux

Le bloc est le concept structural principal

Un bloc peut être réparti en sous-blocs

Un canal est un moyen de transport des signaux

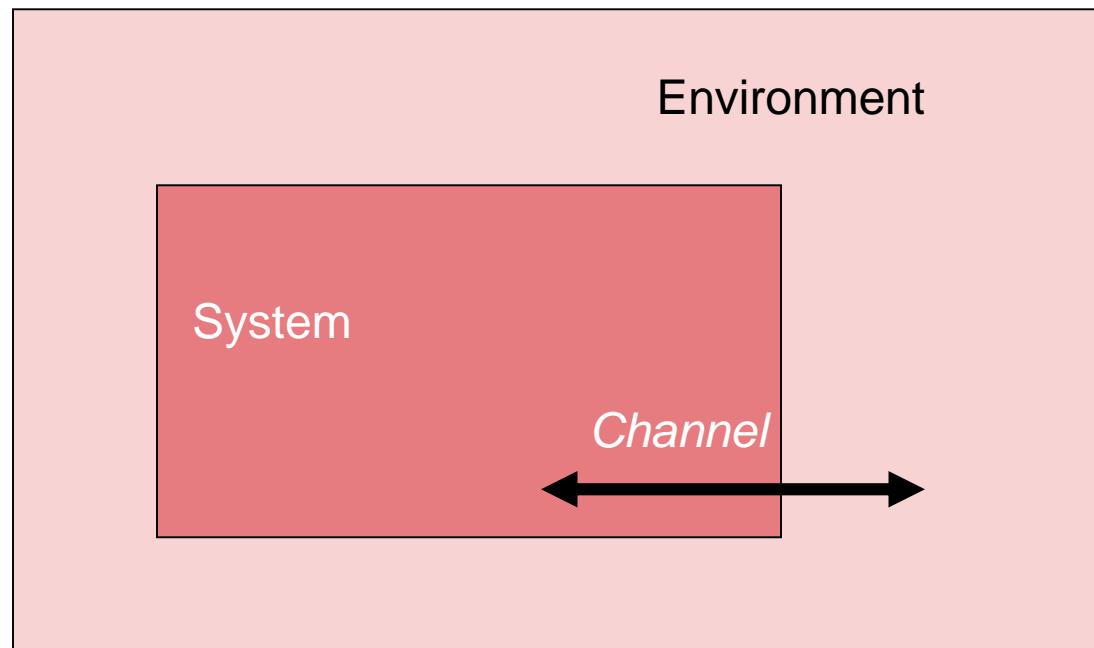


uOttawa

L'Université canadienne
Canada's university

SYSTÈME

Le système représente une machine abstraite qui communique avec son environnement



SYSTÈME

System name

System Deamongame

1(1)

/* This system is a game having any number of playes. The players belong to the environment of the system. A "Deamon" in the environment of the system sends Bump signals randomly to the system. A player has to guess whether the number of the Bump signals received by the system is odd or even. The guess is made by sending a Probe signal to the system. The system relies by the signal Win if the number received Bump signals is odd, otherwise by the signal Lose.

The system keeps track of the score of each player. Aplayer can ask for the current value of his score by the signal Result, which is answered by the system with the signal score.

Before a player can start playing, he must log in. This is accomplished by the signal Newgame . A player logs out by the signal Endgame. */

SIGNAL Newgame, Probe, Result, Endgame, Gameid, Win, Lose,
Score (Integer), Bump;

Text symbol

Signal declaration

Channel

Deamonserver



Block

Signal

[Bump]

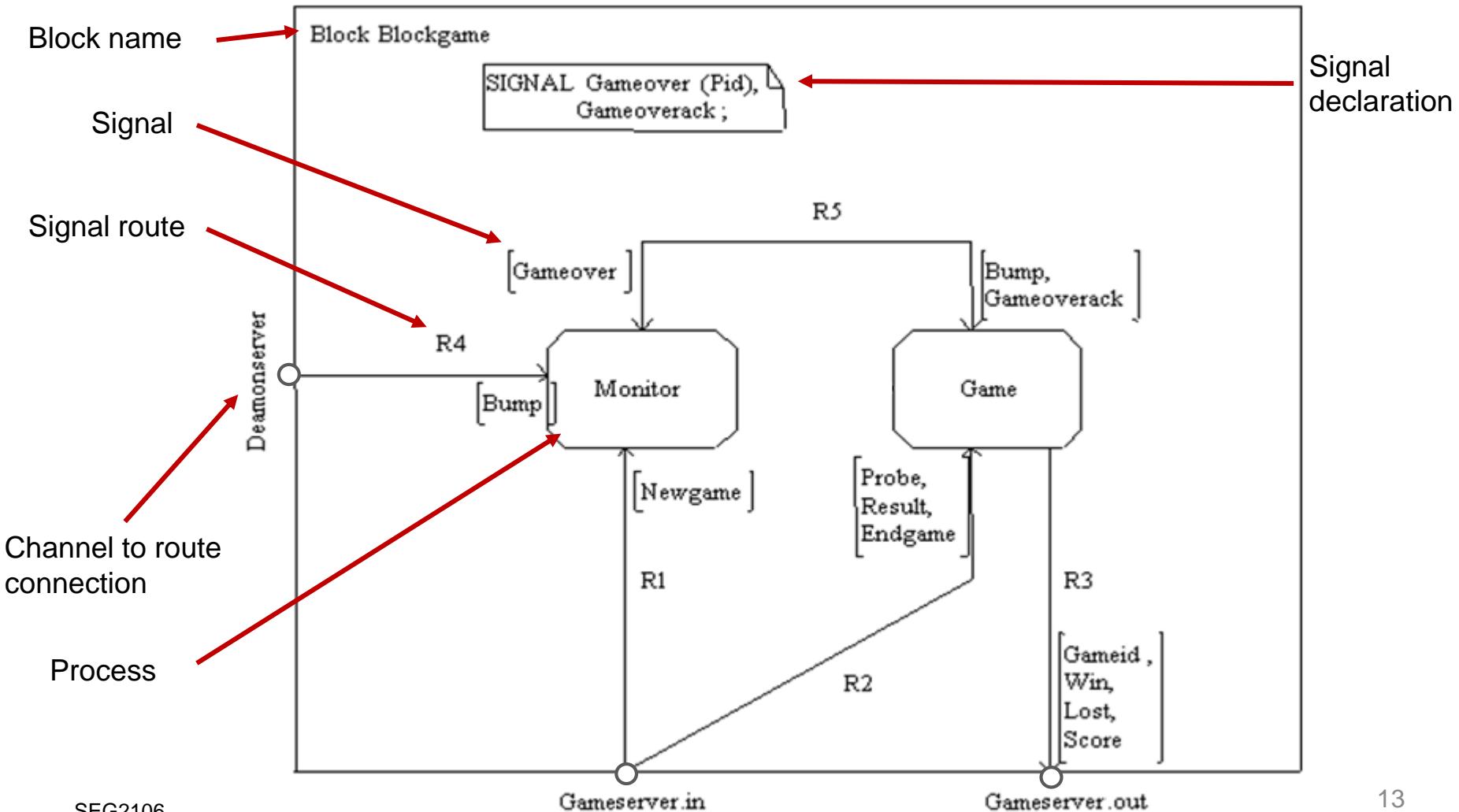
Gameserver.in

Gameserver.out

Newgame,
Probe,
Result,
Endgame

Gameid,
Win,
Lose,
Score

BLOC





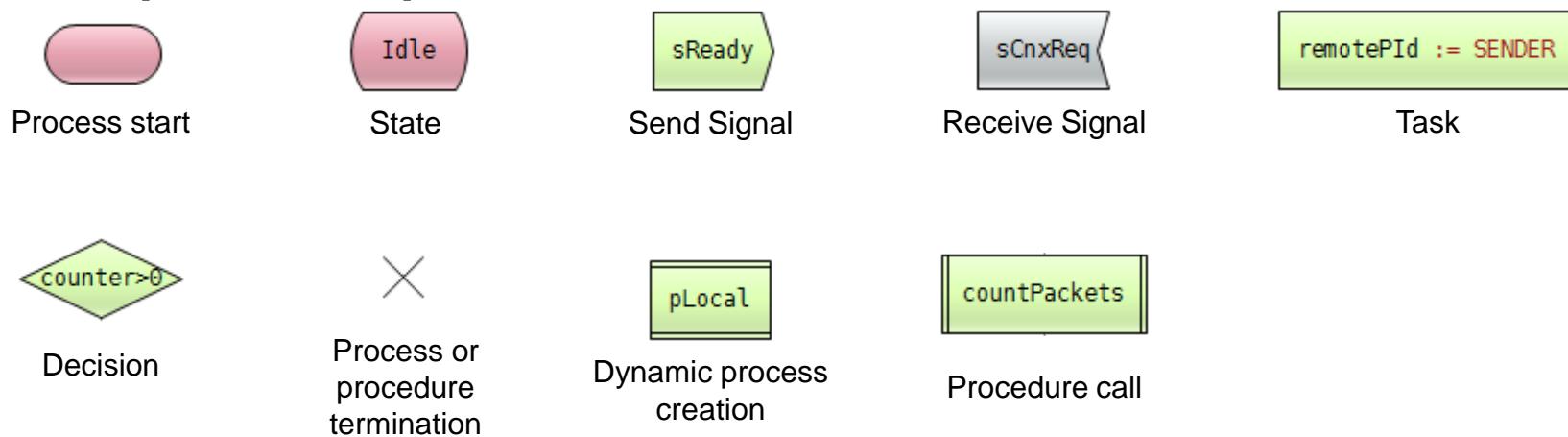
PROCESSUS

Un processus dans SDL est une machine d'état finie étendue

Le comportement d'une machine d'état finie est décrite par les états et les transitions

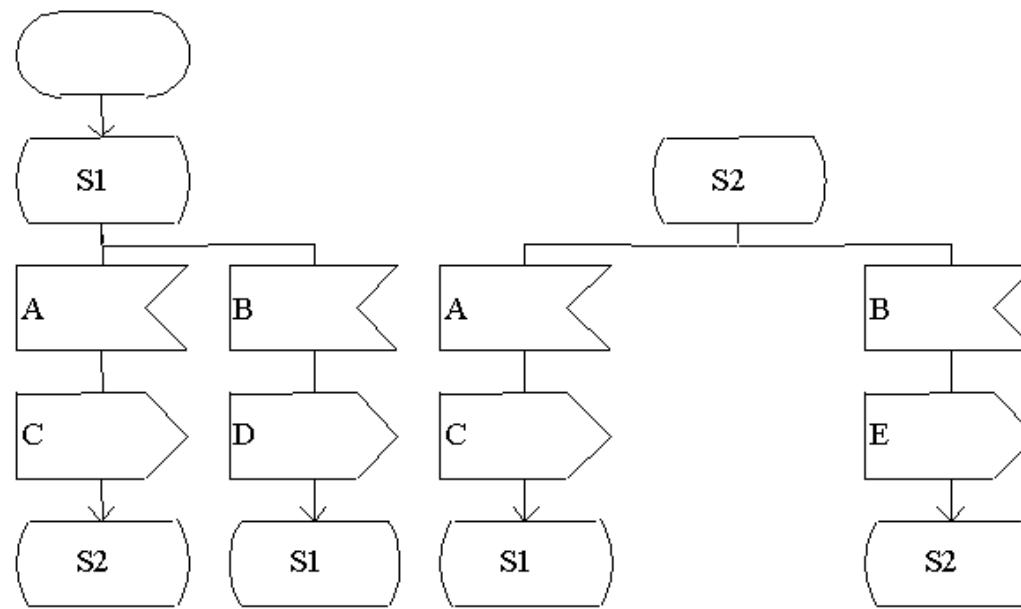
Une description de processus est livrée à travers un diagramme de processus

Ci-dessous sont les éléments de base utilisées pour la description d'un processus:



PROCESSUS

Process Example



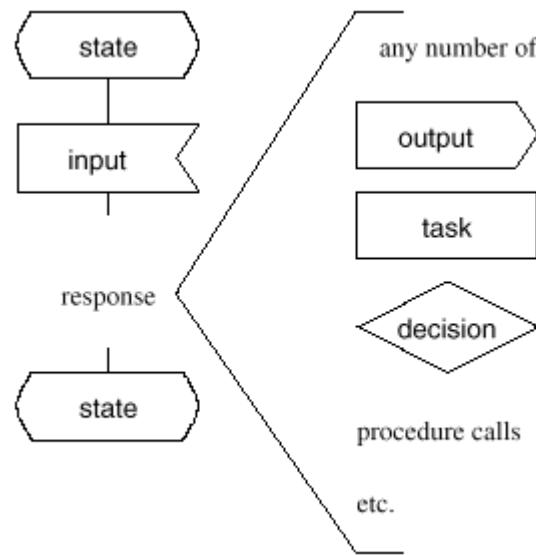


uOttawa

L'Université canadienne
Canada's university

FORMULE DU DIAGRAMME DE PROCESSUS

En général, chaque transition d'état possède la formule suivante



Dans une transition d'état, un processus reste dans son état actuel jusqu'à ce qu'un événement de données prévu est reçu.



uOttawa

L'Université canadienne
Canada's university

PROCÉDURE

La construction de procédure dans SDL est similaire à celle connue à partir des langues de programmation

Une procédure est une machine d'état finie en dedans d'un processus

- Elle est créée lorsqu'un appel de procédure est interprété, et meurt lorsqu'il est terminé

Une description de procédure est similaire à une description de processus, avec quelques exceptions.

Le symbole de départ est remplacé par le symbole de départ de la procédure





uOttawa

L'Université canadienne
Canada's university

DÉcrire le comportement avec SDL

Le comportement d'un système est constitué par la combinaison de comportements des processus dans le système

Un processus est défini comme étant une machine d'état finie, qui travaille d'une façon autonome et concurrentielle avec d'autres processus

Un processus réagit aux stimuli externes en accord avec sa description

Un processus est soit dans un état d'attente, ou il effectue une transition entre les deux états



uOttawa

L'Université canadienne
Canada's university

DÉcrire le comportement avec SDL

La coopération entre les processus est effectuée d'une façon asynchrone à l'aide de messages discrets appelés signaux

Chaque processus possède une file infinie de données associée, qui agit comme une file FIFO

Tout signal qui arrive au processus rentre dans sa file d'attente de données

- L'ensemble de signaux de données prédéfinis est défini comme étant les signaux que le processus se prépare à accepter

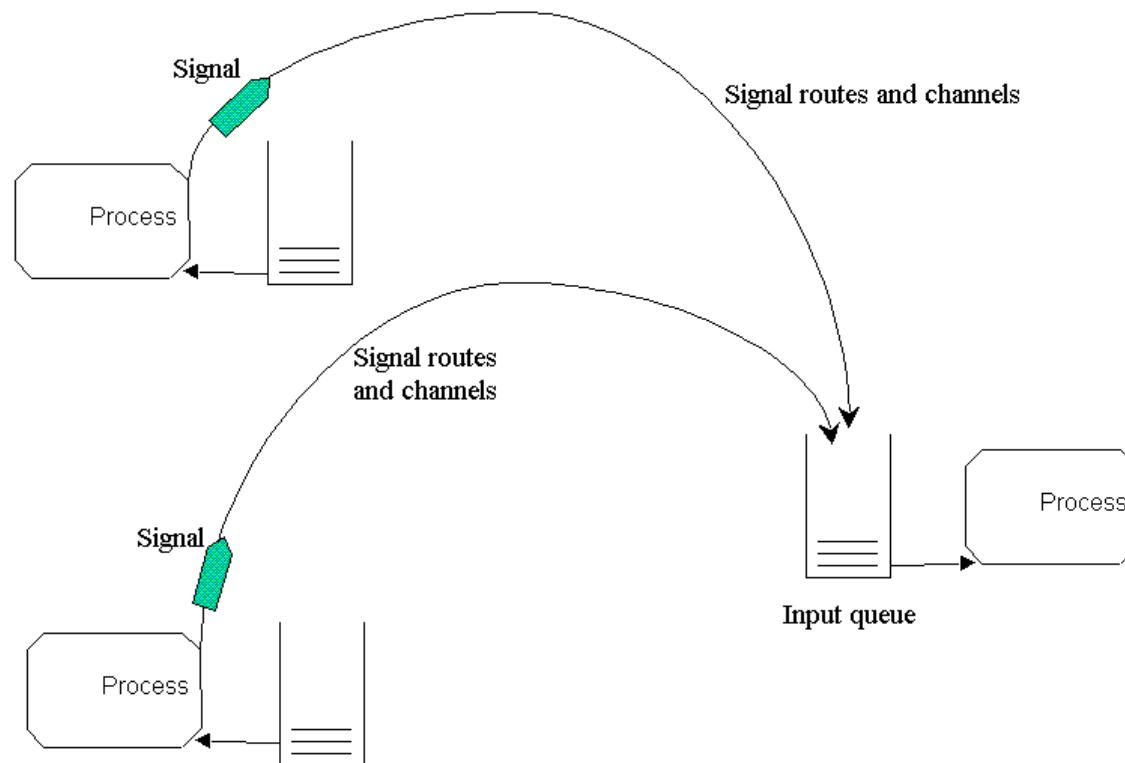
DÉcrire le comportement avec SDL



uOttawa

L'Université canadienne
Canada's university

Lorsqu'un signal a initié une transition, il est éliminé de la file d'attente de données





uOttawa

L'Université canadienne
Canada's university

VARIABLES DU PROCESSUS

Un processus peut utiliser et manipuler localement les données stockées dans les variables

- Celles-ci sont définies, dans un symbole de texte ajouté à la définition du processus
- Le mot-clé DCL introduit la définition d'une ou plusieurs variables dans une boîte de texte
- DCL représente Digital Command Language (langue digitale de commande)

```
DCL
Counter Integer := 0;
```

Lorsqu'il y a plus d'une variable définie après DCL, elles sont séparées par des virgules

- La liste de définition se termine par un point-virgule

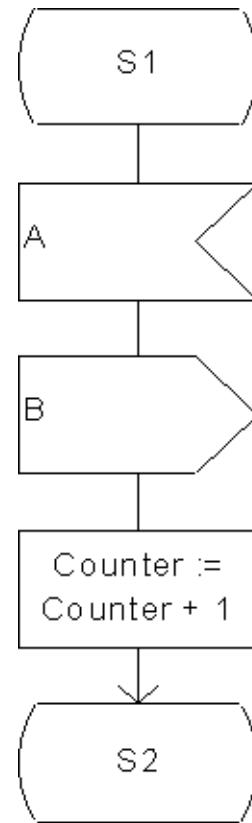


uOttawa

L'Université canadienne
Canada's university

VARIABLES DU PROCESSUS

Durant une transition, le processus peut utiliser et manipuler ses propres variables locales, en utilisant l'élément de tâches





uOttawa

L'Université canadienne
Canada's university

IDENTIFICATEUR DE PROCESSUS

Chaque instance de processus a un identificateur unique

L' identificateur n'est pas déterminée par le modélisateur

- Elle est créée par la machine SDL (i.e. programme utilisé pour la simulation) durant la création du processus
- La machine SDL garantit que chaque instance de processus reçoit un identificateur unique

L' identificateur d'une instance de processus n'est pas identique au nom du processus

- Il existe plusieurs instances de processus à partir d'une seule définition de processus



uOttawa

L'Université canadienne
Canada's university

IDENTIFICATEUR DE PROCESSUS

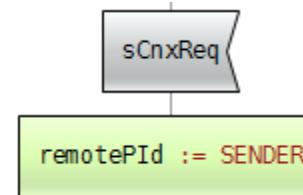
Afin de faire référence à un instance de processus, vous pouvez utiliser les mots-clés suivants

- SELF: retourne l' identificateur de l'instance de processus lui-même
- SENDER: retourne l'identificateur de l'instance de processus duquel le dernier signal consommé a été envoyé
- OFFSPRING: retourne l' identificateur de l'instance de processus qui a récemment été créé par ce processus
- PARENT: retourne l'adresse du processus créateur

L' identificateur de l'instance de processus peut être assignée à un variable de type PID

```
DCL
remotePID PID := NULL,
```

Déclare un variable de type PID



Affecte SENDER à la variable PID déclaré précédemment (sauvegarde pour faire référence dans le future)



uOttawa

L'Université canadienne
Canada's university

ADRESSAGE DU SIGNAL

Pour tout signal envoyé par un processus, il doit y avoir une seule destination

La destination peut être spécifiée:

- Implicitement
- Explicitement



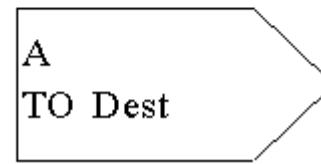
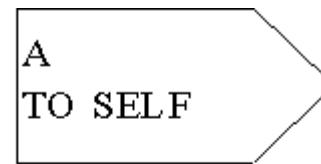
uOttawa

L'Université canadienne
Canada's university

ADRESSAGE EXPLICITE

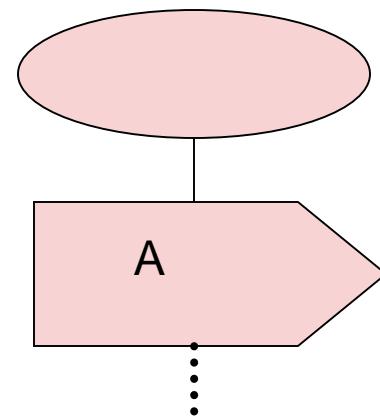
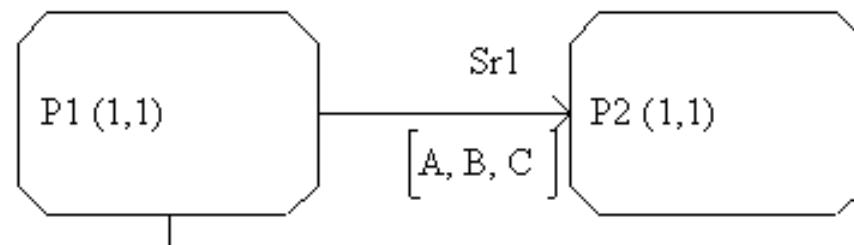
Le SDL définit le mot clé « TO » pour l'adressage explicite des processus

Le mot-clé TO est utilisé dans une donnée, et il est suivi par une expression qui contient l'identificateur de la destination du processus



ADRESSAGE IMPLICITE

La spécification explicite d'une adresse de spécification n'est pas nécessaire si la destination est uniquement définie par la structure du système

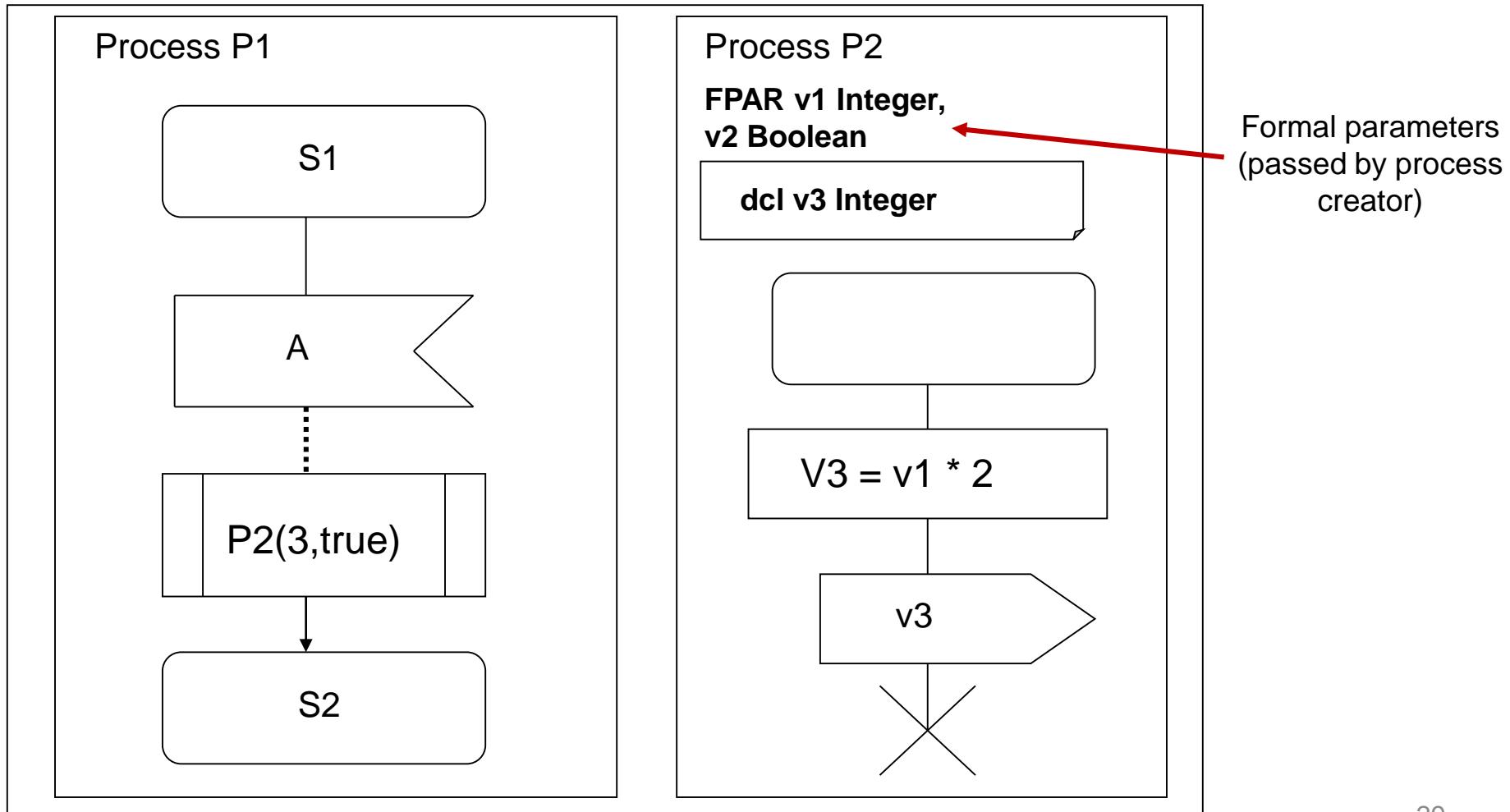


CRÉATION DE PROCESSUS/ TERMINAISON

Les processus peuvent être créés par d'autres processus dynamiquement au temps de l'interprétation

Les processus créateur et créé doivent appartenir au même bloc

CRÉATION DE PROCESSUS/ TERMINAISON





uOttawa

L'Université canadienne
Canada's university

EXAMPLE DE SMART HOME (MAISON INTELLIGENTE)

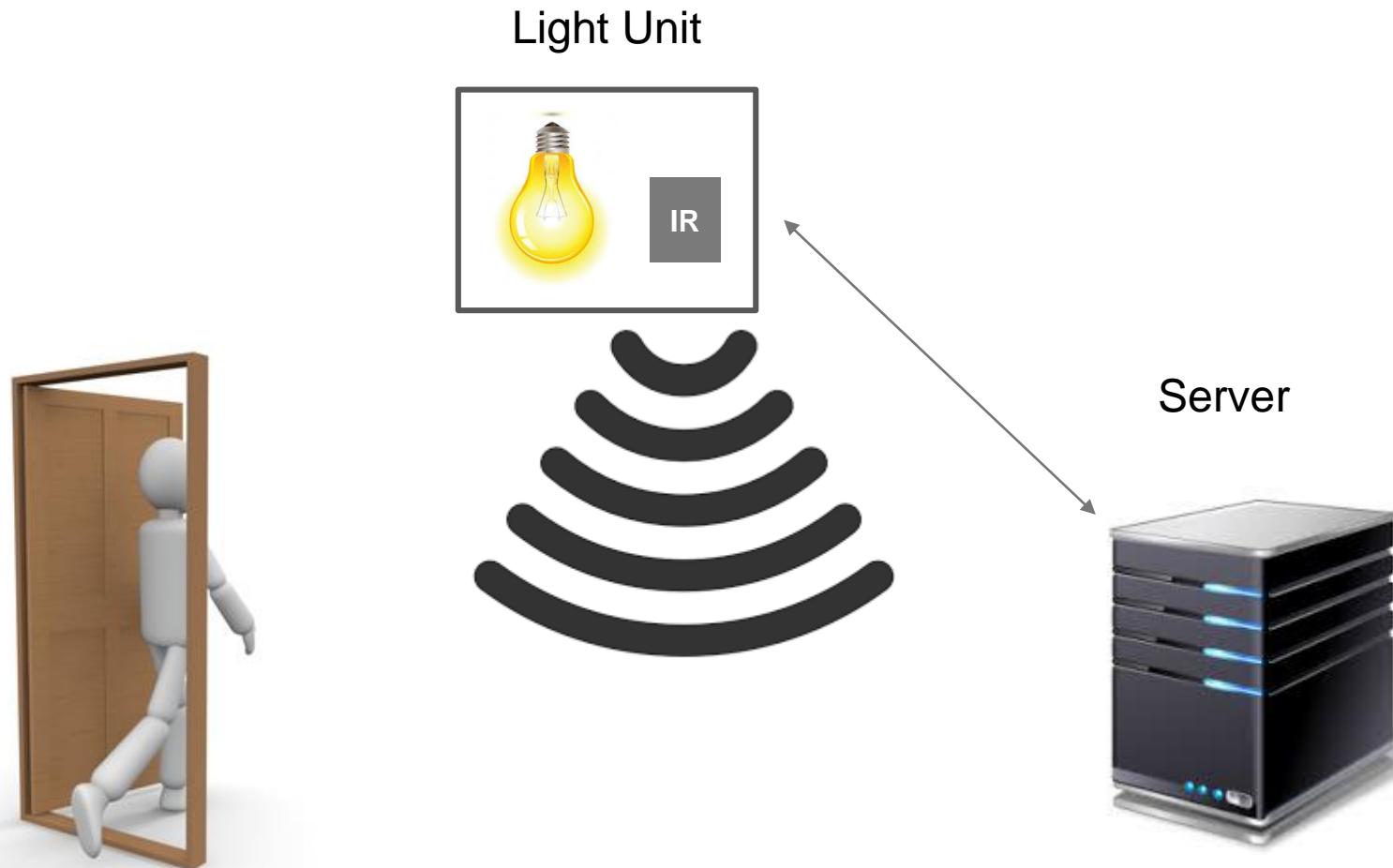
Nous allons modéliser un petit module de système de maison intelligente

Dans une maison intelligente, un processus central qui exécute sur un serveur contrôle plusieurs objets intelligents en dedans de la maison

Nous allons modéliser l'interaction entre le processus du serveur et une unité d'éclairage

- L'unité d'éclairage s'allume automatiquement lorsque quelqu'un entre dans la chambre
- L'unité d'éclairage a un capteur infra-rouge qui détecte le mouvement
- La lumière s'éteint lorsque s'il n'y a pas de détection de mouvement pendant 5 secondes

L'INTERACTION ENTRE LE SERVEUR ET L'UNITÉ D'ÉCLAIRAGE

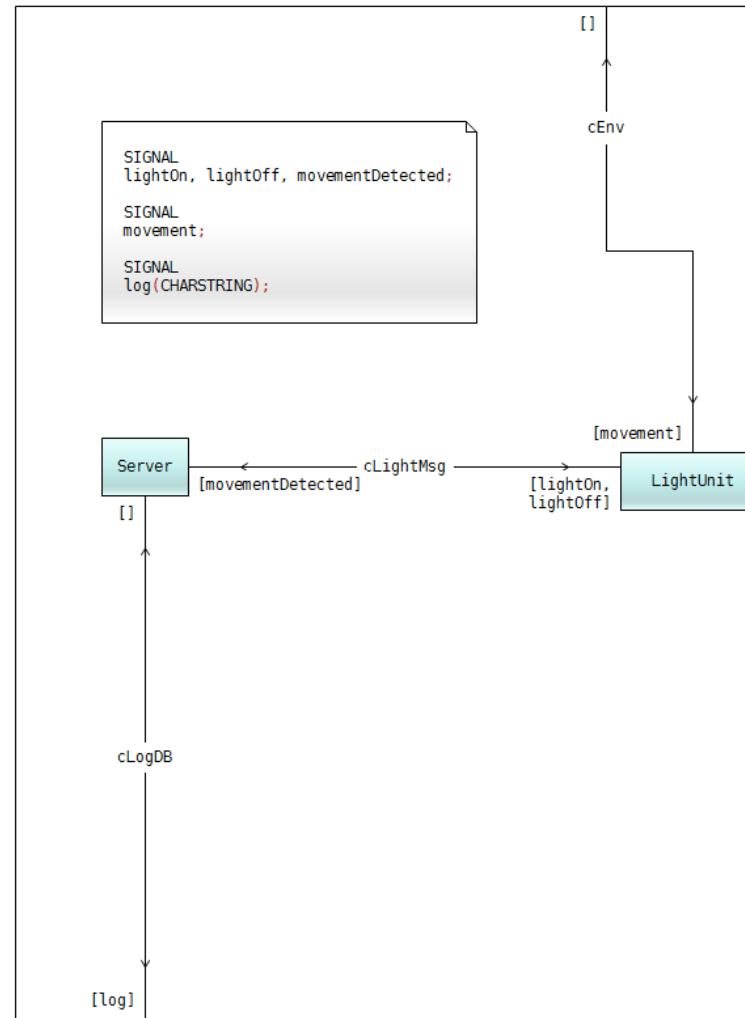




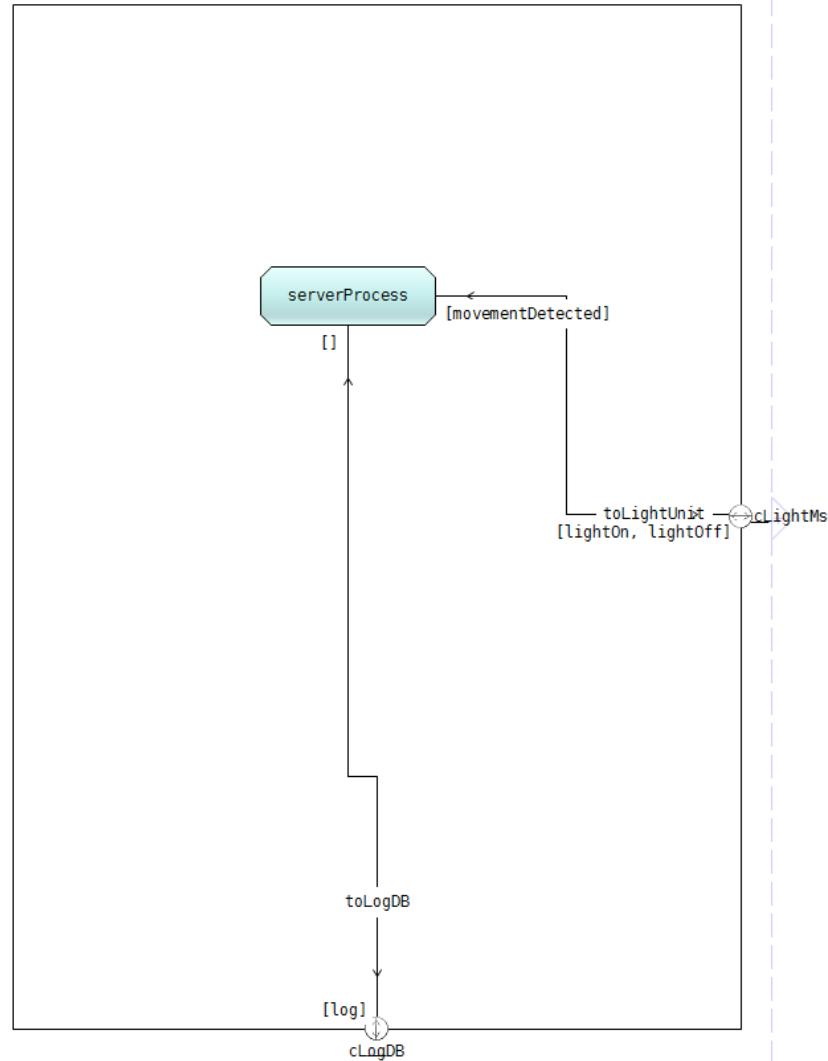
uOttawa

L'Université canadienne
Canada's university

DÉFINITION DU SYSTÈME



DÉFINITION DU BLOC DE SERVEUR

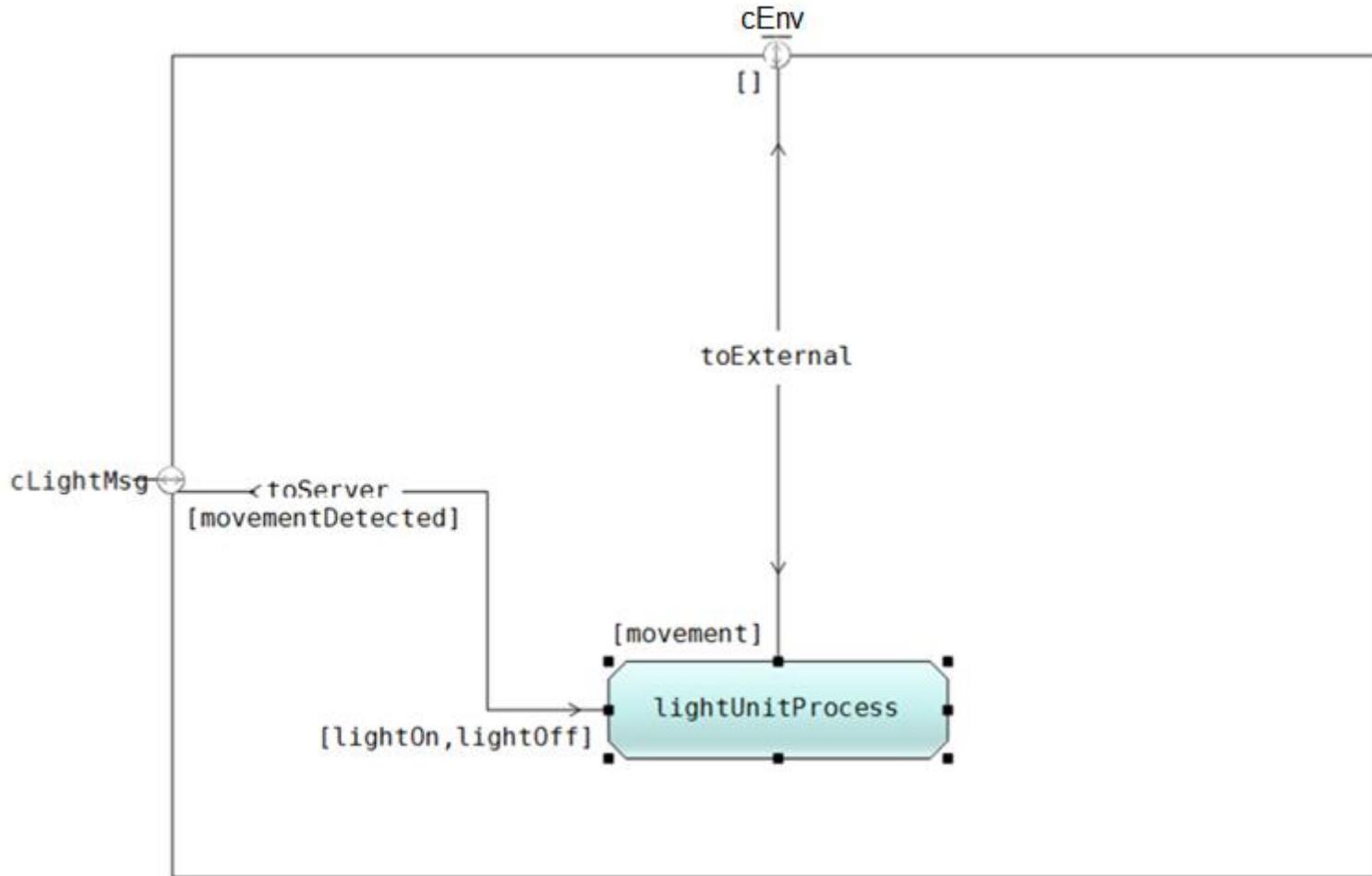


BLOC DE L'UNITÉ D'ÉCLAIRAGE



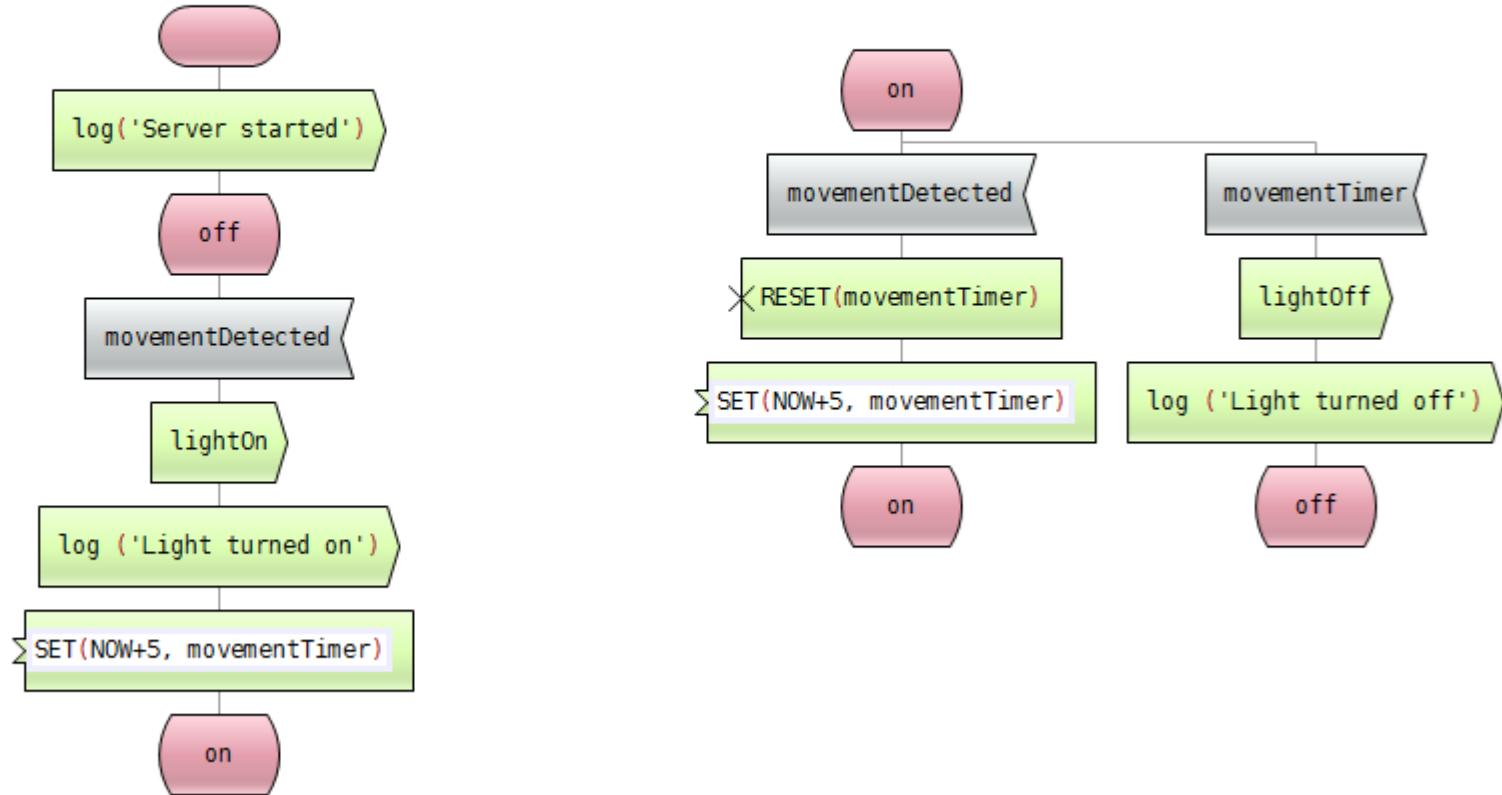
uOttawa

L'Université canadienne
Canada's university

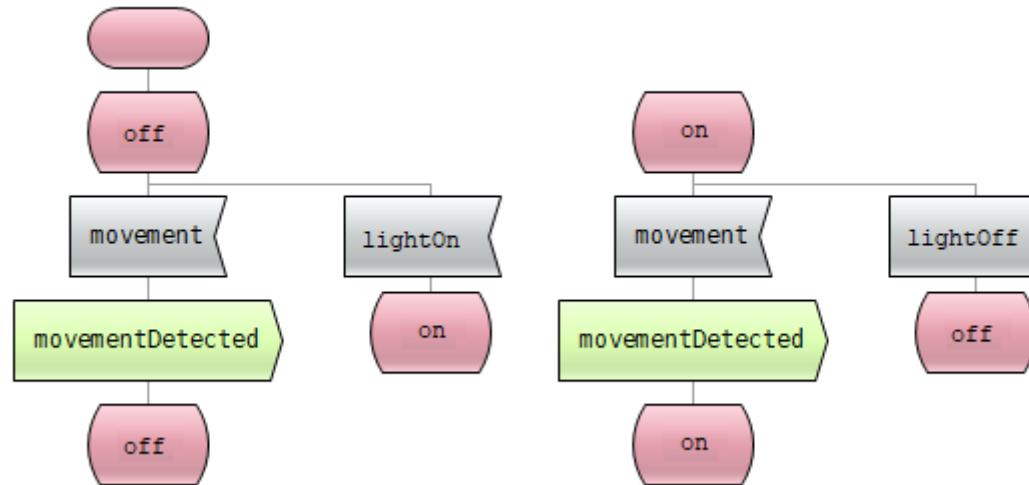


DÉFINITION DU SERVEUR DE PROCESSUS

```
TIMER  
movementTimer;
```



PROCESSUS DE L'UNITÉ D'ÉCLAIRAGE



MERCI!

QUESTIONS?

SÉANCE 7

INTRODUCTION AUX COMPILEURS



uOttawa

L'Université canadienne
Canada's university

SUJETS

Langages naturels

- Entités lexicales
- Syntax et sémantiques

Langages d'ordinateur

- Analyse lexique
- Analyse syntaxique
- Analyse sémantique

Compilateurs

- Exigences de base du compilateur
- Processus de compilation

LANGAGES NATURELS - BASE

Dans un langage (naturel):

- Une phrase est une séquence de mots
- Un mot (aussi appelé unité lexicale) est une séquence de caractères (possiblement un seul)

**L'ensemble de caractères utilisés dans un langage est fini
(connu sous le nom d'alphabet)**

L'ensemble de phrases possibles dans un langage est infini

Un lexique liste tous les mots (unités lexicales) d'un langage

- Les mots sont classifiés en différentes catégories lexicales:
verbe, nom, pronom, préposition....

LANGAGES NATURELS - BASE

La grammaire (aussi considérée l'ensemble de règles syntaxiques) est utilisée pour déterminer quelles séquences des mots sont bien formées

- Les séquences doivent avoir une structure qui obéit au règles de grammaire

Les phrases bien formées ont généralement un sens que les humains comprennent

On essaie d'apprendre aux machines notre langage naturel



Avec des résultats mixtes!!



uOttawa

L'Université canadienne
Canada's university

ANALYSE DES PHRASES

Analyse lexicale: identification de mots formés de caractères

- Les mots sont classifiés en plusieurs catégories: articles, noms, verbes, adjectifs, prépositions, pronoms...

Analyse syntaxique: règles pour combiner des mots afin de former des phrases

Analyse du sens: difficile à formaliser

- Facile pour les humains
- Difficile pour les machines (malgré l'évolution du traitement de langage naturel)
 - Grand domaine de recherche pour ceux intéressés dans les études supérieures...

TRAITEMENT DU LANGAGE D'ORDINATEUR

Dans les langages d'ordinateur (ou programmation), on parle d'un programme (correspondant à une phrase longue ou paragraphe)

- Séquence d'unités lexicales
- Les unités lexicales sont des séquences de caractères

Les règles lexicales d'un langage déterminent quelles sont les unités lexicales valides de ce langage

- Il existe plusieurs **catégories lexicales**: identificateur, nombre, chaînes de caractères, opérateur...
- Les catégories lexicales sont aussi nommées « **tokens** »

TRAITEMENT DU LANGAGE D'ORDINATEUR

Les règles syntaxiques d'un langage déterminent quelles séquences d'unités lexicales sont des programmes bien formés

Le sens d'un programme bien formé est aussi appelé sémantiques

- Un programme peut être bien formé, mais ses énoncés ne font pas de sens
- Exemple:

`int x = 0;`

`x = 1;`

`x = 0;`

- Syntaxiquement, le code ci-haut est valide, mais quel est son sens??

TRAITEMENT DU LANGAGE D'ORDINATEUR

Les compilateurs doivent attraper les erreurs lexicales et syntaxique

Les compilateurs peuvent se plaindre des erreurs sémantiques communes:

```
public boolean test (int x){  
    boolean result;  
    if (x > 100)  
        result = true;  
    return result;  
}
```



Message d'erreur:
The local variable result may
have not been initialized

Vos collègues ou bien le client se plaindront du reste!!



uOttawa

L'Université canadienne
Canada's university

COMPILEURS

Définition d'un compilateur

- Un programme qui traduit un programme exécutable dans un langage vers un programme exécutable dans un autre langage
- On prévoit que le programme produit par le compilateur soit meilleur que l'original

Définition d'un interpréteur

- Un programme qui lit un programme exécutable et produit les résultats qui proviennent lorsqu'on exécute ce programme

**On se concentre sur les compilateurs dans ce cours
(toutefois, beaucoup de concepts s'appliquent aux deux)**



uOttawa

L'Université canadienne
Canada's university

EXIGENCES DE BASE POUR LES COMPILEURS

Le compilateur doit faire:

- Produire le code correct (code byte dans le cas de Java)
- Exécuter rapidement
- Le code produit doit être optimale
- Achever un temps de compilation proportionnel à la taille du programme
- Bien travailler avec les débogueurs (*absolument nécessaire*)

Le compilateur doit avoir:

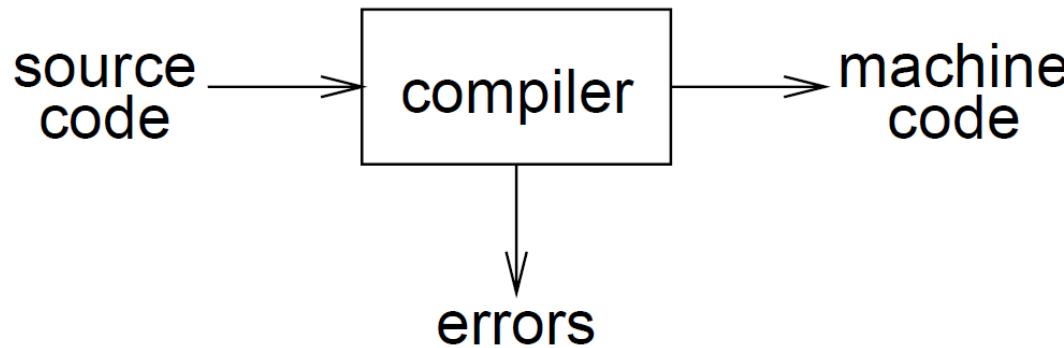
- Bonne diagnostique pour les erreurs lexicale et syntaxiques
- Support pour appel dans autres langages (*voir Java Native Interface si vous êtes intéressés*)



uOttawa

L'Université canadienne
Canada's university

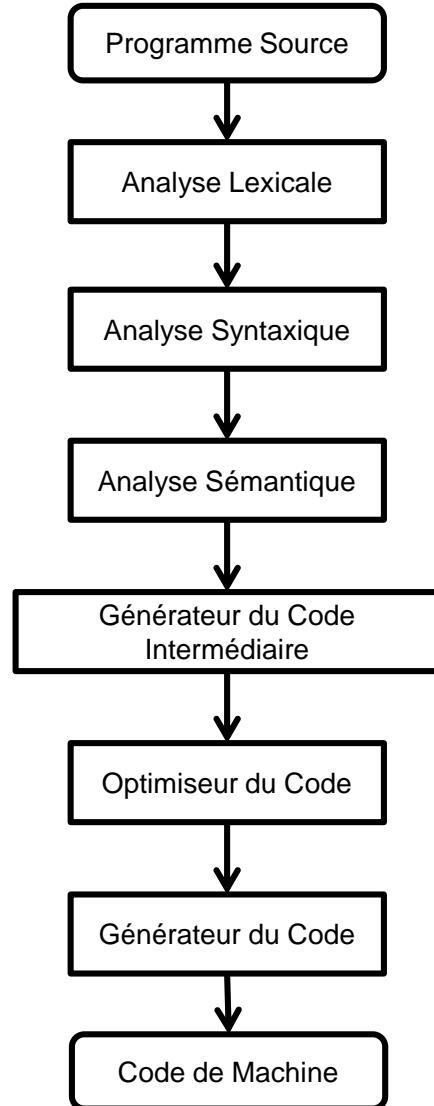
VUE ABSTRAITE DES COMPILATEURS



Un compilateur réalise une traduction en plusieurs étapes; par correspondance, il contient plusieurs composantes

Généralement, un compilateur inclut (au moins) des composantes séparées pour vérifier les règles lexicales et syntaxiques

PROCESSUS DE COMPILATION





uOttawa

L'Université canadienne
Canada's university

PROCESSUS DE COMPILATION

Un cours complet est requis afin de couvrir les détails des phases différentes

Dans ce cours, nous voyons seulement une introduction

- On se concentre sur l'analyse lexicale et syntaxique

QUELQUES DÉFINITIONS IMPORTANTES

Ces définitions, malgré qu'elles sont ennuyantes, sont importantes afin de comprendre les concepts qui seront introduits dans les prochaines séances

Commençons...



uOttawa

L'Université canadienne
Canada's university

ALPHABET

Souvenez-vous du début de la séance (ou de la maternelle):
I'alphabet est un ensemble de caractères qui sont utilisés pour former un mot



Puisque les mathématiciens adorent les symboles Grecs, on va référer à l'alphabet avec Σ



uOttawa

L'Université canadienne
Canada's university

ALPHABET

Σ est un alphabet, ou un ensemble de caractères

- Ensemble fini et consiste de tous les caractères d'entrée ou symboles qui peuvent être arrangés pour former des phrases dans un langage

Langage de programmation: généralement un ensemble d'ordinateur bien-défini comme le ASCII

CHAÎNES DANS UN ALPHABET

$$\Sigma = \{a, b, c, d\}$$

Chaînes possibles de Σ incluent:

- aaa
- aabbccdd
- d
- cba
- abab
- ccccccccccacccc
- Malgré que ceci est amusant, je pense que vous saisissez l'idée...



uOttawa

L'Université canadienne
Canada's university

LANGAGES FORMELS

Σ : alphabet, c'est un ensemble fini composé de tous les caractères d'entrée ou symboles

Σ^* : fermeture de l'alphabet, l'ensemble de toutes les chaînes possibles dans Σ , incluant la chaîne vide ϵ

Un langage (formel) est un sous-ensemble spécifique de Σ^*

MERCI!

QUESTIONS?

SÉANCE 8

ANALYSE LEXICALE



uOttawa

L'Université canadienne
Canada's university

SUJETS

Le rôle de l'analyseur lexical

Spécification des tokens (catégories lexicales)

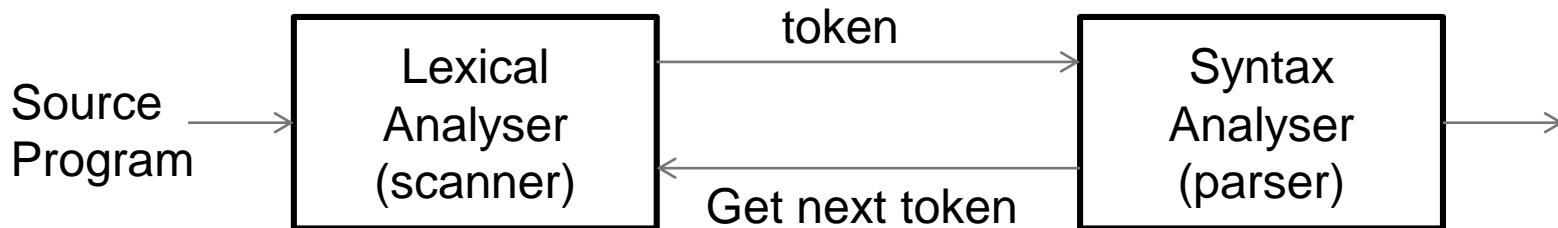
Expressions régulières (expression rationnelle)

Automates finis déterministes et non déterministes

LE RÔLE DE L'ANALYSEUR LEXICAL

L'analyse lexical est la première phase du compilateur

- Rôle: lit les caractères d'entrée et produit une séquence de catégories lexicales que l'analyseur syntaxique utilise pour l'analyse syntaxique
- Enlève les espaces blancs





uOttawa

L'Université canadienne
Canada's university

ANALYSE LEXICALE

Il y a plusieurs raisons pour lesquelles on sépare la phase d'analyse de compilation en analyse lexicale et l'analyse syntaxique:

- Modèle plus simple (en couches)
- Efficacité du compilateur

Des outils spécialisés ont été développés afin d'aider à automatiser la construction des deux séparément



uOttawa

L'Université canadienne
Canada's university

LEXÈMES

Lexème (entité lexicale): séquence de caractères dans le programme source qui est matché par le modèle du token

- Les lexèmes d'un langage de programmation incluent:
 - **Identificateurs**: noms de variables, méthodes, classes, package et interfaces...
 - **Littérales**: valeurs fixes (ex. “1”, “17.56”, “0xFFE” ...)
 - **Opérateurs**: pour opérations mathématique, Booléennes et logiques (ex. “+”, “-”, “&&”, “|” ...)
 - **Mots spéciaux**: mots-clés (ex. “if”, “for”, “public” ...)



uOttawa

L'Université canadienne
Canada's university

TOKENS, MODÈLE, LEXÈMES

Token: catégorie de lexèmes

Un **modèle** est une règle qui décrit l'ensemble de lexèmes qui appartiennent à un token (catégorie lexicale)

LEXÈMES ET TOKEN (CATÉGORIES PLUS DÉTAILLÉS)

```
Index = 2 * count +17;
```

Lexèmes	Tokens
Index	identificateur
=	signe_égal
2	littéral_entier
*	op_multi
Count	identificateur
+	op_plus
17	littéral_entier
;	point_virgule



uOttawa

L'Université canadienne
Canada's university

ERREURS LEXICALES

Peu d'erreurs sont perceptibles au niveau lexical

- L'analyseur lexical possède une vue localisée du programme source

On laisse les autres phases du compilateur s'occuper des erreurs

SPÉCIFICATION DES TOKENS

On a besoin d'une notation puissante afin de spécifier les modèles des tokens

- L'expression régulière au secours !!



Dans le processus d'étude des expressions régulières, on visite les sujets suivants:

- Opération sur les langages formelles
- Définitions régulières
- Raccourcis de notation (shorthand notations) pour les expressions régulières



uOttawa

L'Université canadienne
Canada's university

RAPPEL: LANGAGES

Σ : alphabet, c'est un ensemble fini qui consiste de tous les caractères d'entrée

Σ^* : fermeture de l'alphabet, l'ensemble de toutes les chaînes possibles dans Σ , incluant la chaîne vide ϵ

Un langage (formel) est un sous-ensemble spécifique de Σ^*

OPÉRATIONS SUR LES LANGAGES

Opération	Définition
Union de L et M	$L \cup M = \{s \mid s \in L \text{ ou } s \in M\}$
Concaténation de L et M	$LM = \{st \mid s \in L \text{ et } t \in M\}$
Fermeture de L	$L^* = \bigcup_{i=0}^{\infty} L^i$
Fermeture positive de L	$L^+ = \bigcup_{i=1}^{\infty} L^i$

OPÉRATIONS SUR LES LANGAGES

Format non-mathématique:

- **L'union entre les langages L et M:** l'ensemble des chaînes qui appartiennent à au moins un des deux langages
- **Concaténation des langages L et M:** l'ensemble de toutes les chaînes de forme **st** où **s** est une chaîne de **L** et **t** est une chaîne de **M**
- **Intersection entre les langages L et M:** l'ensemble de toutes les chaînes qui se trouvent dans les deux langages
- **Fermeture Kleene (d'après Stephen Kleene):** l'ensemble de toutes les chaînes qui sont des concaténations de **0 ou plus** de chaînes du langage original
- **Fermeture positive :** l'ensemble de toutes les chaînes qui sont des concaténations de **1 ou plus** de chaînes du langage original



uOttawa

L'Université canadienne
Canada's university

EXPRESSIONS RÉGULIÈRES

Une expression régulière est une notation compacte pour décrire une chaîne.

Typiquement, un identificateur est une lettre suivie d'un zéro ou plus de lettres ou de chiffres → lettre (lettre | chiffre)*

| : ou

*: zéro ou plus de



uOttawa

L'Université canadienne
Canada's university

RÈGLES

ε Est une expression régulière qui dénote $\{\varepsilon\}$, l'ensemble contenant une chaîne vide

Si a est un caractère dans Σ , donc a est une expression régulière qui dénote $\{a\}$, l'ensemble contenant la chaîne a

Supposons r et s sont des expressions régulières qui dénotent les langages L et M , donc

- $(r) |(s)$ est une expression régulière qui dénote $L \cup M$.
- $(r)(s)$ est une expression régulière qui dénote LM .
- $(r)^*$ est une expression régulière qui dénote L^* .



uOttawa

L'Université canadienne
Canada's university

CONVENTIONS DE PRÉCÉDENCE

L'opérateur unaire * possède la plus haute précédence et il est associative à gauche.

La concaténation possède la deuxième plus haute précédence et il est associative à gauche.

| possède la plus basse précédence et est associative à gauche.

$$(a)|(b)*(c) \rightarrow a|b*c$$



uOttawa

L'Université canadienne
Canada's university

PROPRIÉTÉS D'UNE EXPRESSION RÉGULIÈRE

Axiome	Description
$r s = s r$	est commutative
$r (s t) = (r s) t$	est associative
$(rs)t = r(st)$	Concaténation est associative
$r(s t) = rs rt$	Concaténation distribue sur
$(s t)r = sr tr$	
$\epsilon r = r$	ϵ ϵ est l'identité pour la concaténation
$r\epsilon = r$	
$r^* = (r \epsilon)^*$	Relation entre * et ϵ
$r^{**} = r^*$	* est idempotent



uOttawa

L'Université canadienne
Canada's university

EXEMPLES D'EXPRESSIONS RÉGULIÈRES

Si $\Sigma = \{a,b\}$

1. a dénote $\{a\}$
2. $a \mid b$ dénote $\{a,b\}$
3. $(a \mid b)$ ($a \mid b$) dénote $\{aa, ab, ba, bb\}$
4. a^* dénote $\{\varepsilon, a, aa, aaa, aaaa\dots\}$
5. $(a \mid b)^*$ dénote l'ensemble de toutes les chaînes de a et b (incluant ε)
6. $a \mid a^*b$ dénote $\{a, b, ab, aab, aaab, aaaab, \dots\}$



uOttawa

L'Université canadienne
Canada's university

DÉFINITIONS RÉGULIÈRES

Si Σ est un alphabet de symboles de base, alors une **définition régulière** est une séquence de définitions de forme:

$$d_1 \rightarrow r_1$$

$$d_2 \rightarrow r_2$$

...

$$d_n \rightarrow r_n$$

Où d_i est un nom distinct, et chaque r_i est une expression régulière à travers les symboles dans $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$,

- i.e., les symboles de base et les noms pré-définis.



EXEMPLE DE DÉFINITIONS RÉGULIÈRES

letter $\rightarrow (a \mid b \mid c \mid \dots \mid z \mid A \mid B \mid C \mid \dots \mid Z)$

digit $\rightarrow (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)$

id $\rightarrow \text{letter} (\text{ letter} \mid \text{digit})^*$

integer $\rightarrow (+ \mid - \mid \epsilon) (0 \mid (1 \mid 2 \mid 3 \mid \dots \mid 9) \text{ digit}^*)$

decimal $\rightarrow \text{integer} . (\text{ digit})^*$

real $\rightarrow (\text{ integer} \mid \text{ decimal}) \text{ E } (+ \mid -) \text{ digit}^*$



uOttawa

L'Université canadienne
Canada's university

RACCOURCIS

Certaines modèles sont très fréquemment exprimés par des expressions régulières qu'il est convenable d'introduire des notations abrégées pour eux

Nous avons déjà vu quelques-unes de ces notations:

- 1. Un ou plusieurs instances:** a^+ dénote l'ensemble de toutes les chaînes de un ou plusieurs a's
- 2. Zéro ou plusieurs instances:** a^* dénote toutes les chaînes de zéro ou plusieurs a's
- 3. Classes de caractères:** la notation $[abc]$ où a, b et c dénotent l'expression régulière $a \mid b \mid c$
- 4. Classes de caractères abréviés:** la notation $[a-z]$ dénote l'expression régulière $a \mid b \mid \dots \mid z$



uOttawa

L'Université canadienne
Canada's university

RACCOURCIS

En utilisant les classes de caractères, on peut décrire les identificateurs comme étant des chaînes décrites par l'expression régulière suivante:

[A-Za-z][A-Za-z0-9]*



uOttawa

L'Université canadienne
Canada's university

AUTOMATE FINIE

Maintenant que nous avons appris les expressions régulières

- Comment peut-on savoir si une chaîne (ou lexème) suit un modèle d'expression régulière ou non?

Nous allons de nouveau utiliser les machines d'état!

- Cette fois, elles ne sont pas des machines d'état UML ou réseaux de Petri
- Nous les appelons: Automate Finie

Le programme qui exécute cette machine d'état est appelé un *reconnaisseur (recognizer)*



uOttawa

L'Université canadienne
Canada's university

AUTOMATE FINI

Un *reconnaisseur* pour un langage est un programme qui prend comme entrée une chaîne x et répond

- “Oui” si x est un lexème du langage
- “Non” si au contraire

Nous compilons une expression régulière dans un *reconnaisseur* en construisant un diagramme appelé un *automate fini*

Un automate fini peut être *déterministe* ou *non-déterministe*

- Non-déterministe veut dire que plus d'une transition d'un état est possible sur le même symbole d'entrée

AUTOMATE FINI NON-DÉTERMINISTE (AFN)

Un ensemble d'états S

Un ensemble de symboles d'entrée qui appartiennent à l'alphabet Σ

Un ensemble de transitions qui sont déclenchées par le traitement d'un caractère

Un état simple s_0 qui est reconnu comme l'état de départ (*initial*)

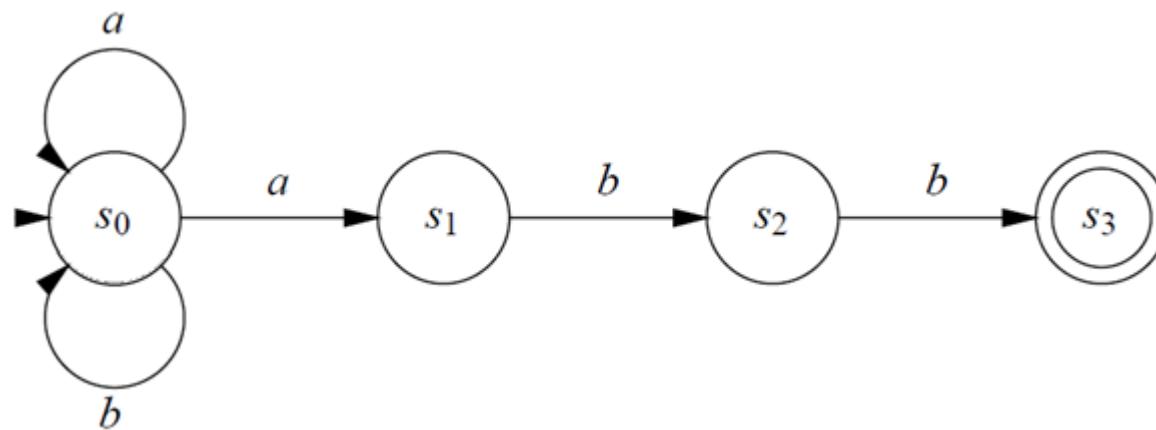
Un ensemble d'états F qui sont reconnus comme les états d'acceptation (*final*).

EXEMPLE D'UN AFN

L'expression régulière suivante

$$(a|b)^*abb$$

Peut être décrite en utilisant un AFN avec le diagramme suivant:



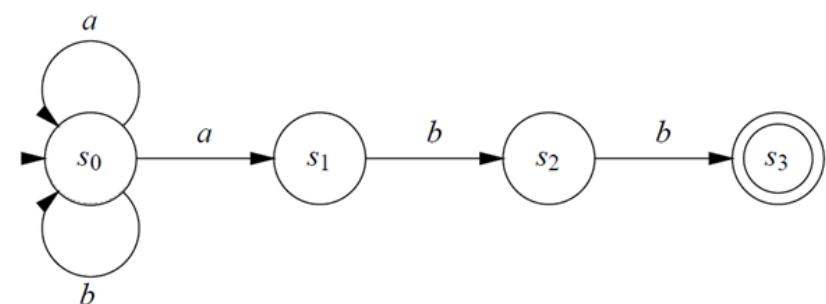
EXEMPLE D'UN AFN

Le diagramme précédent peut être décrit en utilisant le tableau suivant aussi

Souvenez-vous que l'expression régulière était:

$$(a|b)^*abb$$

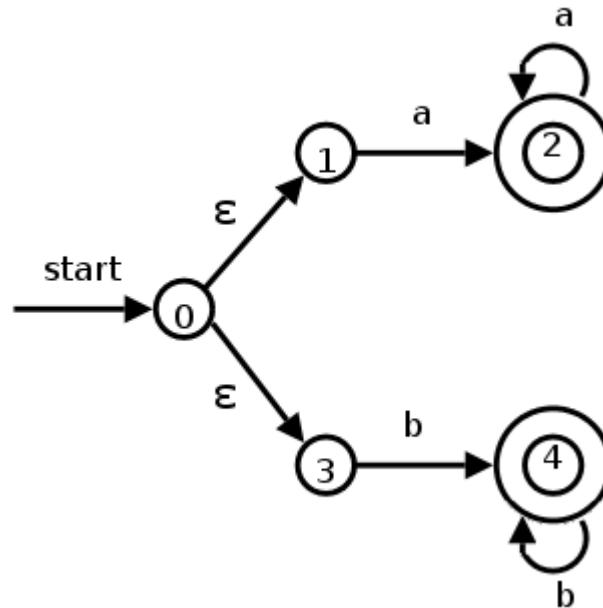
	a	b
s_0	$\{s_0, s_1\}$	$\{s_0\}$
s_1	—	$\{s_2\}$
s_2	—	$\{s_3\}$



UN AUTRE EXEMPLE DE AFN

Un AFN qui accepte l'expression régulière suivante:

$aa^*|bb^*$





uOttawa

L'Université canadienne
Canada's university

AUTOMATE FINI DÉTERMINISTE (AFD)

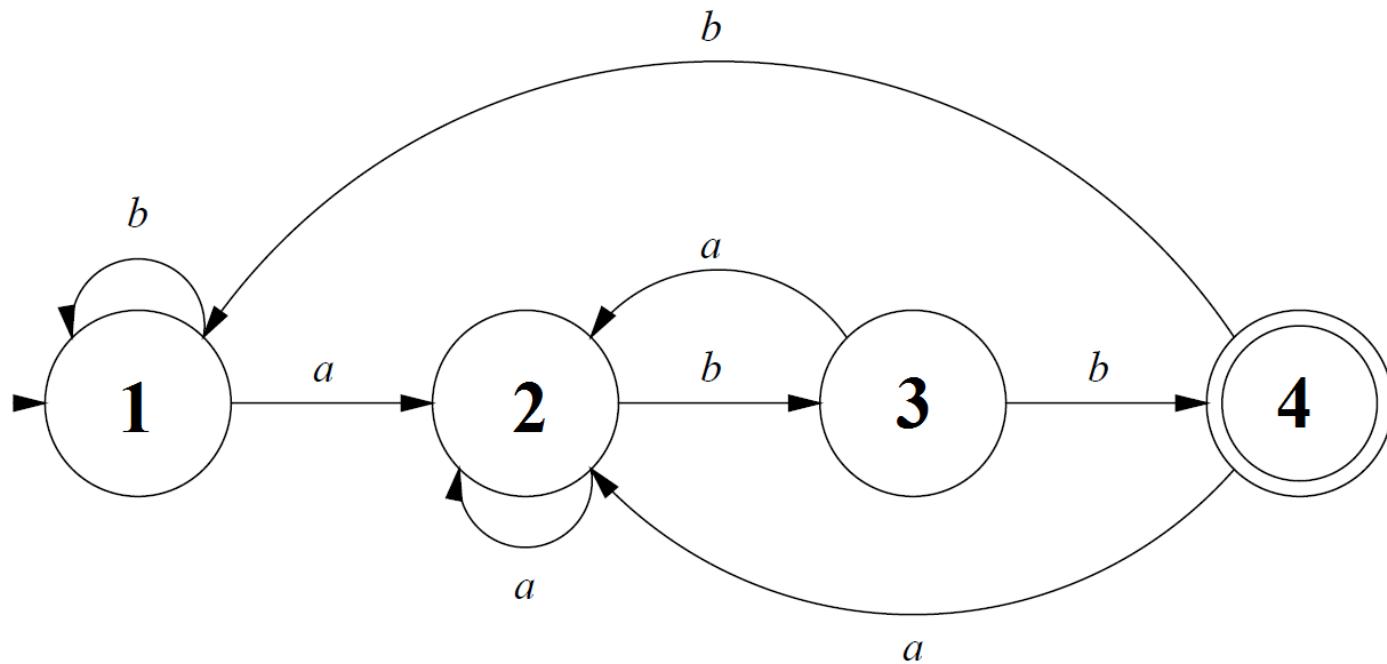
Un AFD est un cas spécial de AFN dans lequel

- Aucun état a une transition ϵ
- Pour chaque état s et symbole d'entrée a , il existe au plus une transition marqué a qui quitte s

UN AUTRE EXEMPLE DE AFD

Pour la même expression régulière vu précédemment

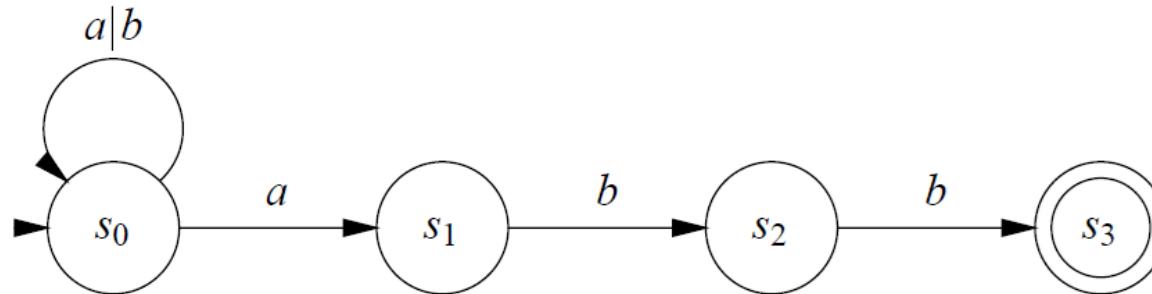
$$(a|b)^*abb$$



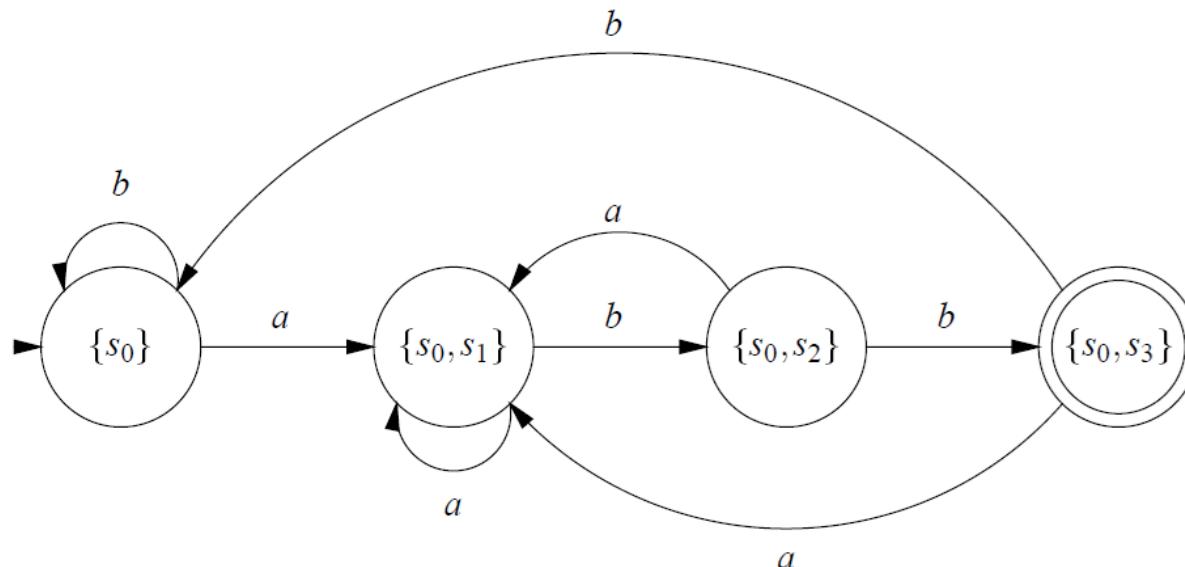
AFN VS AFD

Toujours avec l'expression régulière: $(a|b)^*abb$

AFN:

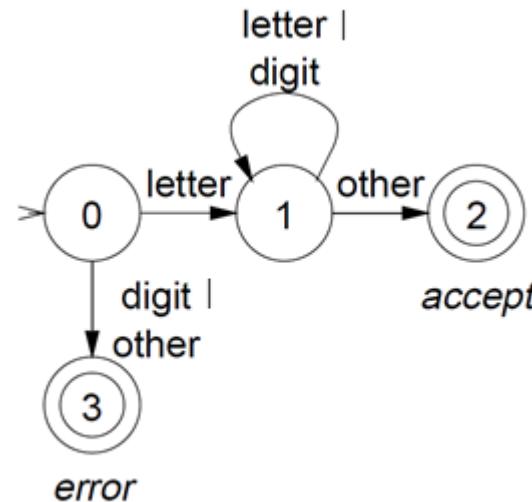


AFD:



EXEMPLE DE AFD

Reconnaisseur pour un identificateur:



identifier

letter $\rightarrow (a \mid b \mid c \mid \dots \mid z \mid A \mid B \mid C \mid \dots \mid Z)$

digit $\rightarrow (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)$

id $\rightarrow \text{letter} (\text{letter} \mid \text{digit})^*$



uOttawa

L'Université canadienne
Canada's university

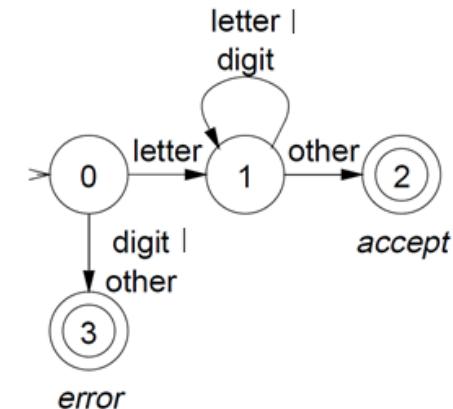
TABLEAUX POUR LE RECONNAISSEUR

char_class:

	<i>a – z</i>	<i>A – Z</i>	<i>0 – 9</i>	other
value	letter	letter	digit	other

next_state:

class	0	1	2	3
letter	1	1	—	—
digit	3	1	—	—
other	3	2	—	—



Pour changer l'expression régulière, nous pouvons simplement changer de tableaux...

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

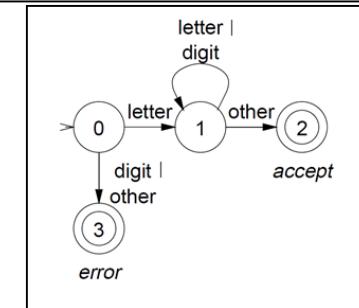
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d
state←0
done←false
token_value←""

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

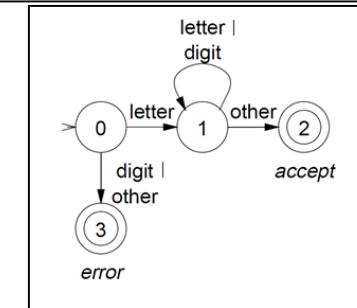
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d
state←0
done←false
token_value←""
class←letter

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

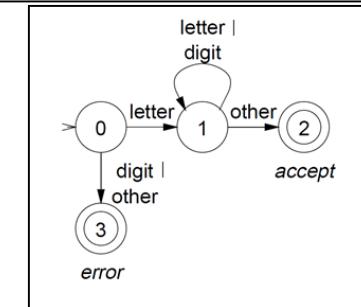
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d
state←0 1
done←false
token_value←""
class←letter

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

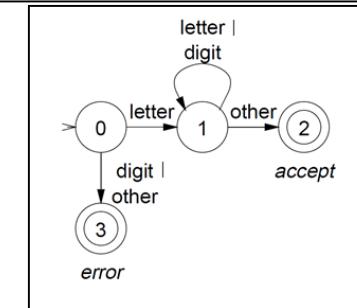
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d
state←0 1
done←false
token_value←"d"
class←letter

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

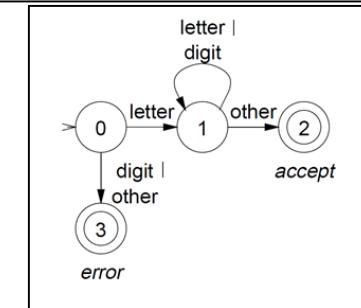
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d 8
state←0 1
done←false
token_value←"d"
class←letter

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

Algorithme du reconnaiseur:

```

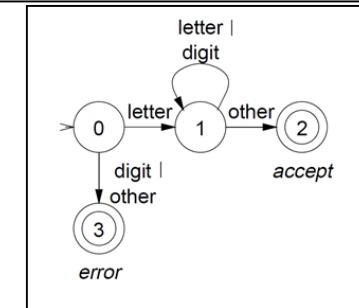
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;

```

Référence:

char_class:		a – z	A – Z	0 – 9	other
	value	letter	letter	digit	other

next_state:		class	0	1	2	3
	letter	1	1	—	—	
	digit	3	1	—	—	
	other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d 8
state←0 1
done←false
token_value←"d"
class←letter digit

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

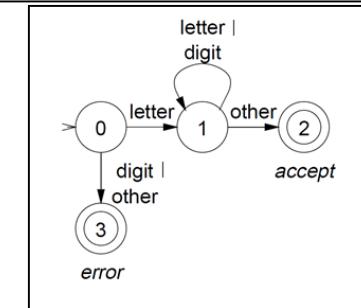
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

char←d 8
state←0 4 1
done←false
token_value←"d"
class←letter digit

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

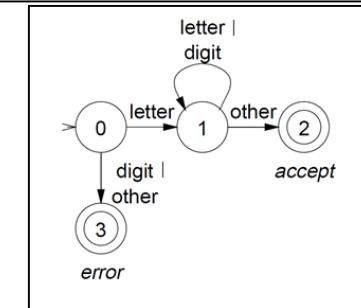
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8**;

char←d 8
state←0 + 1
done←false
token_value←"d8"
class←letter digit

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

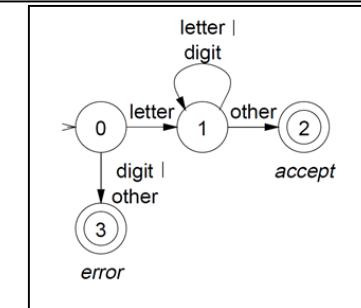
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a – z	A – Z	0 – 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8**;

char←d 8 ;
state←0 + 1
done←false
token_value←"d8"
class←letter digit

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

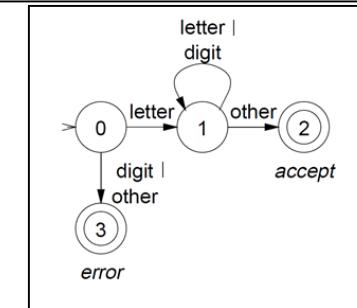
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:		a - z	A - Z	0 - 9	other
value	letter	letter	digit	other	

next_state:	class	0	1	2	3
letter	1	1	—	—	
digit	3	1	—	—	
other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8**;

char←d 8 ;
state←0 + 1
done←false
token_value←"d8"
class←letter digit other

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

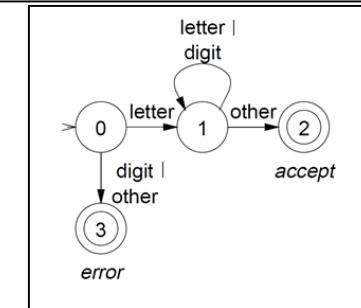
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:	a - z	A - Z	0 - 9	other
	value	letter	letter	digit

next_state:	class	0	1	2	3
	letter	1	1	—	—
letter	digit	3	1	—	—
	other	3	2	—	—



Exemple:

Pour la séquence de caractères: **d8**;

char←d 8 ;
state←0 1 1 2
done←false
token_value←"d8"
class←letter digit other

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

Algorithme du reconnaiseur:

```

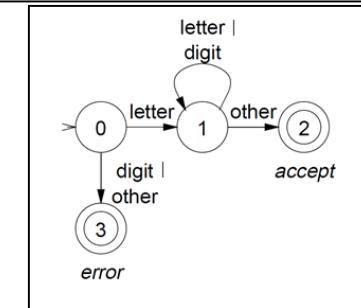
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;

```

Référence:

char_class:		<i>a – z</i>	<i>A – Z</i>	<i>0 – 9</i>	other
	value	letter	letter	digit	other

next_state:		class	0	1	2	3
	letter	1	1	—	—	
	digit	3	1	—	—	
	other	3	2	—	—	



Exemple:

Pour la séquence de caractères: **d8;**

```

char←d 8 ;
state←0 1 1 2
done←false
token_value←"d8"
class←letter digit other

```

token_type←identifier

CODE POUR LE RECONNAISSEUR



uOttawa

L'Université canadienne
Canada's university

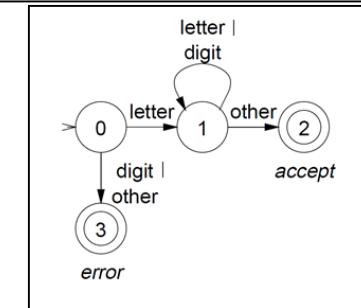
Algorithme du reconnaiseur:

```
char ← next_char();
state ← 0;          /* code for state 0 */
done ← false;
token_value ← ""    /* empty string */
while( not done ) {
    class ← char_class[char];
    state ← next_state[class,state];
    switch(state) {
        case 1:      /* building an id */
            token_value ← token_value + char;
            char ← next_char();
            break;
        case 2:      /* accept state */
            token_type = identifier;
            done = true;
            break;
        case 3:      /* error */
            token_type = error;
            done = true;
            break;
    }
}
return token_type;
```

Référence:

char_class:	a - z	A - Z	0 - 9	other
	value	letter	letter	digit

next_state:	class	0	1	2	3
	letter	1	1	—	—
digit		3	1	—	—
other		3	2	—	—



Exemple:

Pour la séquence de caractères: **d8;**

char←d 8 ;
state←0 1 1 2
done←false true
token_value←"d8"
class←letter digit other

token_type←identifier

MERCI!

QUESTIONS?

SÉANCE 9

AUTOMATE À ÉTAT FINI



uOttawa

L'Université canadienne
Canada's university

SUJETS

Algorithme pour créer des AFNs à partir d'expressions régulières

Algorithme pour convertir les AFNs au AFDs

Algorithme pour minimiser les AFDs

Plusieurs exemples....



uOttawa

L'Université canadienne
Canada's university

CRÉER DES AUTOMATES FINIS DÉTERMINISTES (AFD)

Afin de créer un AFD, on doit effectuer le suivant:

- Créer un automate fini non-déterministe (AFN) à partir d'une expression régulière
- Convertir le AFN en AFD

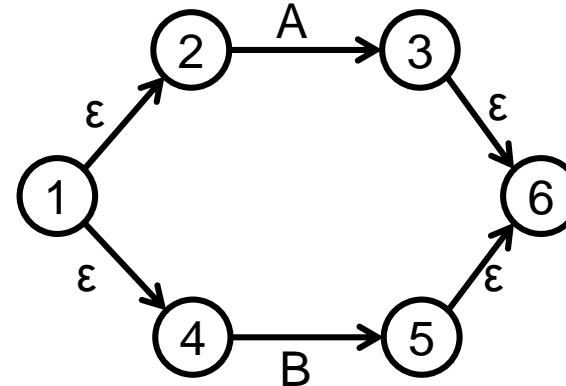
RÈGLES POUR LA CRÉATION DE AFN



uOttawa

L'Université canadienne
Canada's university

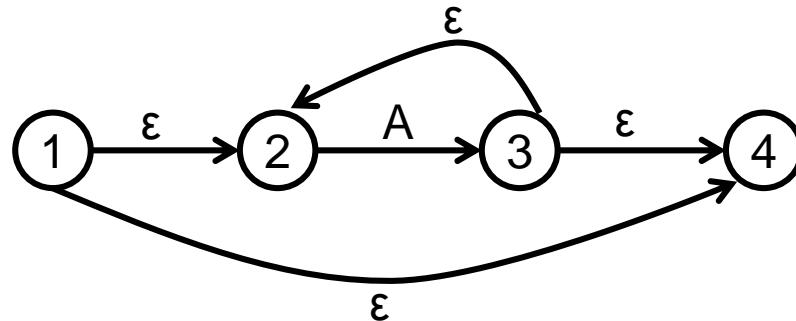
A | B



AB



A^*



EXEMPLES DE CRÉATION DE AFN

x | yz

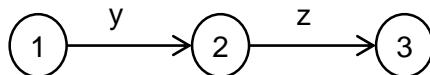
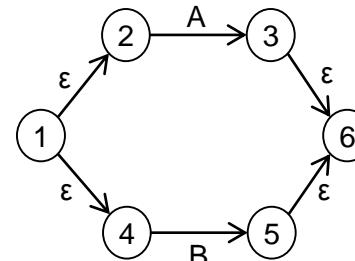
Selon les règles de précédence, ceci est équivalent à:

x | (yz)

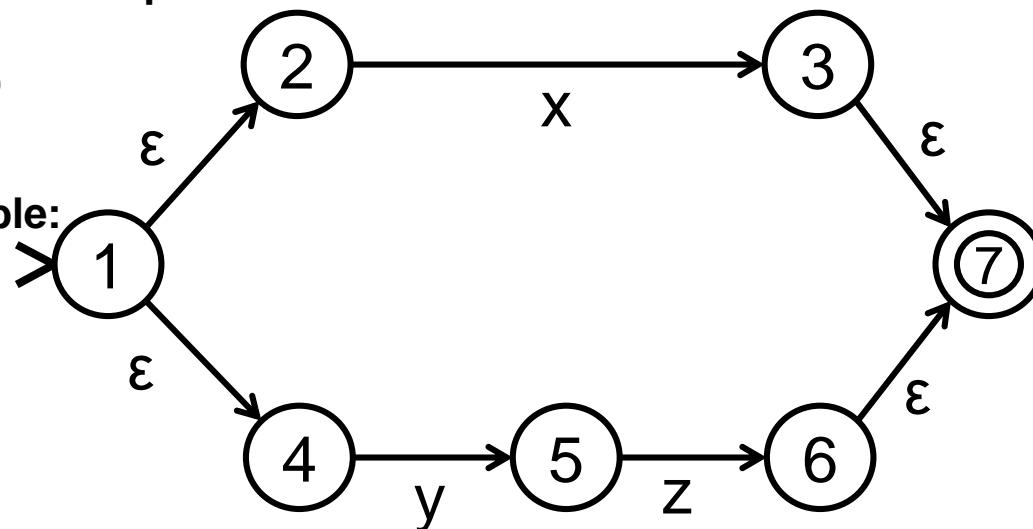
Ceci possède la même forme que

A | B:

Et B peut être représenté tel que:



Mettant le tout ensemble:



EXEMPLES DE CRÉATION DE AFN

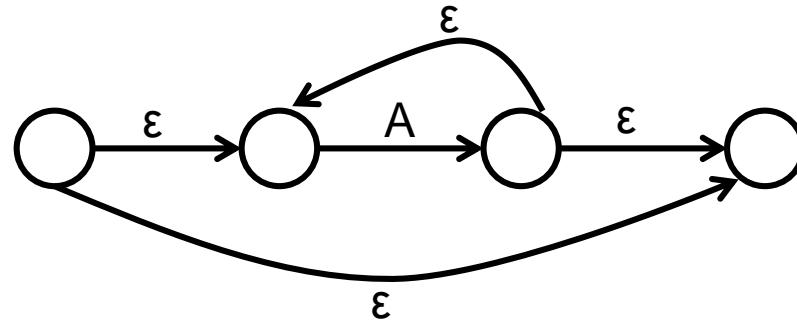


uOttawa

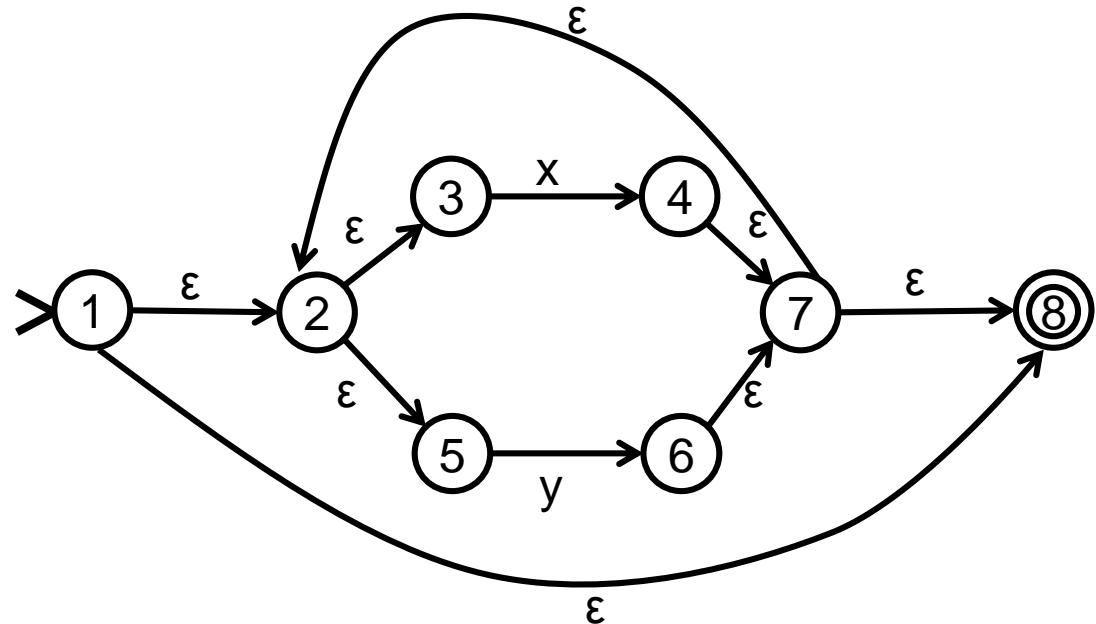
L'Université canadienne
Canada's university

$(x \mid y)^*$

Nous avons vu A^* :



Donc, $(x \mid y)^*$:



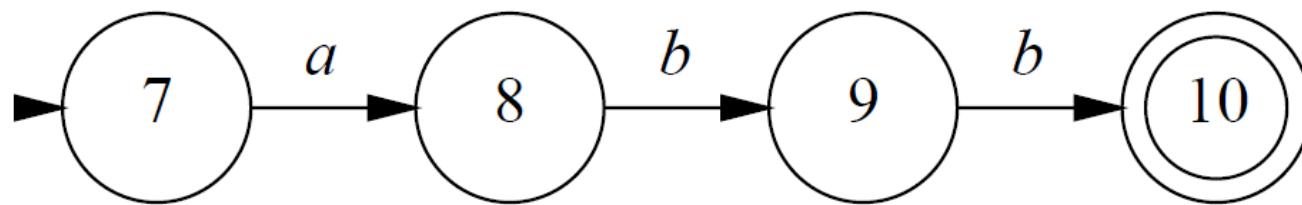
EXEMPLES DE CRÉATION DE AFN



uOttawa

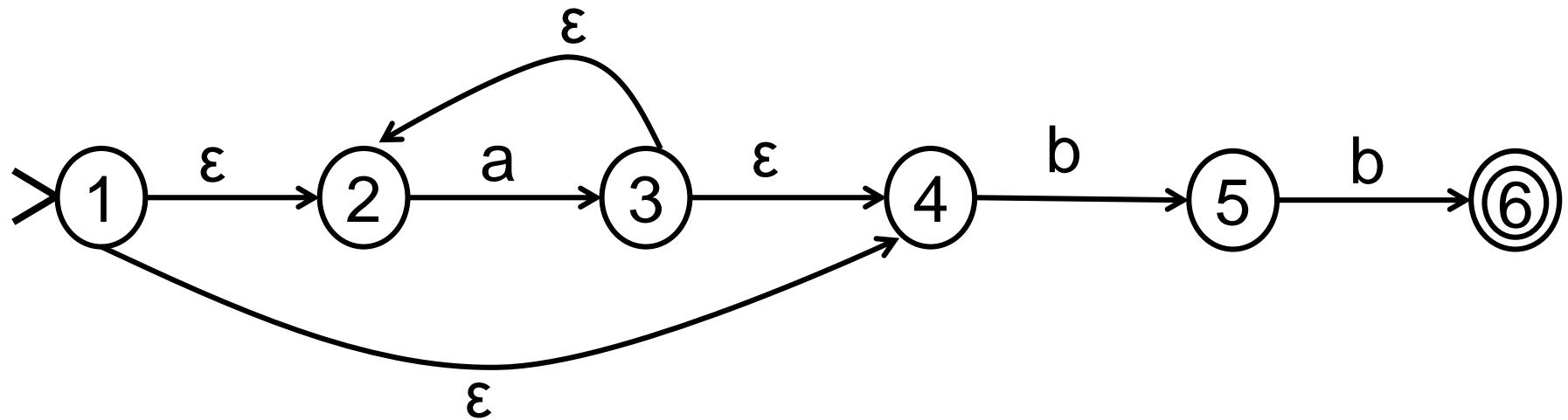
L'Université canadienne
Canada's university

abb



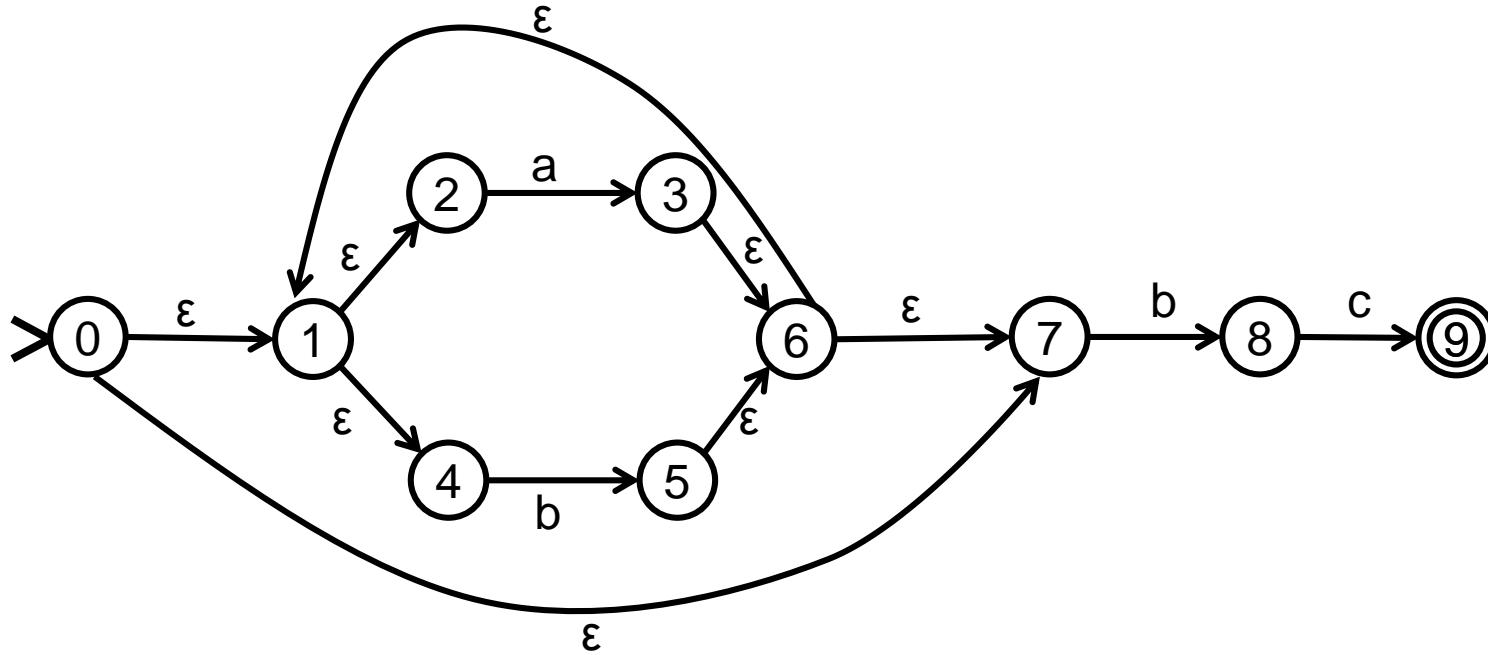
EXEMPLES DE CRÉATION DE AFN

a^*bb



EXEMPLES DE CRÉATION DE AFN

$(a|b)^*bc$





uOttawa

L'Université canadienne
Canada's university

CONVERSION D'UN AFN EN UN AFD

AFN est facile à construire mais difficile à interpréter par un ordinateur

- On a besoin de convertir un AFN à un AFD
- *L'algorithme de construction de sous-ensembles* réalise cette conversion

Dans le tableau de transition d'un AFN, chaque entrée est un ensemble d'états

Dans le tableau de transition d'un AFD, chaque entrée est un seul état.

**Idée générale concernant la conversion d'AFN en AFD:
chaque état AFD correspond à un ensemble d'états AFN**

ALGORITHME DE CONSTRUCTION DE SOUS-ENSEMBLES

Algorithme: Construction de sous-ensembles –
Utilisé pour construire un AFD à partir d'un AFN

Entrée: Un AFN “ N ”

Sortie: Un AFD “ D ” qui accepte le même langage

ALGORITHME DE CONSTRUCTION DE SOUS-ENSEMBLES

Méthode:

- Soit s un état dans “ N ” et “ T ” un ensemble d’états, et utilisons les opérations suivantes:

Operation	Definition
ϵ -closure(s)	Ensembles des états dans N qui peuvent être atteints de l’ état s avec des transitions ϵ
ϵ -closure(T)	Ensembles des états dans N qui peuvent être atteints de l’ensemble des états T avec des transitions ϵ
move(T, a)	Ensembles des états dans N qui peuvent être atteints de l’ensemble des états T avec une seule transition a

CONSTRUCTION DE SOUS-ENSEMBLES *(ALGORITHME PRINCIPAL)*

add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $D\text{states}$

while \exists unmarked state T in $D\text{states}$

 mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin D\text{states}$ **then** add U to $D\text{states}$ unmarked

$D\text{trans}[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

CONSTRUCTION DE SOUS-ENSEMBLES

(COMPUTATION DE ϵ -CLOSURE)

push all states in T onto stack

initialize ϵ -closure(T) to T

while stack is not empty

 pop t , the top element off the stack

for each state u with an edge from t to u labeled ϵ

if u is not in ϵ -closure(T)

 add u to ϵ -closure(T)

 push u onto stack

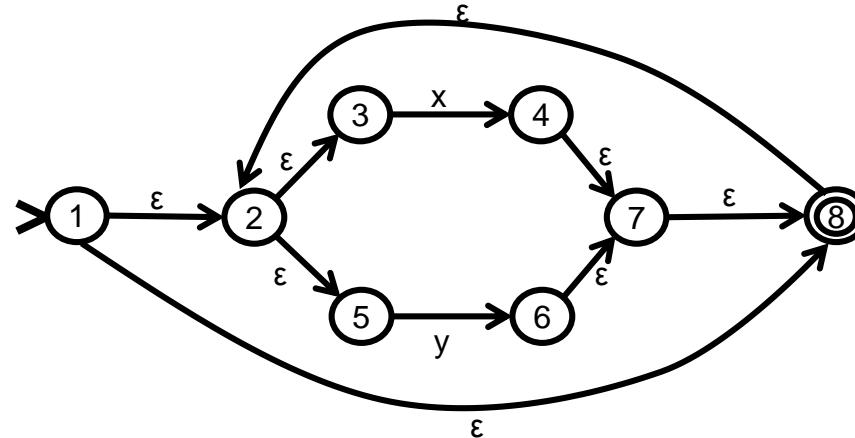
endif

endfor

endwhile

EXEMPLE DE CONVERSION

Expression Régulière:
 $(x \mid y)^*$



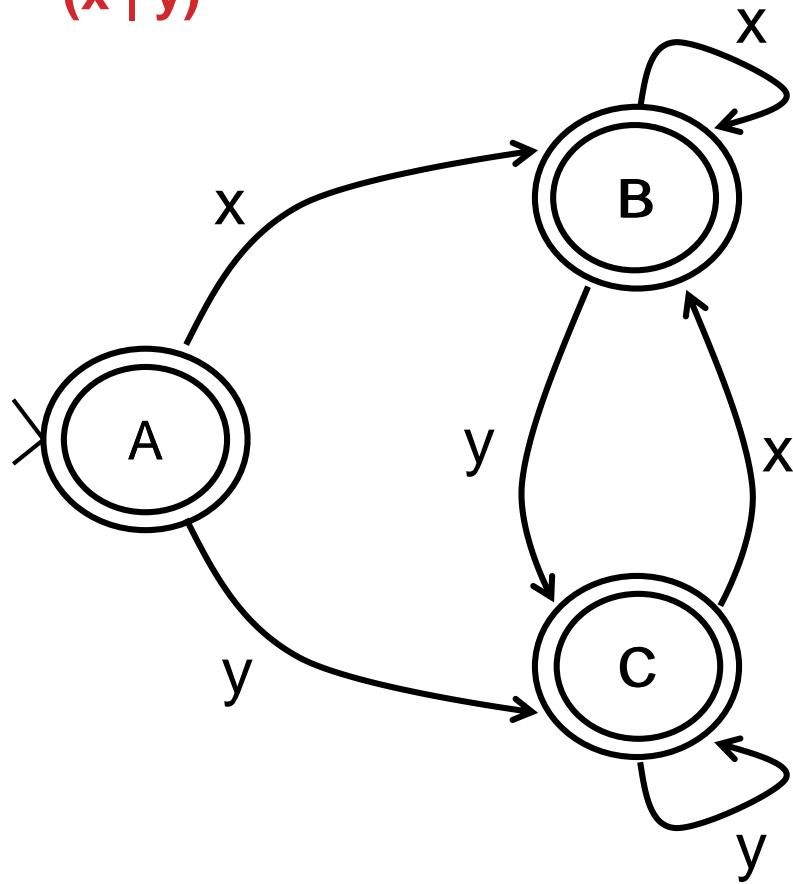
États $D = \{A, B, C\}$, où

- $A = (1, 2, 3, 5, 8)$
- $B = (2, 3, 4, 5, 7, 8)$
- $C = (2, 3, 5, 6, 7, 8)$

	x	y
A	B	C
B	B	C
C	B	C

EXEMPLE DE CONVERSION

Expression Régulièrē:
 $(x \mid y)^*$

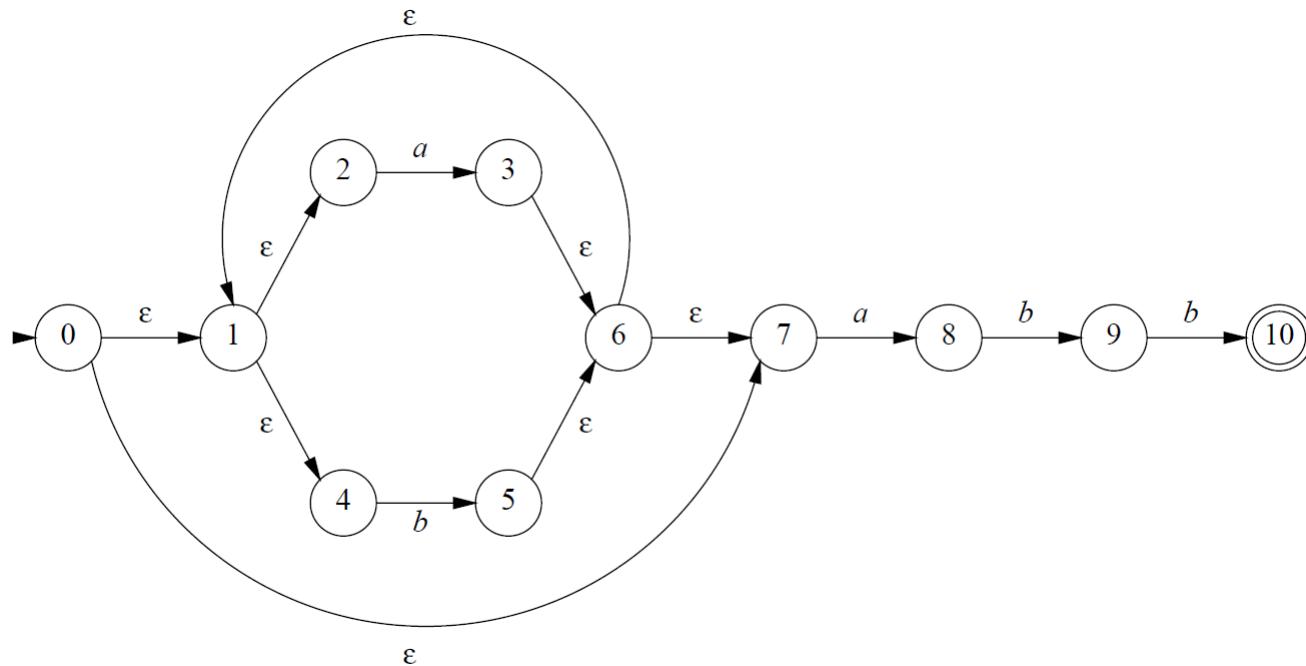


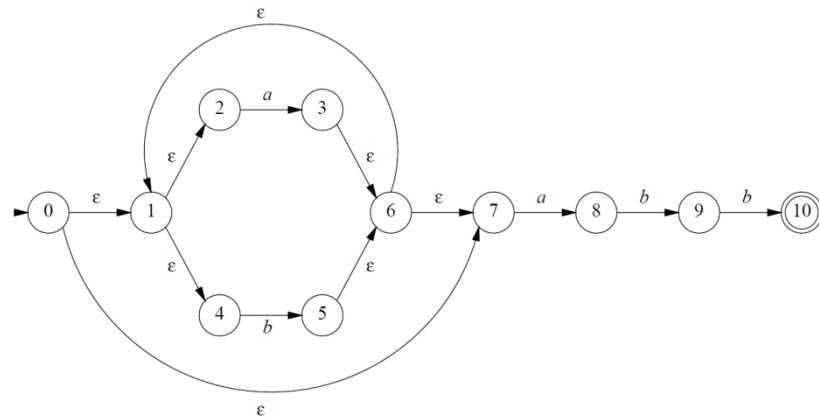
	x	y
A	B	C
B	B	C
C	B	C

AUTRE EXEMPLE DE CONVERSION

Expression Régulière:

$(a \mid b)^*abb$

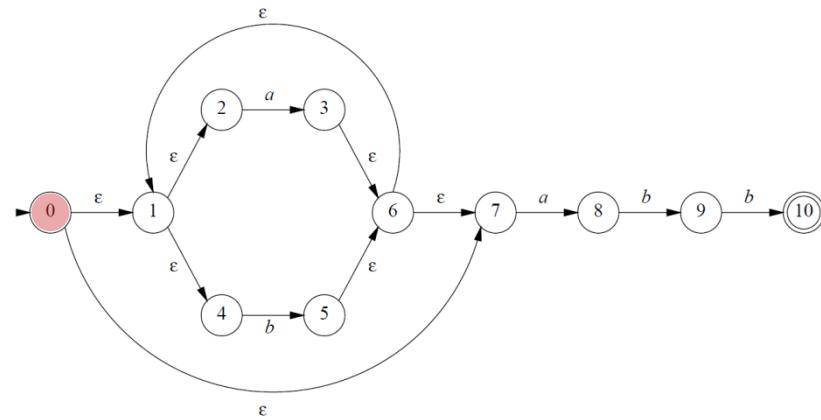




add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
for each input symbol a
 $U = \varepsilon\text{-closure}(\text{move}(T, a))$
if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
endfor
endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

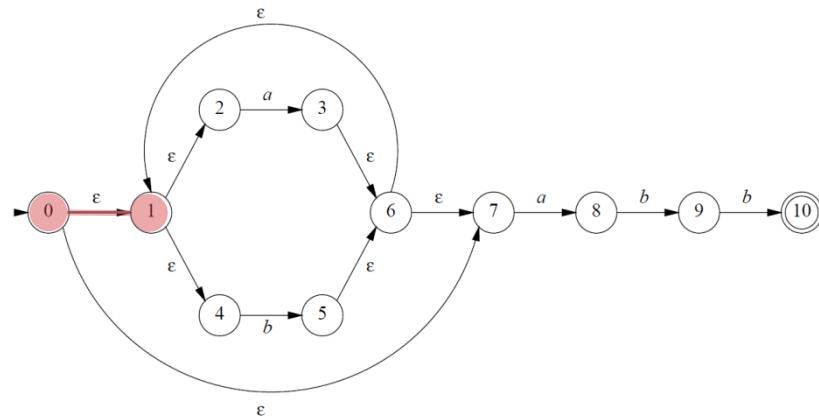
$T = \varepsilon\text{-closure}(0)$



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
for each input symbol a
 $U = \varepsilon\text{-closure}(\text{move}(T, a))$
if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
endfor
endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

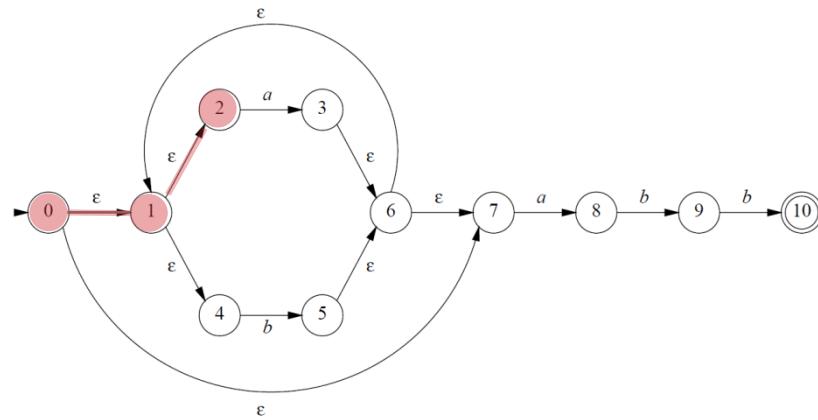
$T = \varepsilon\text{-closure}(0)$
 $= \{0,$



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
 for each input symbol a
 $U = \varepsilon\text{-closure}(\text{move}(T, a))$
 if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
 endfor
endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

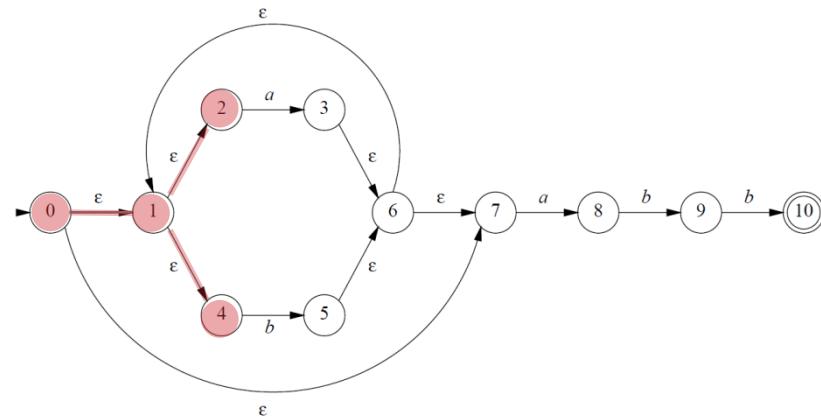
$$\begin{aligned}
 T &= \varepsilon\text{-closure}(0) \\
 &= \{0, 1,
 \end{aligned}$$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
for each input symbol a
 $U = \epsilon\text{-closure}(\text{move}(T, a))$
if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
endfor
endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

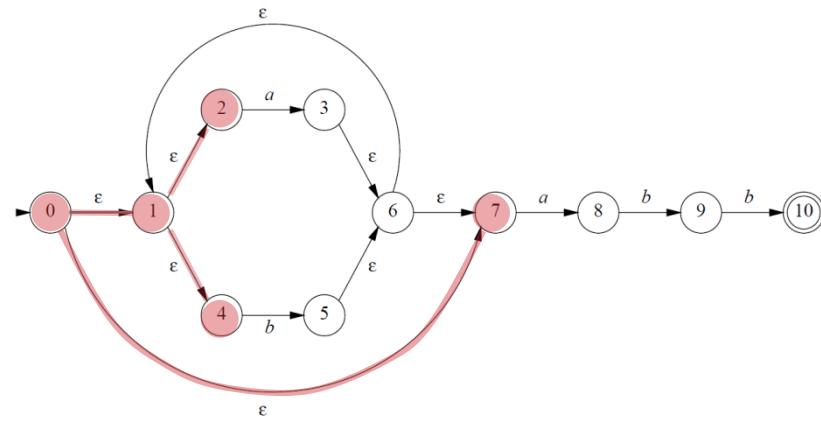
$$\begin{aligned}
 T &= \epsilon\text{-closure}(0) \\
 &= \{0, 1, 2,
 \end{aligned}$$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
for each input symbol a
 $U = \epsilon\text{-closure}(\text{move}(T, a))$
if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
endfor
endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

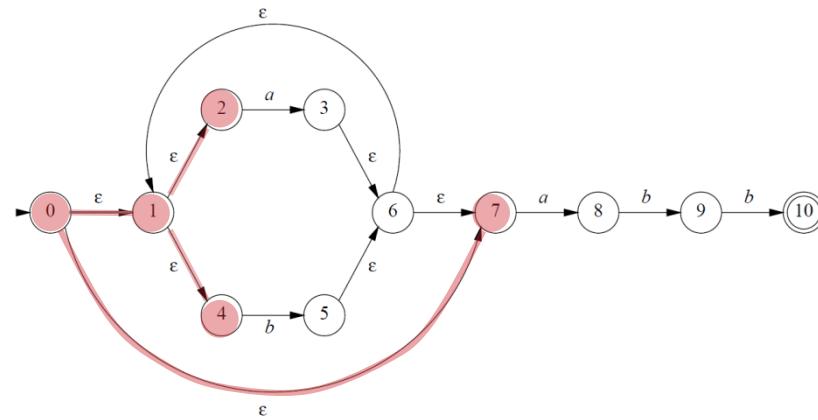
$$\begin{aligned}
 T &= \epsilon\text{-closure}(0) \\
 &= \{0, 1, 2, 4,
 \end{aligned}$$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
for each input symbol a
 $U = \epsilon\text{-closure}(\text{move}(T, a))$
if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
endfor
endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned}
 T &= \epsilon\text{-closure}(0) \\
 &= \{0, 1, 2, 4, 7\}
 \end{aligned}$$



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
 for each input symbol a
 $U = \varepsilon\text{-closure}(\text{move}(T, a))$
 if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
 endfor
endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

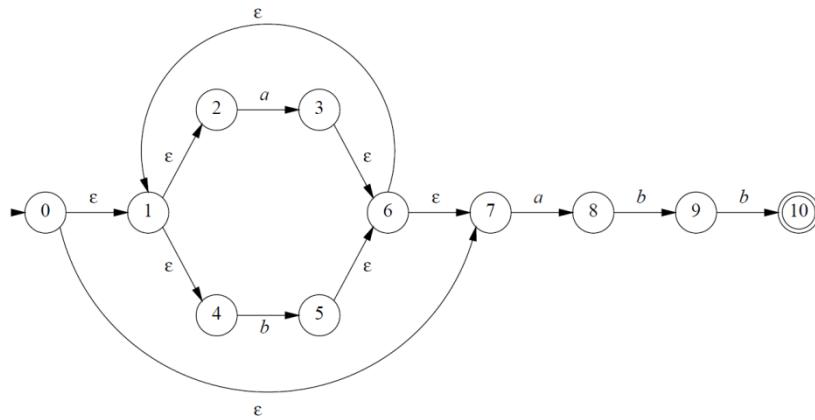
$$\begin{aligned}
 T &= \varepsilon\text{-closure}(0) \\
 &= \{0, 1, 2, 4, 7\} = A
 \end{aligned}$$

Dstates:
A \square

Cheat Sheet

A = {0,1,2,4,7}





add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$Dstates:$

A □

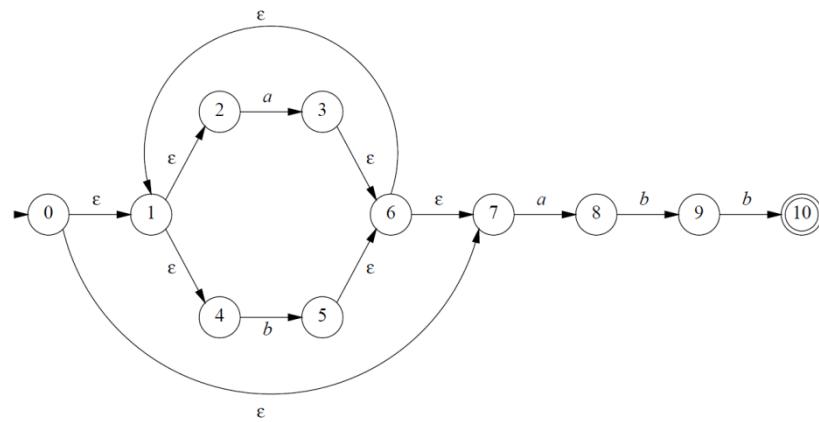
Cheat Sheet

A = {0, 1, 2, 4, 7}

⋮

⋮

⋮



```

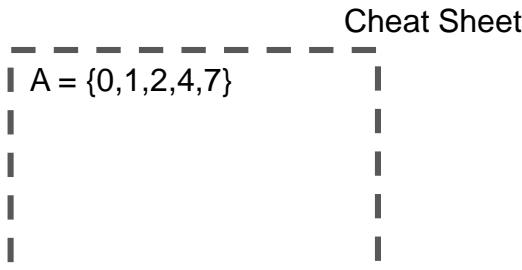
add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

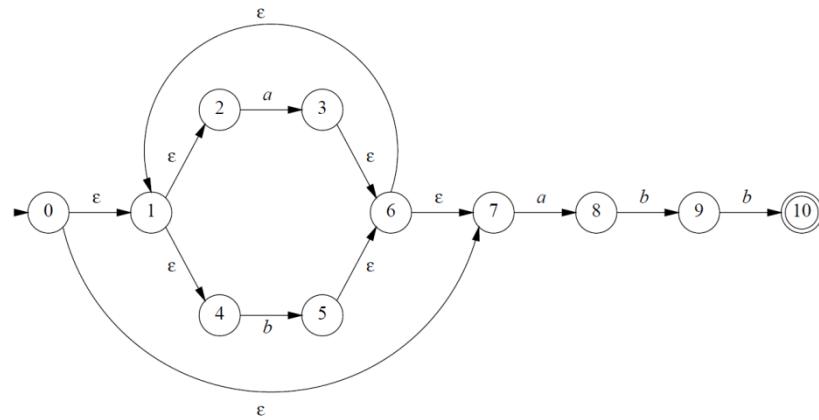
```

$\epsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

Dstates:
 A





```

add state  $T = \varepsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \varepsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\varepsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \varepsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

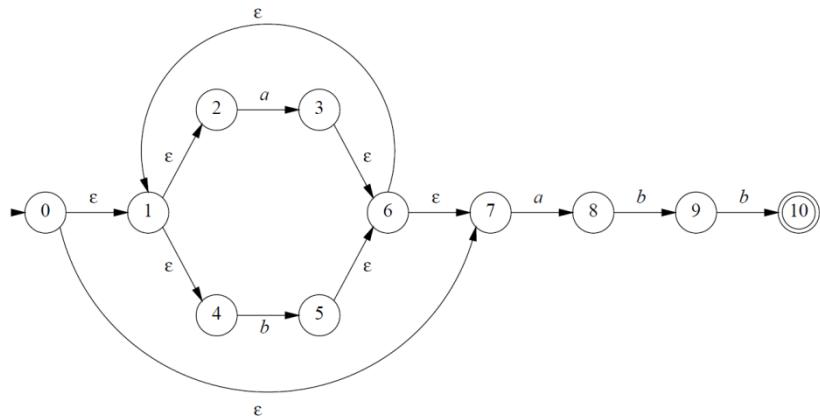
Dstates:

A

Cheat Sheet

A = {0, 1, 2, 4, 7}





```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

 $\epsilon\text{-closure}(s_0)$  is the start state of  $D$ 
A state of  $D$  is accepting if it contains at least one accepting state in  $N$ 

```

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

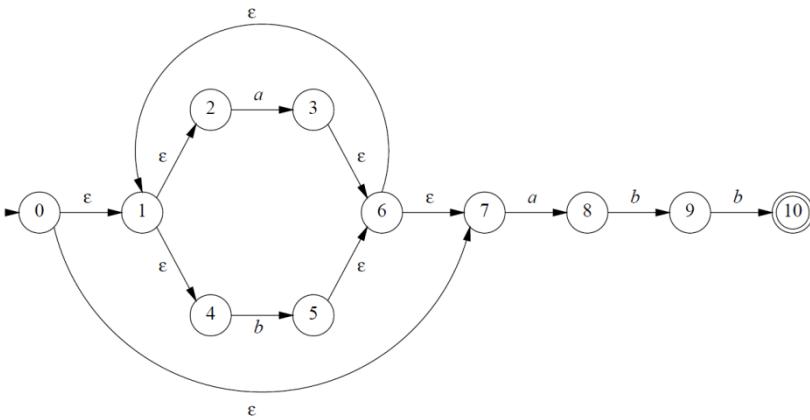
$$U = \epsilon\text{-closure}(\text{move}(T, a))$$

Dstates:

A

Cheat Sheet

A = {0, 1, 2, 4, 7}



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

Dstates:

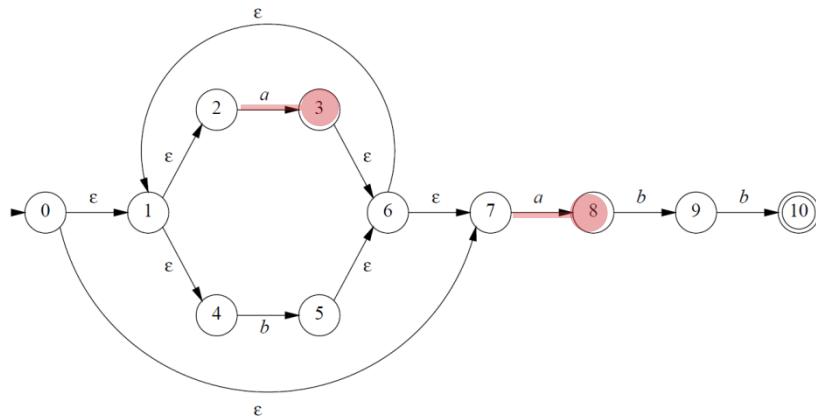
A

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a)) \end{aligned}$$

Cheat Sheet

A = {0, 1, 2, 4, 7}





add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$
while \exists unmarked state T in $Dstates$
 mark T
 for each input symbol a
 $U = \epsilon\text{-closure}(\text{move}(T, a))$
 if $U \notin Dstates$ **then** add U to $Dstates$ unmarked
 $Dtrans[T, a] = U$
 endfor
endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D
 A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned}
 T &= \epsilon\text{-closure}(0) \\
 &= \{0, 1, 2, 4, 7\} = A
 \end{aligned}$$

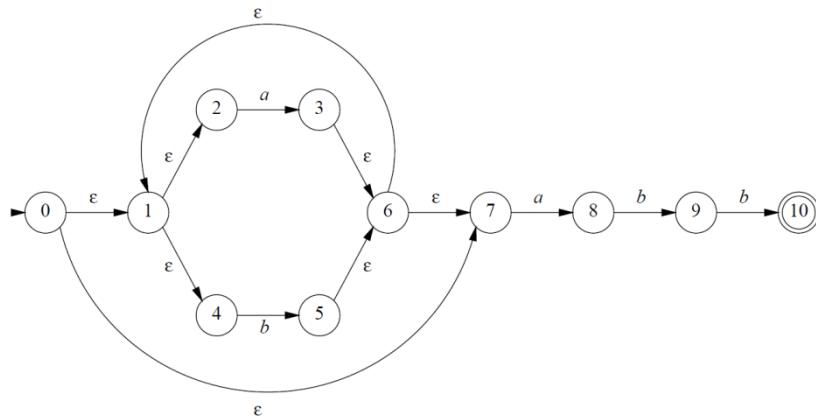
Dstates:
 A

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, a)) \\
 &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a))
 \end{aligned}$$

Cheat Sheet

A = {0, 1, 2, 4, 7}





add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \end{aligned}$$

Dstates:

A

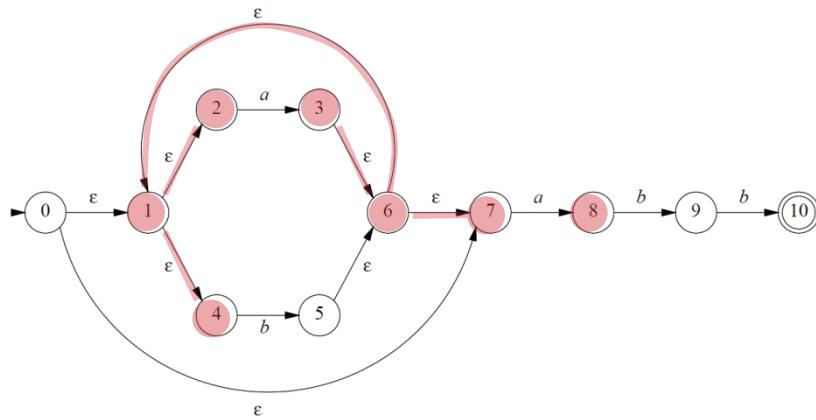
Cheat Sheet

I A = {0, 1, 2, 4, 7} I

I I

I I

I I



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \\ &= \{1, 2, 3, 4, 6, 7, 8\} \end{aligned}$$

Dstates:

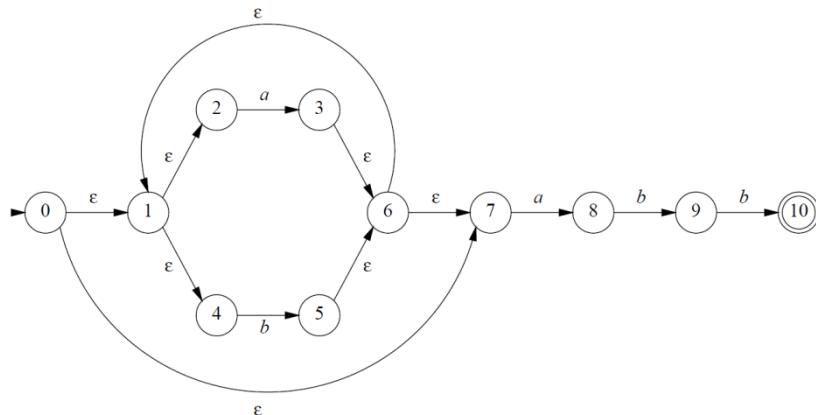
A

Cheat Sheet

A = {0, 1, 2, 4, 7}

⋮

⋮



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \\ &= \{1, 2, 3, 4, 6, 7, 8\} = B \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

Cheat Sheet

| A = {0, 1, 2, 4, 7}

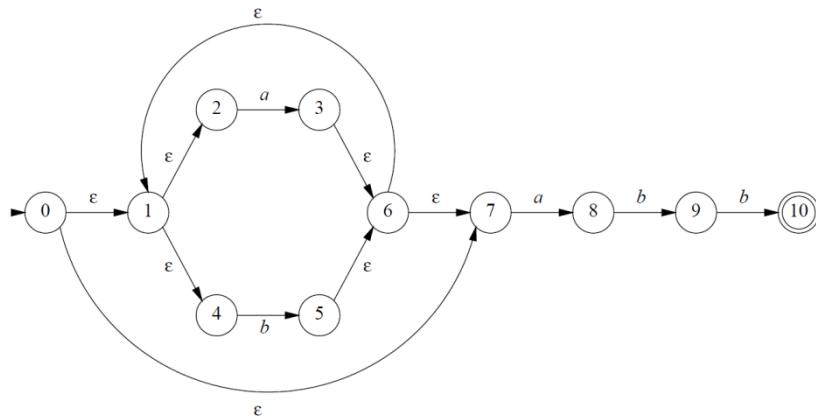
| B = {1, 2, 3, 4, 6, 7, 8}

|

|

|

|



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \varepsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \varepsilon\text{-closure}(\text{move}(T, a)) \\ &= \varepsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, a)) \\ &= \varepsilon\text{-closure}(\{3, 8\}) \\ &= \{1, 2, 3, 4, 6, 7, 8\} = B \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

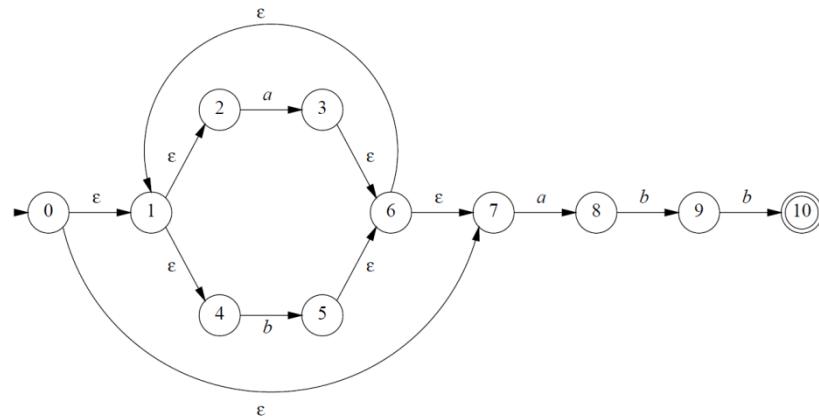
	a	b
A	B	

Cheat Sheet

| A = {0, 1, 2, 4, 7}

| B = {1, 2, 3, 4, 6, 7, 8}

| $\varepsilon\text{-closure}(\{3, 8\}) = B$



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \varepsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

Dstates:

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input type="checkbox"/> |

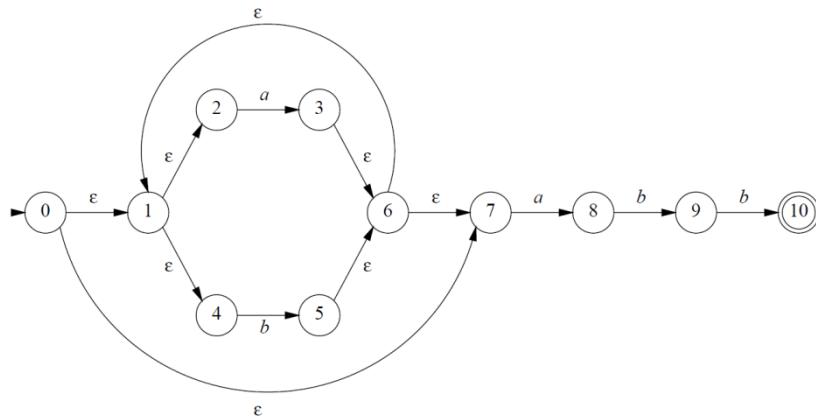
Cheat Sheet

| A = {0,1,2,4,7}

| B = {1,2,3,4,6,7,8}

| $\varepsilon\text{-closure}(\{3,8\}) = B$

	a	b
A	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, b)) \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

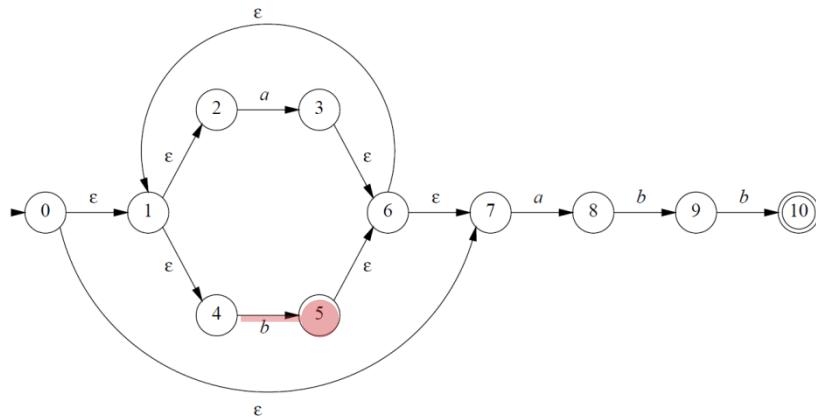
Cheat Sheet

$$A = \{0, 1, 2, 4, 7\}$$

$$B = \{1, 2, 3, 4, 6, 7, 8\}$$

$$\epsilon\text{-closure}(\{3, 8\}) = B$$

	a	b
A	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, b)) \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

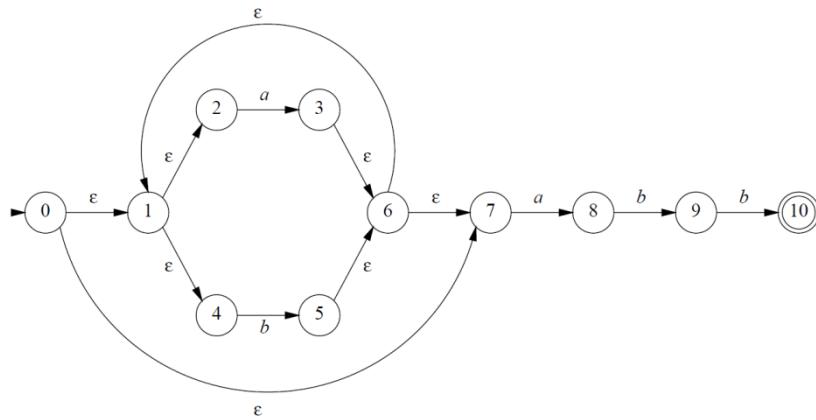
Cheat Sheet

$$A = \{0, 1, 2, 4, 7\}$$

$$B = \{1, 2, 3, 4, 6, 7, 8\}$$

$$\epsilon\text{-closure}(\{3, 8\}) = B$$

	a	b
A	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, b)) \\ &= \epsilon\text{-closure}(5) \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

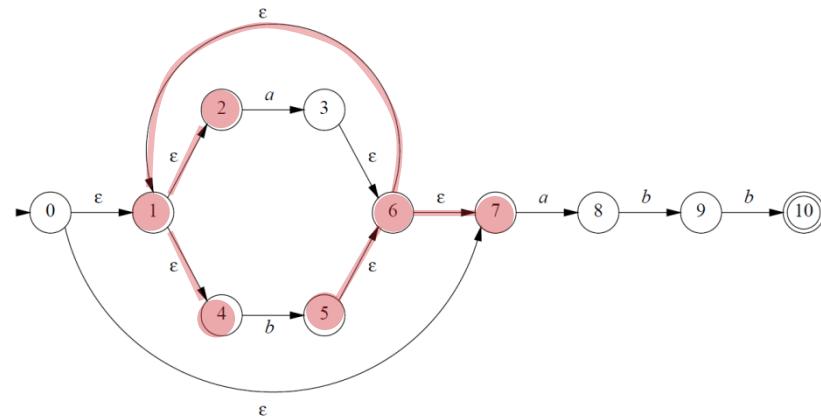
Cheat Sheet

| A = {0, 1, 2, 4, 7}

| B = {1, 2, 3, 4, 6, 7, 8}

| $\epsilon\text{-closure}(\{3, 8\}) = B$

	a	b
A	B	



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \varepsilon\text{-closure}(0) \\ &= \{0, 1, 2, 4, 7\} = A \end{aligned}$$

$$\begin{aligned} U &= \varepsilon\text{-closure}(\text{move}(T, b)) \\ &= \varepsilon\text{-closure}(\text{move}(\{0, 1, 2, 4, 7\}, b)) \\ &= \varepsilon\text{-closure}(5) \\ &= \{1, 2, 4, 5, 6, 7\} \end{aligned}$$

Dstates:

A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>

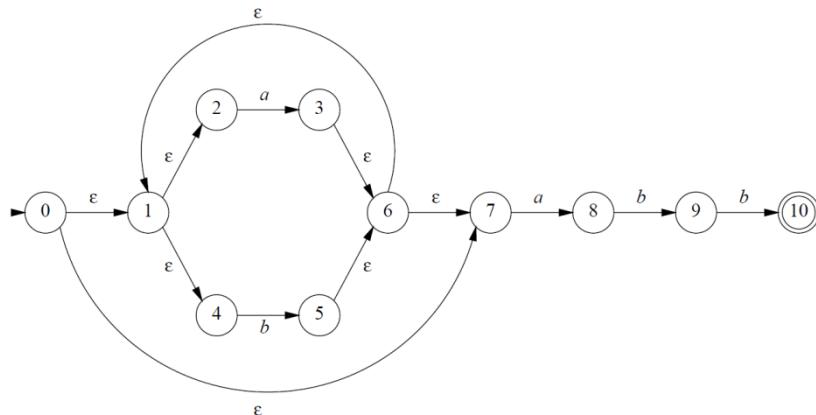
	a	b
A	B	

Cheat Sheet

| A = {0, 1, 2, 4, 7}

| B = {1, 2, 3, 4, 6, 7, 8}

| $\varepsilon\text{-closure}(\{3, 8\}) = B$



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0,1,2,4,7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0,1,2,4,7\}, b)) \\ &= \epsilon\text{-closure}(5) \\ &= \{1,2,4,5,6,7\} = C \end{aligned}$$

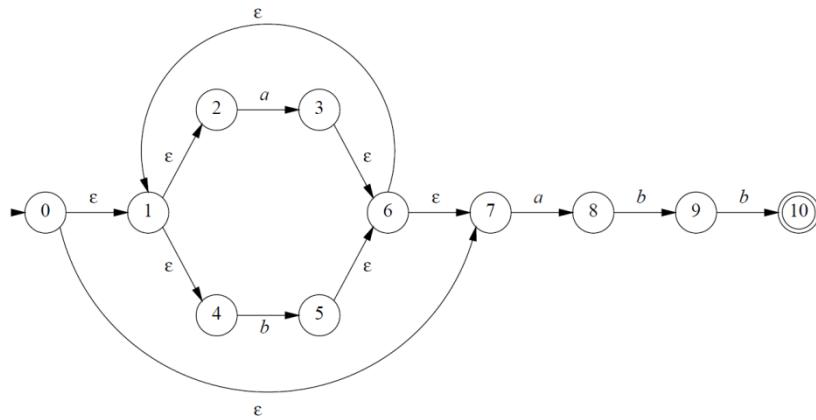
$Dstates:$

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input type="checkbox"/> |
| C | <input type="checkbox"/> |

	a	b
A	B	

Cheat Sheet

- | | |
|-----------------------|--|
| I A = {0,1,2,4,7} | I $\epsilon\text{-closure}(\{3,8\}) = B$ |
| I B = {1,2,3,4,6,7,8} | I $\epsilon\text{-closure}(\{5\}) = C$ |
| I C = {1,2,4,5,6,7} | |
| I | |
| I | |



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0,1,2,4,7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0,1,2,4,7\}, b)) \\ &= \epsilon\text{-closure}(5) \\ &= \{1,2,4,5,6,7\} = C \end{aligned}$$

Dstates:

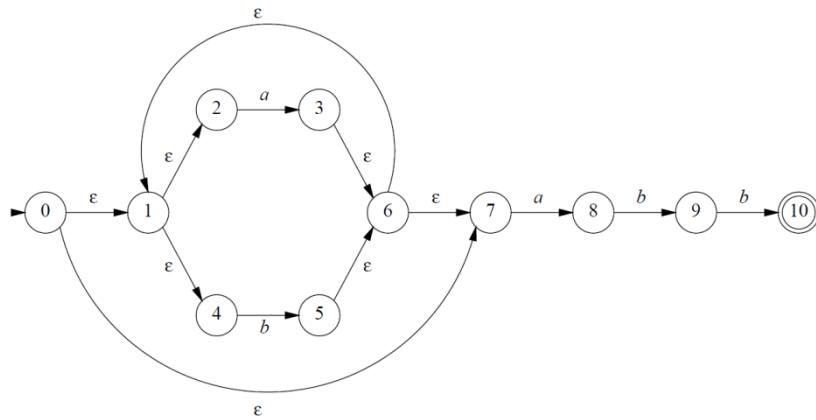
A	<input checked="" type="checkbox"/>
B	<input type="checkbox"/>
C	<input type="checkbox"/>

	a	b
A	B	C

Cheat Sheet

| A = {0,1,2,4,7}
 | B = {1,2,3,4,6,7,8}
 | C = {1,2,4,5,6,7}

| $\epsilon\text{-closure}(\{3,8\}) = B$
 | $\epsilon\text{-closure}(\{5\}) = C$



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$$\begin{aligned} T &= \epsilon\text{-closure}(0) \\ &= \{0,1,2,4,7\} = A \end{aligned}$$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{0,1,2,4,7\}, b)) \\ &= \epsilon\text{-closure}(5) \\ &= \{1,2,4,5,6,7\} = C \end{aligned}$$

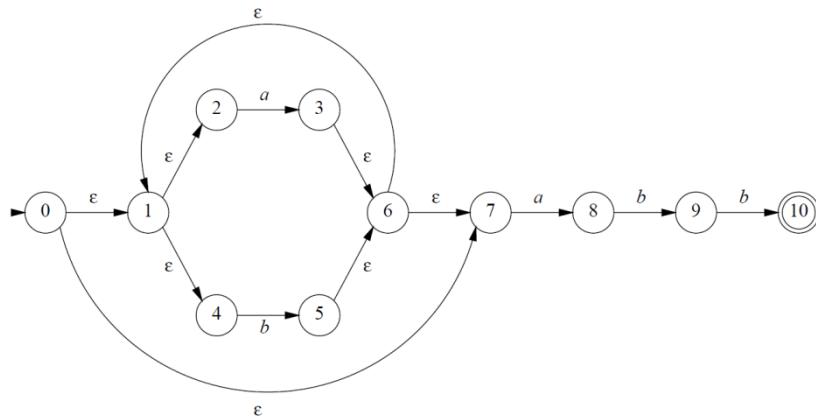
$Dstates:$

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input type="checkbox"/> |
| C | <input type="checkbox"/> |

	a	b
A	B	C

Cheat Sheet

- | | |
|-----------------------|--|
| I A = {0,1,2,4,7} | I $\epsilon\text{-closure}(\{3,8\}) = B$ |
| I B = {1,2,3,4,6,7,8} | I $\epsilon\text{-closure}(\{5\}) = C$ |
| I C = {1,2,4,5,6,7} | |
| I | |
| I | |



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$Dstates:$

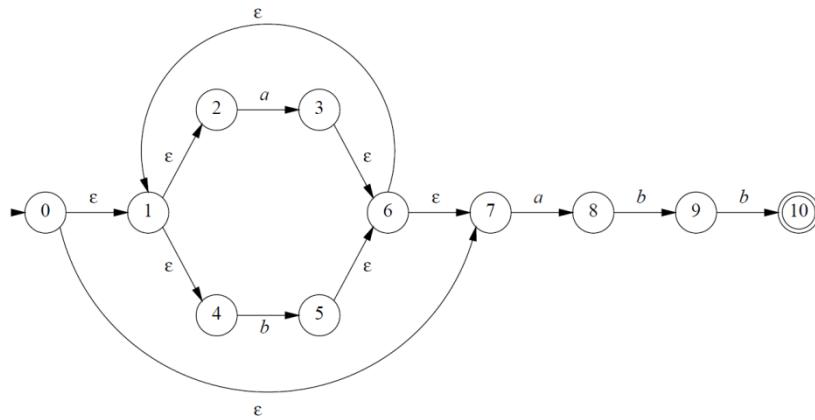
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$Dstates:$

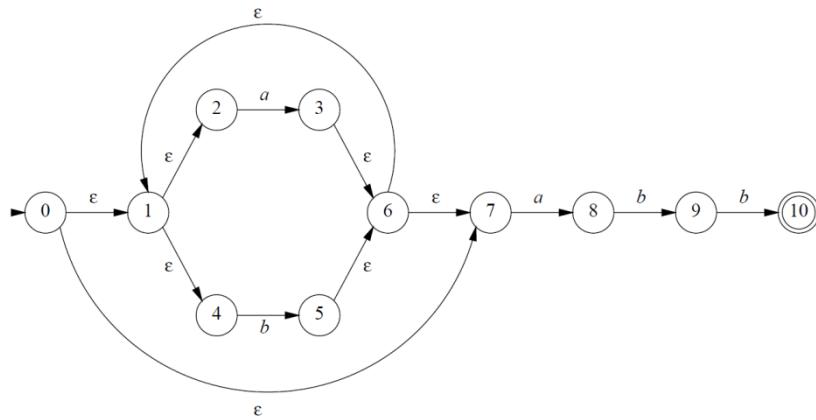
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$Dstates:$

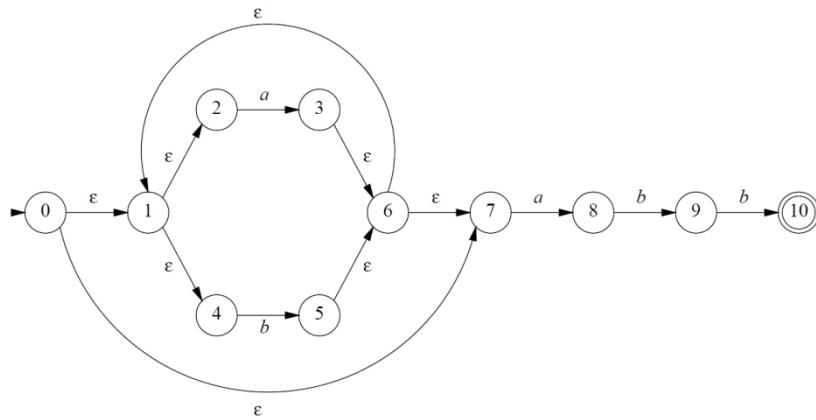
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$U = \epsilon\text{-closure}(\text{move}(T, a))$

$= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a))$

$Dstates:$

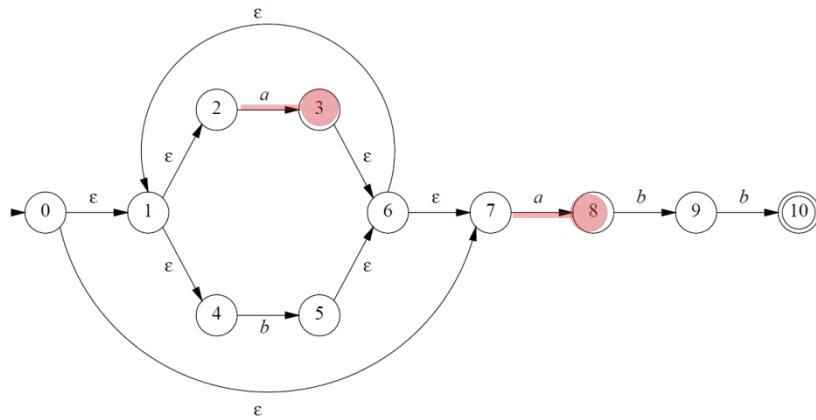
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input type="checkbox"/>

Cheat Sheet

| A = {0,1,2,4,7}
 | B = {1,2,3,4,6,7,8}
 | C = {1,2,4,5,6,7}

| $\epsilon\text{-closure}(\{3,8\}) = B$
 | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$U = \epsilon\text{-closure}(\text{move}(T, a))$

$= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a))$

$Dstates:$

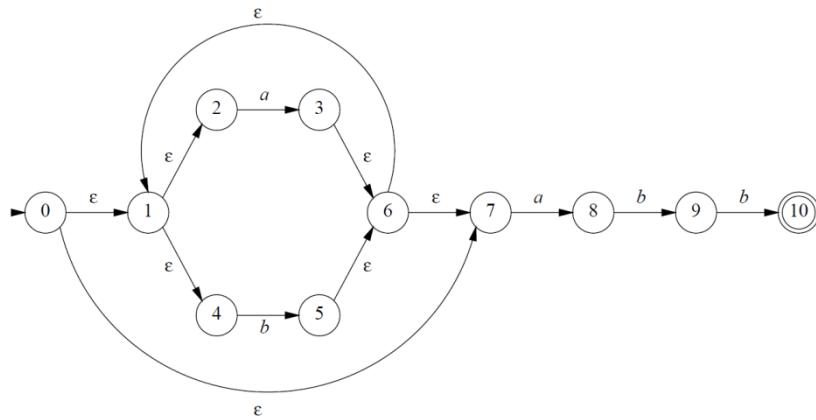
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

$| A = \{0, 1, 2, 4, 7\}$
 $| B = \{1, 2, 3, 4, 6, 7, 8\}$
 $| C = \{1, 2, 4, 5, 6, 7\}$
 $|$
 $|$

$| \epsilon\text{-closure}(\{3, 8\}) = B$
 $| \epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a)) \\ &= \epsilon\text{-closure}(\{3,8\}) \end{aligned}$$

$Dstates:$

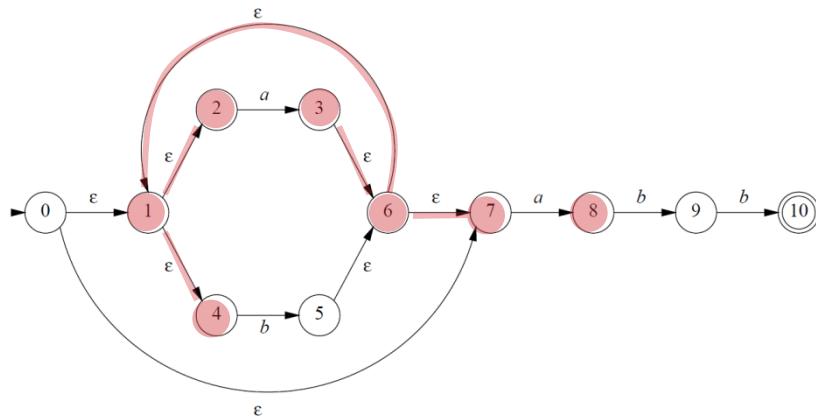
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input type="checkbox"/>

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, a)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a)) \\
 &= \epsilon\text{-closure}(\{3,8\}) \\
 &= \{1,2,3,4,6,7,8\} = B
 \end{aligned}$$

$Dstates:$

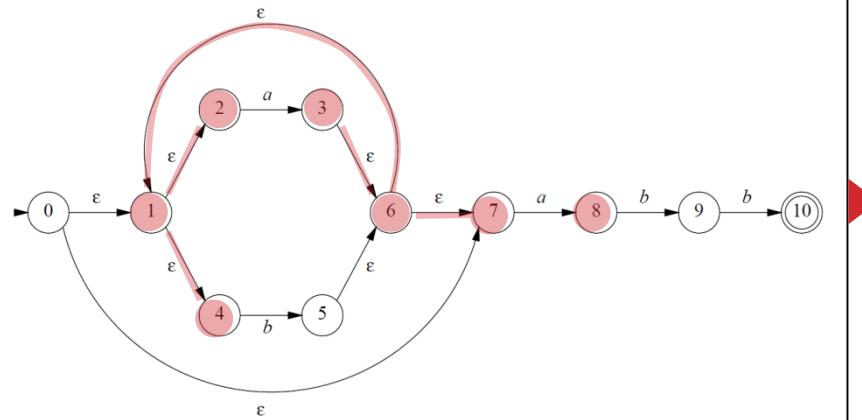
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

	a	b
A	B	C

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$



```

add state  $T = \varepsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \varepsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

 $\varepsilon\text{-closure}(s_0)$  is the start state of  $D$ 
A state of  $D$  is accepting if it contains at least one accepting state in  $N$ 

```

$T = B$

$$\begin{aligned}
 U &= \varepsilon\text{-closure}(\text{move}(T, a)) \\
 &= \varepsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a)) \\
 &= \varepsilon\text{-closure}(\{3,8\}) \\
 &= \{1,2,3,4,6,7,8\} = B
 \end{aligned}$$

$Dstates:$

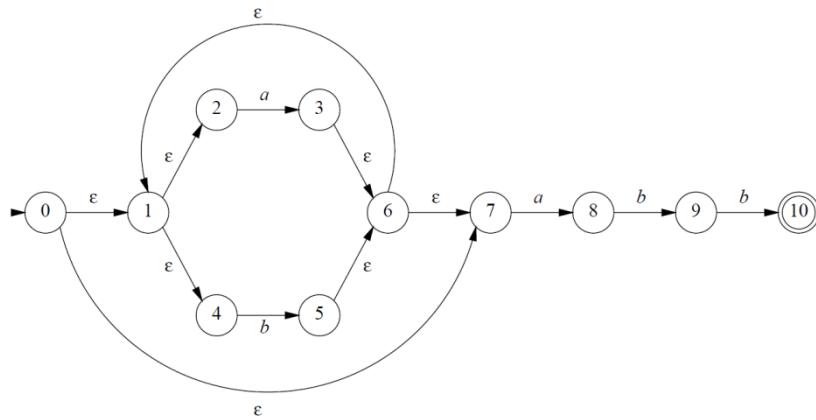
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

	a	b
A	B	C

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\varepsilon\text{-closure}(\{3,8\}) = B$
- | $\varepsilon\text{-closure}(\{5\}) = C$



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, a)) \\
&= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, a)) \\
&= \epsilon\text{-closure}(\{3,8\}) \\
&= \{1,2,3,4,6,7,8\} = B
\end{aligned}$$

$Dstates:$

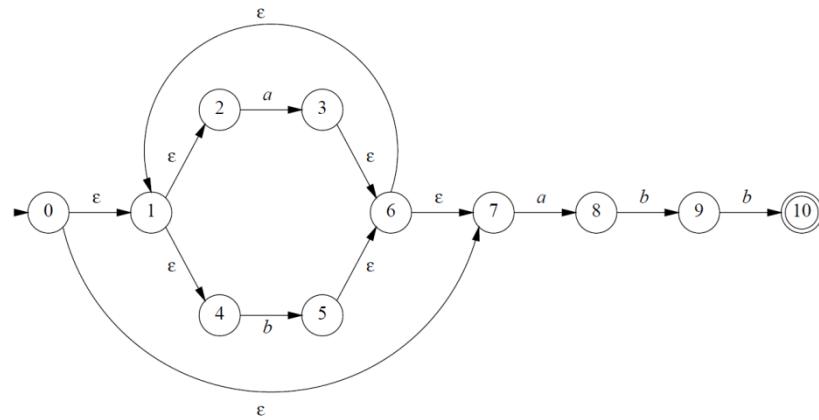
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$Dstates:$

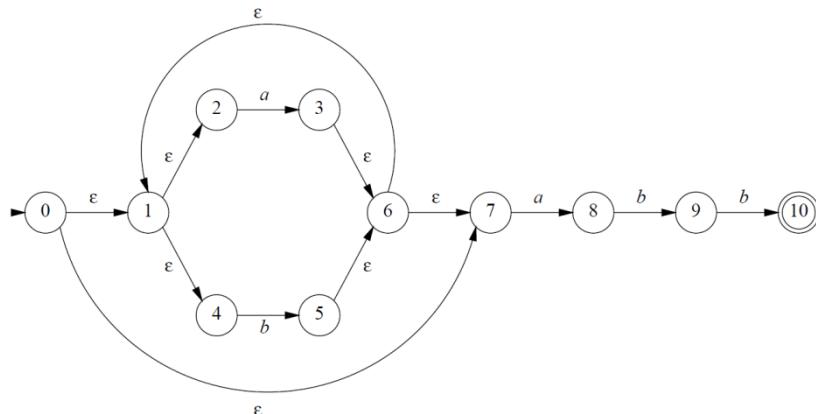
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$U = \epsilon\text{-closure}(\text{move}(T, b))$

$= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b))$

$Dstates:$

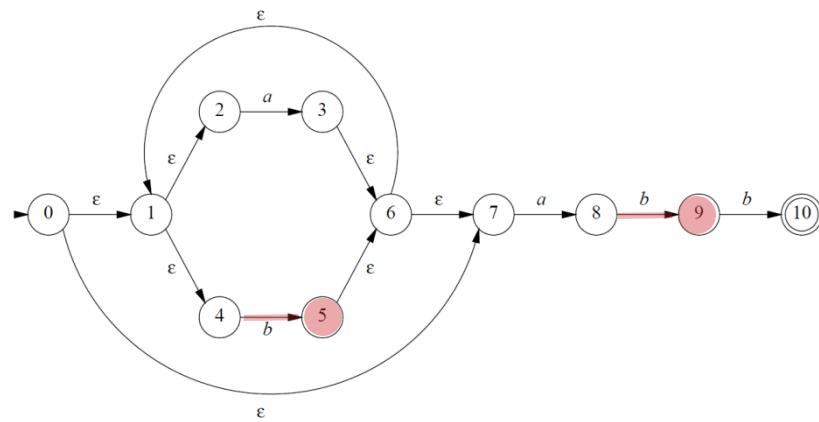
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile
  
```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = B$

$U = \epsilon\text{-closure}(\text{move}(T, b))$
 $= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b))$

$Dstates:$

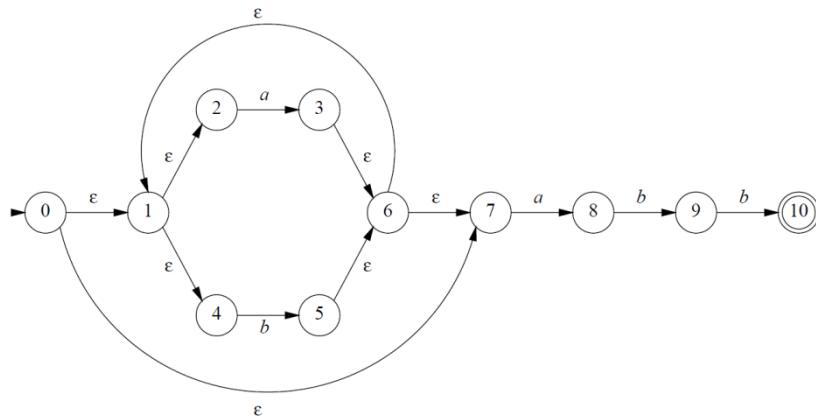
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

| A = {0,1,2,4,7}
| B = {1,2,3,4,6,7,8}
| C = {1,2,4,5,6,7}

| $\epsilon\text{-closure}(\{3,8\}) = B$
| $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b)) \\ &= \epsilon\text{-closure}(\{5, 9\}) \end{aligned}$$

$Dstates:$

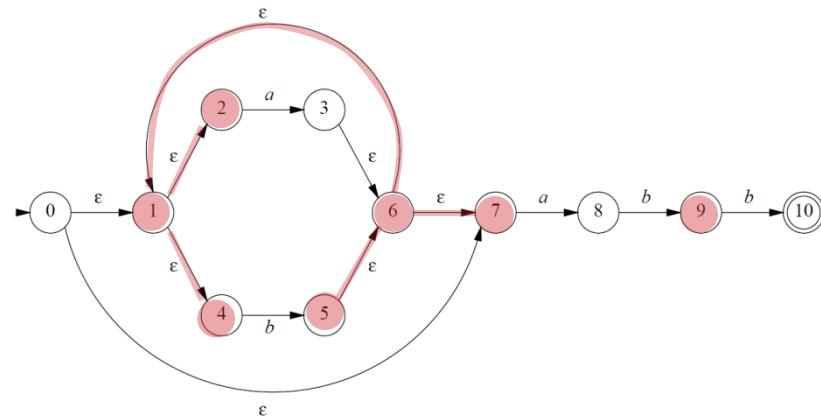
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned} U &= \varepsilon\text{-closure}(\text{move}(T, b)) \\ &= \varepsilon\text{-closure}(\text{move}(\{1, 2, 3, 4, 6, 7, 8\}, b)) \\ &= \varepsilon\text{-closure}(\{5, 9\}) \end{aligned}$$

$Dstates:$

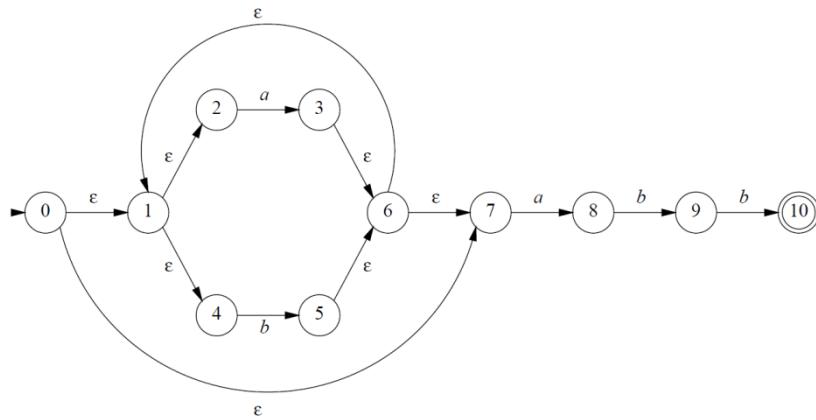
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- |
- |

- | $\varepsilon\text{-closure}(\{3, 8\}) = B$
- | $\varepsilon\text{-closure}(\{5\}) = C$

	a	b
A	B	C
B	B	



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
&= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b)) \\
&= \epsilon\text{-closure}(\{5, 9\}) \\
&= \{1,2,4,5,6,7,9\}
\end{aligned}$$

$Dstates:$

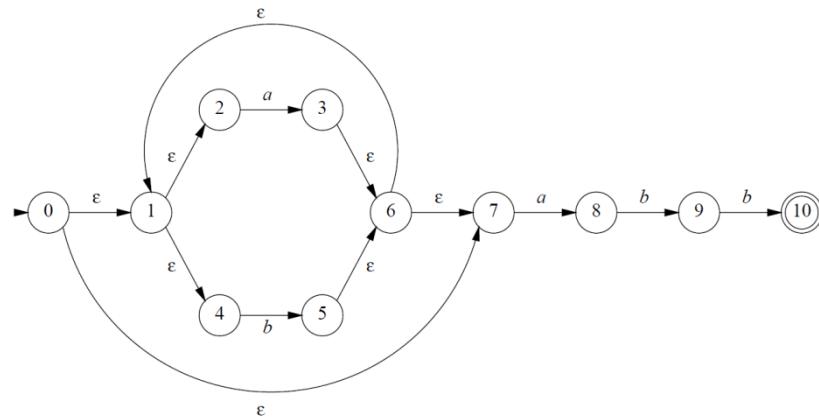
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |

	a	b
A	B	C
B	B	

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- |
- |

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b)) \\
 &= \epsilon\text{-closure}(\{5, 9\}) \\
 &= \{1,2,4,5,6,7,9\} = D
 \end{aligned}$$

$Dstates:$

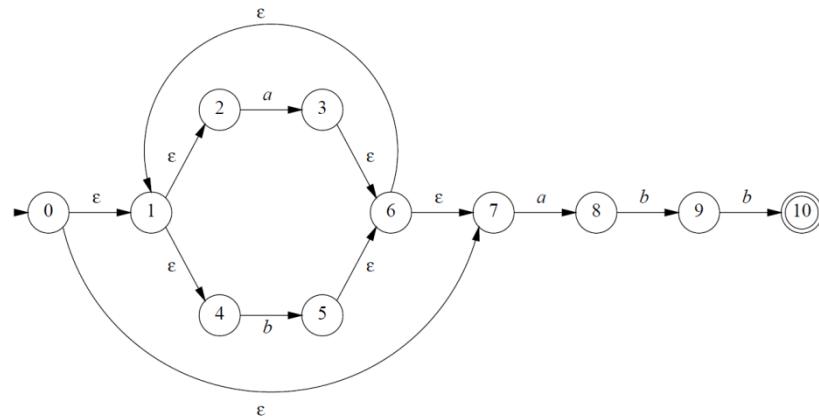
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |
| D | <input type="checkbox"/> |

	a	b
A	B	C
B	B	

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = B$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
&= \epsilon\text{-closure}(\text{move}(\{1,2,3,4,6,7,8\}, b)) \\
&= \epsilon\text{-closure}(\{5, 9\}) \\
&= \{1,2,4,5,6,7,9\} = D
\end{aligned}$$

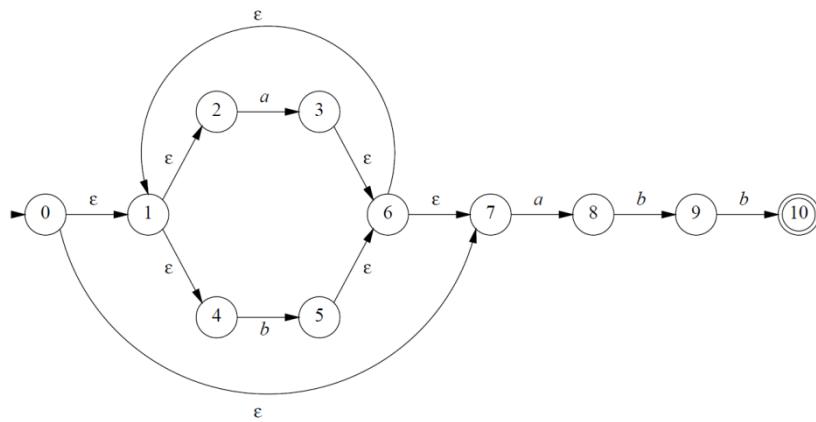
$Dstates:$

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |
| D | <input type="checkbox"/> |

	a	b
A	B	C
B	B	D

Cheat Sheet

- | | |
|-----------------------|----------------------|
| I A = {0,1,2,4,7} | ε-closure({3,8}) = B |
| I B = {1,2,3,4,6,7,8} | ε-closure({5}) = C |
| I C = {1,2,4,5,6,7} | ε-closure({5,9}) = D |
| I D = {1,2,4,5,6,7,9} | |



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = B$

$Dstates:$

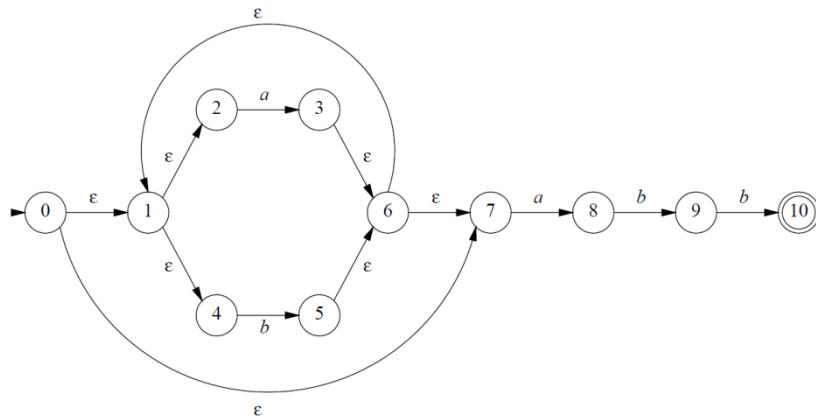
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$Dstates:$

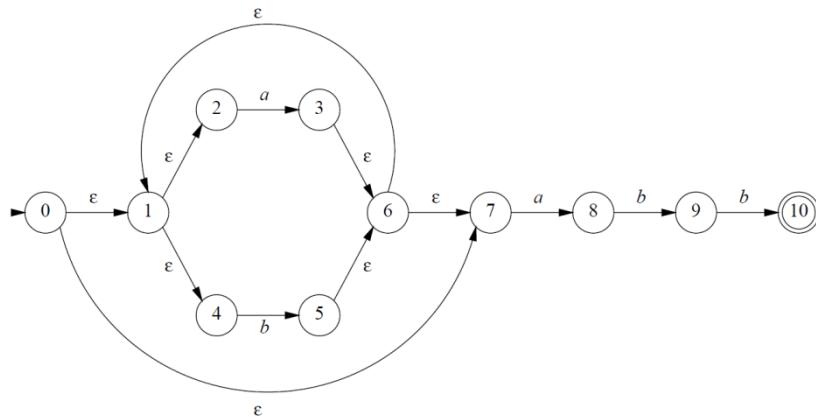
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$Dstates:$

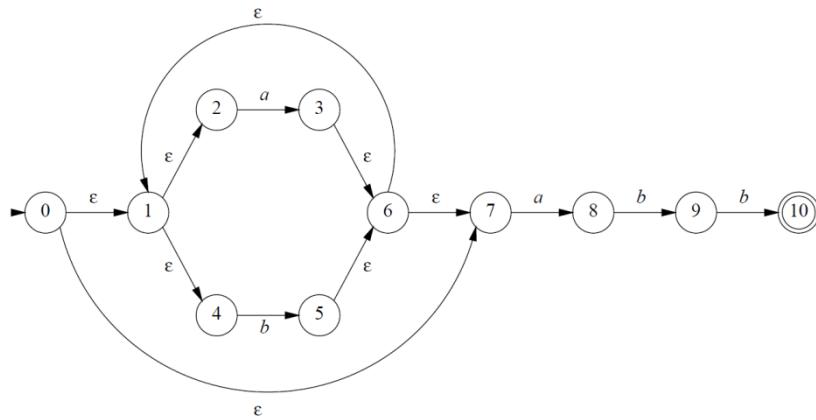
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$Dstates:$

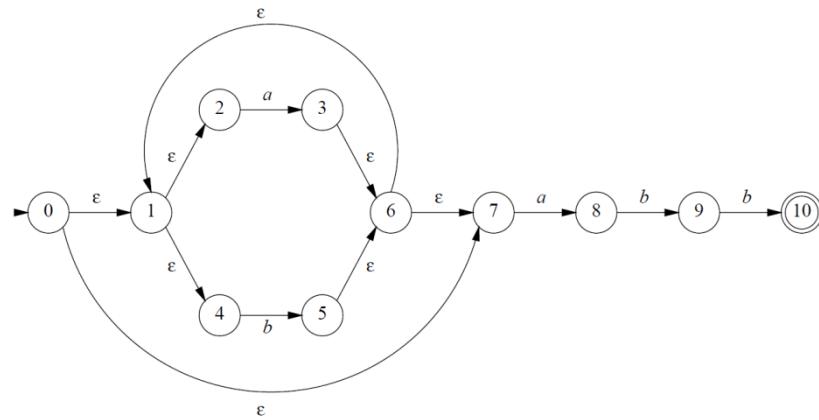
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$U = \epsilon\text{-closure}(\text{move}(T, a))$
 $= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, a))$

$Dstates:$

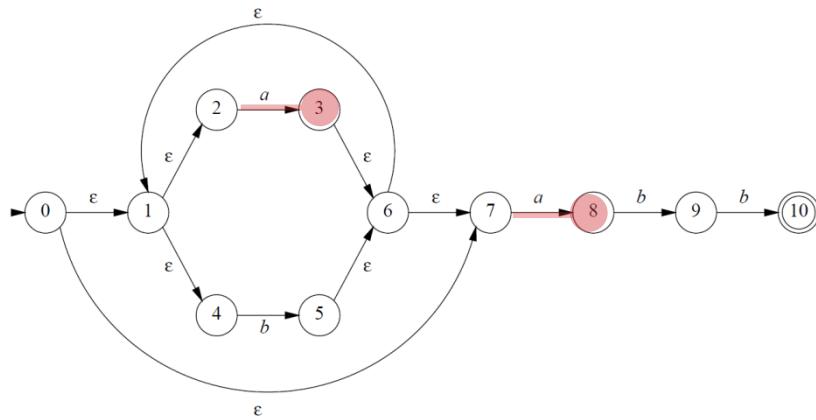
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

	a	b
A	B	C
B	B	D

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \end{aligned}$$

$Dstates:$

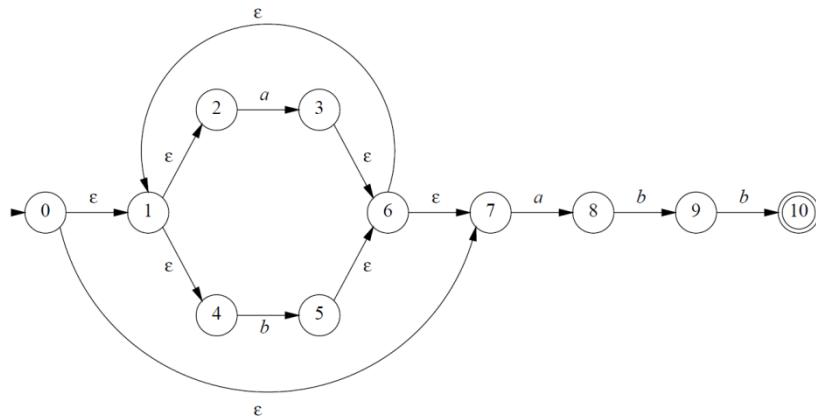
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

	a	b
A	B	C
B	B	D

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

$Dstates:$

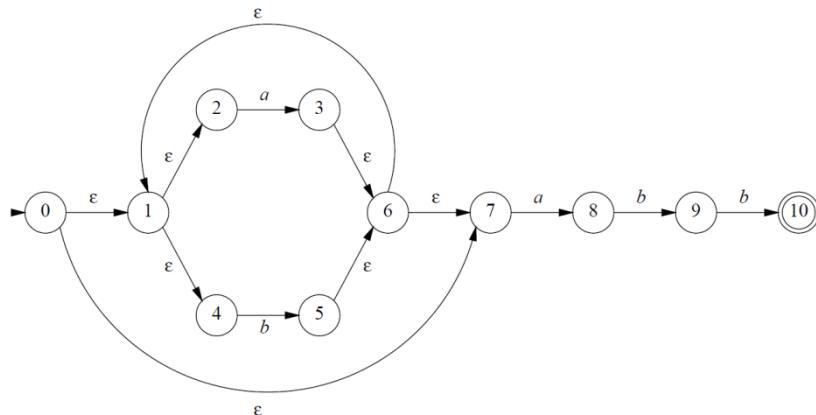
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input type="checkbox"/>

	a	b
A	B	C
B	B	D

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

$Dstates:$

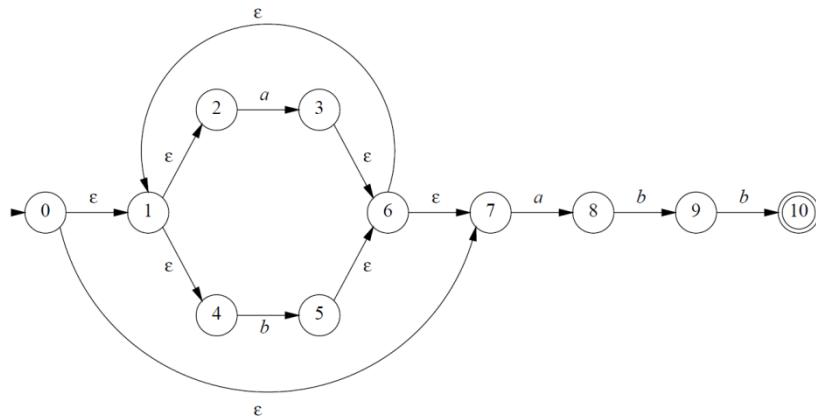
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

	a	b
A	B	C
B	B	D

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7\}, a)) \\ &= \epsilon\text{-closure}(\{3,8\}) = B \end{aligned}$$

$Dstates:$

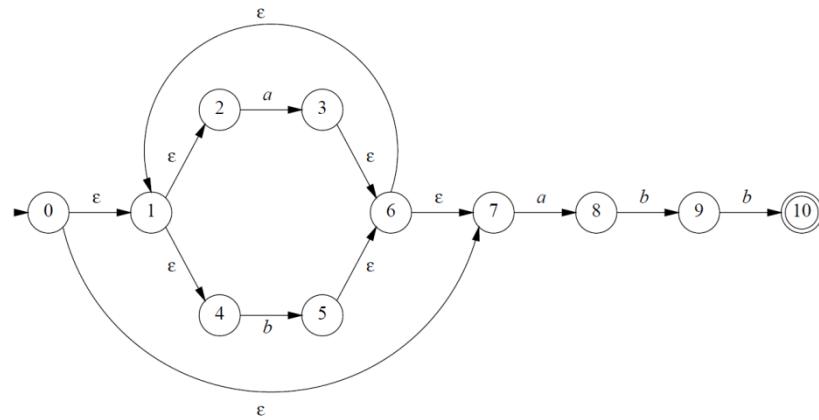
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input type="checkbox"/>

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$Dstates:$

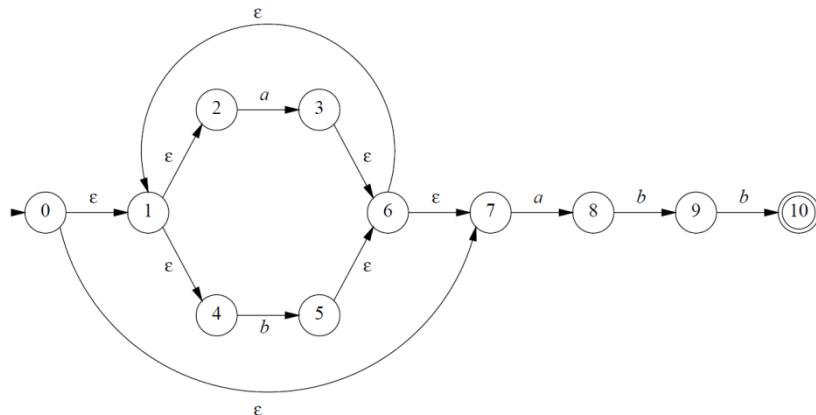
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$U = \epsilon\text{-closure}(\text{move}(T, b)) \\ = \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7\}, b))$$

$Dstates:$

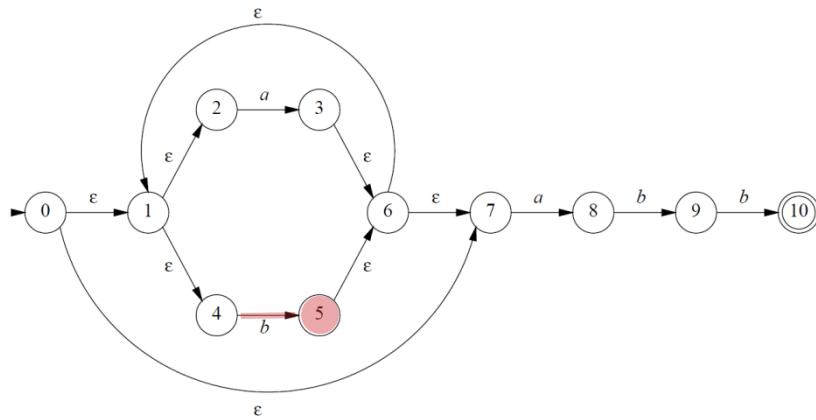
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, b)) \\ &= \epsilon\text{-closure}(5) \end{aligned}$$

$Dstates:$

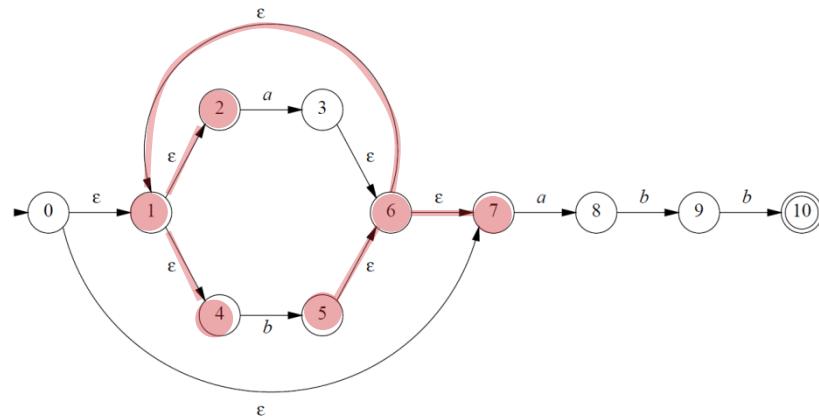
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \varepsilon\text{-closure}(\text{move}(T, b)) \\ &= \varepsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, b)) \\ &= \varepsilon\text{-closure}(5) = C \end{aligned}$$

$Dstates:$

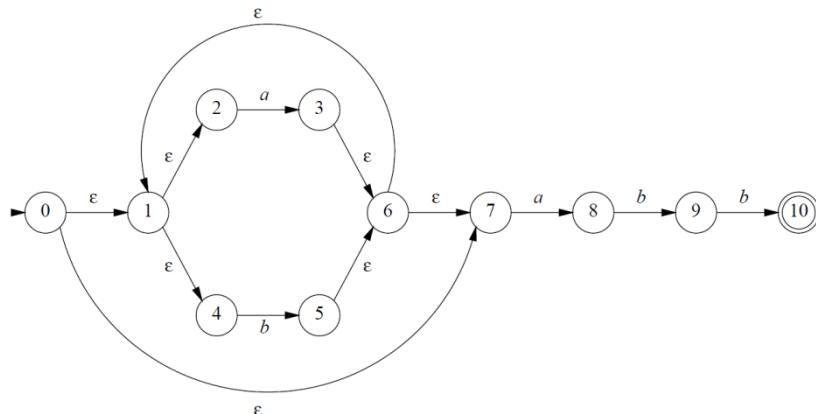
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}

- I $\varepsilon\text{-closure}(\{3, 8\}) = B$
- I $\varepsilon\text{-closure}(\{5\}) = C$
- I $\varepsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, b)) \\ &= \epsilon\text{-closure}(5) = C \end{aligned}$$

$Dstates:$

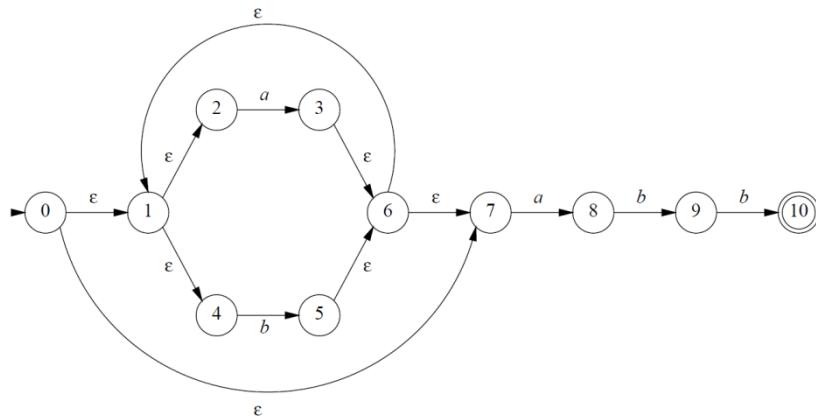
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	



```

add state  $T = \varepsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \varepsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

```

$\varepsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = C$

$$\begin{aligned}
U &= \varepsilon\text{-closure}(\text{move}(T, b)) \\
&= \varepsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7\}, b)) \\
&= \varepsilon\text{-closure}(5) = C
\end{aligned}$$

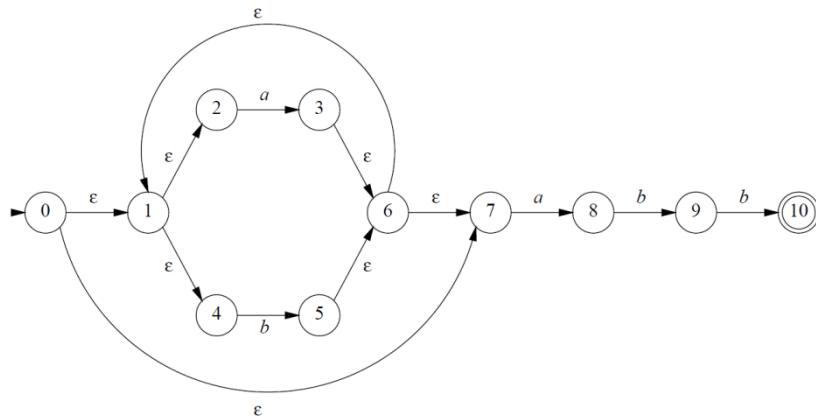
$Dstates:$

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | | |
|-----------------------------|--|
| I A = {0, 1, 2, 4, 7} | I $\varepsilon\text{-closure}(\{3, 8\}) = B$ |
| I B = {1, 2, 3, 4, 6, 7, 8} | I $\varepsilon\text{-closure}(\{5\}) = C$ |
| I C = {1, 2, 4, 5, 6, 7} | I $\varepsilon\text{-closure}(\{5, 9\}) = D$ |
| I D = {1, 2, 4, 5, 6, 7, 9} | |

	a	b
A	B	C
B	B	D
C	B	C



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = C$

$Dstates:$

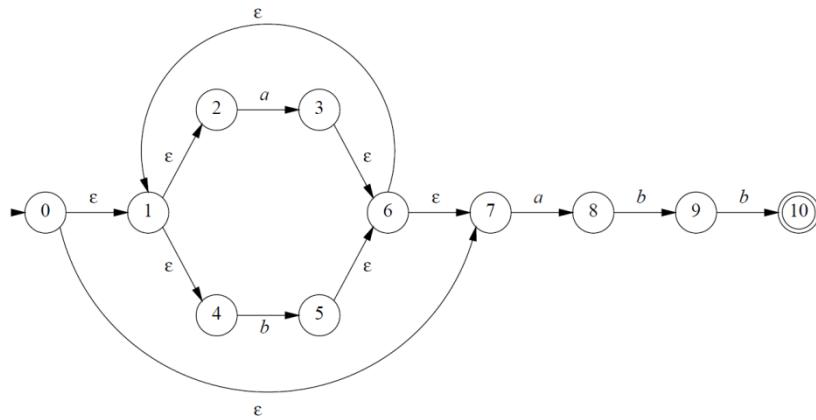
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$Dstates:$

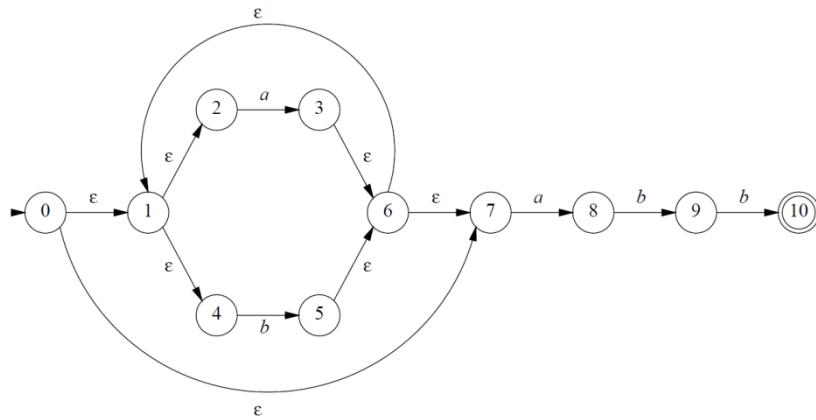
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$Dstates:$

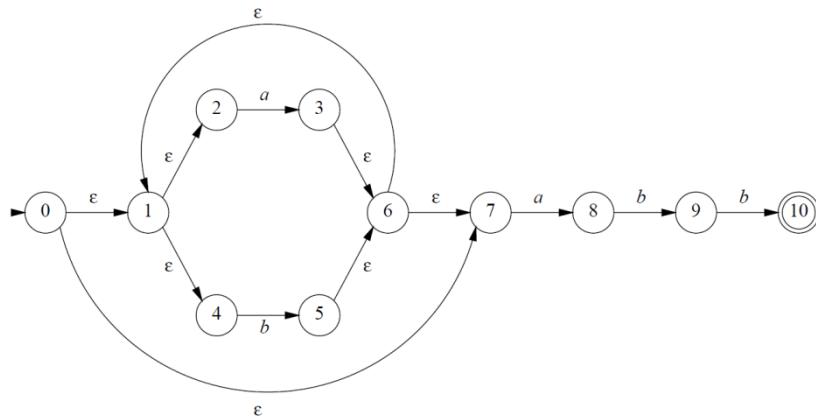
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$Dstates:$

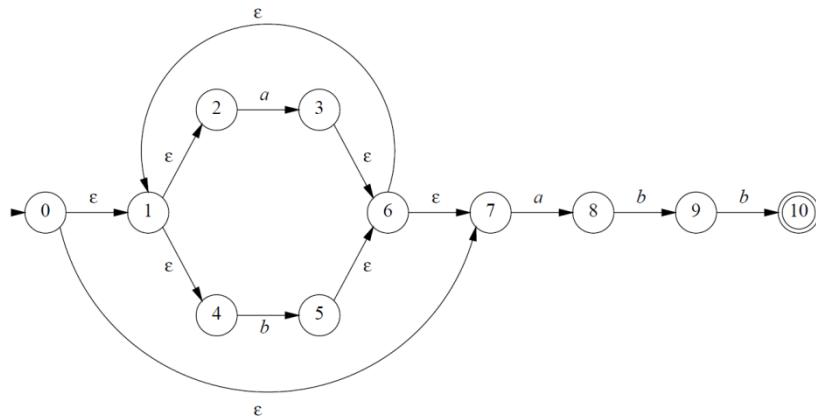
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$U = \epsilon\text{-closure}(\text{move}(T, a))$
 $= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, a))$

$Dstates:$

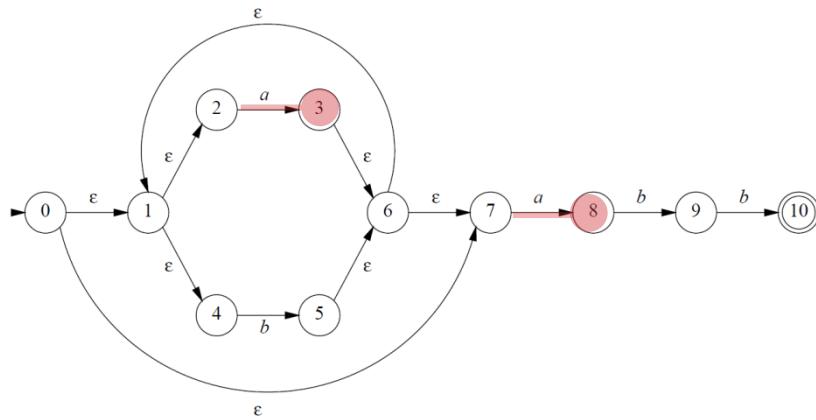
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$U = \epsilon\text{-closure}(\text{move}(T, a))$
 $= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,9\}, a))$

$Dstates:$

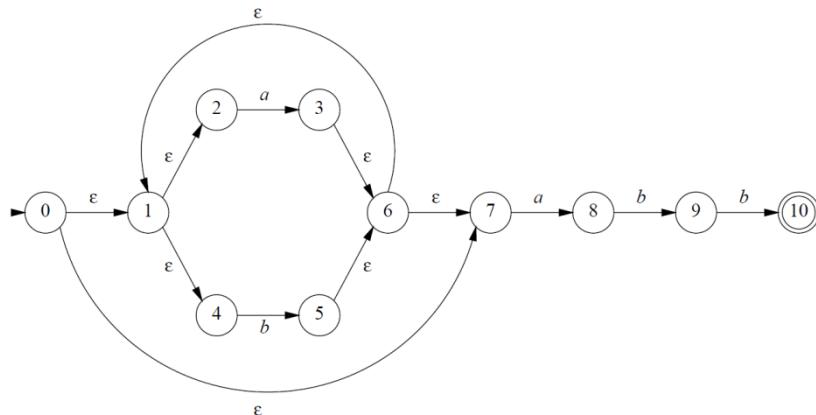
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \end{aligned}$$

$Dstates:$

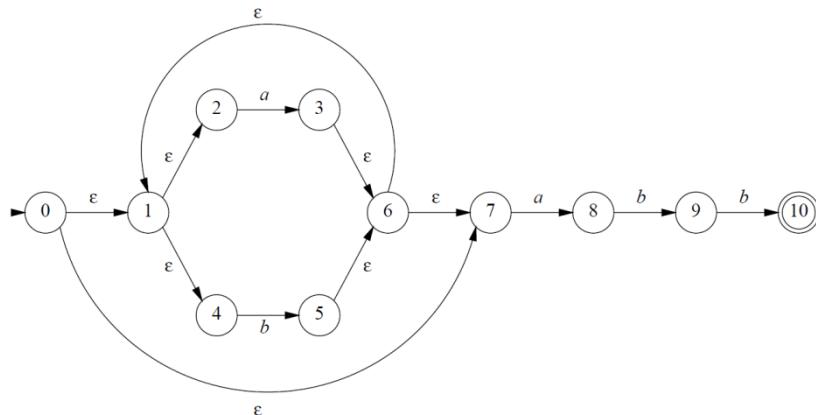
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

$Dstates:$

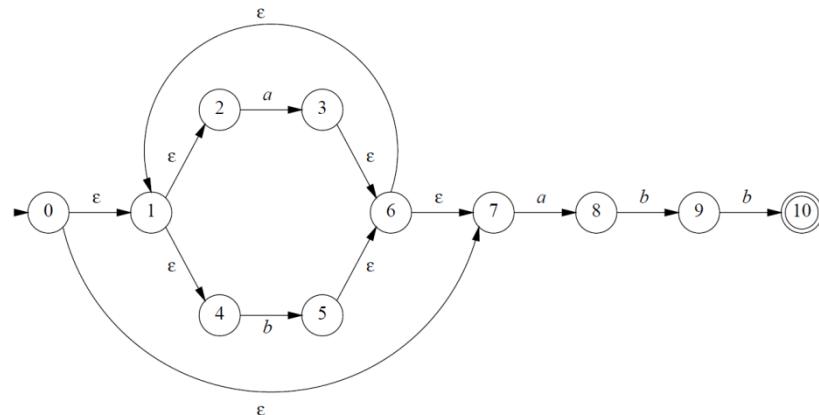
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

$Dstates:$

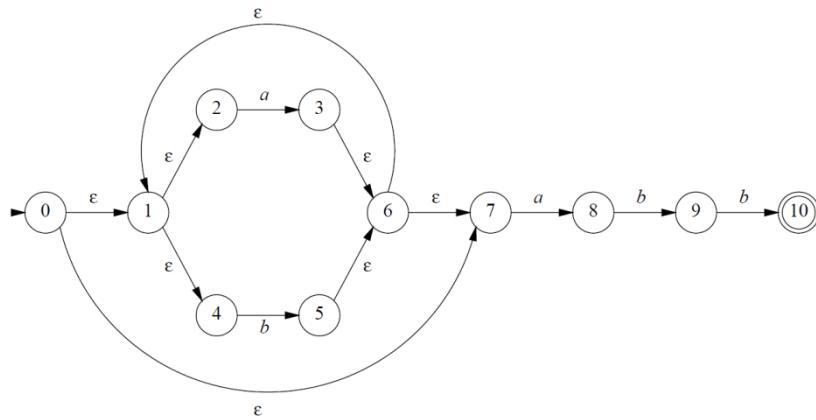
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, a)) \\
&= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, a)) \\
&= \epsilon\text{-closure}(\{3, 8\}) = B
\end{aligned}$$

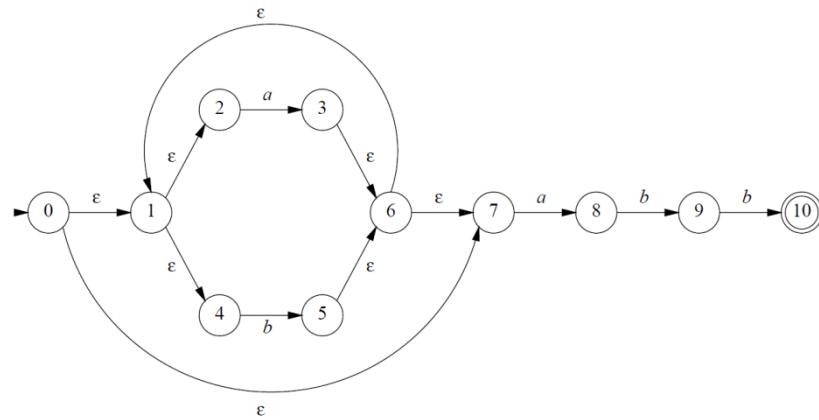
$Dstates:$

- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

	a	b
A	B	C
B	B	D
C	B	C
D	B	

Cheat Sheet

- | | |
|-----------------------------|---|
| I A = {0, 1, 2, 4, 7} | $\epsilon\text{-closure}(\{3, 8\}) = B$ |
| I B = {1, 2, 3, 4, 6, 7, 8} | $\epsilon\text{-closure}(\{5\}) = C$ |
| I C = {1, 2, 4, 5, 6, 7} | $\epsilon\text{-closure}(\{5, 9\}) = D$ |
| I D = {1, 2, 4, 5, 6, 7, 9} | |



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$Dstates:$

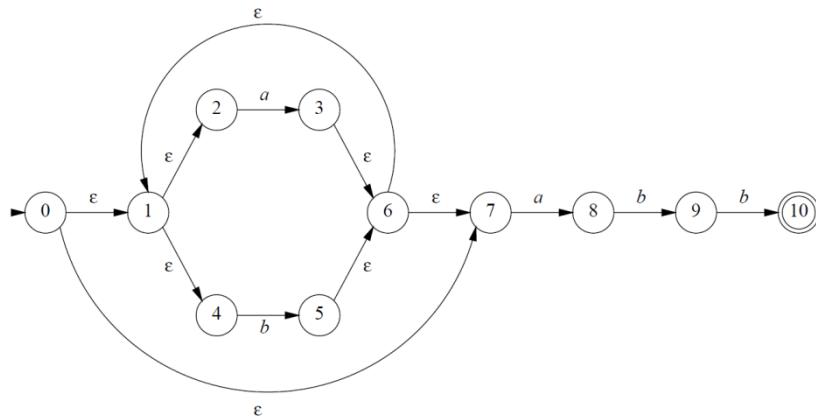
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$U = \epsilon\text{-closure}(\text{move}(T, b)) \\ = \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, b))$$

$Dstates:$

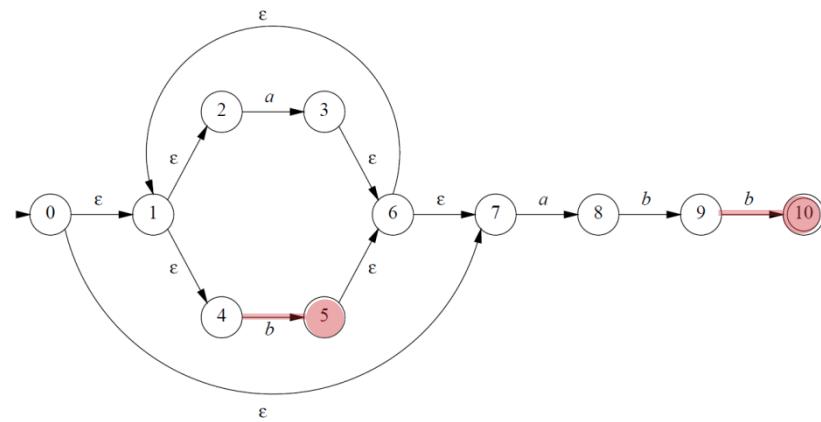
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$U = \epsilon\text{-closure}(\text{move}(T, b)) \\ = \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 9\}, b))$$

$Dstates:$

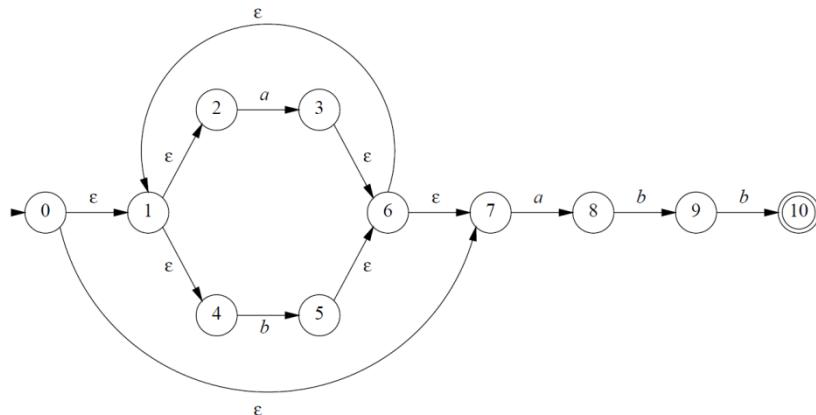
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0, 1, 2, 4, 7}
- | B = {1, 2, 3, 4, 6, 7, 8}
- | C = {1, 2, 4, 5, 6, 7}
- | D = {1, 2, 4, 5, 6, 7, 9}

- | $\epsilon\text{-closure}(\{3, 8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5, 9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,9\}, b)) \\ &= \epsilon\text{-closure}(\{5,10\}) \end{aligned}$$

$Dstates:$

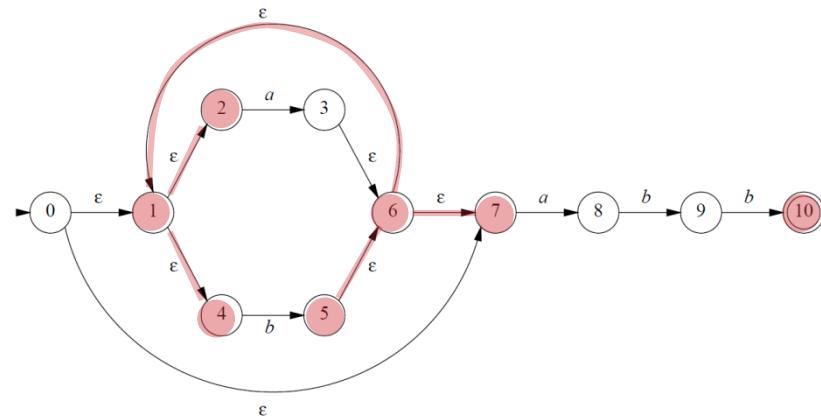
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,9\}, b)) \\
 &= \epsilon\text{-closure}(\{5,10\}) \\
 &= \{1,2,4,5,6,7,10\}
 \end{aligned}$$

$Dstates:$

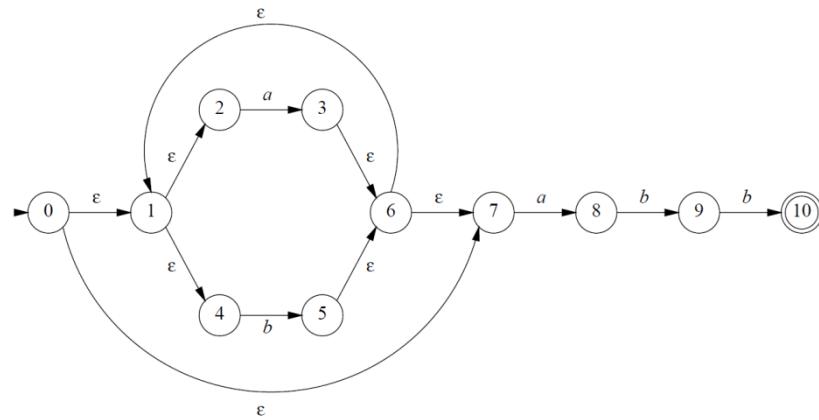
- | | |
|---|-------------------------------------|
| A | <input checked="" type="checkbox"/> |
| B | <input checked="" type="checkbox"/> |
| C | <input checked="" type="checkbox"/> |
| D | <input checked="" type="checkbox"/> |

Cheat Sheet

- | A = {0,1,2,4,7}
- | B = {1,2,3,4,6,7,8}
- | C = {1,2,4,5,6,7}
- | D = {1,2,4,5,6,7,9}

- | $\epsilon\text{-closure}(\{3,8\}) = B$
- | $\epsilon\text{-closure}(\{5\}) = C$
- | $\epsilon\text{-closure}(\{5,9\}) = D$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,9\}, b)) \\
 &= \epsilon\text{-closure}(\{5,10\}) \\
 &= \{1,2,4,5,6,7,10\} = E
 \end{aligned}$$

$Dstates:$

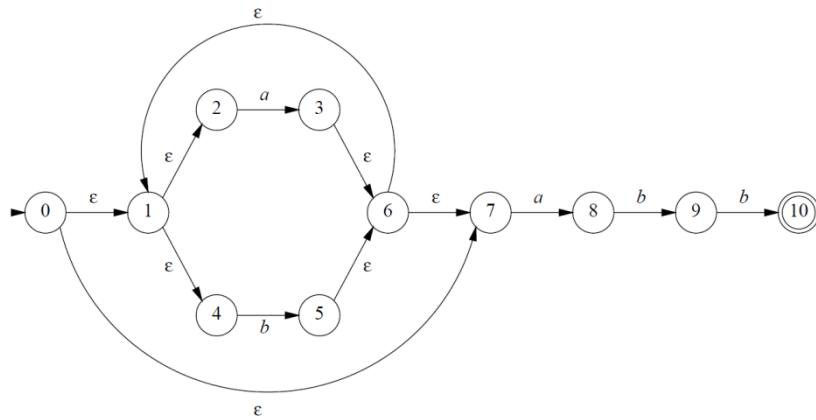
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,9\}, b)) \\
 &= \epsilon\text{-closure}(\{5,10\}) \\
 &= \{1,2,4,5,6,7,10\} = E
 \end{aligned}$$

$Dstates:$

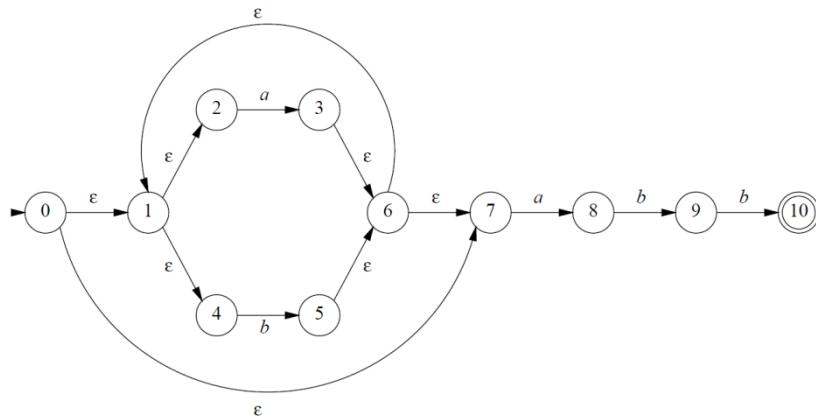
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = D$

$Dstates:$

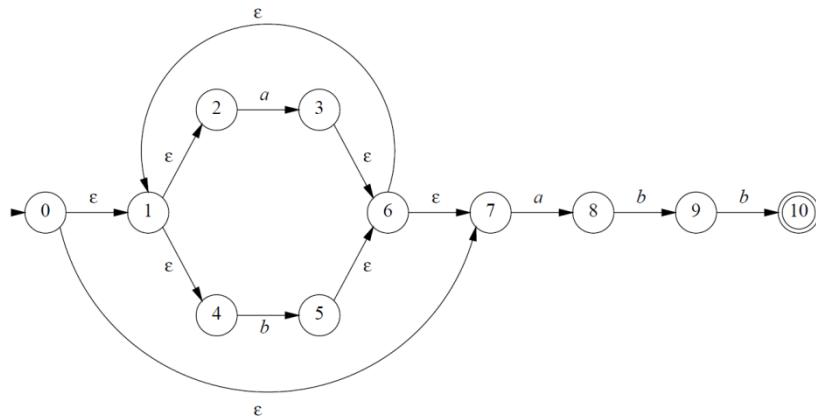
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$Dstates:$

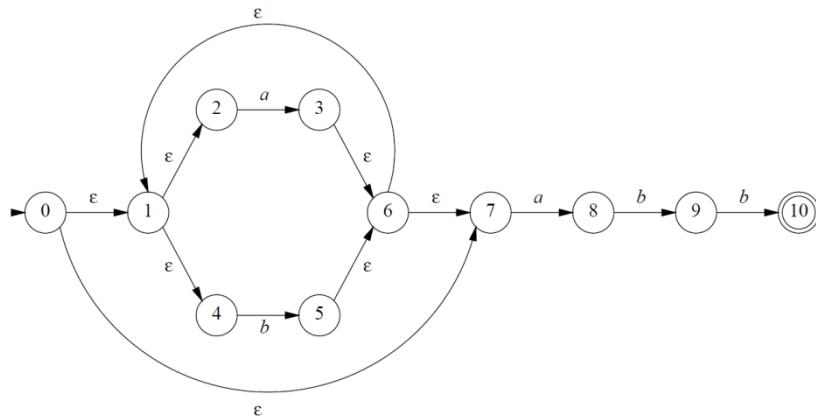
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$Dstates:$

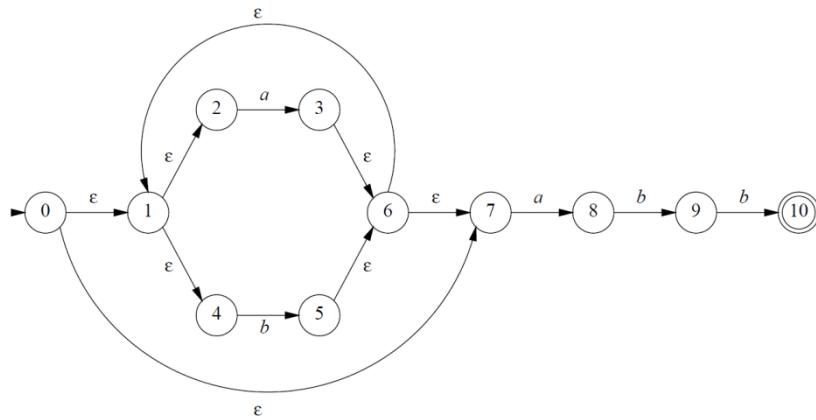
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$Dstates:$

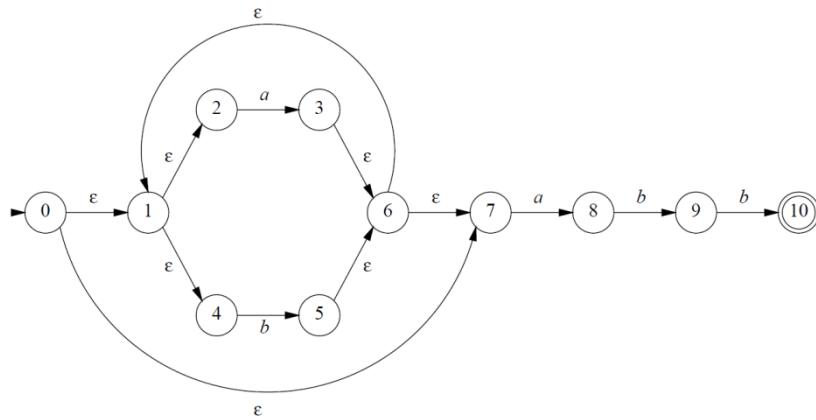
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$U = \epsilon\text{-closure}(\text{move}(T, a))$
 $= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, a))$

$Dstates:$

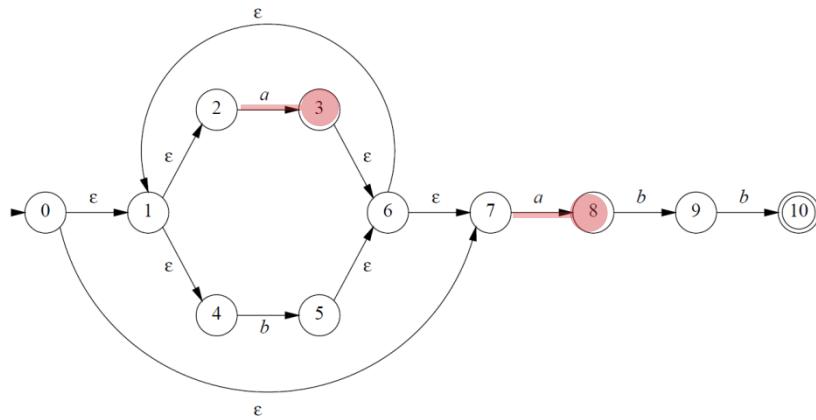
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \varepsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \varepsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\varepsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$U = \varepsilon\text{-closure}(\text{move}(T, a)) \\ = \varepsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, a))$$

$Dstates:$

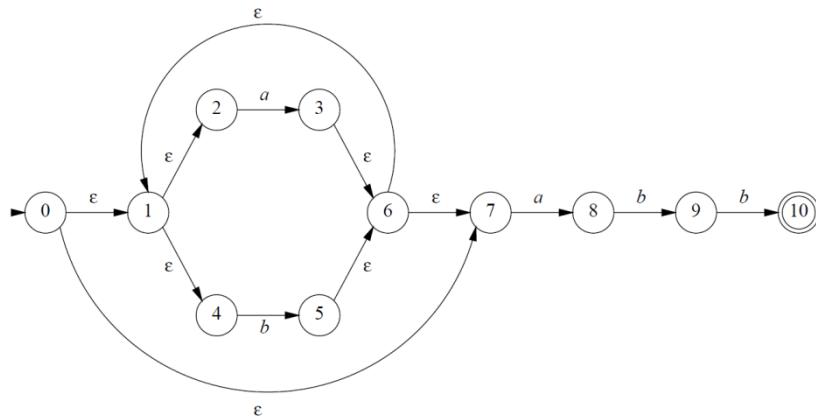
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}
- I E = {1, 2, 4, 5, 6, 7, 10}

- I $\varepsilon\text{-closure}(\{3, 8\}) = B$
- I $\varepsilon\text{-closure}(\{5\}) = C$
- I $\varepsilon\text{-closure}(\{5, 9\}) = D$
- I $\varepsilon\text{-closure}(\{5, 10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) \end{aligned}$$

$Dstates:$

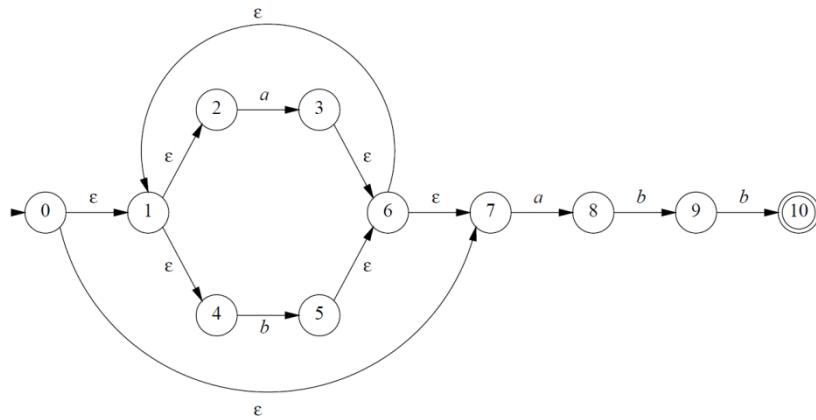
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}
- I E = {1, 2, 4, 5, 6, 7, 10}

- I $\epsilon\text{-closure}(\{3, 8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5, 9\}) = D$
- I $\epsilon\text{-closure}(\{5, 10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, a)) \\ &= \epsilon\text{-closure}(\{3,8\}) = B \end{aligned}$$

$Dstates:$

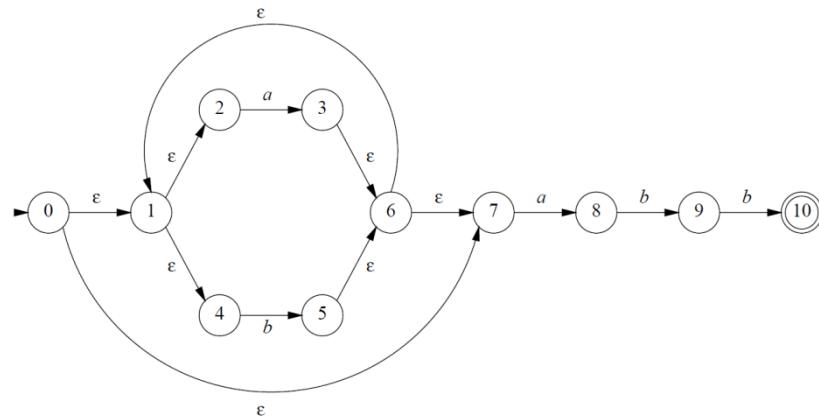
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

$Dstates:$

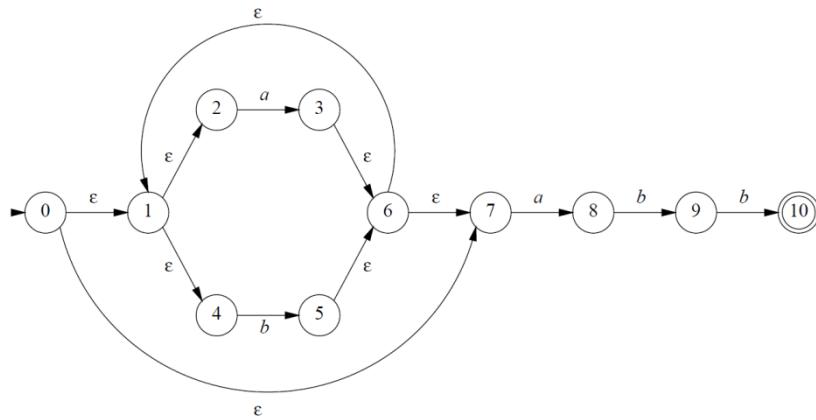
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}
- I E = {1, 2, 4, 5, 6, 7, 10}

- I $\epsilon\text{-closure}(\{3, 8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5, 9\}) = D$
- I $\epsilon\text{-closure}(\{5, 10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, a)) \\ &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, a)) \\ &= \epsilon\text{-closure}(\{3, 8\}) = B \end{aligned}$$

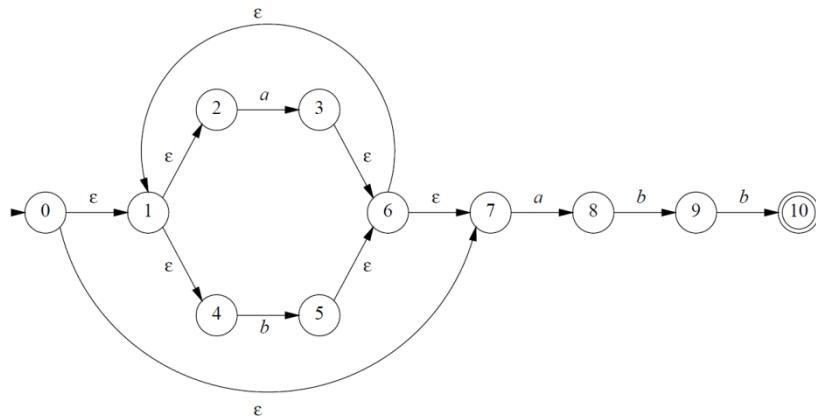
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

I A = {0, 1, 2, 4, 7}	I ε-closure({3, 8}) = B
I B = {1, 2, 3, 4, 6, 7, 8}	I ε-closure({5}) = C
I C = {1, 2, 4, 5, 6, 7}	I ε-closure({5, 9}) = D
I D = {1, 2, 4, 5, 6, 7, 9}	I ε-closure({5, 10}) = E
I E = {1, 2, 4, 5, 6, 7, 10}	

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$Dstates:$

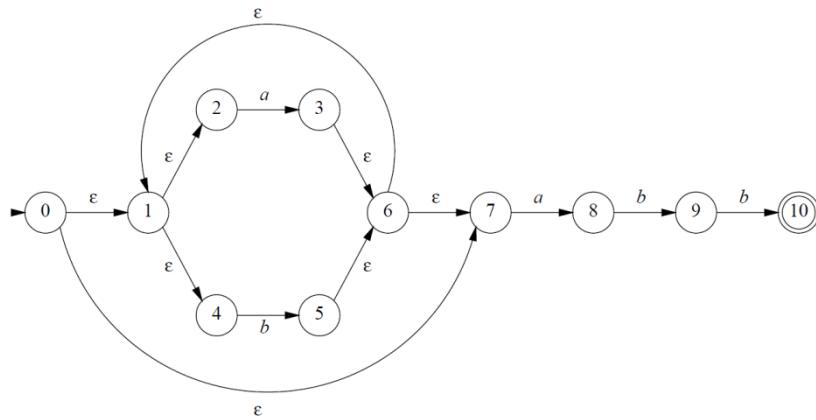
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$U = \epsilon\text{-closure}(\text{move}(T, b))$
 $= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, b))$

$Dstates:$

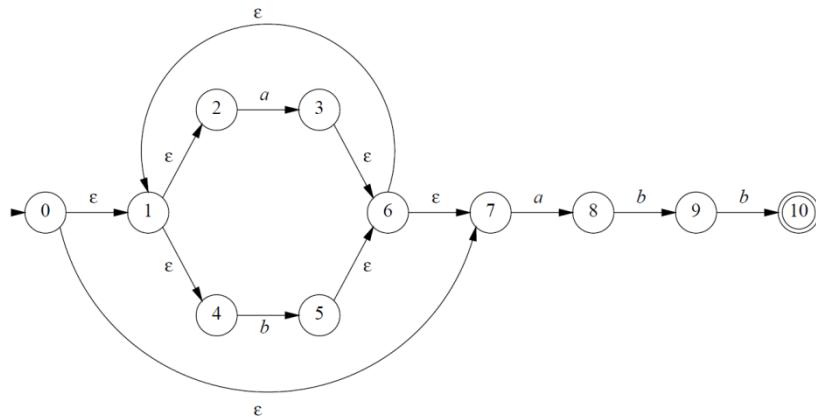
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}
- I E = {1, 2, 4, 5, 6, 7, 10}

- I $\epsilon\text{-closure}(\{3, 8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5, 9\}) = D$
- I $\epsilon\text{-closure}(\{5, 10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, b)) \\ &= \epsilon\text{-closure}(5) = C \end{aligned}$$

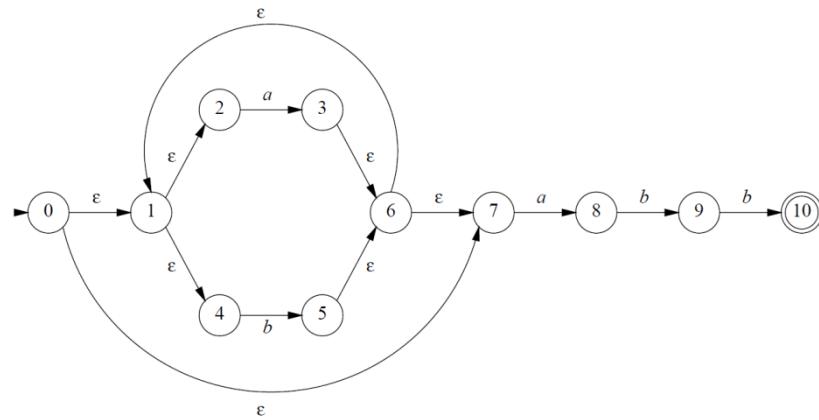
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

I A = {0,1,2,4,7}	I ε-closure({3,8}) = B
I B = {1,2,3,4,6,7,8}	I ε-closure({5}) = C
I C = {1,2,4,5,6,7}	I ε-closure({5,9}) = D
I D = {1,2,4,5,6,7,9}	I ε-closure({5,10}) = E
I E = {1,2,4,5,6,7,10}	

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, b)) \\ &= \epsilon\text{-closure}(5) = C \end{aligned}$$

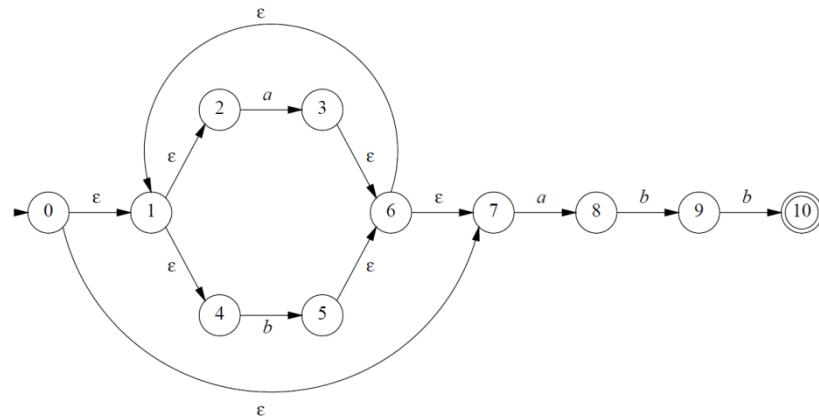
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

I A = {0,1,2,4,7}	I ε-closure({3,8}) = B
I B = {1,2,3,4,6,7,8}	I ε-closure({5}) = C
I C = {1,2,4,5,6,7}	I ε-closure({5,9}) = D
I D = {1,2,4,5,6,7,9}	I ε-closure({5,10}) = E
I E = {1,2,4,5,6,7,10}	

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
&= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, b)) \\
&= \epsilon\text{-closure}(5) = C
\end{aligned}$$

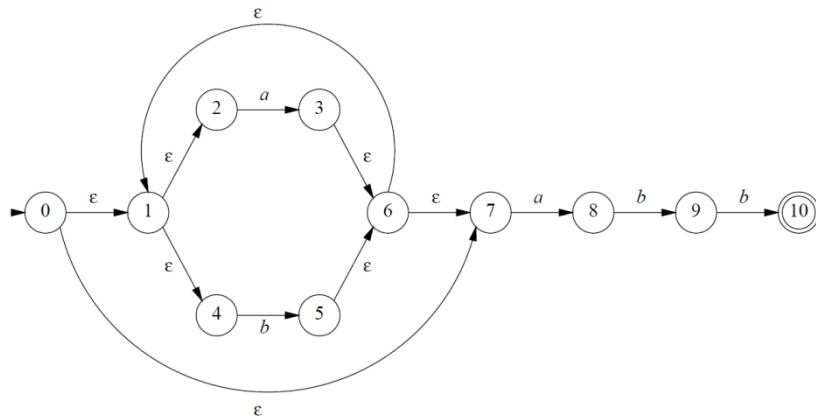
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

Cheat Sheet

- | | |
|--------------------------|---|
| $A = \{0,1,2,4,7\}$ | $\epsilon\text{-closure}(\{3,8\}) = B$ |
| $B = \{1,2,3,4,6,7,8\}$ | $\epsilon\text{-closure}(\{5\}) = C$ |
| $C = \{1,2,4,5,6,7\}$ | $\epsilon\text{-closure}(\{5,9\}) = D$ |
| $D = \{1,2,4,5,6,7,9\}$ | $\epsilon\text{-closure}(\{5,10\}) = E$ |
| $E = \{1,2,4,5,6,7,10\}$ | |



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

```

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned}
 U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
 &= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, b)) \\
 &= \epsilon\text{-closure}(5) = C
 \end{aligned}$$

$Dstates:$

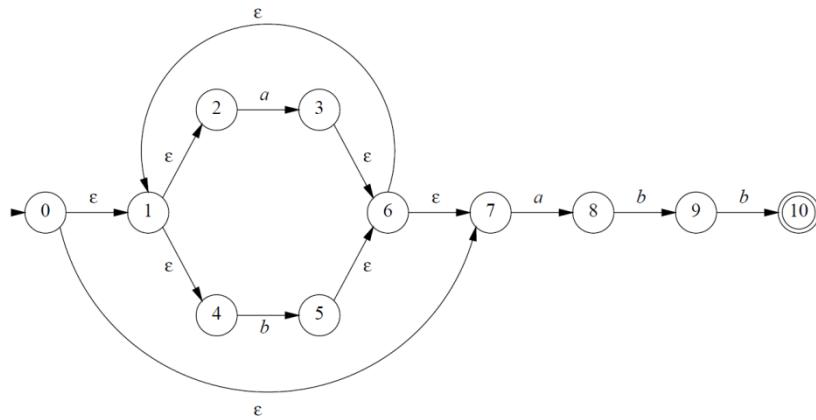
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0, 1, 2, 4, 7}
- I B = {1, 2, 3, 4, 6, 7, 8}
- I C = {1, 2, 4, 5, 6, 7}
- I D = {1, 2, 4, 5, 6, 7, 9}
- I E = {1, 2, 4, 5, 6, 7, 10}

- I $\epsilon\text{-closure}(\{3, 8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5, 9\}) = D$
- I $\epsilon\text{-closure}(\{5, 10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, b)) \\ &= \epsilon\text{-closure}(5) = C \end{aligned}$$

$Dstates:$

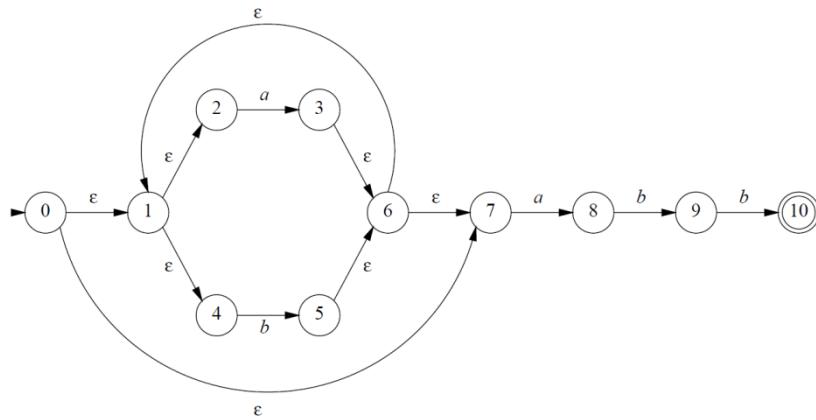
A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- I A = {0,1,2,4,7}
- I B = {1,2,3,4,6,7,8}
- I C = {1,2,4,5,6,7}
- I D = {1,2,4,5,6,7,9}
- I E = {1,2,4,5,6,7,10}

- I $\epsilon\text{-closure}(\{3,8\}) = B$
- I $\epsilon\text{-closure}(\{5\}) = C$
- I $\epsilon\text{-closure}(\{5,9\}) = D$
- I $\epsilon\text{-closure}(\{5,10\}) = E$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



```

add state  $T = \varepsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
  mark  $T$ 
  for each input symbol  $a$ 
     $U = \varepsilon\text{-closure}(\text{move}(T, a))$ 
    if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
     $Dtrans[T, a] = U$ 
  endfor
endwhile

```

$\varepsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned}
U &= \varepsilon\text{-closure}(\text{move}(T, b)) \\
&= \varepsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, b)) \\
&= \varepsilon\text{-closure}(5) = C
\end{aligned}$$

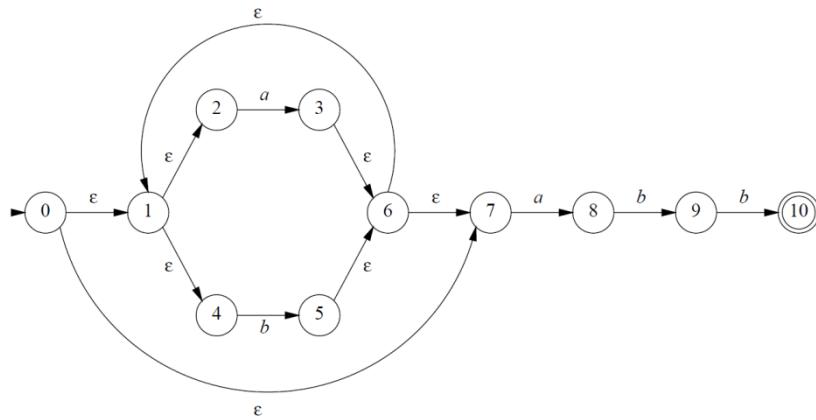
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

Cheat Sheet

- | | |
|--------------------------------|---|
| $A = \{0, 1, 2, 4, 7\}$ | $\varepsilon\text{-closure}(\{3, 8\}) = B$ |
| $B = \{1, 2, 3, 4, 6, 7, 8\}$ | $\varepsilon\text{-closure}(\{5\}) = C$ |
| $C = \{1, 2, 4, 5, 6, 7\}$ | $\varepsilon\text{-closure}(\{5, 9\}) = D$ |
| $D = \{1, 2, 4, 5, 6, 7, 9\}$ | $\varepsilon\text{-closure}(\{5, 10\}) = E$ |
| $E = \{1, 2, 4, 5, 6, 7, 10\}$ | |



```

add state  $T = \epsilon\text{-closure}(s_0)$  unmarked to  $Dstates$ 
while  $\exists$  unmarked state  $T$  in  $Dstates$ 
    mark  $T$ 
    for each input symbol  $a$ 
         $U = \epsilon\text{-closure}(\text{move}(T, a))$ 
        if  $U \notin Dstates$  then add  $U$  to  $Dstates$  unmarked
         $Dtrans[T, a] = U$ 
    endfor
endwhile

```

► $\epsilon\text{-closure}(s_0)$ is the start state of D
A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned}
U &= \epsilon\text{-closure}(\text{move}(T, b)) \\
&= \epsilon\text{-closure}(\text{move}(\{1, 2, 4, 5, 6, 7, 10\}, b)) \\
&= \epsilon\text{-closure}(5) = C
\end{aligned}$$

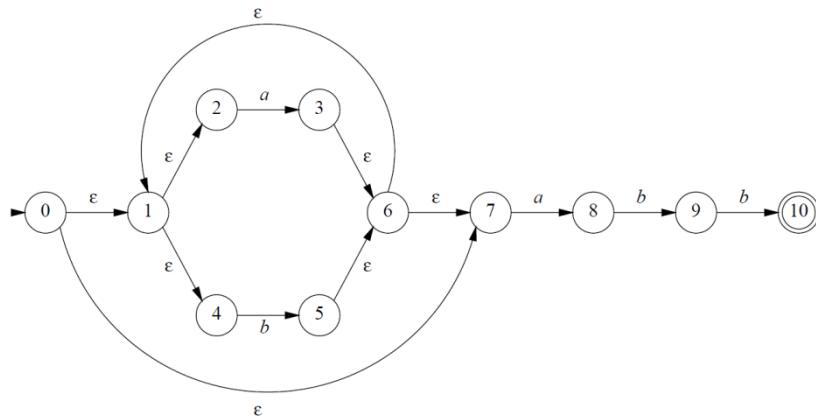
$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

- | | |
|------------------------------|--|
| I A = {0, 1, 2, 4, 7} | I $\epsilon\text{-closure}(\{3, 8\}) = B$ |
| I B = {1, 2, 3, 4, 6, 7, 8} | I $\epsilon\text{-closure}(\{5\}) = C$ |
| I C = {1, 2, 4, 5, 6, 7} | I $\epsilon\text{-closure}(\{5, 9\}) = D$ |
| I D = {1, 2, 4, 5, 6, 7, 9} | I $\epsilon\text{-closure}(\{5, 10\}) = E$ |
| I E = {1, 2, 4, 5, 6, 7, 10} | |

	a	b
>A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



add state $T = \epsilon\text{-closure}(s_0)$ unmarked to $Dstates$

while \exists unmarked state T in $Dstates$

mark T

for each input symbol a

$U = \epsilon\text{-closure}(\text{move}(T, a))$

if $U \notin Dstates$ **then** add U to $Dstates$ unmarked

$Dtrans[T, a] = U$

endfor

endwhile

$\epsilon\text{-closure}(s_0)$ is the start state of D

A state of D is accepting if it contains at least one accepting state in N

$T = E$

$$\begin{aligned} U &= \epsilon\text{-closure}(\text{move}(T, b)) \\ &= \epsilon\text{-closure}(\text{move}(\{1,2,4,5,6,7,10\}, b)) \\ &= \epsilon\text{-closure}(5) = C \end{aligned}$$

$Dstates:$

A	<input checked="" type="checkbox"/>
B	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
D	<input checked="" type="checkbox"/>
E	<input checked="" type="checkbox"/>

Cheat Sheet

I A = {0,1,2,4,7}	I ε-closure({3,8}) = B
I B = {1,2,3,4,6,7,8}	I ε-closure({5}) = C
I C = {1,2,4,5,6,7}	I ε-closure({5,9}) = D
I D = {1,2,4,5,6,7,9}	I ε-closure({5,10}) = E
I E = {1,2,4,5,6,7,10}	

	a	b
>A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



uOttawa

L'Université canadienne
Canada's university

SOMMAIRE

Expression Régulière:

(a | b)*abb

$$A = \{0, 1, 2, 4, 7\}$$

$$B = \{1, 2, 3, 4, 6, 7, 8\}$$

$$C = \{1, 2, 4, 5, 6, 7\}$$

$$D = \{1, 2, 4, 5, 6, 7, 9\}$$

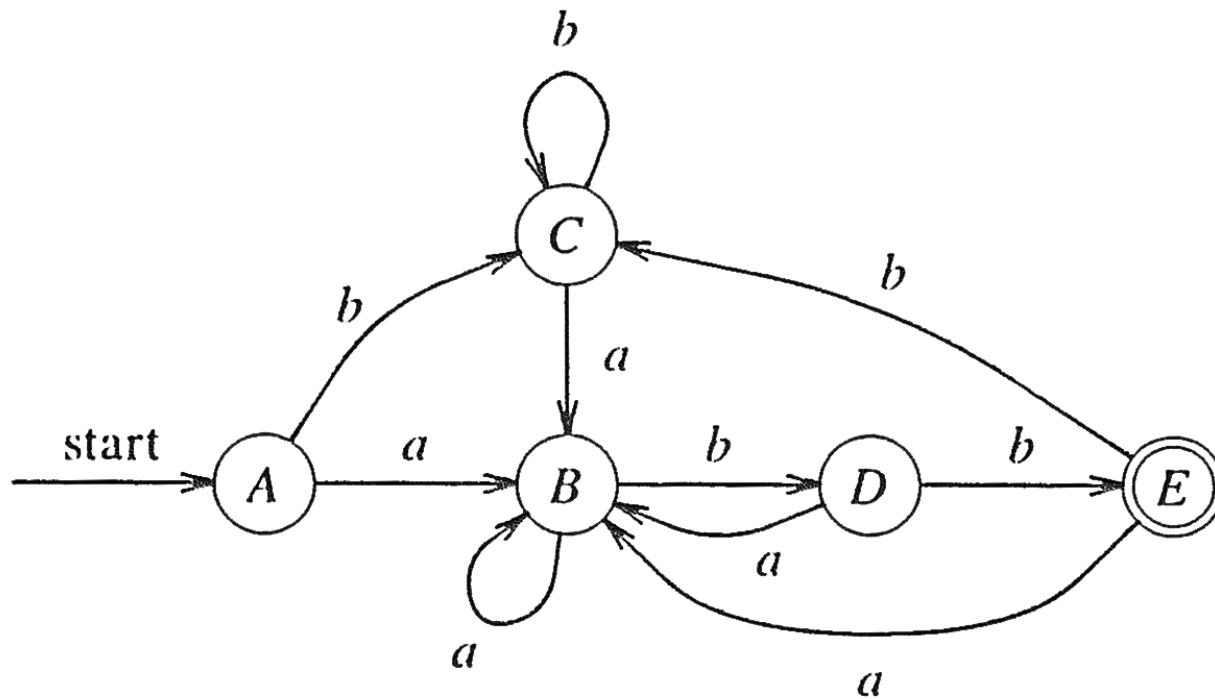
$$E = \{1, 2, 4, 5, 6, 7, 10\}$$

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

AFD RÉSULTANT

Expression Régulière:

$(a \mid b)^*abb$





uOttawa

L'Université canadienne
Canada's university

MINIMISER LE NOMBRE D'ÉTATS D'UN AFD

Minimiser le nombre d'états d'un AFD en trouvant tous les groupes d'états qui peuvent être distingués par une chaîne de caractères d'entrée

Chaque groupe d'états qui ne peut pas être distingué est fusionné en un état simple



uOttawa

L'Université canadienne
Canada's university

MINIMISER LE NOMBRE D'ÉTATS D'UN AFD

Algorithme: Minimiser le nombre d'états d'un AFD

Entrée: Un AFD “ D ” avec un ensemble d'états S

Sortie: Un AFD “ M ” qui accepte le même langage que “ D ”
mais ayant aussi peu d'états que possible



uOttawa

L'Université canadienne
Canada's university

MINIMISER LE NOMBRE D'ÉTATS D'UN AFD

Méthode:

1. Construire une partition initiale Π de l'ensemble d'états avec deux groupes:
 - Le groupe d'états acceptants (c'est-à-dire finaux)
 - Tous les autres groupes d'états
2. Partitionner Π en Π_{new} (en utilisant la procédure montrée dans la diapo suivante)
3. Si $\Pi_{\text{new}} \neq \Pi$, alors $\Pi = \Pi_{\text{new}}$ et répéter l'étape (2). Sinon, aller à l'étape (4)
4. Créer un seul état dans *AFD M* à partir de chaque groupe dans Π
5. Spécifier les états acceptants de *AFD M*. Un état d'acceptation dans *AFD M* correspond à un groupe dans Π qui contient un état d'acceptation dans *AFD D*
6. Spécifier l'état initial de *AFD M*. Un état initial dans *AFD M* correspond à un groupe dans Π qui contient un état initial dans *AFD D*

PROCÉDURE POUR CONSTRUIRE UNE NOUVELLE PARTITION

for chaque groupe G de Π **do begin**

Partitionnez G dans des sous-groupes tels que deux états s et t de G sont dans le même sous-groupe **si et seulement si**

for chaque symbole “ a ” $\in \Sigma$ **do begin**

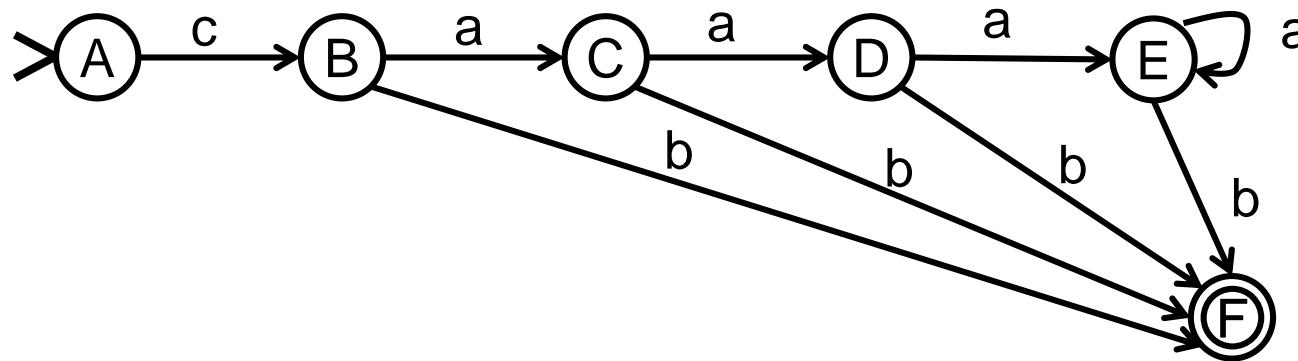
1. s et t n'ont pas des transitions sur “ a ”, **ou**
2. s et t ont des transitions sur “ a ” aux états dans le même groupe

endfor

Remplacez G dans Π_{new} par l'ensemble de tous les sous-groupes formés

endfor

EXEMPLE DE MINIMISATION DE AFD



$\Pi = \{$

- A, B, C, D, E
- F

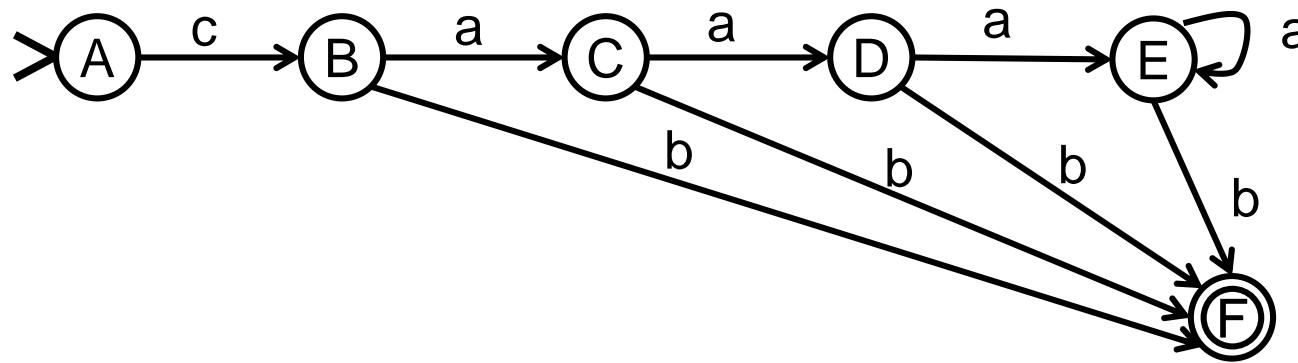
Pour **A** et **B**

- Transition “a” à partir de l'état **A** n'existe pas
- Transition “a” à partir de l'état **B** mène au groupe {A,B, C D, E}

Donc, **A** et **B** ne peuvent pas appartenir au même groupe dans Π_{new}

On arrive à la même conclusion concernant A et C, A et D, A et E

EXEMPLE DE MINIMISATION DE AFD



$\Pi = \left\{ \begin{array}{l} A, B, C, D, E \\ F \end{array} \right.$

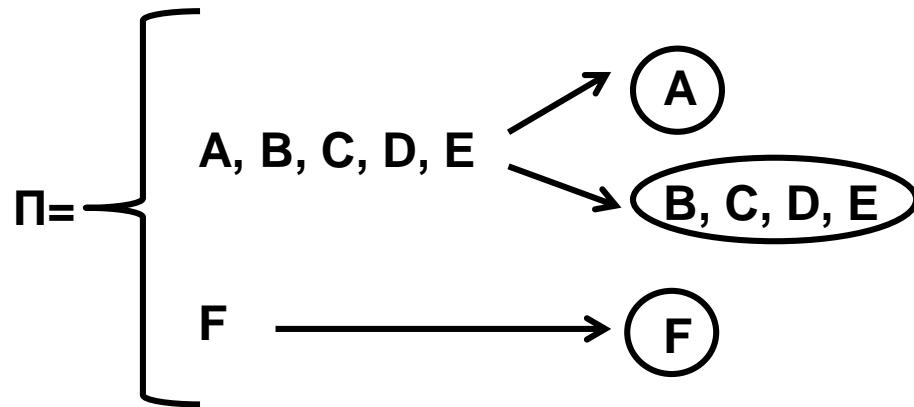
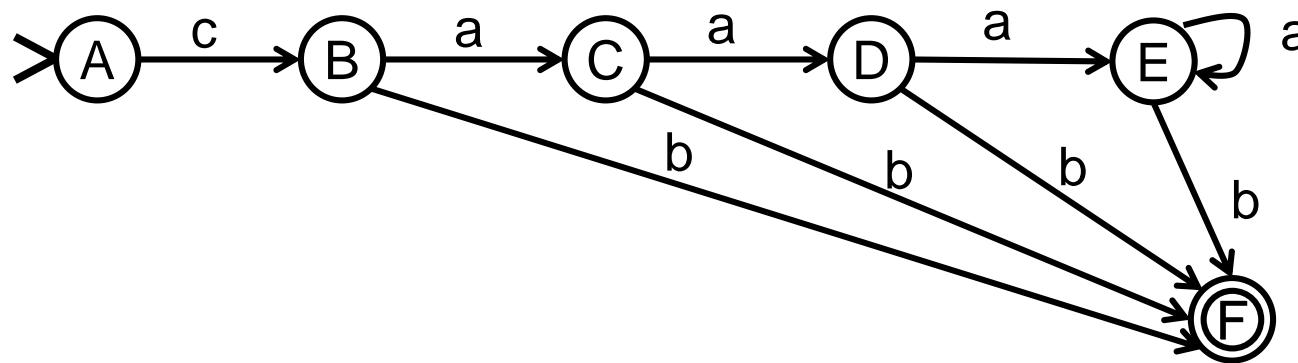
Pour **B** et **C**

- Transition “a” à partir de l’état **B** mène au groupe {A,B, C D, E}
- Transition “a” à partir de l’état **C** mène au groupe {A,B, C D, E}
- Transition “b” à partir de l’état **B** mène au groupe {F}
- Transition “b” à partir de l’état **C** mène au groupe {F}
- Transition “c” à partir de l’état **B** n’existe pas
- Transition “c” à partir de l’état **C** n’existe pas

Donc, **B** et **C** doivent appartenir au même groupe dans Π_{new}

On continue cette analyse pour toutes les paires d’états.

EXEMPLE DE MINIMISATION DE AFD





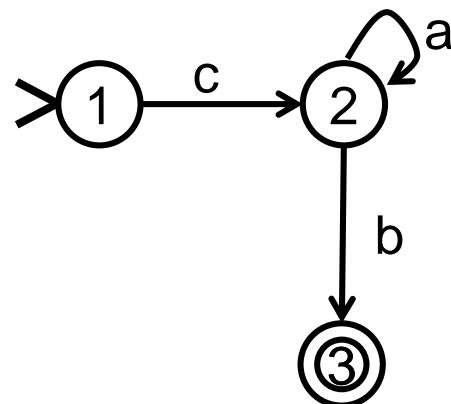
uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE MINIMISATION DE AFD

AFD minimisé, où:

- 1: A
- 2: B, C, D, E
- 3: F



MERCI!

QUESTIONS?

SÉANCE 10

**EXPRESSIONS RÉGULIÈRES
PRATIQUES**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Notations pratiques qui sont souvent utilisées dans les expressions régulières

Quelques exercices de pratique



uOttawa

L'Université canadienne
Canada's university

TRUCS POUR LES EXPRESSIONS RÉGULIÈRES PRATIQUES

Nous verrons des trucs pour les expressions régulières pratiques qui sont supportées par la plupart des librairies de regex

Souvenez-vous, les expressions régulières ne sont pas utilisées uniquement dans le contexte des compilateurs

- Nous les utilisons souvent pour extraire l'information d'un texte

Exemple: Imaginez que vous cherchez dans un fichier log qui accumule des entrées depuis deux mois pour un motif d'erreur particulier

- Sans les expressions régulières, ceci sera une tâche ennuyante



uOttawa

L'Université canadienne
Canada's university

MATCHER LES CHIFFRES

Afin de matcher un chiffre simple, comme on a vu précédemment, on peut utiliser l'expression régulière suivante:

[0-9]

Cependant, puisque l'opération de matcher un chiffre est une opération très commune, on peut utiliser la notation suivante:

\d

Un slash est un caractère d'échappement utilisé pour le distinguer de la lettre d

Similairement, afin de matcher un caractère non-chiffre, on peut utiliser la notation:

\D



uOttawa

L'Université canadienne
Canada's university

CARACTÈRES ALPHANUMÉRIQUES

Afin de matcher un caractère alphanumérique, on peut utiliser la notation:

[a-zA-Z0-9]

Ou on peut utiliser le raccourci suivant:

\w

Similairement, on peut représenter n'importe quel caractère non-alphanumérique comme suit:

\W



uOttawa

L'Université canadienne
Canada's university

MATCHER LES LETTRES

Vous pouvez matcher n'importe quel lettre de l'alphabet Français/Anglais:

[a-zA-Z]

Cependant, il existe de nombreux lettres définis dans Unicode au-delà des 26 lettres de l'alphabet Français

- Vous pouvez les matcher en utilisant:

\p{L}



uOttawa

L'Université canadienne
Canada's university

CARACTÈRE GÉNÉRIQUE (WILDCARD)

Un caractère générique est défini pour matcher n'importe quel caractère simple (lettre, chiffre, espace blanc ...)

Il est représenté par le caractère **.** (point)

Alors, afin de matcher un point, vous devez utiliser le caractère d'échappement: \.



uOttawa

L'Université canadienne
Canada's university

EXCLUSION

Nous avons déjà vu que [abc] est équivalent à (a | b | c)

Mais, parfois on veut matcher tout sauf un ensemble de caractères

Pour accomplir ceci, on peut utiliser la notation: [^abc]

- Ceci matche n'importe quel caractère simple sauf a, b ou c

Cette notation peut aussi être utilisée avec les classes de caractères abréviés

- [^a-z] matche n'importe quel caractère sauf les lettres minuscules



uOttawa

L'Université canadienne
Canada's university

RÉPÉTITIONS

Comment peut-on matcher une lettre ou une chaîne qui se répète plusieurs fois :

- Ex. ababab

Jusqu'à maintenant, nous avons implémenter des répétitions à travers trois mécanismes:

- Concaténation: simplement concaténer la chaîne ou le caractère avec lui-même (ne fonctionne pas si vous ne connaissez pas le nombre exact de répétitions)
- Fermeture étoile Kleene: pour matcher des lettres ou des chaînes répétés 0 ou plusieurs fois
- Fermeture positive: pour matcher des lettres ou des chaînes répétés 1 ou plusieurs fois



uOttawa

L'Université canadienne
Canada's university

RÉPÉTITIONS

On peut aussi spécifier une gamme de combien de fois une lettre ou une chaîne peuvent être répétés

Exemple, si on veut matcher des chaînes de répétition de la lettre « a » entre 1 et 3 fois, on peut utiliser la notation: **a {1,3}**

- Alors, **a {1,3}** matche les chaînes
 - a
 - aa
 - aaa

On peut aussi spécifier un nombre exact de répétitions au lieu d'une gamme

- **(ab) {3}** matche la chaîne **ababab**



uOttawa

L'Université canadienne
Canada's university

CARACTÈRES OPTIONNELS

Le concept du caractère optionnel est plus ou moins similaire à l'étoile kleene

- L'opérateur étoile matche **0 ou plus** cas de l'opérande
- L'opérateur optionnel, **représenté par ? (point d'interrogation)**, matche **0 ou un** cas de l'opérande

Exemple: le motif ab?c matchera soit les chaînes "abc" ou "ac" parce que b est considéré optionnel.



uOttawa

L'Université canadienne
Canada's university

ESPACE BLANC

Souvent, on veut facilement identifier les espaces blancs

- Soit pour les enlever ou pour identifier le début ou la fin de mots

Les espaces blanc les plus utilisés avec les expressions régulières:

- Espace _, le caractère de tabulation \t, la nouvelle ligne \n et le retour chariot \r

Un caractère spécial d'espace blanc \s matchera n'importe quel espace blanc spécifiez ci-haut

Similairement, vous pouvez matcher n'importe quel autre caractère en utilisant la notation \S



uOttawa

L'Université canadienne
Canada's university

MATCHER LE DÉBUT ET LA FIN D'UNE LIGNE

Vous pouvez spécifier qu'une expression régulière matche seulement le début ou à la fin de la ligne

Si un caret (^) est au début de toute l'expression régulière, il correspond au début d'une ligne

Si un signe dollar (\$) se trouve à la fin de toute l'expression régulière, il correspond à la fin d'une ligne

Si une expression régulière entière est entourée d'un caret et d'un signe dollar (^ c'est une phrase \$), elle correspond à une ligne entière



uOttawa

L'Université canadienne
Canada's university

MATCHER LE LIMITES DES MOTS

Des fois, c'est important de matcher les limites des mots

- Cela nous permet de distinguer les chaînes qui apparaissent au début d'un mot de celles qui apparaissent au milieu

Le début d'un mot est matché en utilisant \b

- Par exemple: `\bart` matche la chaîne “art” mais non “start”

La fin d'un mot est aussi matché par \b

- Par exemple: `art\b` matche la chaîne “start” mais non “arts”

De même, \B matche le milieu d'un mot

- Par exemple: `\Bart` matche la chaîne “start” mais non “art”

QUELQUES EXERCICES - 1

Étant donné la phrase:

- Error, computer will now shut down...

Développez une expression régulière qui matchera tous les mots dans la phrase

Réponse:

[A-Za-z]+

ou

\p{L}+

QUELQUES EXERCICES - 2

Étant donné la phrase:

- Error, computer will now shut down...

Développez une expression régulière qui matchera tous les caractères non-alphanumériques

Réponse:

\W+



uOttawa

L'Université canadienne
Canada's university

QUELQUES EXERCICES - 3

Étant donné le fichier log:

- [Sunday Feb. 2 2020] Program starting up
- [Monday Feb. 3 2020] Entered initialization phase
- [Tuesday Feb. 4 2020] Error 5: cannot open XML file
- [Thursday Feb. 6 2020] Warning 5: response time is too slow
- [Friday Feb. 7 2020] Error 9: major error occurred, system will shut down

Matcher n'importe quel message d'erreur (Error) ou d'alerte (Warning) qui termine par le terme “shut down”

Réponse:

(Error|Warning).*(shut down)



uOttawa

L'Université canadienne
Canada's university

QUELQUES EXERCICES - 4

Étant donné le fichier log:

- [Sunday Feb. 2 2020] Program starting up
- [Monday Feb. 3 2020] Entered initialization phase
- [Tuesday Feb. 4 2020] Error 5: cannot open XML file
- [Thursday Feb. 6 2020] Warning 5: response time is too slow
- [Friday Feb. 7 2020] Error 9: major error occurred, system will shut down

Matcher n'importe quel erreur ou alerte entre le 1 et 6 février

Réponse:

\[\w+ Feb\.\. [1-6] 2017\] (Error|Warning)



uOttawa

L'Université canadienne
Canada's university

QUELQUES EXERCICES - 4

Spécifiez l'expression régulière pour les mots de passe qui doivent commencer par une lettre suivie de caractères alphanumériques

Le mot de passe doit contenir au moins 8 caractères et un maximum de 15 caractères

\b[a-zA-Z]\w{7,14}\b

MERCI!

QUESTIONS?

SÉANCE 11

**INTRODUCTION À
L'ANALYSE SYNTAXIQUE**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Grammaires non-contextuelles

Dérivations

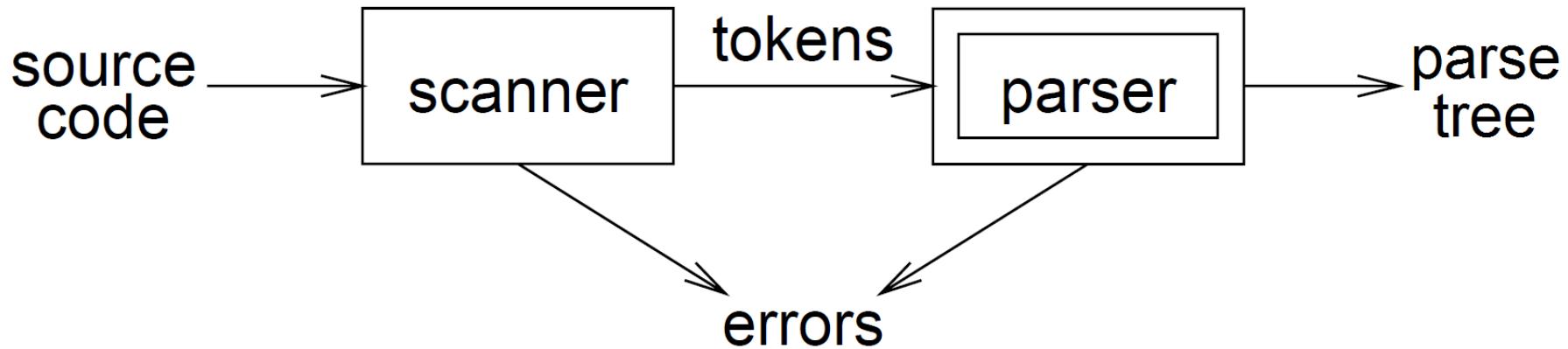
Arbres de parage

Ambiguité

Parseur descendant

Récursivité gauche

LE RÔLE DE L'ANALYSEUR SYNTAXIQUE



GRAMMAIRES NON-CONTEXTUELLES

Une grammaire non-contextuelles (GNC) se consiste de:

- Terminaux
- Non-terminaux
- Symbole de début
- Productions

Un langage qui peut être généré par une grammaire non-contextuelle est appelé un langage non-contextuel

GRAMMAIRES NON-CONTEXTUELLES

Terminaux: sont les symboles de base d'où les chaînes sont formées

- Ceux-ci sont les tokens qui ont été produits par l'Analyseur Lexical

Non-terminaux: sont des variables syntaxiques qui dénote une série de symboles de grammaire

- Un de ces non-terminaux est distingué comme étant le **symbole de départ**

Les productions d'une grammaire spécifient la manière dont le terminal et non-terminal peuvent être combinés afin de former des chaînes



uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE GRAMMAIRE

La grammaire avec les productions suivantes définit des expressions arithmétiques simples

```
<expr> ::= <expr> <op> <expr>
<expr> ::= id
<expr> ::= num
<op> ::= +
<op> ::= -
<op> ::= *
<op> ::= /
```

Dans cette grammaire, les symboles terminaux sont
*num, id + - * /*

Les symboles non-terminaux sont $\langle \text{expr} \rangle$ et $\langle \text{op} \rangle$, et $\langle \text{expr} \rangle$ est le symbole de départ

```
<expr> ::= <expr>  <op>  <expr>  
<expr> ::= id  
<expr> ::= num  
  <op> ::= +  
  <op> ::= -  
  <op> ::= *  
  <op> ::= /
```



uOttawa

L'Université canadienne
Canada's university

DÉRIVATIONS

$\langle \text{expr} \rangle \Rightarrow \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

est lue "expr derives expr op expr"

$$\begin{aligned}\langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \\ &\Rightarrow \text{id } \langle \text{op} \rangle \langle \text{expr} \rangle \\ &\Rightarrow \text{id } * \langle \text{expr} \rangle \\ &\Rightarrow \text{id*id}\end{aligned}$$

est appelée une **dérivation** de id*id d'après expr .



uOttawa

L'Université canadienne
Canada's university

DÉRIVATIONS

Si $A ::= \gamma$ est une production et α et β sont des chaînes arbitraires de symboles de grammaire, on peut dire que:

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

Si $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$, on dit que α_1 *dérive* α_n .



uOttawa

L'Université canadienne
Canada's university

DÉRIVATIONS

⇒ veut dire “dérive en une seule étape.”

*⇒ veut dire “dérive en zéro ou plusieurs étapes.”

- si $\alpha \xrightarrow{*} \beta$ et $\beta \Rightarrow \gamma$ donc $\alpha \xrightarrow{*} \gamma$

⊕⇒ veut dire “dérive en une ou plusieurs étapes.”

Si $S \xrightarrow{*} \alpha$, où α peut contenir des non-terminaux, alors on dit que α est en forme phrasique

- Si α ne contient pas des non-terminaux, on dit que α est une phrase



uOttawa

L'Université canadienne
Canada's university

DÉRIVATIONS

G: grammaire

S: symbole de départ

L(G): le langage généré par G

Les chaînes dans L(G) peuvent contenir uniquement des symboles terminaux de G

Une chaîne de terminaux w appartient à L(G) si et seulement si $S \xrightarrow{+} w$

- Comme on a déjà mentionné, la chaîne w est appelée une phrase de G

Un langage qui peut être généré par une grammaire est connu comme étant un langage non-contextuel

- Si deux grammaires génèrent le même langage, les grammaires sont considérées équivalentes



uOttawa

L'Université canadienne
Canada's university

DÉRIVATIONS

Nous avons déjà vu les règles de production suivantes:

$$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \mid \text{id} \mid \text{num}$$
$$\langle \text{op} \rangle ::= + \mid - \mid * \mid /$$

La chaîne **id+id** est une phrase de grammaire ci-haut parce que

$$\begin{aligned}\langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\ &\Rightarrow \text{id} \ \langle \text{op} \rangle \ \langle \text{expr} \rangle \\ &\Rightarrow \text{id} \ + \ \langle \text{expr} \rangle \\ &\Rightarrow \text{id} \ + \ \text{id}\end{aligned}$$

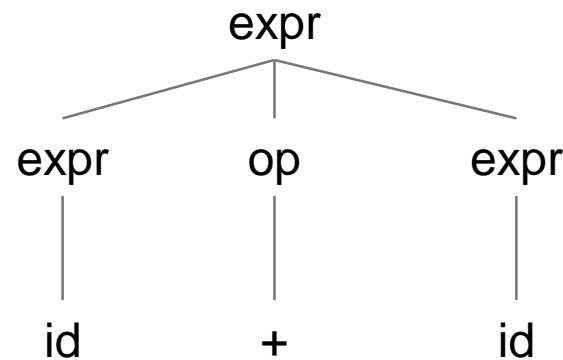
On écrit $\langle \text{expr} \rangle \stackrel{*}{\Rightarrow} \text{id+id}$



uOttawa

L'Université canadienne
Canada's university

ARBRE DE PARSAGE



Ceci est appelé:

Dérivation la plus à gauche



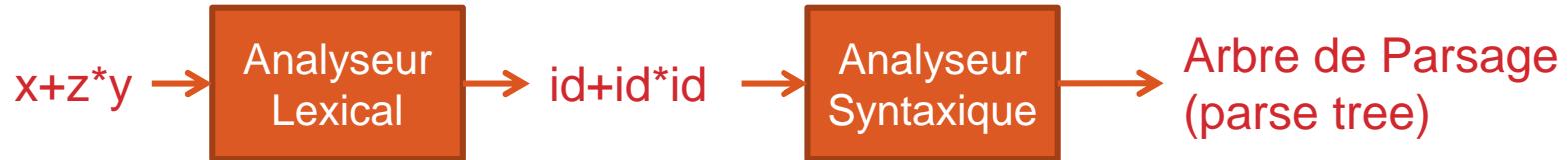
DEUX ARBRES DE PARSAGE

Considérons encore une fois l'expression arithmétique de grammaire.

Pour la ligne de code:

$x+2*y$

(nous ne considérons pas le point-virgule pour le moment)



Grammaire:

$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \text{ } \langle \text{op} \rangle \text{ } \langle \text{expr} \rangle \mid \text{id} \mid \text{num}$
 $\langle \text{op} \rangle ::= + \mid - \mid * \mid /$

DEUX ARBRES DE PARSAGE

Considérons encore une fois l'expression arithmétique de grammaire.

La phrase $\text{id} + \text{id} * \text{id}$ possède deux dérivations à gauche distinctes:

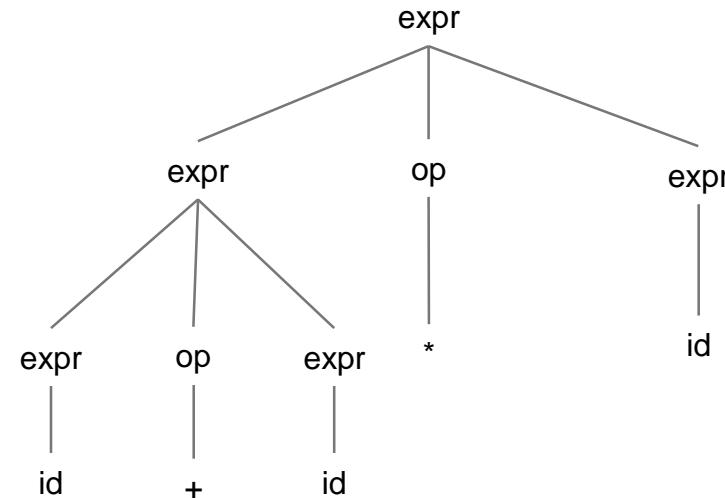
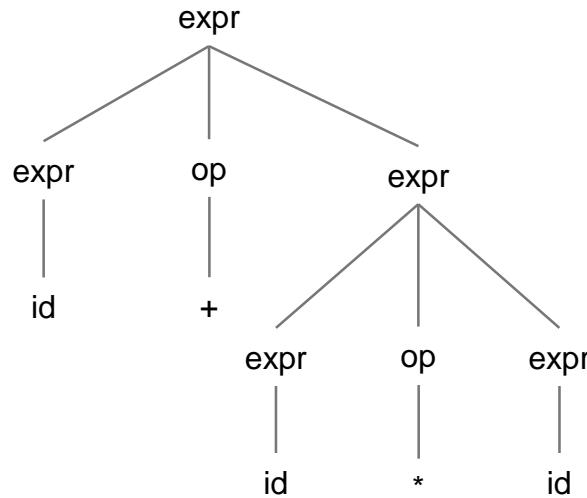
```
<expr>  => <expr> <op> <expr>
         => id <op> <expr>
         => id + <expr>
         => id + <expr> <op> <expr>
         => id + id <op> <expr>
         => id + id * <expr>
         => id + id * id
```

```
<expr>  => <expr> <op> <expr>
         => <expr> <op> <expr> <op> <expr>
         => id <op> <expr> <op> <expr>
         => id + <expr> <op> <expr>
         => id + id <op> <expr>
         => id + id * <expr>
         => id + id * id
```

Grammaire:

```
<expr> ::= <expr> <op> <expr> | id | num
<op> ::= + | - | * | /
```

DEUX ARBRES DE PARSAGE



Equivalent à:
 $\text{id} + (\text{id} * \text{id})$

Grammaire:

$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle \text{ } \langle \text{op} \rangle \text{ } \langle \text{expr} \rangle \mid \text{id} \mid \text{num}$
 $\langle \text{op} \rangle ::= + \mid - \mid * \mid /$

Equivalent à:
 $(\text{id} + \text{id})^* \text{id}$ X



uOttawa

L'Université canadienne
Canada's university

PRÉCÉDENCE

L'exemple précédent surligne un problème dans la grammaire:

- Elle n'impose pas la précédence
- Elle n'implique pas l'ordre d'évaluation

On peut améliorer les règles de production pour ajouter la précédence

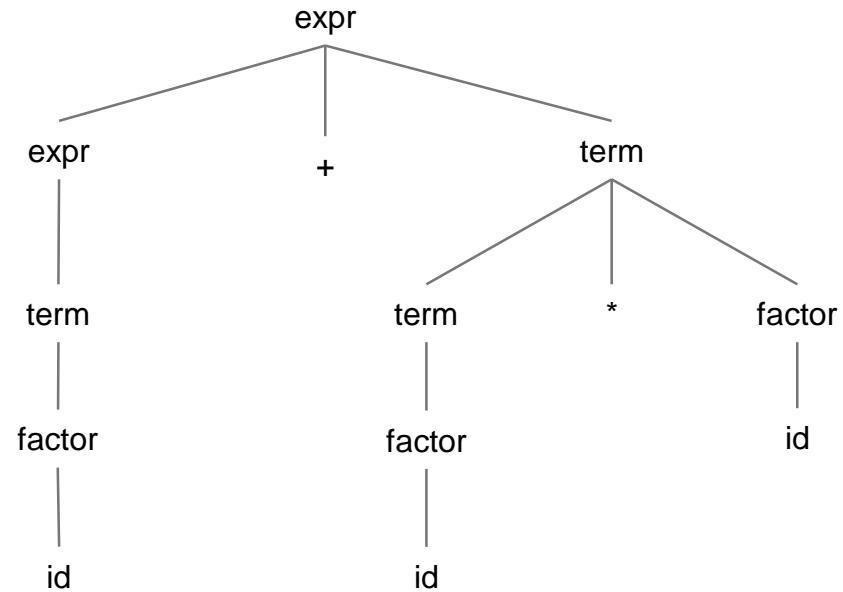
$$\begin{array}{lcl} \langle \text{expr} \rangle & ::= & \langle \text{expr} \rangle + \langle \text{term} \rangle \\ & | & \langle \text{expr} \rangle - \langle \text{term} \rangle \\ & | & \langle \text{term} \rangle \\ \langle \text{term} \rangle & ::= & \langle \text{term} \rangle * \langle \text{factor} \rangle \\ & | & \langle \text{term} \rangle / \langle \text{factor} \rangle \\ & | & \langle \text{factor} \rangle \\ \langle \text{factor} \rangle & ::= & \text{num} \\ & | & \text{id} \end{array}$$



APPLIQUER LA MISE À JOUR DE PRÉCÉDENCE

La phrase $\text{id} + \text{id} * \text{id}$ possède seulement une seule dérivation à gauche maintenant:

```
<expr>  ::= <expr> + <term>
          ::= <term> + <term>
          ::= <factor> + <term>
          ::= id + <term>
          ::= id + <term> * <factor>
          ::= id + <factor> * <factor>
          ::= id + id * <factor>
          ::= id + id * id
```



Grammaire:

```
<expr>      ::= <expr> + <term> | <expr> - <term> | <term>
<term>      ::= <term> * <factor> | <term> / <factor> | <factor>
<factor>    ::= num | id
```



uOttawa

L'Université canadienne
Canada's university

AMBIGUITÉ

Une grammaire qui produit plus qu'un arbre de parsage pour une phrase est considérée d'être *ambiguë*

Exemple:

```
<stmt> ::= if <expr>then <stmt>
          | if <expr>then <stmt>else <stmt>
          | other stmts
```

Considérez la déclaration suivante:

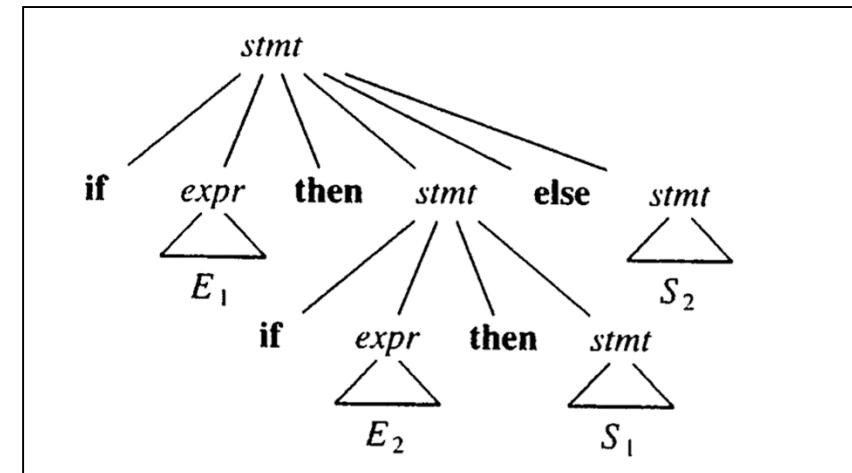
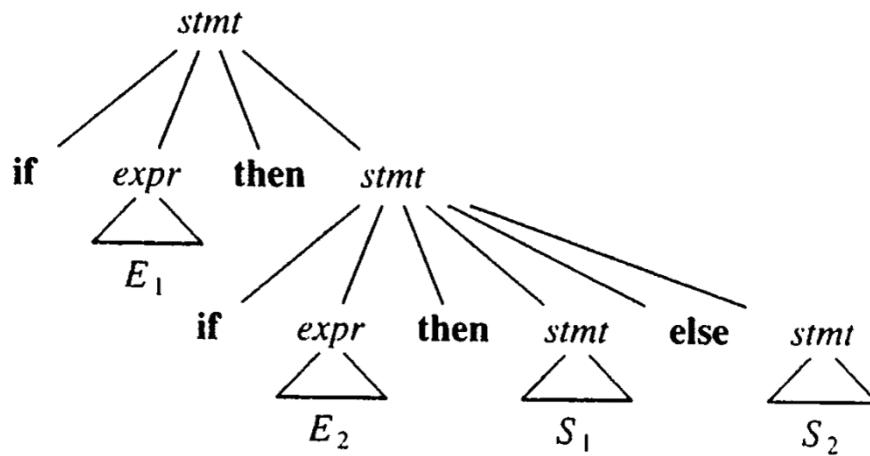
$$\text{if } E_1 \text{ then if } E_2 \text{ then } S_1 \text{ else } S_2$$

Elle possède deux dérivations

- C'est une ambiguïté non-contextuelle

AMBIGUITÉ

Une grammaire qui produit plus qu'un arbre de parsage pour une phrase est considérée d'être *ambiguë*





uOttawa

L'Université canadienne
Canada's university

ÉLIMINER L'AMBIGUITÉ

Parfois, une grammaire ambiguë peut être réécrite afin d'éliminer l'ambiguité.

- Ex. “matcher chaque « else » avec « then » le plus proche”
- Ceci est probablement l'intention du programmeur

```
⟨stmt⟩      ::=  ⟨matched⟩  
            |  ⟨unmatched⟩  
⟨matched⟩   ::=  if ⟨expr⟩ then ⟨matched⟩ else ⟨matched⟩  
            |  other stmts  
⟨unmatched⟩ ::=  if ⟨expr⟩ then ⟨stmt⟩  
            |  if ⟨expr⟩ then ⟨matched⟩ else ⟨unmatched⟩
```



uOttawa

L'Université canadienne
Canada's university

APPLICATION DANS JAVA

Dans Java, les règles de grammaire sont un peu différentes de l'exemple précédent

Une version (très simplifiée) de ces règles est présentée ci-dessous

```
<stmt>      ::= <matched>
              | <unmatched>

<matched>    ::= if ( <expr> ) <matched> else <matched>
              | other stmts

<unmatched>  ::= if ( <expr> ) <stmt>
              | if ( <expr> ) <matched> else <unmatched>
```



uOttawa

L'Université canadienne
Canada's university

APPLICATION DANS JAVA

Pour la portion de code suivante

```
if (x==0)  
  
    if (y==0)  
  
        z = 0;  
  
    else  
  
        z = 1;
```

Après avoir exécuter l'analyseur lexical, nous obtenons la liste de tokens suivante:

```
if ( id == num ) if (id == num) id = num ; else id = num ;
```

EXEMPLE JAVA

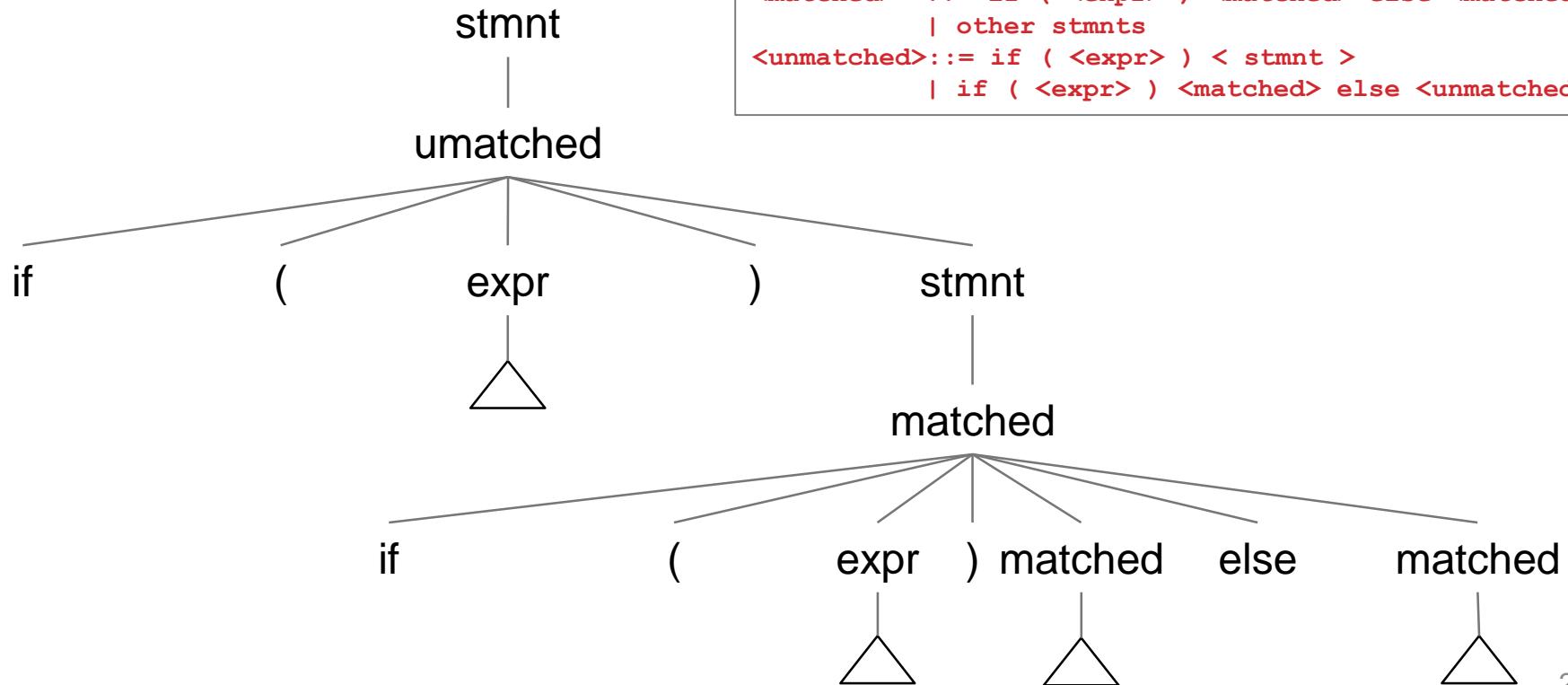
Chaîne de tokens

```
if ( id == num ) if (id == num) id = num ; else id = num ;
```

Grammaire

```

<stmtnt>      ::= <matched>
                  | <unmatched>
<matched>     ::= if ( <expr> ) <matched> else <matched>
                  | other stmtnts
<unmatched> ::= if ( <expr> ) < stmtnt >
                  | if ( <expr> ) <matched> else <unmatched>
```





uOttawa

L'Université canadienne
Canada's university

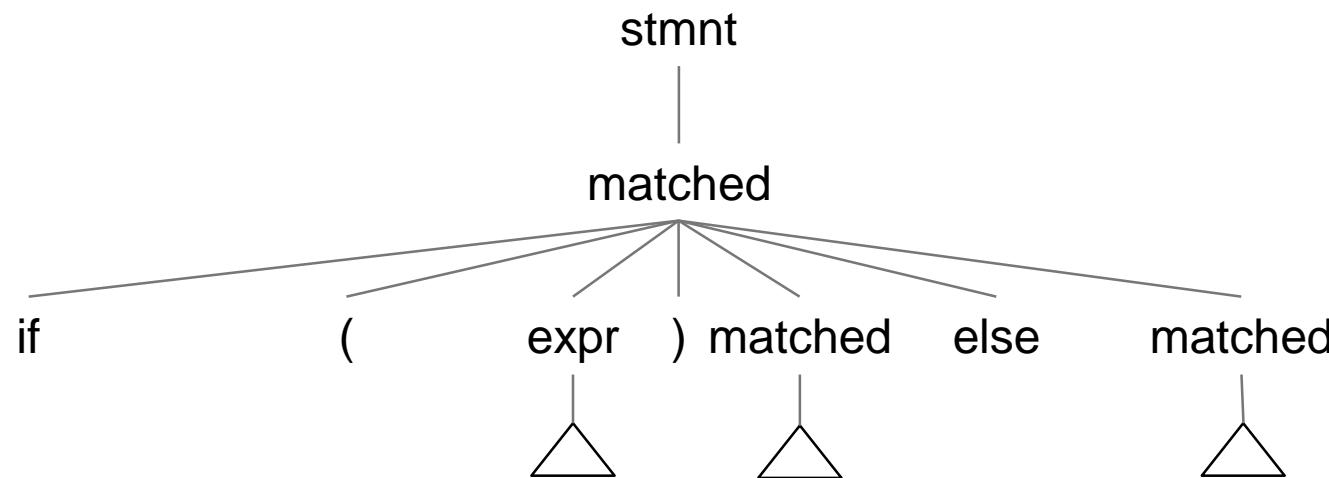
UN AUTRE EXEMPLE JAVA

Chaîne de tokens

```
if ( id == num ) else id = num ;
```

Grammaire

```
<stmt>      ::= <matched>
              | <unmatched>
<matched>   ::= if ( <expr> ) <matched> else <matched>
              | other stmts
<unmatched> ::= if ( <expr> ) <stmt>
              | if ( <expr> ) <matched> else <unmatched>
```





uOttawa

L'Université canadienne
Canada's university

PARSEUR DESCENDANT

Un parseur descendant commence avec la racine de l'arbre de parage, marquée avec le symbole de départ

Pour construire un arbre de parage, on répète les étapes suivantes jusqu'à ce que les feuilles de l'arbre de parage matches la chaîne de données

1. À un nœud marqué par A , sélectionnez une production $A ::= \alpha$ et construisez l'enfant approprié pour chaque symbole de α
2. Lorsqu'un terminal ajouté à l'arbre de parage ne matche pas la chaîne de données, reculer
3. Trouvez le non-terminal suivant à être développer



uOttawa

L'Université canadienne
Canada's university

PARSEUR DESCENDANT

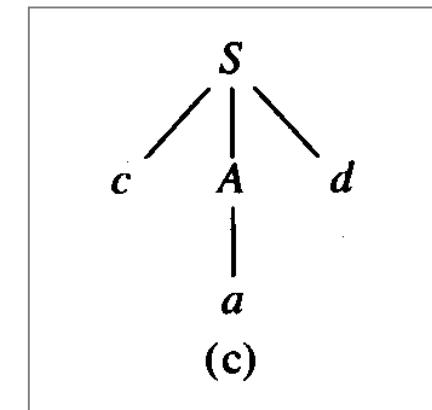
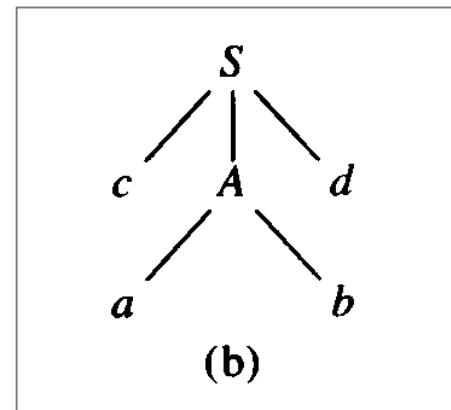
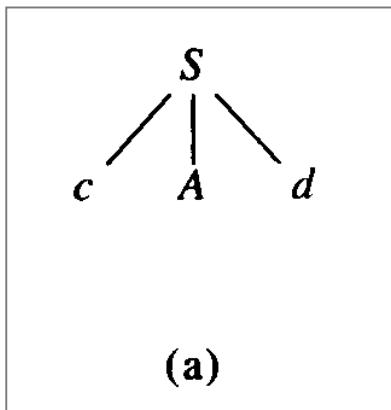
Le parsage descendant peut être considéré comme une tentative pour trouver une dérivation à gauche pour une chaîne de données

Exemple:

Grammaire:

$\langle S \rangle ::= c \langle A \rangle d$
 $\langle A \rangle ::= ab \mid a$

Chaîne de données
cad



On a besoin de reculer!

GRAMMAIRE DE L'EXPRESSION

Rappelez-vous de notre grammaire pour les expressions simples:

```
 $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle$ 
|    $\langle \text{expr} \rangle - \langle \text{term} \rangle$ 
|    $\langle \text{term} \rangle$ 
 $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$ 
|    $\langle \text{term} \rangle / \langle \text{factor} \rangle$ 
|    $\langle \text{factor} \rangle$ 
 $\langle \text{factor} \rangle ::= \text{num}$ 
|    $\text{id}$ 
```

Considérez la chaîne de données:

`id - num * id`

EXAMPLE



Prod'n	Sentential form	Input
1	$\langle \text{expr} \rangle$	$\uparrow \text{id} - \text{num} * \text{id}$

Reference Grammar

```
 $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle$ 
          |  $\langle \text{expr} \rangle - \langle \text{term} \rangle$ 
          |
          |  $\langle \text{term} \rangle$ 
 $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$ 
          |  $\langle \text{term} \rangle / \langle \text{factor} \rangle$ 
          |
          |  $\langle \text{factor} \rangle$ 
 $\langle \text{factor} \rangle ::= \text{num}$ 
          |
          |  $\text{id}$ 
```

EXAMPLE



Reference Grammar

```

<expr> ::= <expr> + <term>
          | <expr> - <term>
          | <term>
<term> ::= <term> * <factor>
          | <term> / <factor>
          | <factor>
<factor> ::= num
           | id
    
```

Prod'n	Sentential form	Input
1	<expr>	↑ id - num * id
2	<expr> + <term>	↑ id - num * id
4	<term> + <term>	↑ id - num * id
7	<factor> + <term>	↑ id - num * id
9	id + <term>	↑ id - num * id
-	id + <term>	id ↑ - num * id
-	<expr>	↑ id - num * id
3	<expr> - <term>	↑ id - num * id
4	<term> - <term>	↑ id - num * id
7	<factor> - <term>	↑ id - num * id
9	id - <term>	↑ id - num * id
-	id - <term>	id ↑ - num * id
-	id - <term>	id - ↑num * id
7	id - <factor>	id - ↑num * id
8	id - num	id - ↑num * id
-	id - num	id - num ↑ * id
-	id - <term>	id - ↑num * id
5	id - <term> * <factor>	id - ↑num * id
7	id - <factor> * <factor>	id - ↑num * id
8	id - num * <factor>	id - ↑num * id
-	id - num * <factor>	id - num ↑ * id
-	id - num * <factor>	id - num * ↑id
9	id - num * id	id - num * ↑id
-	id - num * id	id - num * id ↑



EXEMPLE

Une autre analyse syntaxique possible pour $\text{id} - \text{num} * \text{id}$

Prod'n	Sentential form	Input	Reference Grammar
1	$\langle \text{expr} \rangle$	$\uparrow \text{id} - \text{num} * \text{id}$	$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle$
2	$\langle \text{expr} \rangle + \langle \text{term} \rangle$	$\uparrow \text{id} - \text{num} * \text{id}$	$ \quad \langle \text{expr} \rangle - \langle \text{term} \rangle$
2	$\langle \text{expr} \rangle + \langle \text{term} \rangle + \langle \text{term} \rangle$	$\uparrow \text{id} - \text{num} * \text{id}$	$ \quad \langle \text{term} \rangle$
2	$\langle \text{expr} \rangle + \langle \text{term} \rangle + \dots$	$\uparrow \text{id} - \text{num} * \text{id}$	$\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$
2	$\langle \text{expr} \rangle + \langle \text{term} \rangle + \dots$	$\uparrow \text{id} - \text{num} * \text{id}$	$ \quad \langle \text{term} \rangle / \langle \text{factor} \rangle$
2	\dots	$\uparrow \text{id} - \text{num} * \text{id}$	$ \quad \langle \text{factor} \rangle$
			$\langle \text{factor} \rangle ::= \text{num}$
			$ \quad \text{id}$

Si l'analyseur syntaxique fait les mauvais choix, l'expansion ne se termine pas

- Ceci n'est pas une bonne propriété pour l'analyseur syntaxique
- Les analyseurs syntaxiques devraient se terminer, finalement...



uOttawa

L'Université canadienne
Canada's university

RÉCURSIVITÉ À GAUCHE

Une grammaire est récursive à gauche si:

“Elle possède un non-terminal **A** de façon qu'il existe une dérivation $A \xrightarrow{+} A\alpha$ pour une chaîne **α** ”

Les parseurs descendants avec dérivation à gauche ne peuvent pas traiter la récursivité à gauche dans une grammaire



uOttawa

L'Université canadienne
Canada's university

ÉLIMINER LA RÉCURSIVITÉ À GAUCHE

Considérez le fragments de grammaire:

$$\begin{array}{lcl} \langle \text{foo} \rangle & ::= & \langle \text{foo} \rangle \alpha \\ & | & \beta \end{array}$$

Où α et β ne commence pas par $\langle \text{foo} \rangle$

On peut réécrire ceci comme:

$$\begin{array}{lcl} \langle \text{foo} \rangle & ::= & \beta \langle \text{bar} \rangle \\ \langle \text{bar} \rangle & ::= & \alpha \langle \text{bar} \rangle \\ & | & \varepsilon \end{array}$$

Où $\langle \text{bar} \rangle$ est un nouveau non-terminal

Ce Fragment ne contient pas de récursivité à gauche



uOttawa

L'Université canadienne
Canada's university

EXAMPLE

Notre grammaire d'expressions contient deux cas de récursivité à gauche

```
 $\langle \text{expr} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle$ 
|  $\langle \text{expr} \rangle - \langle \text{term} \rangle$ 
|  $\langle \text{term} \rangle$ 
 $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$ 
|  $\langle \text{term} \rangle / \langle \text{factor} \rangle$ 
|  $\langle \text{factor} \rangle$ 
```

Appliquer la transformation nous donne

```
 $\langle \text{expr} \rangle ::= \langle \text{term} \rangle \langle \text{expr}' \rangle$ 
 $\langle \text{expr}' \rangle ::= + \langle \text{term} \rangle \langle \text{expr}' \rangle$ 
|  $- \langle \text{term} \rangle \langle \text{expr}' \rangle$ 
|  $\epsilon$ 
 $\langle \text{term} \rangle ::= \langle \text{factor} \rangle \langle \text{term}' \rangle$ 
 $\langle \text{term}' \rangle ::= * \langle \text{factor} \rangle \langle \text{term}' \rangle$ 
|  $/ \langle \text{factor} \rangle \langle \text{term}' \rangle$ 
|  $\epsilon$ 
```

Avec cette grammaire, un parseur descendant va

- Se terminer (assurément)
- Reculer pour quelques données

ALGORITHMES PRÉDICTIFS

Nous avons vu que les parseurs descendants peuvent reculer lorsqu'ils choisissent la mauvaise production

Conséquemment, nous devons utiliser les algorithmes prédictifs afin d'éviter de reculer

Les parseur prédictifs sont utiles

- LL(1): scanner de gauche à droite, dérivation gauche, 1-token de prélecture (look ahead)
- LR(1): scanner de gauche à droite, dérivation droite, 1-token de prélecture (look ahead)

MERCI!

QUESTIONS?

SÉANCE 12

PARSEUR LL(1)



uOttawa

L'Université canadienne
Canada's university

SUJETS

Grammaire LL(1)

Éliminer la récursivité à gauche

Factorisation gauche

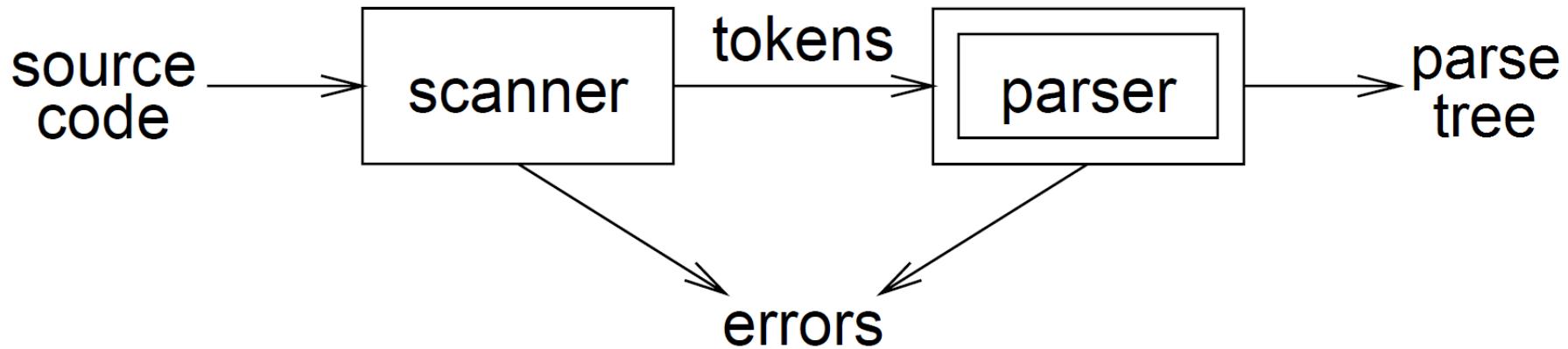
Les ensembles FIRST et FOLLOW

Tableaux syntaxiques

Parseur LL(1)

Plusieurs exemples...

RÉVISION: LE RÔLE DU PARSEUR





uOttawa

L'Université canadienne
Canada's university

PARSEURS PRÉDICTIFS

Nous avons vu que les parseurs descendants ont besoin de reculer lorsqu'ils choisissent la mauvaise production

Nous voulons éviter de reculer.

Les parseurs prédictifs sont utiles dans ce cas

- LL(1): scanning de gauche à droite, dérivation à gauche, 1-token de prélecture (look ahead)
- LR(1): scanning de gauche à droite, dérivation à droite, 1-token de prélecture (look ahead)

GRAMMAIRE LL(1)

Afin d'utiliser les parseurs LL(1), la grammaire non-contextuelle doit être:

- Non ambiguë (nous avons déjà discuté de l'ambiguïté)
- Sans récursivité à gauche (nous avons déjà discuté de l'élimination de la récursivité à gauche)
- Factorisée à gauche (nous allons discuter de la factorisation gauche aujourd'hui)



Les méthodes ci-hauts convertissent plusieurs grammaires en forme LL(1) mais pas toutes... Il existe plusieurs exceptions.

RÉVISION: RÉCURSIVITÉ À GAUCHE

Une grammaire est récursive à gauche si:

“Elle possède un non-terminal **A** de sorte qu'il existe une dérivation $A \xrightarrow{+} A\alpha$ pour une chaîne α ”

Les parseurs descendants ne peuvent pas traiter la récursivité à gauche dans une grammaire



uOttawa

L'Université canadienne
Canada's university

ÉLIMINER LA RÉCURSIVITÉ À GAUCHE

Considérez le fragment de grammaire:

$$\begin{array}{lcl} \langle \text{foo} \rangle & ::= & \langle \text{foo} \rangle \alpha \\ & | & \beta \end{array}$$

Où α et β ne commencent pas par $\langle \text{foo} \rangle$

On peut réécrire ceci de la façon suivante:

$$\begin{array}{lcl} \langle \text{foo} \rangle & ::= & \beta \langle \text{bar} \rangle \\ \langle \text{bar} \rangle & ::= & \alpha \langle \text{bar} \rangle \\ & | & \varepsilon \end{array}$$

Où $\langle \text{bar} \rangle$ est un non-terminal

Ce fragment ne contient aucune récursivité à gauche



uOttawa

L'Université canadienne
Canada's university

FACTORIZATION GAUCHE

Pour toutes deux productions $A \rightarrow \alpha \mid \beta$, on veut avoir une façon distincte pour choisir la bonne production à étendre

On définit **FIRST(α)** comme étant l'ensemble de terminaux qui apparaissent au début d'une chaîne de symboles quelconque dérivée de α

Pour un terminal w , on peut dire que:

$$w \in \text{FIRST}(\alpha) \text{ iff } \alpha \xrightarrow{*} wz$$



uOttawa

L'Université canadienne
Canada's university

FACTORIZATION GAUCHE

Maintenant, retournons à nos deux productions:
 $A \rightarrow \alpha$ et $A \rightarrow \beta$, on veut:

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$$

Ceci permet le parseur de faire le bon choix avec une vue d'avance d'un seul symbole



uOttawa

L'Université canadienne
Canada's university

FACTORIZATION GAUCHE

Étant donné cette grammaire:

1	$\langle \text{expr} \rangle$	$::=$	$\langle \text{term} \rangle + \langle \text{expr} \rangle$
2			$\langle \text{term} \rangle - \langle \text{expr} \rangle$
3			$\langle \text{term} \rangle$
4	$\langle \text{term} \rangle$	$::=$	$\langle \text{factor} \rangle * \langle \text{term} \rangle$
5			$\langle \text{factor} \rangle / \langle \text{term} \rangle$
6			$\langle \text{factor} \rangle$
7	$\langle \text{factor} \rangle$	$::=$	num
8			id

Le parseur ne peut pas choisir entre les productions 1, 2 et 3 étant donné un token d'entrée de num ou id

$$\text{FIRST}(1) \cap \text{FIRST}(2) \cap \text{FIRST}(3) \neq \emptyset$$

La factorisation gauche est nécessaire pour résoudre ce problème!





uOttawa

L'Université canadienne
Canada's university

FACTORIZATION GAUCHE

Alors, comment ça fonctionne?

Pour chaque non-terminal **A**, trouvez le plus long préfixe α commun à deux de ses alternatives ou plus

Si $\alpha \neq \epsilon$, alors remplacez toutes les productions A

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \dots \mid \alpha\beta_n$$

Par

$$A \rightarrow \alpha \ A'$$

$$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \dots \mid \beta_n$$

Où A' est un nouveau non-terminal

Répétez jusqu'à ce qu'aucune deux alternatives pour un non-terminal simple possède un préfixe commun



uOttawa

L'Université canadienne
Canada's university

FACTORISATION GAUCHE

Donc, dans notre grammaire:

$$\begin{aligned}\langle \text{expr} \rangle & ::= \langle \text{term} \rangle + \langle \text{expr} \rangle \\ & | \quad \langle \text{term} \rangle - \langle \text{expr} \rangle \\ & | \quad \langle \text{term} \rangle\end{aligned}$$
$$\begin{aligned}\langle \text{term} \rangle & ::= \langle \text{factor} \rangle * \langle \text{term} \rangle \\ & | \quad \langle \text{factor} \rangle / \langle \text{term} \rangle \\ & | \quad \langle \text{factor} \rangle\end{aligned}$$

Lorsqu'on effectue la factorisation gauche (sur expr et term), on obtient:

$$\begin{aligned}\langle \text{expr} \rangle & ::= \langle \text{term} \rangle \langle \text{expr}' \rangle \\ \langle \text{expr}' \rangle & ::= + \langle \text{expr} \rangle \\ & | \quad - \langle \text{expr} \rangle \\ & | \quad \epsilon\end{aligned}$$
$$\begin{aligned}\langle \text{term} \rangle & ::= \langle \text{factor} \rangle \langle \text{term}' \rangle \\ \langle \text{term}' \rangle & ::= * \langle \text{term} \rangle \\ & | \quad / \langle \text{term} \rangle \\ & | \quad \epsilon\end{aligned}$$



uOttawa

L'Université canadienne
Canada's university

PARSEUR LL(1)

On sait maintenant comment prendre une grammaire non-contextuelle et la transformer en grammaire LL(1) (au moins on peut essayer...)





uOttawa

L'Université canadienne
Canada's university

PARSEUR LL(1)

On veut implémenter un parseur LL(1) qui est capable d'analyser le syntaxe d'une chaîne de donnée de tokens sans reculer

- Évidemment, étant donné que la grammaire est compatible avec ce parseur

Afin d'effectuer ceci, on doit trouver deux ensembles pour chaque non-terminal:

- FIRST (on a déjà parlé de cet ensemble brièvement)
- FOLLOW

CALCUL DE L'ENSEMBLE FIRST

Règles pour calculer l'ensemble FIRST:

1. FIRST (terminal) $\rightarrow \{ \text{terminal} \}$

2. Si $A \rightarrow a\alpha$, et a est un terminal:

$$\{ a \} \in \text{FIRST}(A)$$

3. Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ N'EXISTE PAS:

$$\text{FIRST}(B) \in \text{FIRST}(A)$$

4. Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ EXISTE:

$$\{ (\text{FIRST}(B) - \epsilon) \cup \text{FIRST}(\alpha) \} \in \text{FIRST}(A)$$

CALCUL DE L'ENSEMBLE FIRST

Appliquons ces règles à un exemple.

Étant donné la grammaire:

$\langle S \rangle ::= \langle X \rangle \langle Y \rangle$

$\langle X \rangle ::= a$

$\langle Y \rangle ::= b$

$\text{FIRST}(X) = \{a\}$ (on applique la 2^e règle)

$\text{FIRST}(Y) = \{b\}$ (on applique la 2^e règle)

$\text{FIRST}(S) = \text{FIRST}(X)$

$= \{a\}$ (on applique la 3^e règle)

1) $\text{FIRST}(\text{terminal}) \rightarrow \{\text{terminal}\}$

2) Si $A \rightarrow a$, et a est un terminal:
 $\{a\} \in \text{FIRST}(A)$

3) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ N'EXISTE PAS:
 $\text{FIRST}(B) \in \text{FIRST}(A)$

4) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ EXISTE:
 $\{(\text{FIRST}(B) - \epsilon) \cup \text{FIRST}(\alpha)\} \in \text{FIRST}(A)$

CALCUL DE L'ENSEMBLE FIRST

Un autre exemple...

Étant donné la grammaire:

$\langle S \rangle ::= \langle X \rangle \langle Y \rangle$
 $\langle X \rangle ::= a \mid \epsilon$
 $\langle Y \rangle ::= b$

$\text{FIRST}(X) = \{a, \epsilon\}$ (2^e règle)

$\text{FIRST}(Y) = \{b\}$ (2^e règle)

$\text{FIRST}(S) = [\text{FIRST}(X) - \epsilon] \cup \text{FIRST}(Y)$
 $= \{a, b\}$ (4^e règle)

- 1) $\text{FIRST}(\text{terminal}) \rightarrow \{\text{terminal}\}$
- 2) Si $A \rightarrow a\alpha$, et a est un terminal:
 $\{a\} \in \text{FIRST}(A)$
- 3) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ **N'EXISTE PAS**:
 $\text{FIRST}(B) \in \text{FIRST}(A)$
- 4) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \epsilon$ **EXISTE**:
 $\{(\text{FIRST}(B) - \epsilon) \cup \text{FIRST}(\alpha)\} \in \text{FIRST}(A)$



uOttawa

L'Université canadienne
Canada's university

ENSEMBLE FOLLOW

Définition informelle:

*L'ensemble **Follow** d'un non-terminal **A** (c'est-à-dire **Follow(A)**) contient tous les **terminaux** qui apparaissent APRÈS **A** dans n'importe quelle chaîne générée par la grammaire **G**.*

CALCUL DE L'ENSEMBLE FOLLOW

Règles pour calculer l'ensemble FOLLOW:

1. $\{ \$ \} \in \text{FOLLOW}(S)$ (où S est le symbole de départ)

2. Si $A \rightarrow \alpha B$:

$$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$$

3. Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \varepsilon$ N'EXISTE PAS:

$$\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$$

4. Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \varepsilon$ EXISTE:

$$\{ (\text{FIRST}(\gamma) - \varepsilon) \cup \text{FOLLOW}(A) \} \in \text{FOLLOW}(B)$$

CALCUL DE L'ENSEMBLE FOLLOW

Appliquons ces règles à un exemple.

Étant donné la grammaire:

$\langle S \rangle ::= \langle X \rangle \langle Y \rangle$

$\langle X \rangle ::= a$

$\langle Y \rangle ::= b$

$\text{FOLLOW}(S) = \{\$ \}$ (1^e règle)

$\text{FOLLOW}(X) = \text{FIRST}(Y)$
 $= \{b\}$ (3^e règle)

$\text{FOLLOW}(Y) = \text{FOLLOW}(S)$
 $= \{\$ \}$ (2^e règle)

1) $\{ \$ \} \in \text{FOLLOW}(S)$

2) Si $A \rightarrow \alpha B$:

$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$

3) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ N'EXISTE PAS:
 $\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$

4) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ EXISTE:
 $\{ (\text{FIRST}(\gamma) - \epsilon) \cup \text{FOLLOW}(A) \} \in \text{FOLLOW}(B)$

CALCUL DE L'ENSEMBLE FOLLOW

Un autre exemple...

Étant donné la grammaire:

$\langle S \rangle ::= \langle X \rangle \langle Y \rangle$

$\langle X \rangle ::= a$

$\langle Y \rangle ::= b \mid \epsilon$

$\text{FOLLOW}(S) = \{\$\}$ (1^e règle)

1) $\{\$\} \in \text{FOLLOW}(S)$

2) Si $A \rightarrow \alpha B$:

$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$

3) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ N'EXISTE PAS:
 $\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$

4) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ EXISTE:
 $\{ (\text{FIRST}(\gamma) - \epsilon) \cup \text{FOLLOW}(A) \} \in \text{FOLLOW}(B)$

$\text{FOLLOW}(X) = [\text{FIRST}(Y) - \epsilon] \cup \text{FOLLOW}(S)$
 $= \{b, \$\}$ (4^e règle)

$\text{FOLLOW}(Y) = \text{FOLLOW}(S)$
 $= \{\$\}$ (2^e règle)



FIRST ET FOLLOW

Calculons **FIRST** et **FOLLOW** pour chaque non-terminal dans notre fameuse grammaire:

Trouvez ceux qui sont faciles en premier:

$\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

$\text{FIRST}(\text{term}') = \{\ast, /, \varepsilon\}$

$\text{FIRST}(\text{expr}') = \{+, -, \varepsilon\}$

Ensuite, trouvez ceux qui sont plus difficiles:

$\text{FIRST}(\text{term}) = \text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

$\text{FIRST}(\text{expr}) = \text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$

Grammaire	
$\langle \text{expr} \rangle$	$::= \langle \text{term} \rangle \langle \text{expr}' \rangle$
$\langle \text{expr}' \rangle$	$::= + \langle \text{expr} \rangle$
	$- \langle \text{expr} \rangle$
	ε
$\langle \text{term} \rangle$	$::= \langle \text{factor} \rangle \langle \text{term}' \rangle$
$\langle \text{term}' \rangle$	$::= * \langle \text{term} \rangle$
	$/ \langle \text{term} \rangle$
	ε
$\langle \text{factor} \rangle$	$::= \text{num}$
	id

“Règles” de FIRST

- 1) $\text{FIRST}(\text{terminal}) \rightarrow \{\text{terminal}\}$
- 2) Si $A \rightarrow a\alpha$, et a est un terminal:
 $\{a\} \in \text{FIRST}(A)$
- 3) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \varepsilon$ N'EXISTE PAS:
 $\text{FIRST}(B) \in \text{FIRST}(A)$
- 4) Si $A \rightarrow B\alpha$, et la règle $B \rightarrow \varepsilon$ EXISTE:
 $\{(\text{FIRST}(B) - \varepsilon) \cup \text{FIRST}(\alpha)\} \in \text{FIRST}(A)$



FIRST ET FOLLOW

Calculons **FIRST** et **FOLLOW** pour chaque non-terminal dans notre fameuse grammaire:

Commencez avec ceux qui sont faciles:

$$\text{FOLLOW}(\text{expr}) = \{ \$ \}$$

$$\text{FOLLOW}(\text{expr}') = \text{FOLLOW}(\text{expr}) = \{ \$ \}$$

Grammaire	
$\langle \text{expr} \rangle$	$::= \langle \text{term} \rangle \langle \text{expr}' \rangle$
$\langle \text{expr}' \rangle$	$::= +\langle \text{expr} \rangle$ $ -\langle \text{expr} \rangle$ $ \epsilon$
$\langle \text{term} \rangle$	$::= \langle \text{factor} \rangle \langle \text{term}' \rangle$
$\langle \text{term}' \rangle$	$::= *\langle \text{term} \rangle$ $ /\langle \text{term} \rangle$ $ \epsilon$
$\langle \text{factor} \rangle$	$::= \text{num}$ $ \text{id}$

“Règles” de FOLLOW

1) $\{ \$ \} \in \text{FOLLOW}(S)$

2) Si $A \rightarrow \alpha B$:

$$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$$

3) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ N’EXISTE PAS:

$$\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$$

4) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \epsilon$ EXISTE:

$$\{ (\text{FIRST}(\gamma) - \epsilon) \cup \text{FOLLOW}(A) \} \in \text{FOLLOW}(B)$$



FIRST ET FOLLOW

Calculons **FIRST** et **FOLLOW** pour chaque non-terminal dans notre fameuse grammaire:

Trouvez ceux qui sont plus difficiles:

$$\text{FOLLOW}(\text{term}) = [\text{FIRST}(\text{expr}') - \varepsilon] \cup \text{FOLLOW}(\text{expr})$$

$$= \{ +, -, \$ \}$$

$$\text{FOLLOW}(\text{factor}) = [\text{FIRST}(\text{term}') - \varepsilon] \cup \text{FOLLOW}(\text{term})$$

$$= \{ *, /, +, -, \$ \}$$

$$\text{FOLLOW}(\text{term}') = \text{FOLLOW}(\text{term})$$

$$= \{ +, -, \$ \}$$

Grammaire	
$\langle \text{expr} \rangle$	$::= \langle \text{term} \rangle \langle \text{expr}' \rangle$
$\langle \text{expr}' \rangle$	$::= + \langle \text{expr} \rangle$ $- \langle \text{expr} \rangle$ ε
$\langle \text{term} \rangle$	$::= \langle \text{factor} \rangle \langle \text{term}' \rangle$
$\langle \text{term}' \rangle$	$::= * \langle \text{term} \rangle$ $/ \langle \text{term} \rangle$ ε
$\langle \text{factor} \rangle$	$::= \text{num}$ id

“Règles” de FOLLOW

1) $\{ \$ \} \in \text{FOLLOW}(S)$

2) Si $A \rightarrow \alpha B$:

$$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$$

3) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \varepsilon$ N’EXISTE PAS:

$$\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$$

4) Si $A \rightarrow \alpha B \gamma$, et la règle $\gamma \rightarrow \varepsilon$ EXISTE:

$$\{ (\text{FIRST}(\gamma) - \varepsilon) \cup \text{FOLLOW}(A) \} \in \text{FOLLOW}(B)$$



uOttawa

L'Université canadienne
Canada's university

FIRST ET FOLLOW

Résumé:

$\text{FIRST(expr)} = \{\text{num}, \text{id}\}$

$\text{FIRST(expr')} = \{+, -, \epsilon\}$

$\text{FIRST(term)} = \{\text{num}, \text{id}\}$

$\text{FIRST(term')} = \{*, /, \epsilon\}$

$\text{FIRST(factor)} = \{\text{num}, \text{id}\}$

$\text{FOLLOW(expr)} = \{\$\}$

$\text{FOLLOW(expr')} = \{\$\}$

$\text{FOLLOW(term)} = \{+, -, \$\}$

$\text{FOLLOW(term')} = \{+, -, \$\}$

$\text{FOLLOW(factor)} = \{*, /, +, -, \$\}$

En utilisant ces ensembles, on construit un tableau syntaxique

- Ce tableau syntaxique est nécessaire pour effectuer l'analyse syntaxique LL(1)



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

```
FIRST(expr) = {num, id}
FIRST(expr') = {+, -, ε}
FIRST(term) = {num, id}
FIRST(term') = {*, /, ε}
FIRST(factor) = {num, id}
```

FOLLOW Sets

```
FOLLOW(expr) = { $ }
FOLLOW(expr') = { $ }
FOLLOW(term) = { +, -, $ }
FOLLOW(term') = { +, -, $ }
FOLLOW(factor) = { *, /, +, -, $ }
```

Grammar

```
 $\langle \text{expr} \rangle ::= \langle \text{term} \rangle \langle \text{expr}' \rangle$ 
 $\langle \text{expr}' \rangle ::= +\langle \text{expr} \rangle$ 
 $\quad |$ 
 $\quad -\langle \text{expr} \rangle$ 
 $\quad |$ 
 $\quad \epsilon$ 
 $\langle \text{term} \rangle ::= \langle \text{factor} \rangle \langle \text{term}' \rangle$ 
 $\langle \text{term}' \rangle ::= *\langle \text{term} \rangle$ 
 $\quad |$ 
 $\quad /\langle \text{term} \rangle$ 
 $\quad |$ 
 $\quad \epsilon$ 
 $\langle \text{factor} \rangle ::= \text{num}$ 
 $\quad |$ 
 $\quad \text{id}$ 
```

On ajoute deux données associées avec num et id

	num	id	+	-	*	/	\$
expr							
expr'							
term							
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

```
FIRST(expr) = {num, id}
FIRST(expr') = {+, -, ε}
FIRST(term) = {num, id}
FIRST(term') = {*, /, ε}
FIRST(factor) = {num, id}
```

FOLLOW Sets

```
FOLLOW(expr) = { $ }
FOLLOW(expr') = { $ }
FOLLOW(term) = {+, -, $}
FOLLOW(term') = {+, -, $}
FOLLOW(factor) = {*, /, +, -, $}
```

Grammar

```
 $\langle \text{expr} \rangle ::= \langle \text{term} \rangle \langle \text{expr}' \rangle$ 
 $\langle \text{expr}' \rangle ::= + \langle \text{expr} \rangle$ 
 $\quad |$ 
 $\quad - \langle \text{expr} \rangle$ 
 $\quad |$ 
 $\quad \epsilon$ 
 $\langle \text{term} \rangle ::= \langle \text{factor} \rangle \langle \text{term}' \rangle$ 
 $\langle \text{term}' \rangle ::= * \langle \text{term} \rangle$ 
 $\quad |$ 
 $\quad / \langle \text{term} \rangle$ 
 $\quad |$ 
 $\quad \epsilon$ 
 $\langle \text{factor} \rangle ::= \text{num}$ 
 $\quad |$ 
 $\quad \text{id}$ 
```

On ajoute deux entrées associées avec num et id

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'							
term							
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +
          | -
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *
          | /
          | ε
⟨factor⟩ ::= num
           | id
    
```

Remplir le expr' de la même façon

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'							
term							
term'							
factor							



TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Remplir le expr' de la même façon

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$			
term							
term'							
factor							



TABLEAU SYNTAXIQUE

FIRST Sets

FIRST(expr) = {num, id}
FIRST(expr') = {+, -, ε}
FIRST(term) = {num, id}
FIRST(term') = {*, /, ε}
FIRST(factor) = {num, id}

FOLLOW Sets

FOLLOW(expr) = { \$ }
FOLLOW(expr') = { \$ }
FOLLOW(term) = {+, -, \$}
FOLLOW(term') = {+, -, \$}
FOLLOW(factor) = {*, /, +, -, \$}

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Qu'en est-il de epsilon? On utilise l'ensemble FOLLOW pour ajouter une règle epsilon...

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$			
term							
term'							
factor							



TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Qu'en est-il de epsilon? On utilise l'ensemble FOLLOW pour ajouter une règle epsilon...

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'				$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$		$\langle \text{expr}' \rangle \rightarrow \epsilon$
term							
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

$\langle \text{expr} \rangle ::=$	$\langle \text{term} \rangle \langle \text{expr}' \rangle$
$\langle \text{expr}' \rangle ::=$	$+ \langle \text{expr} \rangle$
	$- \langle \text{expr} \rangle$
	$ $
	ϵ
$\langle \text{term} \rangle ::=$	$\langle \text{factor} \rangle \langle \text{term}' \rangle$
$\langle \text{term}' \rangle ::=$	$* \langle \text{term} \rangle$
	$/ \langle \text{term} \rangle$
	$ $
	ϵ
$\langle \text{factor} \rangle ::=$	num
	$ $
	id

Pas d' ϵ , on utilise l'ensemble FIRST

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'				$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$		$\langle \text{expr}' \rangle \rightarrow$ ϵ
term							
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Pas d'epsilon, on utilise l'ensemble FIRST

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'				$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$		$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$					
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Celle-ci possède un epsilon, on utilise les ensembles FIRST et FOLLOW

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$			$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$					
term'							
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

$\langle \text{expr} \rangle ::=$	$\langle \text{term} \rangle \langle \text{expr}' \rangle$
$\langle \text{expr}' \rangle ::=$	$+ \langle \text{expr} \rangle$
	$- \langle \text{expr} \rangle$
	$ $
	ϵ
$\langle \text{term} \rangle ::=$	$\langle \text{factor} \rangle \langle \text{term}' \rangle$
$\langle \text{term}' \rangle ::=$	$* \langle \text{term} \rangle$
	$/ \langle \text{term} \rangle$
	$ $
	ϵ
$\langle \text{factor} \rangle ::=$	num
	$ $
	id

Celle-ci possède un epsilon, on utilise les ensembles FIRST et FOLLOW

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$			$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$					
term'			$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ ϵ
factor							



TABLEAU SYNTAXIQUE

FIRST Sets

FIRST(expr) = { num, id }
FIRST(expr') = { +, -, ε }
FIRST(term) = { num, id }
FIRST(term') = { *, /, ε }
FIRST(factor) = { num, id }

FOLLOW Sets

FOLLOW(expr) = { \$ }
FOLLOW(expr') = { \$ }
FOLLOW(term) = { +, -, \$ }
FOLLOW(term') = { +, -, \$ }
FOLLOW(factor) = { *, /, +, -, \$ }

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Remplir la rangée pour factor...

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$			$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$					
term'			$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ ϵ
factor							



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST}(\text{expr}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{expr}') = \{+, -, \epsilon\}$
 $\text{FIRST}(\text{term}) = \{\text{num}, \text{id}\}$
 $\text{FIRST}(\text{term}') = \{*, /, \epsilon\}$
 $\text{FIRST}(\text{factor}) = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW}(\text{expr}) = \{\$\}$
 $\text{FOLLOW}(\text{expr}') = \{\$\}$
 $\text{FOLLOW}(\text{term}) = \{+, -, \$\}$
 $\text{FOLLOW}(\text{term}') = \{+, -, \$\}$
 $\text{FOLLOW}(\text{factor}) = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

Remplir la rangée pour factor...

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$					
expr'			$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$			$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$					
term'			$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ ϵ
factor	factor \rightarrow num	factor \rightarrow id					



TABLEAU SYNTAXIQUE

FIRST Sets

$\text{FIRST(expr)} = \{\text{num}, \text{id}\}$
 $\text{FIRST(expr')} = \{+, -, \epsilon\}$
 $\text{FIRST(term)} = \{\text{num}, \text{id}\}$
 $\text{FIRST(term')} = \{*, /, \epsilon\}$
 $\text{FIRST(factor)} = \{\text{num}, \text{id}\}$

FOLLOW Sets

$\text{FOLLOW(expr)} = \{\$\}$
 $\text{FOLLOW(expr')} = \{\$\}$
 $\text{FOLLOW(term)} = \{+, -, \$\}$
 $\text{FOLLOW(term')} = \{+, -, \$\}$
 $\text{FOLLOW(factor)} = \{*, /, +, -, \$\}$

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | −⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= ∗⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

On ajoute des tirets aux cellules restantes pour indiquer:
aucun entrées

	num	id	+	-	*	/	\$
expr	$<\text{expr}> \rightarrow$ $<\text{term}> <\text{expr}'>$	$<\text{expr}> \rightarrow$ $<\text{term}> <\text{expr}'>$					
expr'			$<\text{expr}'> \rightarrow$ $+ <\text{expr}>$	$<\text{expr}'> \rightarrow$ $- <\text{expr}>$			$<\text{expr}'> \rightarrow$ ϵ
term	$<\text{term}> \rightarrow$ $<\text{factor}> <\text{term}'>$	$<\text{term}> \rightarrow$ $<\text{factor}> <\text{term}'>$					
term'			$<\text{term}'> \rightarrow$ ϵ	$<\text{term}'> \rightarrow$ ϵ	$<\text{term}'> \rightarrow$ $* <\text{term}>$	$<\text{term}'> \rightarrow$ $/ <\text{term}>$	$<\text{term}'> \rightarrow$ ϵ
factor	factor \rightarrow num	factor \rightarrow id					



uOttawa

L'Université canadienne
Canada's university

TABLEAU SYNTAXIQUE

FIRST Sets

FIRST(expr) = { num, id }
FIRST(expr') = { +, -, ε }
FIRST(term) = { num, id }
FIRST(term') = { *, /, ε }
FIRST(factor) = { num, id }

FOLLOW Sets

FOLLOW(expr) = { \$ }
FOLLOW(expr') = { \$ }
FOLLOW(term) = { +, -, \$ }
FOLLOW(term') = { +, -, \$ }
FOLLOW(factor) = { *, /, +, -, \$ }

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +⟨expr⟩
          | -⟨expr⟩
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *⟨term⟩
          | /⟨term⟩
          | ε
⟨factor⟩ ::= num
           | id
    
```

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow$ ϵ
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ ϵ	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ ϵ
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

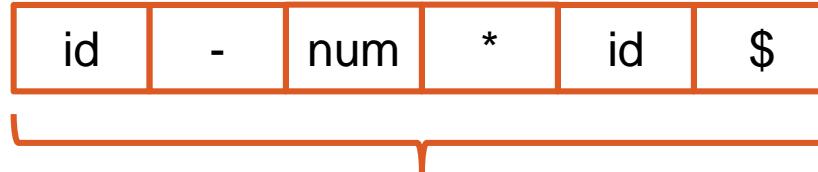
ANALYSE SYNTAXIQUE UTILISANT LL(1)

Afin d'implémenter un parseur LL(1), on doit utiliser les structures de données suivantes:

- Tableau syntaxique (peut être implémenté avec un tableau 2D ou autres structures plus sophistiquées)
- Pile (qui contiendra les dérivations)
- Liste (qui contiendra le flux de token de donnée)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



Liste de flux des tokens

Pile de dérivations

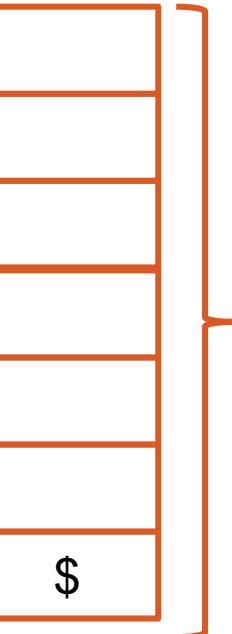


Table Syntaxique

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-
							41

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	-	num	*	id	\$
----	---	-----	---	----	----

Commencez en poussant le symbole de départ (but) dans la pile

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr'} \rightarrow + \langle \text{expr} \rangle$	$\text{<expr'} \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr'} \rightarrow \epsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow * \langle \text{term} \rangle$	$\text{<term'} \rightarrow / \langle \text{term} \rangle$	$\text{<term'} \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



Commencez en poussant le symbole de départ (but) dans la pile

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	-	num	*	id	\$
----	---	-----	---	----	----

Sur la tête du flux de données, on a **id**

Sur le sommet de la pile, on a **expr**

En utilisant le tableau syntaxique, on établit la règle:

$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

expr

\$

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	-	num	*	id	\$
----	---	-----	---	----	----

→ DÉPILE expr et PILE term et expr'

	num	id	+	-	*	/	\$
expr	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	-	-	-	-	-
expr'	-	-	$<\text{expr}'> \rightarrow +<\text{expr}>$	$<\text{expr}'> \rightarrow -<\text{expr}>$	-	-	$<\text{expr}'> \rightarrow \epsilon$
term	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	-	-	-	-	-
term'	-	-	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow * <\text{term}>$	$<\text{term}'> \rightarrow / <\text{term}>$	$<\text{term}'> \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



Sur la tête du flux de données, on a **id**

Sur le sommet de la pile, on a **term**

En utilisant le tableau syntaxique, on établit la règle:

$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	-	num	*	id	\$
----	---	-----	---	----	----

→ DÉPILE term et PILE factor et term'

	num	id	+	-	*	/	\$
expr	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	-	-	-	-	-
expr'	-	-	$<\text{expr}'> \rightarrow +<\text{expr}>$	$<\text{expr}'> \rightarrow -<\text{expr}>$	-	-	$<\text{expr}'> \rightarrow \epsilon$
term	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	-	-	-	-	-
term'	-	-	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow * <\text{term}>$	$<\text{term}'> \rightarrow / <\text{term}>$	$<\text{term}'> \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

term
expr'
\$

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



Sur la tête du flus de données, on a **id**

Sur le sommet de la pile, on a **factor**

En utilisant le tableau syntaxique, on établit la règle: **<factor> → id**

factor
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr}' \rightarrow + \langle \text{expr} \rangle$	$\text{<expr}' \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr}' \rightarrow \epsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term}' \rightarrow \epsilon$	$\text{<term}' \rightarrow \epsilon$	$\text{<term}' \rightarrow * \langle \text{term} \rangle$	$\text{<term}' \rightarrow / \langle \text{term} \rangle$	$\text{<term}' \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	-	num	*	id	\$
----	---	-----	---	----	----

Lorsqu'on a un terminal sur le sommet de la pile, on vérifie s'il matche la tête de la liste

Sinon → le syntaxe ne suit pas la grammaire
Si oui, ENLEVER la tête de la liste et DÉPILER

id
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



Lorsqu'on a un terminal sur le sommet de la pile, on vérifie s'il matche la tête de la liste

Sinon → le syntaxe ne suit pas la grammaire
Si oui, ENLEVER la tête de la liste et DÉPILER

	num	id	+	-	*	/	\$
expr	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	$<\text{expr}> \rightarrow <\text{term}><\text{expr}'>$	-	-	-	-	-
expr'	-	-	$<\text{expr}'> \rightarrow +<\text{expr}>$	$<\text{expr}'> \rightarrow -<\text{expr}>$	-	-	$<\text{expr}'> \rightarrow \epsilon$
term	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	$<\text{term}> \rightarrow <\text{factor}><\text{term}'>$	-	-	-	-	-
term'	-	-	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow \epsilon$	$<\text{term}'> \rightarrow * <\text{term}>$	$<\text{term}'> \rightarrow / <\text{term}>$	$<\text{term}'> \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	50



ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

-	num	*	id	\$
---	-----	---	----	----

Sur la tête du flux de données, on a -

Sur le sommet de la pile, on a **term'**

En utilisant le tableau syntaxique, on établit la règle: $\langle \text{term}' \rangle \rightarrow \epsilon$

Alors, il faut tout simplement dépiler

term'			
expr'			
\$			

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

-	num	*	id	\$
---	-----	---	----	----

Et on continue de la même façon...

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr}' \rightarrow + \langle \text{expr}' \rangle$	$\text{<expr}' \rightarrow - \langle \text{expr}' \rangle$	-	-	$\text{<expr>' \rightarrow } \varepsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term}' \rightarrow \varepsilon$	$\text{<term}' \rightarrow \varepsilon$	$\text{<term}' \rightarrow * \langle \text{term} \rangle$	$\text{<term}' \rightarrow / \langle \text{term} \rangle$	$\text{<term}' \rightarrow \varepsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

-	num	*	id	\$
---	-----	---	----	----

Et on continue de la même façon...

-
expr
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	*	id	\$
-----	---	----	----

Et on continue de la même façon...

expr

\$

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr}' \rightarrow + \langle \text{expr} \rangle$	$\text{<expr}' \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr}' \rightarrow \epsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term}' \rightarrow \epsilon$	$\text{<term}' \rightarrow \epsilon$	$\text{<term}' \rightarrow * \langle \text{term} \rangle$	$\text{<term}' \rightarrow / \langle \text{term} \rangle$	$\text{<term}' \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	*	id	\$
-----	---	----	----

Et on continue de la même façon...

term
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	*	id	\$
-----	---	----	----

Et on continue de la même façon...

factor
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	*	id	\$
-----	---	----	----

Et on continue de la même façon...

num
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

*	id	\$
---	----	----

Et on continue de la même façon...

term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

*	id	\$
---	----	----

Et on continue de la même façon...

*
term
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

Et on continue de la même façon...

term
expr'
\$

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr'} \rightarrow + \langle \text{expr} \rangle$	$\text{<expr'} \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr'} \rightarrow \epsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow * \langle \text{term} \rangle$	$\text{<term'} \rightarrow / \langle \text{term} \rangle$	$\text{<term'} \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	60

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

Et on continue de la même façon...

factor		
term'		
expr'		
\$		

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	$\text{factor} \rightarrow \text{num}$	$\text{factor} \rightarrow \text{id}$	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

Et on continue de la même façon...

id		
term'		
expr'		
\$		

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

\$

Et on continue de la même façon...

term'			
expr'			
\$			

	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr}' \rightarrow + \langle \text{expr} \rangle$	$\text{<expr}' \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr>' \rightarrow } \varepsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term}' \rightarrow \varepsilon$	$\text{<term}' \rightarrow \varepsilon$	$\text{<term}' \rightarrow * \langle \text{term} \rangle$	$\text{<term}' \rightarrow / \langle \text{term} \rangle$	$\text{<term}' \rightarrow \varepsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

\$

Et on continue de la même façon...



	num	id	+	-	*	/	\$
expr	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	$\text{<expr>} \rightarrow \text{<term>} \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\text{<expr'} \rightarrow + \langle \text{expr} \rangle$	$\text{<expr'} \rightarrow - \langle \text{expr} \rangle$	-	-	$\text{<expr'} \rightarrow \epsilon$
term	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	$\text{<term>} \rightarrow \text{<factor>} \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow \epsilon$	$\text{<term'} \rightarrow * \langle \text{term} \rangle$	$\text{<term'} \rightarrow / \langle \text{term} \rangle$	$\text{<term'} \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-



uOttawa

L'Université canadienne
Canada's university

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

\$



On a vérifié que la chaîne de données est une phrase de la grammaire!!

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow \epsilon$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-

MERCI!

QUESTIONS?

SÉANCE 13

**ENCORE SUR L'ANALYSE
SYNTAXIQUE LL**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Reprise en cas d'erreur

Grammaire Non LL(1)



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1)

Qu'est-ce qui arrive lorsque l'analyseur syntaxique découvre une erreur?

- **Approche 1:** arrêter toute activité d'analyse syntaxique et afficher un message d'erreur
- **Approche 2:** essayer de continuer l'analyse syntaxique (si possible) et vérifier s'il y a plus d'erreurs le long du chemin



Jamais!

Quelle approche votre compilateur choisit-il?



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1)

Une erreur est détectée lorsque:

- Le terminal au sommet de la pile ne matche pas le prochain symbole d'entrée
- La cellule du tableau de parage de laquelle on est supposé tirer la prochaine production est vide

Que fait l'analyseur syntaxique?

- Il rentre dans le **mode de panique** de reprise en cas d'erreur
- Basé sur l'idée de sauter des symboles sur l'entrée jusqu'à ce qu'un token dans l'ensemble SYNCH est trouvé



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1)

Soit **S** un ensemble de tokens appelé ensemble de synchronisation (**SYNCH**)

Soit $s \in S \rightarrow s$ est appelé un token de synchronisation

Comment construire l'ensemble de synchronisation?

- Plusieurs méthodes heuristiques ont été proposés
- On couvrira une méthode simple

Placez tous les symboles de FOLLOW(A) dans l'ensemble **SYNCH(A)** pour le non-terminal A

- Si on saute des tokens jusqu'à ce qu'un élément de **SYNCH(A)** est détecté et on pop A de la pile, c'est probable que le parage peut continuer.



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1)

Le mode de panique de reprise en cas d'erreur peut être implémenté en utilisant l'ensemble de SYNCH comme suit:

- **Scenario 1:** S'il y a un non-terminal au sommet de la pile, écarter les tokens d'entrée jusqu'à ce que vous trouvez un token qui appartient à l'ensemble SYNCH, ensuite dépilez le non-terminal
- **Scenario 2 :** S'il y a un terminal au sommet de la pile, on peut essayer de le dépiler afin de déterminer si on peut continuer
 - On suppose que la chaîne d'entrée manque ce terminal



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1) - EXEMPLE

FIRST Sets

```
FIRST(expr) = {num, id}
FIRST(expr') = {+, -, ε}
FIRST(term) = {num, id}
FIRST(term') = {*, /, ε}
FIRST(factor) = {num, id}
```

FOLLOW Sets

```
FOLLOW(expr) = { $ }
FOLLOW(expr') = { $ }
FOLLOW(term) = {+, -, $}
FOLLOW(term') = {+, -, $}
FOLLOW(factor) = {*, /, +, -, $}
```

Grammar

```
<expr> ::= <term><expr'>
<expr'> ::= +
          | -
          | ε
<term> ::= <factor><term'>
<term'> ::= *
          | /
          | ε
<factor> ::= num
           | id
```

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	-
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow$ $ε$
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	-	-	-	-	-
term'	-	-	$\langle \text{term}' \rangle \rightarrow$ $ε$	$\langle \text{term}' \rangle \rightarrow$ $ε$	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $ε$
factor	factor \rightarrow num	factor \rightarrow id	-	-	-	-	-



uOttawa

L'Université canadienne
Canada's university

REPRISE EN CAS D'ERREUR

LL(1) - EXEMPLE

FIRST Sets

```

FIRST(expr) = {num, id}
FIRST(expr') = {+, -, ε}
FIRST(term) = {num, id}
FIRST(term') = {*, /, ε}
FIRST(factor) = {num, id}

```

FOLLOW Sets

```

FOLLOW(expr) = { $ }
FOLLOW(expr') = { $ }
FOLLOW(term) = {+, -, $}
FOLLOW(term') = {+, -, $}
FOLLOW(factor) = {*, /, +, -, $}

```

Grammar

```

⟨expr⟩ ::= ⟨term⟩⟨expr'⟩
⟨expr'⟩ ::= +
          | -
          | ε
⟨term⟩ ::= ⟨factor⟩⟨term'⟩
⟨term'⟩ ::= *
          | /
          | ε
⟨factor⟩ ::= num
           | id

```

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow$ $\langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow$ $+ \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow$ $- \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow$ $\epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow$ $\langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow$ $\epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow$ $\epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow$ $* \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $/ \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow$ $\epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

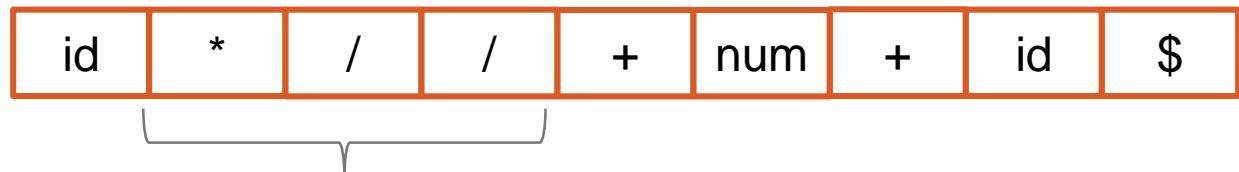
ANALYSE SYNTAXIQUE UTILISANT LL(1)



uOttawa

L'Université canadienne
Canada's university

Exemple:



Chaînes
d'erreurs

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	*	/	/	+	num	+	id	\$
----	---	---	---	---	-----	---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
expr'	$\langle \text{term}' \rangle \rightarrow \langle \text{term} \rangle \langle \text{term}' \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
\$	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 10

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	*	/	/	+	num	+	id	\$
----	---	---	---	---	-----	---	----	----

factor
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	*	/	/	+	num	+	id	\$
----	---	---	---	---	-----	---	----	----

id
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

*	/	/	+	num	+	id	\$
---	---	---	---	-----	---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr} \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

*	/	/	+	num	+	id	\$
---	---	---	---	-----	---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

/	/	+	num	+	id	\$
---	---	---	-----	---	----	----

Erreur: la cellule correspondant à la rangée term et la colonne / est vide!

Commencez à jeter les tokens, jusqu'à ce que vous trouvez un token synch

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
expr'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

/	+	num	+	id	\$
---	---	-----	---	----	----

Erreur: la cellule correspondant à la rangée term et la colonne / est vide!

Commencez à jeter les tokens, jusqu'à ce que vous trouvez un token synch

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
expr'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	num	+	id	\$
---	-----	---	----	----

Erreur: la cellule correspondant à la rangée term et la colonne / est vide!

Commencez à jeter les tokens, jusqu'à ce que vous trouvez un token synch

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
expr'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



On a trouvé un token synch!

Dépilez term de la pile et essayez de continuer...

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
expr'	$\langle \text{term}' \rangle \rightarrow \langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
\$	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	num	+	id	\$
---	-----	---	----	----

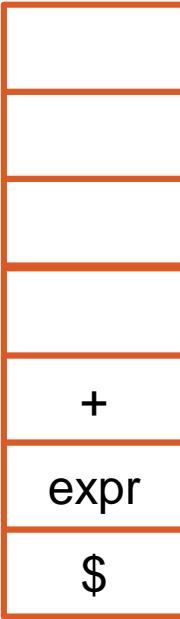
	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	num	+	id	\$
---	-----	---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 20



ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	+	id	\$
-----	---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	+	id	\$
-----	---	----	----

term
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	+	id	\$
-----	---	----	----

factor
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

num	+	id	\$
-----	---	----	----

num
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 24

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	id	\$
---	----	----

term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	id	\$
---	----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

+	id	\$
---	----	----

+
expr
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

term
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

factor
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 30

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

id	\$
----	----

id
term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:

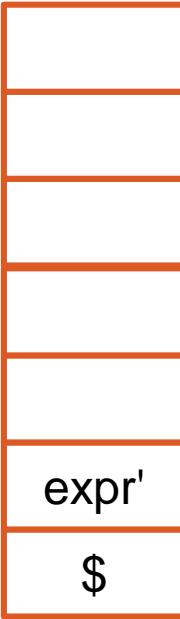


term'
expr'
\$

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr} \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow +\langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow -\langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 32

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s) 33

ANALYSE SYNTAXIQUE UTILISANT LL(1)

Exemple:



On a fait notre mieux pour continuer l'analyse
syntaxique

	num	id	+	-	*	/	\$
expr	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \langle \text{expr}' \rangle$	-	-	-	-	(s)
expr'	-	-	$\langle \text{expr}' \rangle \rightarrow + \langle \text{expr} \rangle$	$\langle \text{expr}' \rangle \rightarrow - \langle \text{expr} \rangle$	-	-	$\langle \text{expr}' \rangle \rightarrow \epsilon \text{ (s)}$
term	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \langle \text{term}' \rangle$	(s)	(s)	-	-	(s)
term'	-	-	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$	$\langle \text{term}' \rangle \rightarrow * \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow / \langle \text{term} \rangle$	$\langle \text{term}' \rangle \rightarrow \epsilon \text{ (s)}$
factor	factor \rightarrow num	factor \rightarrow id	(s)	(s)	(s)	(s)	(s)



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

Considérez la grammaire:

```
<stmt> ::= if <expr> then <stmt>  
          | if <expr> then <stmt> else <stmt>
```

On a besoin de faire la factorisation à gauche, ce qui nous donne:

```
<stmt> ::= if <expr> then <stmt><stmt'>  
<stmt'> ::= else <stmt> | ε
```

Cherchons les ensembles FIRST et FOLLOW

FIRST(stmt) = {if}

FIRST(stmt')={else, ε}

1) FIRST(terminal) is {terminal}

2) If $A \rightarrow a\alpha$, and a is a terminal:
 $\{a\} \in \text{FIRST}(A)$

3) If $A \rightarrow B\alpha$, and rule $B \rightarrow \epsilon$ does NOT exist:
 $\text{FIRST}(B) \in \text{FIRST}(A)$

4) If $A \rightarrow B\alpha$, and rule $B \rightarrow \epsilon$ DOES exist:
 $\{(\text{FIRST}(B) - \epsilon) \cup \text{FIRST}(\alpha)\} \in \text{FIRST}(A)$



GRAMMAIRE NON LL(1)

Considérez la grammaire:

```
<stmt> ::= if <expr> then <stmt>  
          | if <expr> then <stmt> else <stmt>
```

On a besoin de faire la factorisation à gauche, ce qui nous donne:

```
<stmt> ::= if <expr> then <stmt><stmt'>  
<stmt'> ::= else <stmt> | ε
```

Cherchons les ensembles FIRST et FOLLOW

FIRST(stmt) = {if}

FOLLOW(stmt) = {\$, else}

FIRST(stmt')={else, ε}

FOLLOW(stmt')={\$, else}

1) $\{\$\} \in \text{FOLLOW}(S)$

2) If $A \rightarrow \alpha B$:

$\text{FOLLOW}(A) \in \text{FOLLOW}(B)$

3) If $A \rightarrow \alpha B \gamma$, and rule $\gamma \rightarrow \epsilon$ does NOT exist:

$\text{FIRST}(\gamma) \in \text{FOLLOW}(B)$

4) If $A \rightarrow \alpha B \gamma$, and rule $\gamma \rightarrow \epsilon$ DOES exist:

$\{(\text{FIRST}(\gamma) - \epsilon) \cup \text{FOLLOW}(A)\} \in \text{FOLLOW}(B)$

Note: Ceci est une grammaire partielle (utilisée pour démontrer un concept), c'est la raison pour laquelle on n'a pas spécifié les règles de production associée avec expr. Conséquemment, ses ensembles FIRST et FOLLOW ne seront pas calculés.



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

$\text{FIRST}(\text{stmt}) = \{\text{if}\}$
 $\text{FIRST}(\text{stmt}') = \{\text{else}, \epsilon\}$

...

$\text{FOLLOW}(\text{stmt}) = \{\$\text{, else}\}$
 $\text{FOLLOW}(\text{stmt}') = \{\$\text{, else}\}$

...

```
<stmt> ::= if <expr> then <stmt><stmt'>
<stmt'> ::= else <stmt> | ε
```

	if	then	else	\$
stmt				
stmt'				



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

$\text{FIRST}(\text{stmt}) = \{\text{if}\}$
 $\text{FIRST}(\text{stmt}') = \{\text{else}, \epsilon\}$

...

$\text{FOLLOW}(\text{stmt}) = \{\$\text{, else}\}$
 $\text{FOLLOW}(\text{stmt}') = \{\$\text{, else}\}$

...

$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt} \rangle \langle \text{stmt}' \rangle$
 $\langle \text{stmt}' \rangle ::= \text{else } \langle \text{stmt} \rangle \mid \epsilon$

	if	then	else	\$
stmt	$\langle \text{stmt} \rangle \rightarrow$ $\text{if } \langle \text{expr} \rangle \text{ then }$ $\langle \text{stmt} \rangle \langle \text{stmt}' \rangle$	-	-	-
stmt'				



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

$\text{FIRST}(\text{stmt}) = \{\text{if}\}$
 $\text{FIRST}(\text{stmt}') = \{\text{else}, \epsilon\}$

...

$\text{FOLLOW}(\text{stmt}) = \{\$\text{, else}\}$
 $\text{FOLLOW}(\text{stmt}') = \{\$\text{, else}\}$

...

$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt} \rangle \langle \text{stmt}' \rangle$
 $\langle \text{stmt}' \rangle ::= \text{else } \langle \text{stmt} \rangle \mid \epsilon$

	if	then	else	\$
stmt	$\langle \text{stmt} \rangle \rightarrow$ $\text{if } \langle \text{expr} \rangle \text{ then }$ $\langle \text{stmt} \rangle \langle \text{stmt}' \rangle$	-	-	-
stmt'			$\langle \text{stmt}' \rangle \rightarrow$ $\text{else } \langle \text{stmt} \rangle$	



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

$\text{FIRST}(\text{stmt}) = \{\text{if}\}$
 $\text{FIRST}(\text{stmt}') = \{\text{else}, \epsilon\}$

...

$\text{FOLLOW}(\text{stmt}) = \{\$\text{, else}\}$
 $\text{FOLLOW}(\text{stmt}') = \{\$\text{, else}\}$

...

$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt} \rangle \langle \text{stmt}' \rangle$
 $\langle \text{stmt}' \rangle ::= \text{else } \langle \text{stmt} \rangle \mid \epsilon$

	if	then	else	\$
stmt	$\langle \text{stmt} \rangle \rightarrow$ $\text{if } \langle \text{expr} \rangle \text{ then }$ $\langle \text{stmt} \rangle \langle \text{stmt}' \rangle$	-	-	-
stmt'			$\langle \text{stmt}' \rangle \rightarrow$ $\text{else } \langle \text{stmt} \rangle$	$\langle \text{stmt}' \rangle \rightarrow \epsilon$



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

$\text{FIRST}(\text{stmt}) = \{\text{if}\}$
 $\text{FIRST}(\text{stmt}') = \{\text{else}, \epsilon\}$

...

$\text{FOLLOW}(\text{stmt}) = \{\$\text{, else}\}$
 $\text{FOLLOW}(\text{stmt}') = \{\$\text{, else}\}$

...

$\langle \text{stmt} \rangle ::= \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{stmt} \rangle \langle \text{stmt}' \rangle$
 $\langle \text{stmt}' \rangle ::= \text{else } \langle \text{stmt} \rangle \mid \epsilon$

	if	then	else	\$
stmt	$\langle \text{stmt} \rangle \rightarrow$ $\text{if } \langle \text{expr} \rangle \text{ then }$ $\langle \text{stmt} \rangle \langle \text{stmt}' \rangle$	-	-	-
stmt'	-	-	$\langle \text{stmt}' \rangle \rightarrow$ $\text{else } \langle \text{stmt} \rangle,$ $\langle \text{stmt}' \rangle \rightarrow \epsilon$	$\langle \text{stmt}' \rangle \rightarrow \epsilon$



uOttawa

L'Université canadienne
Canada's university

GRAMMAIRE NON LL(1)

On envisage un problème parce que, pour un token d'entrée `else` et un sommet de pile de `stmt'`, on ne sait pas quelle production choisir:

- $\langle \text{stmt}' \rangle \rightarrow \text{else } \langle \text{stmt} \rangle$
- $\langle \text{stmt}' \rangle \rightarrow \epsilon$

Conséquemment, ceci n'est pas une grammaire LL(1)

	if	then	else	\$
stmt	$\langle \text{stmt} \rangle \rightarrow$ $\text{if } \langle \text{expr} \rangle \text{ then}$ $\langle \text{stmt} \rangle \langle \text{stmt}' \rangle$	-	-	-
stmt'	-	-	$\langle \text{stmt}' \rangle \rightarrow$ $\text{else } \langle \text{stmt} \rangle,$ $\langle \text{stmt}' \rangle \rightarrow \epsilon$	$\langle \text{stmt}' \rangle \rightarrow \epsilon$

MERCI!

QUESTIONS?

SÉANCE 14

**INTRODUCTION À LA
CONCURRENCE**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Concepts de la concurrence

Concurrence au niveau sous-programme

Sémaphores

POURQUOI ÉTUDIER LA CONCURRENCE?

Les systèmes d'ordinateur résous des problèmes du monde réel, où les événements se passent en même temps:

- Les systèmes d'exploitation, avec plusieurs processus qui se déroulent en même temps
- Les serveurs Web

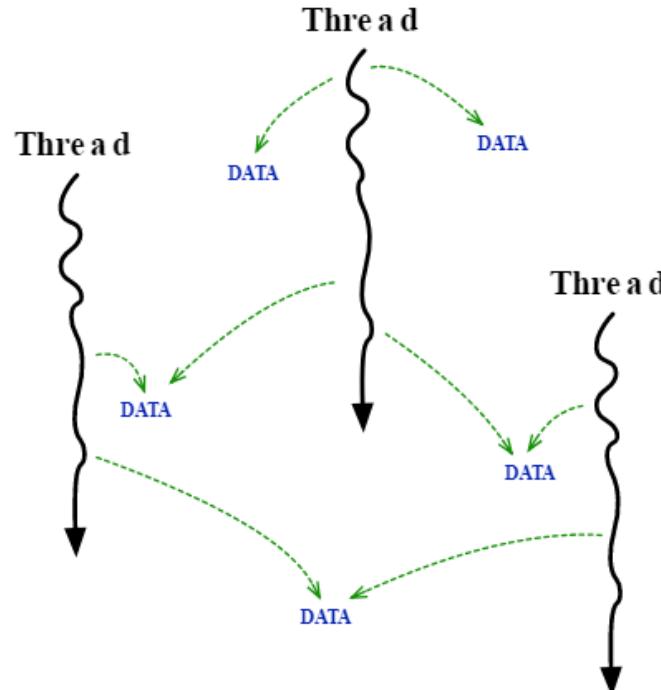
Les ordinateurs à multi-processeurs ou multi-cœurs sont maintenant très populaires

- Ceci nécessite un logiciel qui utilise effectivement ce type de hardware

CONCURRENCE À NIVEAU SOUS-PROGRAMME

Une tâche est une unité de programme qui peut être exécutée en concurrence avec d'autres unités du même programme

- Chaque tâche dans un programme peut fournir un thread de contrôle





uOttawa

L'Université canadienne
Canada's university

SYNCHRONISATION

Un mécanisme pour contrôler l'ordre dans lequel les tâches exécutent

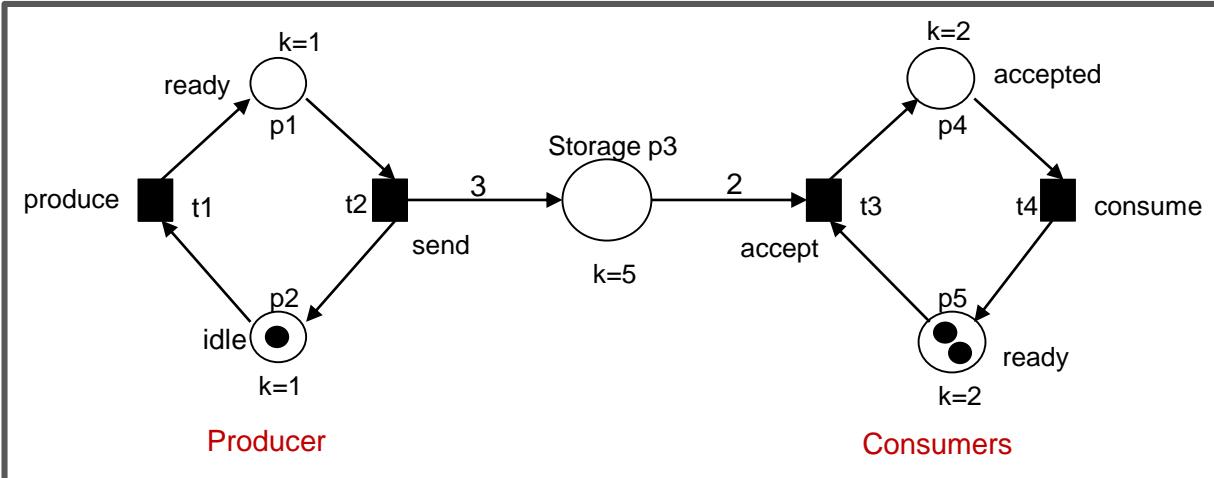
La synchronisation de coopération est requise entre la tâche A et la tâche B lorsque:

- La tâche A doit attendre la tâche B pour compléter une activité spécifique avant que la tâche A puisse continuer l'exécution
- Rappel du problème de réseaux de petri producteur-consommateur

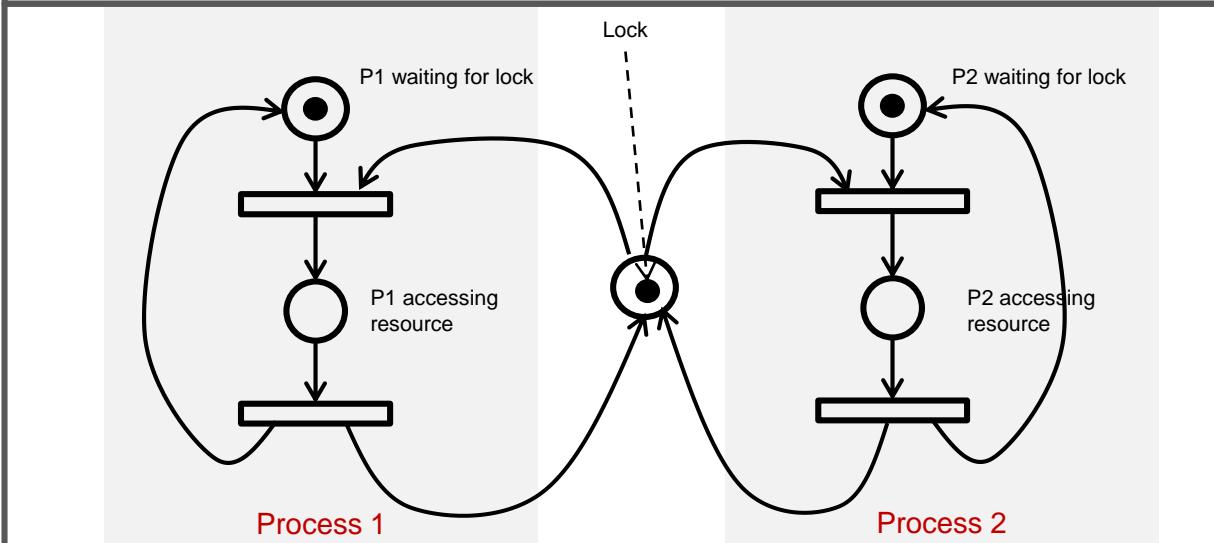
La synchronisation de compétition est requise entre deux tâches lorsque:

- Les deux tâches ont besoin d'utiliser une ressource qui ne peut pas être utilisée simultanément

SYNCHRONISATION (RÉSEAUX DE PETRI)

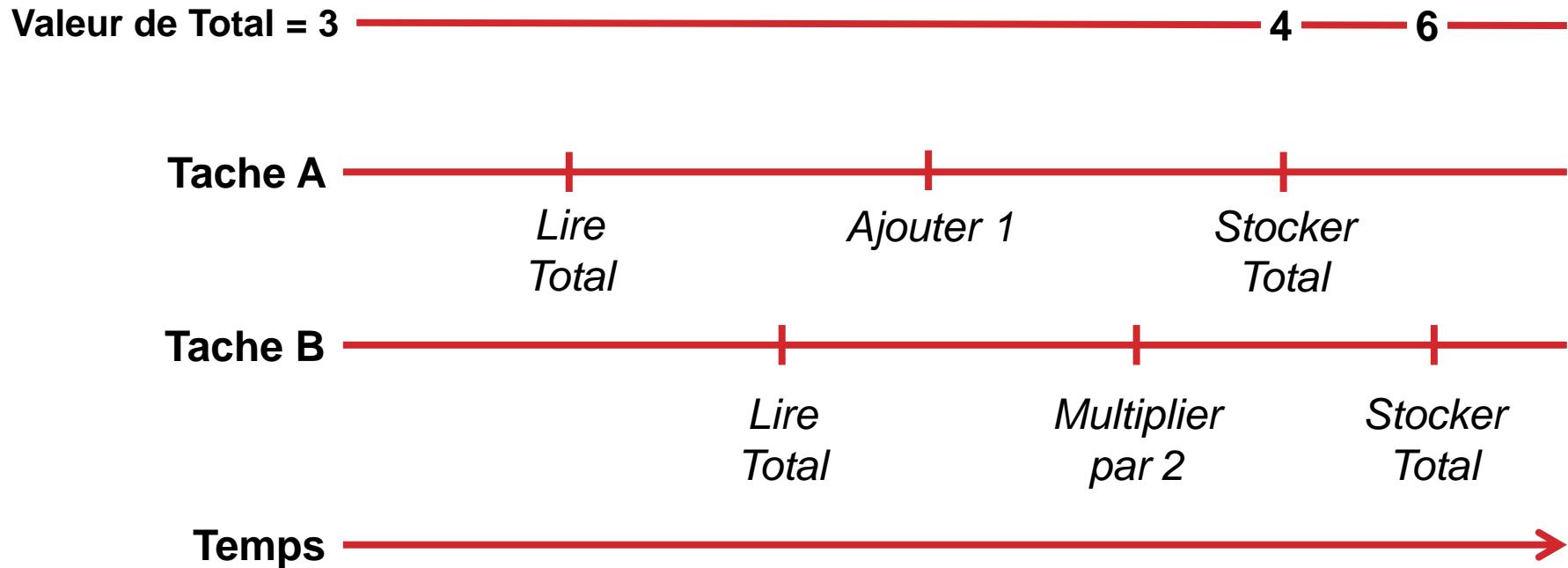


Synchronisation de coopération



Synchronisation de compétition

LA NÉCESSITÉ DE LA SYNCHRONISATION DE COMPÉTITION





uOttawa

L'Université canadienne
Canada's university

SECTION CRITIQUE

Une section du code dans laquelle le thread peut faire:

- Le changement de variables communes,
- La mise à jour d'un tableau,
- L'écriture dans un document,
- Ou la mise à jour de ressources partagées

L'exécution de sections critiques par les threads est mutuellement exclusive dans le temps



uOttawa

L'Université canadienne
Canada's university

ÉTATS DE TÂCHES (THREAD)

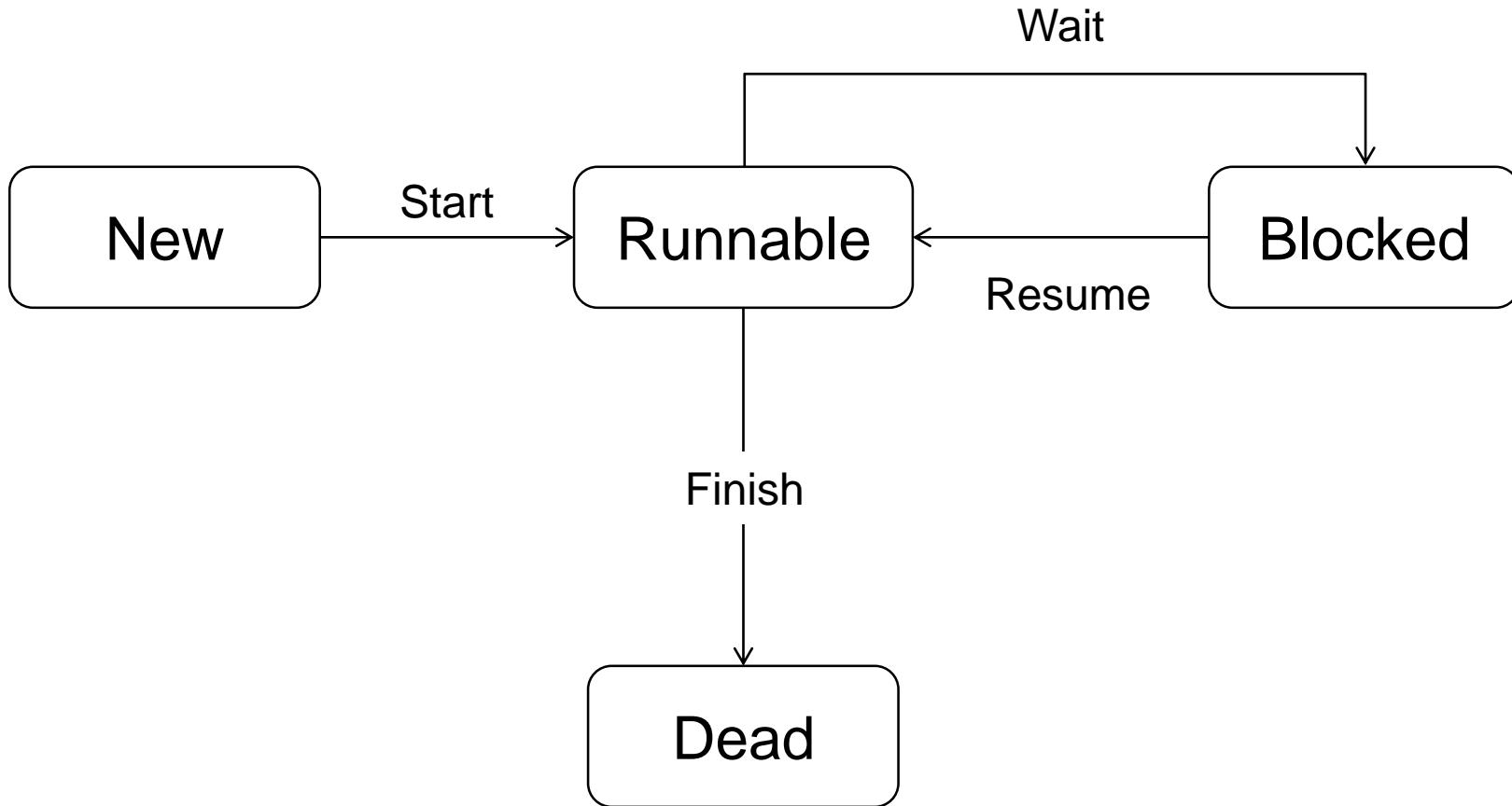
Nouveau (New): il a été créé, mais n'a pas encore commencer son exécution

Exécutable ou prêt (Runnable or Ready): il exécute couramment ou il est prêt à exécuter

Bloqué (Blocked): il exécutait, mais son exécution a été interrompue par un événement parmi plusieurs

Mort (Dead): il n'est plus actif dans aucun sens

ÉTATS DE TÂCHES (THREAD)





uOttawa

L'Université canadienne
Canada's university

SÉMAPHORES

Le Sémaphore est une technique utilisée pour contrôler l'accès à une ressource commune pour plusieurs tâches

- C'est un objet qui consiste d'un nombre entier (*comteur*) et une file qui stocke les descripteurs de tâches

Une tâche qui a besoin de faire accès à une section critique a besoin « d'acquérir » le sémaphore

Classiquement: c'est un système pour communiquer des messages en tenant les bras de deux drapeaux dans certaines positions selon un code alphabétique



uOttawa

L'Université canadienne
Canada's university

SÉMAPHORES

Deux opérations sont toujours associées avec les Sémafores

L'Opération “P” est utilisée pour chercher la séaphore, et “V” pour la relâcher

- **P** (proberen, qui veut dire tester et diminuer le nombre entier)
- **V** (verhogen, qui veut dire augmenter) l'opération

**Alternativement, ces opérations sont appelées:
wait et release**



uOttawa

L'Université canadienne
Canada's university

SÉMAPHORES

Les systèmes d'exploitation distinguent souvent entre les sémaphores comptants et binaires

Le nombre entier d'un **sémaphore comptant peut avoir n'importe quel valeur**

Le nombre entier d'un **sémaphore binaire peut seulement varier entre 0 et 1**



uOttawa

L'Université canadienne
Canada's university

SÉMAPHORES BINAIRES

La stratégie générale pour utiliser un sémaaphore binaire afin de contrôler l'accès à une section critique est comme suit:

```
Semaphore aSemaphore;
```

```
wait(aSemaphore);  
Critical section();  
release(aSemaphore);
```



uOttawa

L'Université canadienne
Canada's university

SYNCHRONISATION AVEC SÉMAPHORE COMPTANT

wait (semaphoreComptant) :

if conteur de semaphoreComptant > 0 **then**

 Décrémente le conteur de semaphoreComptant

else

 Change l'état de la tache à **bloqué (blocked)**

 Met la tache dans la file du semaphoreComptant

end



uOttawa

L'Université canadienne
Canada's university

SYNCHRONISATION AVEC SÉMAPHORE COMPTANT

release (semaphoreComptant) :

if file de semaphoreComptant est vide **then**
incrémenté le conteur de semaphoreComptant

else

Change l'état de la tâche dans la tête du file à **Exécutable (runnable)**

Fait un opération de défilage

end

PRODUCTEUR ET CONSUMMATEUR



uOttawa

L'Université canadienne
Canada's university

```
semaphore fullspots, emptyspots;
fullspots.count := 0;
emptyspots.count := BUFLEN;
task producer;
    loop
        -- produce VALUE --
        wait(emptyspots);      { wait for a space }
        DEPOSIT(VALUE);
        release(fullspots);   { increase filled spaces }
    end loop;
end producer;

task consumer;
    loop
        wait(fullspots);      { make sure it is not empty }
        FETCH(VALUE);
        release(emptyspots); { increase empty spaces }
        -- consume VALUE --
    end loop;
end consumer;
```



uOttawa

L'Université canadienne
Canada's university

SYNCHRONISATION AVEC SÉMAPHORE BINAIRE

wait(semaphoreBinaire) :

if conteur de semaphoreBinaire == 1 **then**

 Change la valeur du conteur de semaphoreBinaire à 0

else

 Change l'état de la tache à **bloqué (blocked)**

 Met la tache dans la file du semaphoreBinaire

end

SYNCHRONISATION AVEC SÉMAPHORE BINAIRE

release (semaphoreBinaire) :

if file de semaphoreBinaire est vide **then**

 conteur = 1

else

 Change l'état de la tâche dans la tête du file à **Exécutable (runnable)**

 Fait un opération de défilage

end



uOttawa

L'Université canadienne
Canada's university

AJOUTER LA SYNCHRONISATION DE COMPÉTITION

```
semaphore access, fullspots, emptyspots;
access.count := 1;
fullspots.count := 0;
emptyspots.count := BUflen;

task producer;
  loop
    -- produce VALUE --
    wait(emptyspots);      { wait for a space }
    wait(access);          { wait for access }
    DEPOSIT(VALUE);
    release(access);       { relinquish access }
    release(fullspots);    { increase filled spaces }
  end loop;
end producer;
```



uOttawa

L'Université canadienne
Canada's university

AJOUTER LA SYNCHRONISATION DE COMPÉTITION

```
task consumer;
  loop
    wait(fullspots);           { make sure it is not empty }
    wait(access);              { wait for access }

    FETCH(VALUE);
    release(access);          { relinquish access }
    release(emptyspots);      { increase empty spaces }
    -- consume VALUE --
  end loop
end consumer;
```



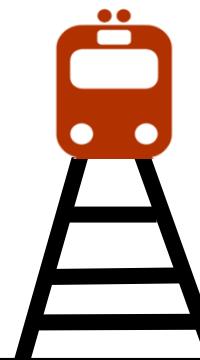
uOttawa

L'Université canadienne
Canada's university

INTER-BLOCAGES

Une loi passée par la législature de Kansas vers le début du 20^e siècle:

“..... Lorsque deux trains s’approchent à un croisement, les deux doivent s’arrêter complètement et aucun d’eux ne peut démarrer jusqu’à ce que l’autre est parti.





uOttawa

L'Université canadienne
Canada's university

CONDITIONS POUR UN INTER-BLOCAGE

Exclusion mutuelle: le fait de permettre à un seul processus d'avoir accès à une ressource partagée

Tenir-et-attendre: il doit y avoir un processus qui tiens au moins une ressource et attend d'acquérir des ressources additionnelles qui sont couramment tenus par d'autres processus

Pas de préemption: le manque de réallocation temporaire de ressource. La ressource peut être relâchée volontairement seulement

Attente circulaire: chaque processus impliqué dans l'impasse attend un autre processus de relâcher volontairement la ressource

EXEMPLE D'INTER-BLOCAGE

```
BinarySemaphore s1, s2;
```

Task A:

```
wait(s1)
    // access resource 1
    wait (s2)
        // access resource 2
    release(s2)
    release(s1)
end task
```

Task B:

```
wait(s2)
    // access resource 2
    wait(s1)
        // access resource 1
    release(s1)
    release(s2)
end task
```

STRATÉGIE DE GESTION DES INTER-BLOCAGES

Prévention: éliminer une des conditions nécessaires

Évitement: Éviter si le système connaît à l'avance la séquence des demandes de ressources associées avec chaque processus actif

Détection: détecter en construisant des graphes de ressources dirigées et chercher les cycles

Recouvrement: lorsque détecté, il doit être démêlé et le système doit retourner à son état normal le plus rapidement possible

- Terminaison du processus
- Préemption de ressources

MERCI!

QUESTIONS?

SÉANCE 15

MONITEURS



uOttawa

L'Université canadienne
Canada's university

SUJETS

Introduction au moniteurs

Comparaison entre les moniteurs et sémaphores

Variables de condition

- Les opérations des variables de condition

Exemples des moniteurs



uOttawa

L'Université canadienne
Canada's university

MONITEUR

Un moniteur est un ensemble de routines qui sont protégés par un verrou d'exclusion mutuelle

- Aucune routine dans le moniteur peut s'exécuter par un thread jusqu'à ce que ce thread obtient le verrou

Tout autre thread doit attendre le thread qui exécute couramment afin d'abandonner le contrôle du verrou

Lorsqu'un moniteur est occupé, le code de synchronisation de compétition est ajouté par le compilateur

- Pourquoi est-ce un avantage?



uOttawa

L'Université canadienne
Canada's university

MONITEUR

Un thread peut se suspendre à l'intérieur d'un moniteur et ensuite attendre qu'un événement se passe

- Si ceci se passe, alors un autre thread a l'opportunité de rentrer dans le moniteur

Habituellement, un thread se suspend en attendant une condition

- Durant l'attente, le thread abandonne temporairement son accès exclusif
- Il doit le réacquérir après que la condition soit remplie



uOttawa

L'Université canadienne
Canada's university

MONITEUR VS SÉMAPHORE

Un sémaphore est une construction plus simple que le moniteur parce qu'il s'agit seulement d'un verrou qui protège une ressource partagée

- Pas un ensemble de routines comme le moniteur

Une tâche doit acquérir (ou attendre pour) un sémaphore avant d'accéder une ressource partagée

Une tâche doit simplement faire appel à une routine (ou procédure) dans le moniteur afin d'accéder une ressource partagée

- Lorsque terminé, vous n'avez rien à relâcher
- Rappelez-vous que vous devez relâcher les sémaphores (si vous oubliez →inter-blocage)

SYNCHRONISATION DE COMPÉTITION

Une des caractéristiques les plus importantes des moniteurs est que l'information partagée réside dans le moniteur

- Tout le code de synchronisation est centralisé dans un seul endroit

Le moniteur garantit la synchronisation en permettant l'accès à une seule tâche à la fois

- Rappelez-vous que par l'utilisation des sémaphores comptants, nous sommes capables de permettre l'accès à plusieurs tâches, et non nécessairement une seule

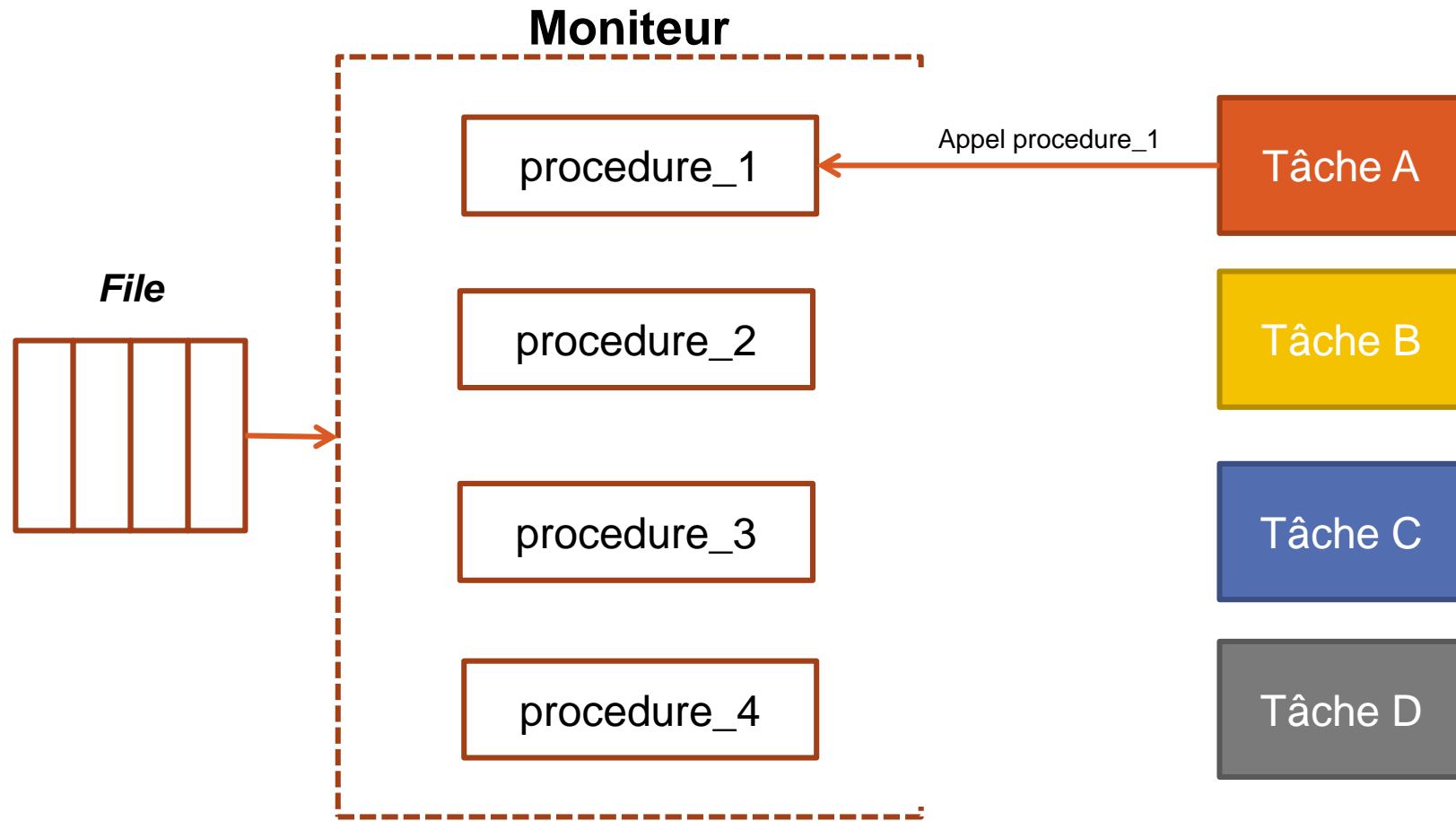
Les appels aux procédures du moniteur sont implicitement filés si le moniteur est occupé au moment de l'appel



uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



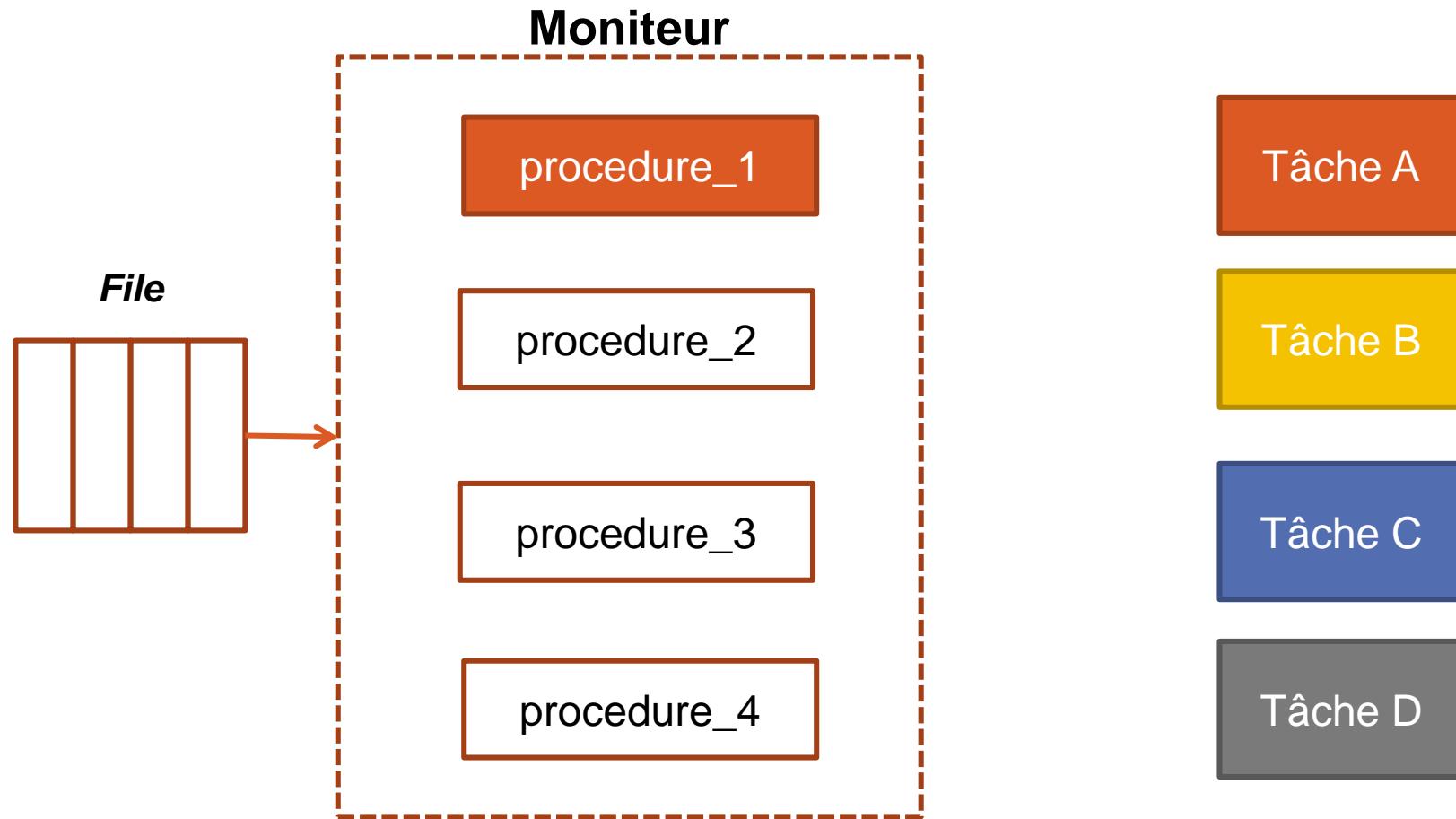
Propriétaire du Moniteur: Aucun



uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR

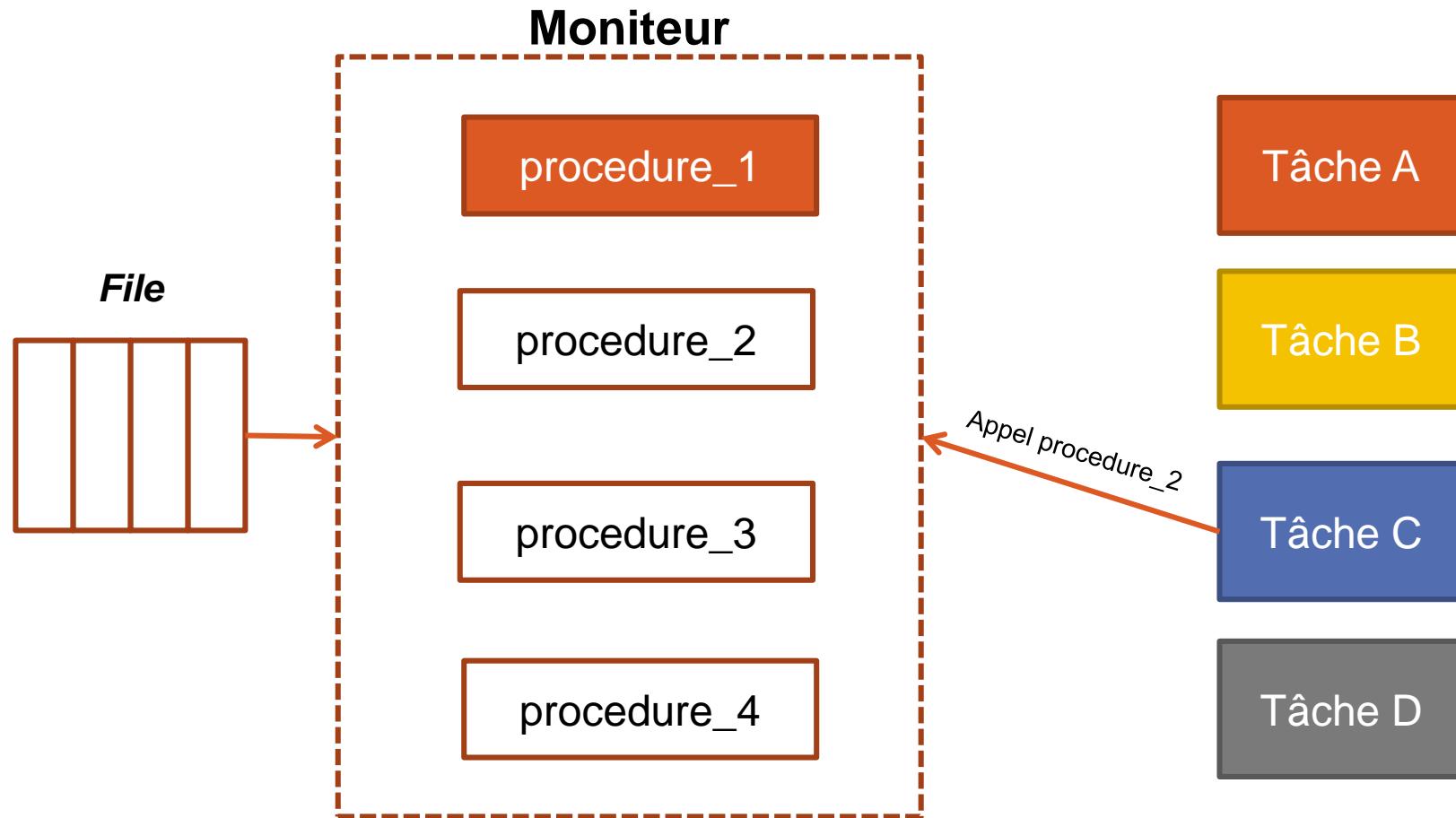




uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



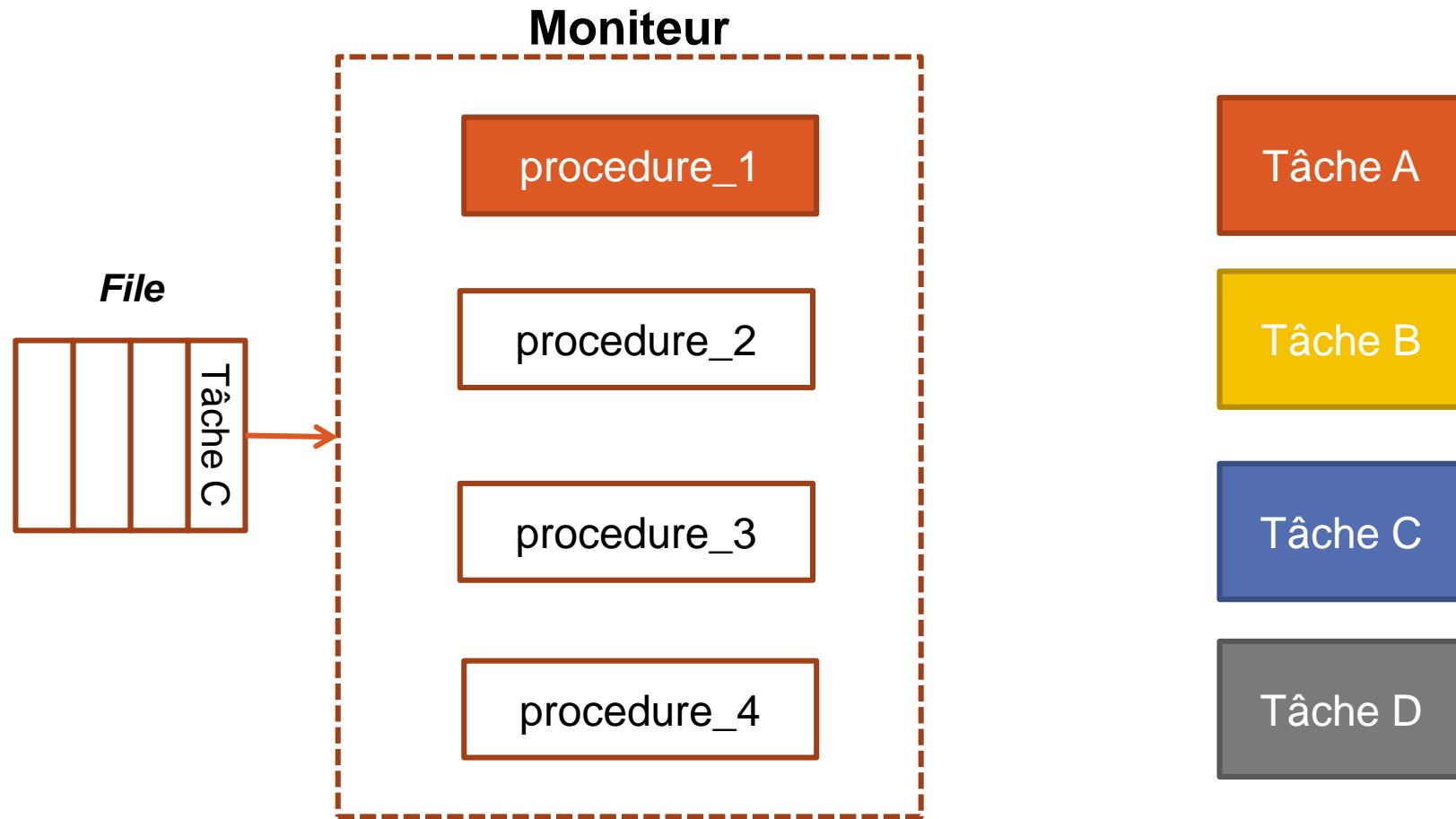
Propriétaire du Moniteur: Tâche A



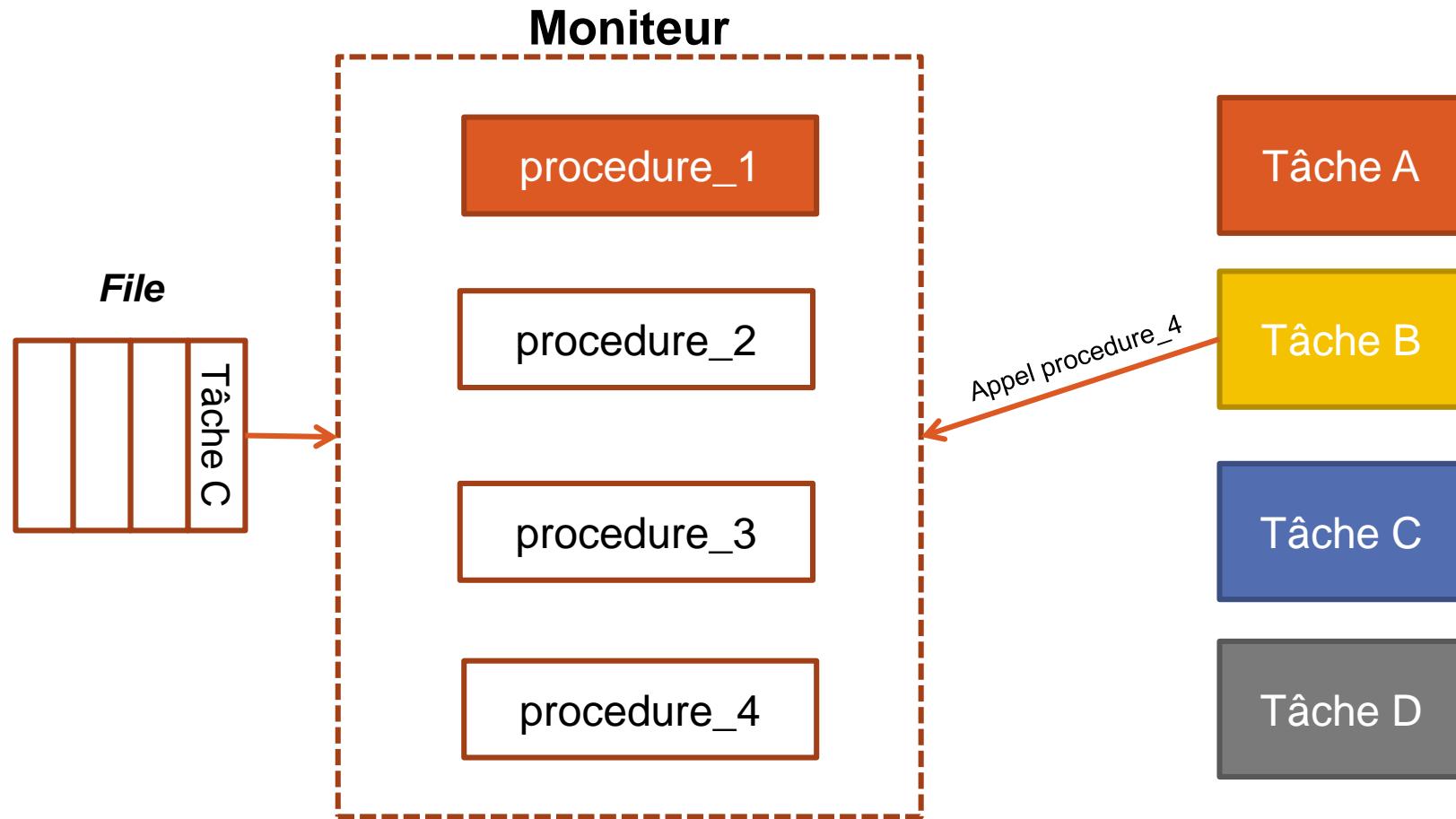
uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



EXEMPLE D'UTILISATION D'UN MONITEUR



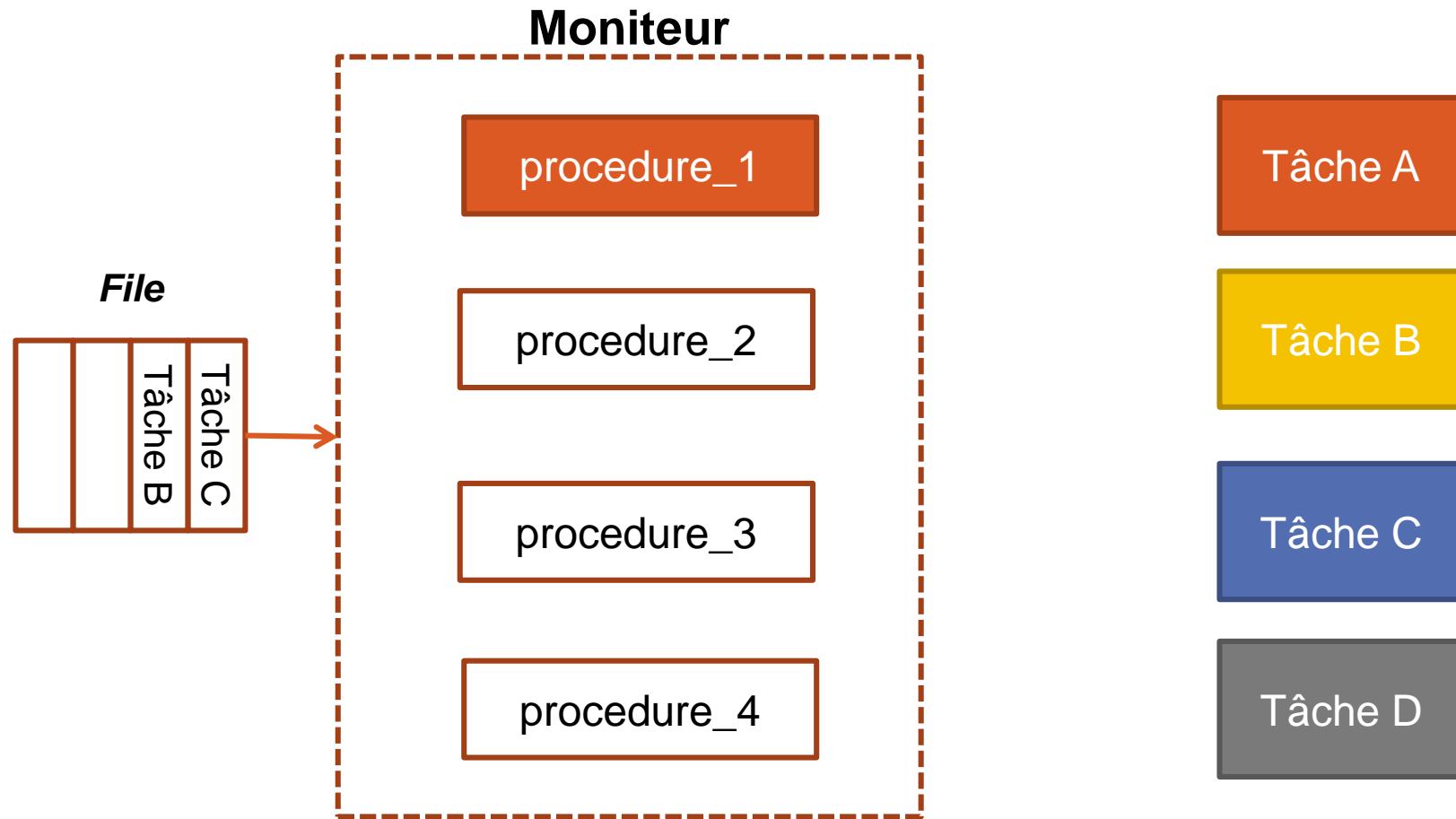
Propriétaire du Moniteur: Tâche A



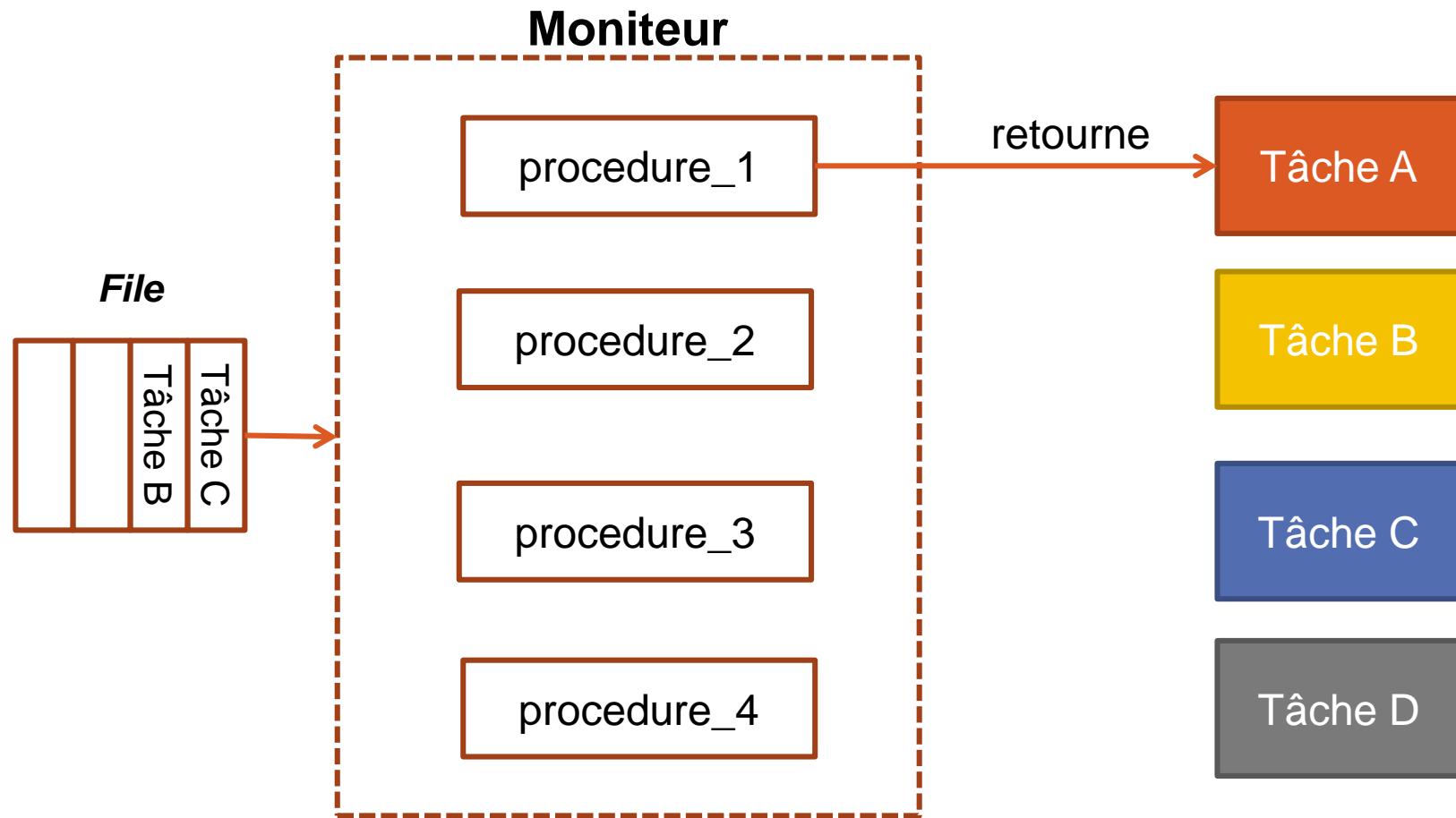
uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



EXEMPLE D'UTILISATION D'UN MONITEUR

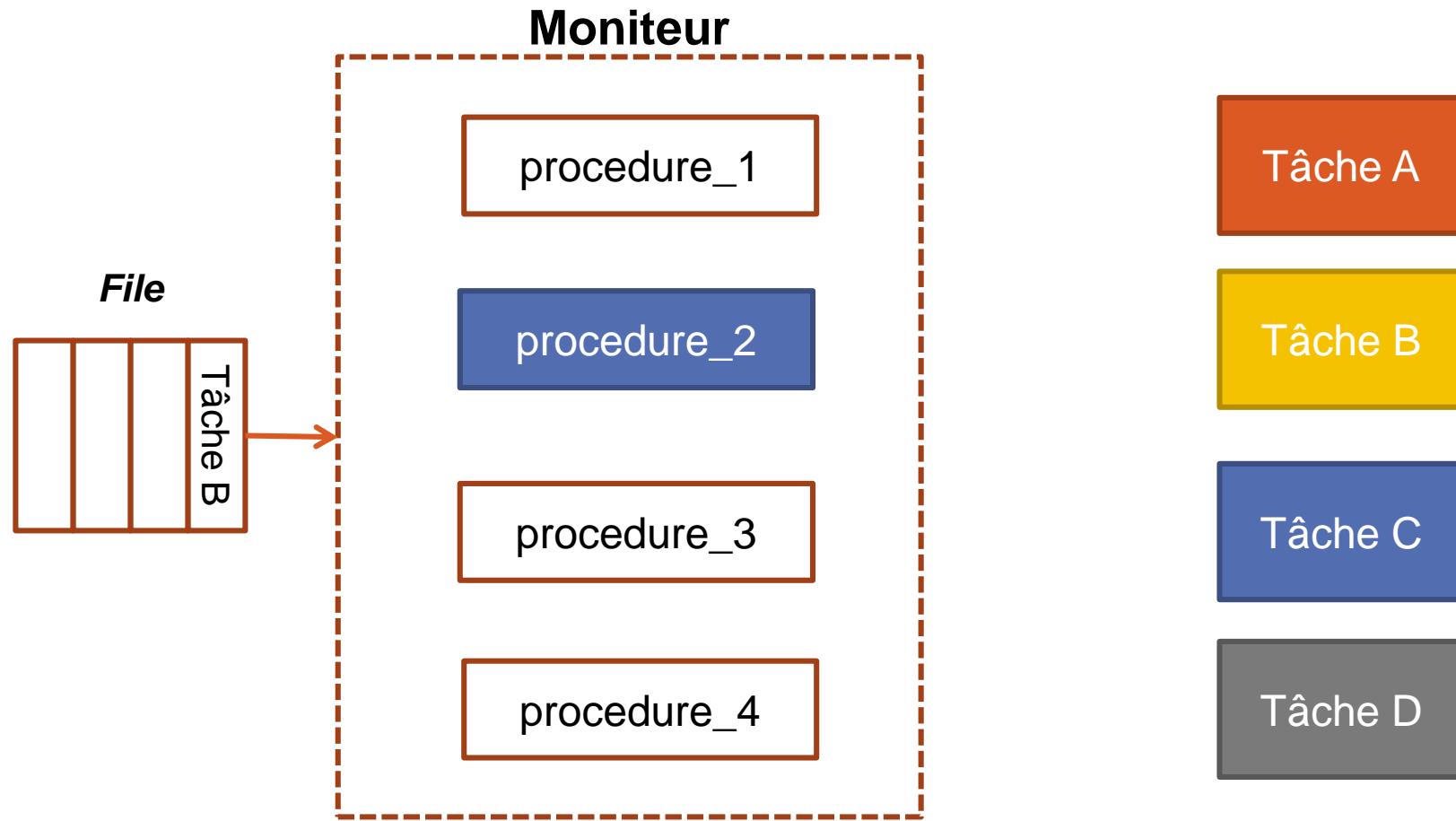




uOttawa

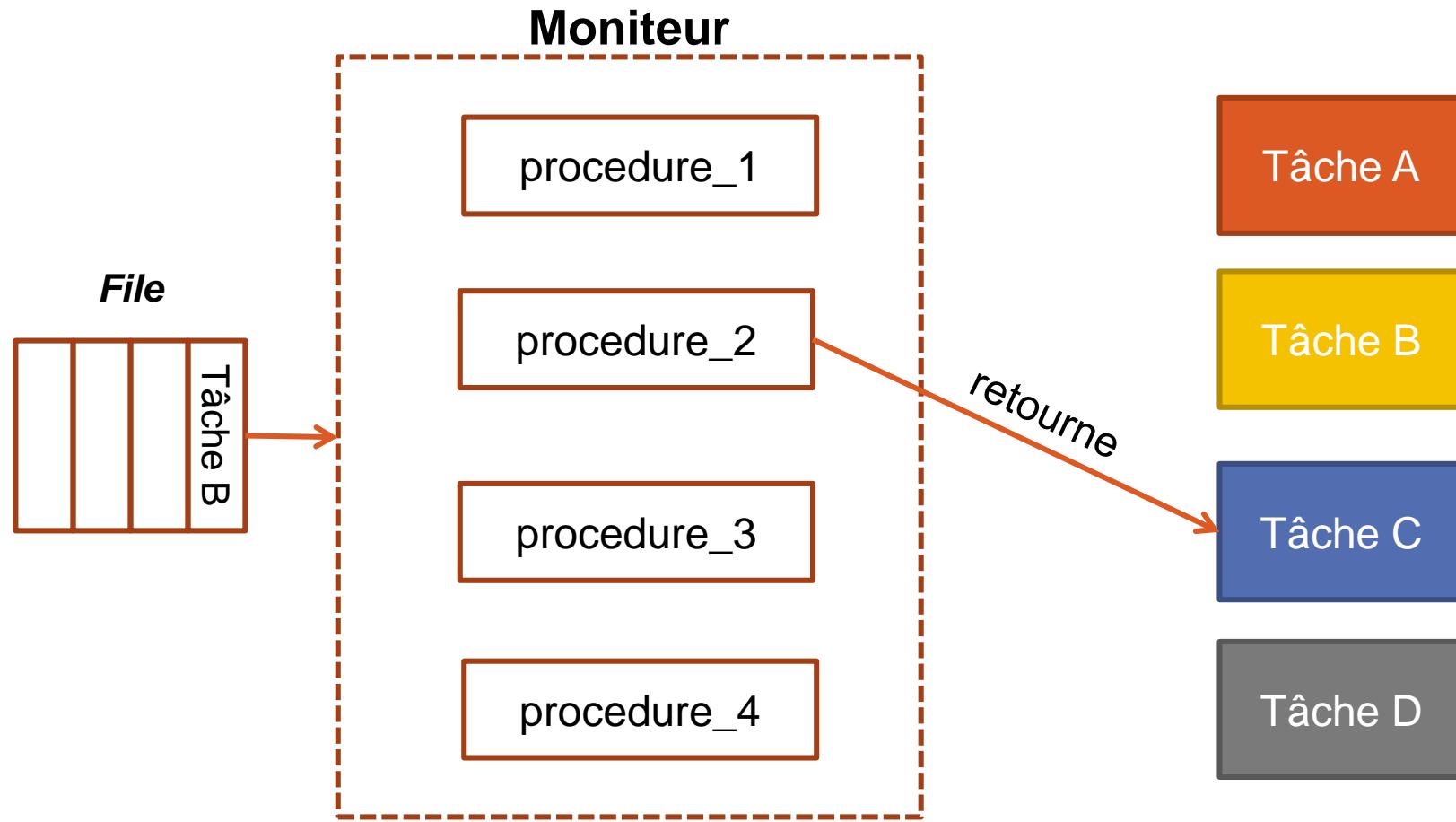
L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



Propriétaire du Moniteur: Tâche C

EXEMPLE D'UTILISATION D'UN MONITEUR



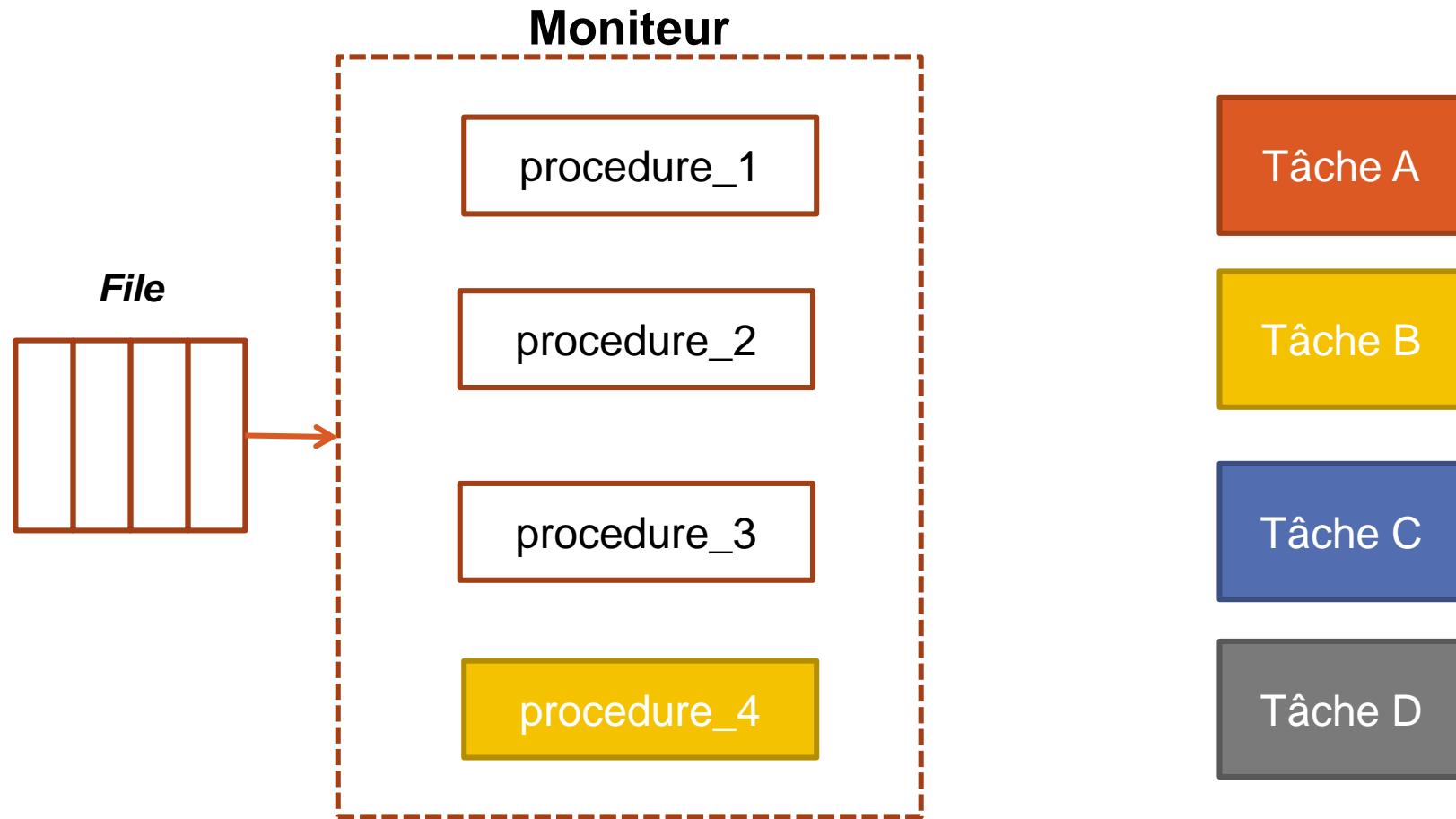
Propriétaire du Moniteur: Aucun



uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



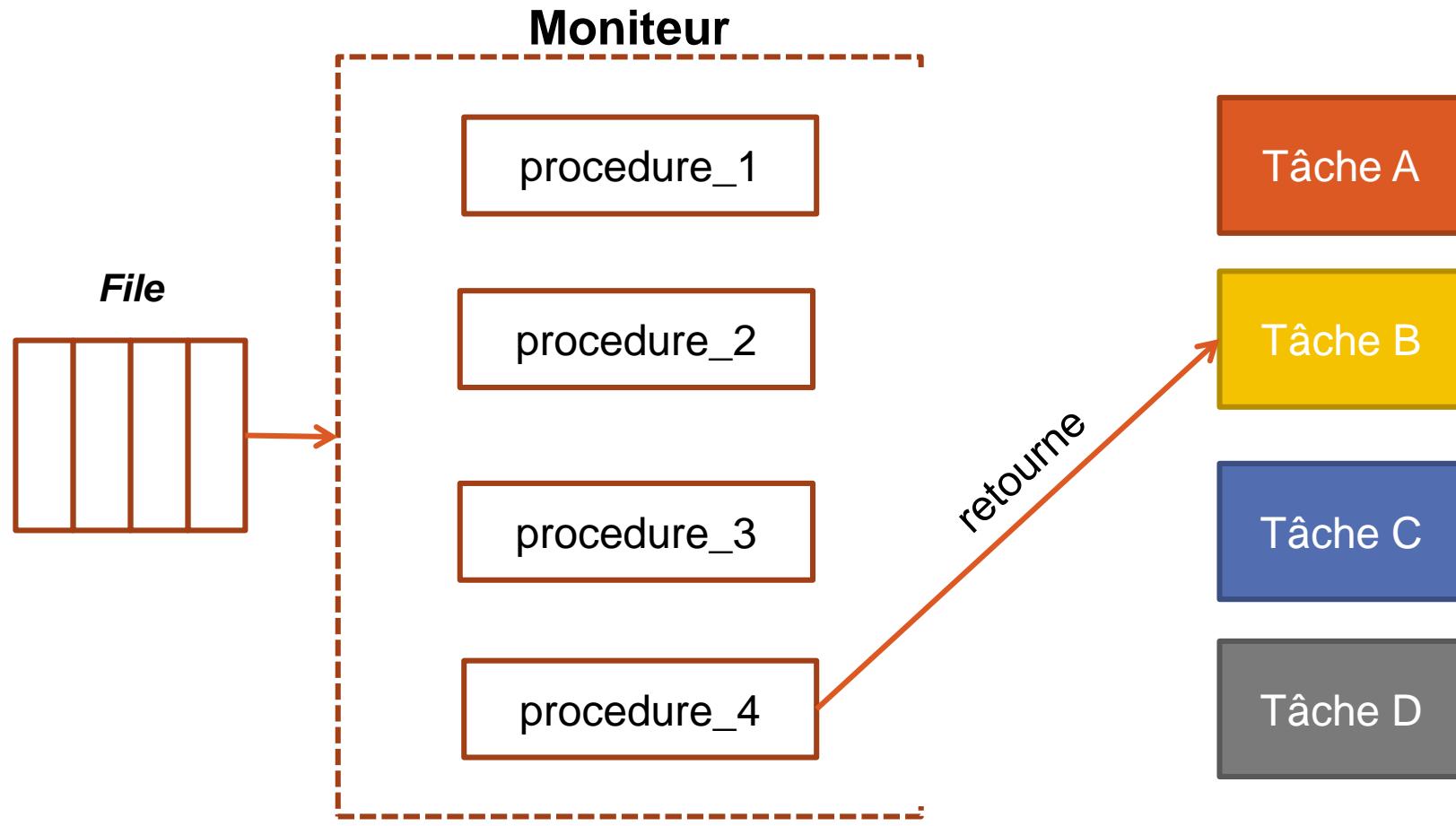
Propriétaire du Moniteur: Tâche B



uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



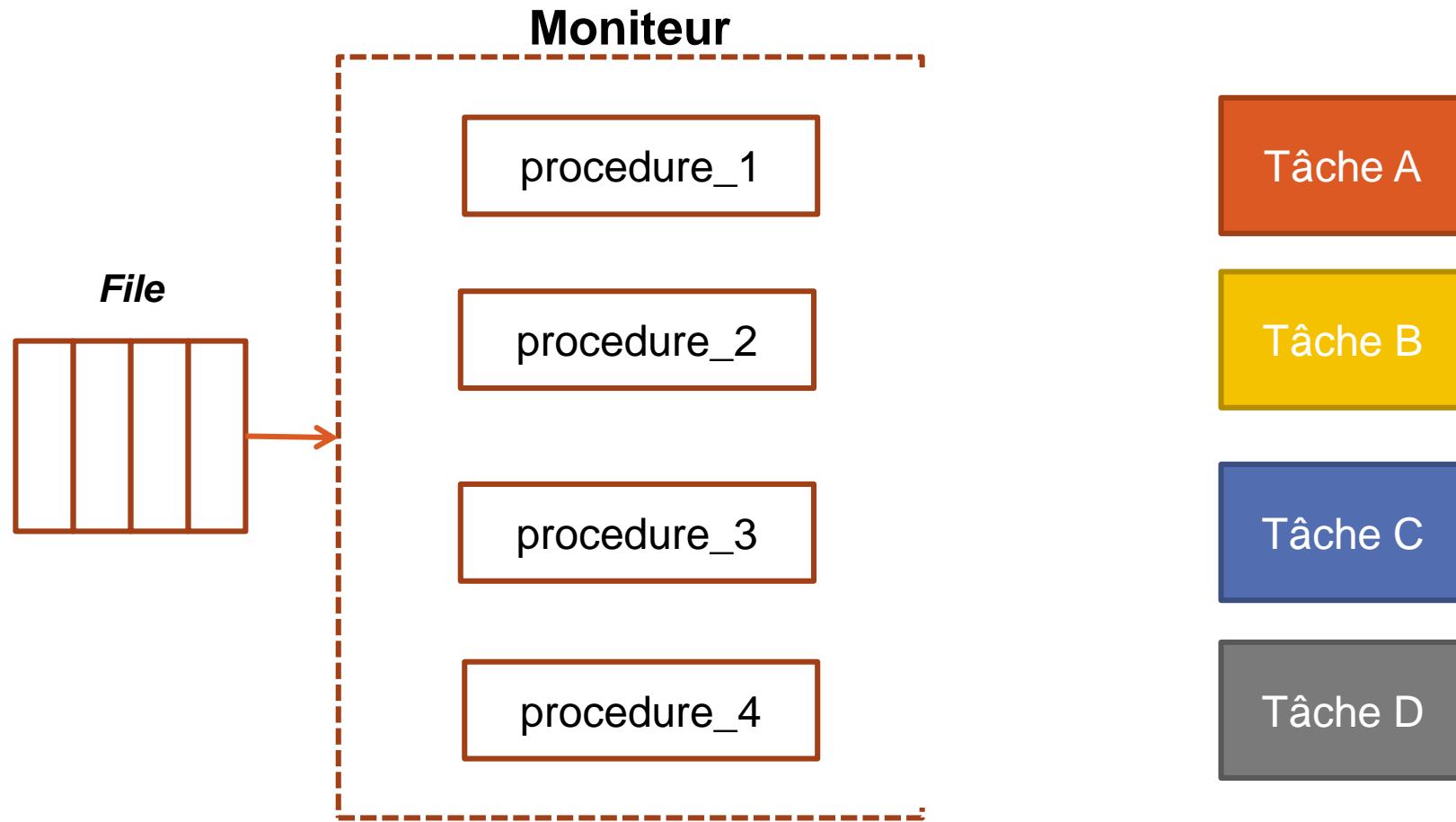
Propriétaire du Moniteur: Aucun



uOttawa

L'Université canadienne
Canada's university

EXEMPLE D'UTILISATION D'UN MONITEUR



Propriétaire du Moniteur: Aucun

EXEMPLE DE COMPTE BANCAIRE

MONITOR: Account

```
double balance

procedure double withdraw(amount )
begin
    balance = balance - amount
    return balance
end procedure
```

withdraw (amount)

balance = balance - amount

withdraw (amount)

withdraw (amount)

return balance

balance = balance – amount
return balance

balance = balance – amount
return balance

SYNCHRONISATION DE COOPÉRATION

Malgré que l'accès mutuellement exclusif à l'information partagée soit intrinsèque avec un moniteur:

- La coopération entre les tâches est toujours la responsabilité du programmeur

Le programmeur doit garantir qu'un tampon partagé ne subit pas un débordement



uOttawa

L'Université canadienne
Canada's university

VARIABLES DE CONDITION

Les variables de condition fournissent un mécanisme d'attente pour les événements

Les variables de condition supportent trois opérations:

- `wait ()`: relâcher le verrou du moniteur et attendre que la variable de condition soit signalée
- `signal()`: réveille un thread en attente
- `broadcast()`: réveiller tous les threads en attente

Chaque variable de condition a sa propre file d'attente

- Une tâche qui attend cette condition est bloquée et son descripteur est stocké dans la file d'attente



uOttawa

L'Université canadienne
Canada's university

EXEMPLE: PRODUCTEUR CONSOMMATEUR

PRODUCTEUR

```
Task Producer

begin

    Loop Forever

        // produce new item

        ...

        bufferMonitor.deposit newItem

    end

End Task
```

CONSOMMATEUR

```
Task Consumer

begin

    Loop Forever

        newItem = bufferMonitor.fetch()

        // consume new item

        ...

    end

End Task
```

EXEMPLE: PRODUCTEUR CONSOMMATEUR – MONITEUR VS SÉMAPHORE

Producteur

```
Task Producer

begin

    Loop Forever

        // produce new item

        bufferMonitor.deposit(newItem)

    end

End Task
```

Consommateur

```
Task Consumer

begin

    Loop Forever

        newItem = bufferMonitor.fetch()

        // consume new item

    end

End Task
```

```
task producer;
loop
-- produce VALUE --
wait(emptyspots);      { wait for a space }
wait(access);          { wait for access }
DEPOSIT(VALUE);
release(access);       { relinquish access }
release(fullspots);    { increase filled spaces }
end loop;
end producer;
```

```
task consumer;
loop
wait(fullspots);      { make sure it is not empty }
wait(access);          { wait for access }
FETCH(VALUE);
release(access);       { relinquish access }
release(emptyspots);   { increase empty spaces }
-- consume VALUE --
end loop;
end consumer;
```



uOttawa

L'Université canadienne
Canada's university

EXEMPLE: PRODUCTEUR CONSOMMATEUR

MONITOR: BufferMonitor

```
const bufferSize = 5

buffer = array [0.. bufferSize-1]

next_in = 0, next_out = 0, filled = 0

condition not_full, not_empty

procedure void deposit (item)

begin

while filled == bufferSize then

    wait(not_full) // block thread and place it in the not_full queue

end

buffer[next_in] = item

next_in = (next_in + 1) mod bufferSize

filled = filled + 1

signal(not_empty) // free a task that has been waiting on not_empty

end procedure
```



uOttawa

L'Université canadienne
Canada's university

EXEMPLE: PRODUCTEUR CONSOMMATEUR

```
procedure Item fetch()

begin

    while filled == 0 then

        wait(not_empty) // block thread and place it in the not_empty queue


    end

    item = buffer[next_out]

    next_out = (next_out + 1) mod bufferSize

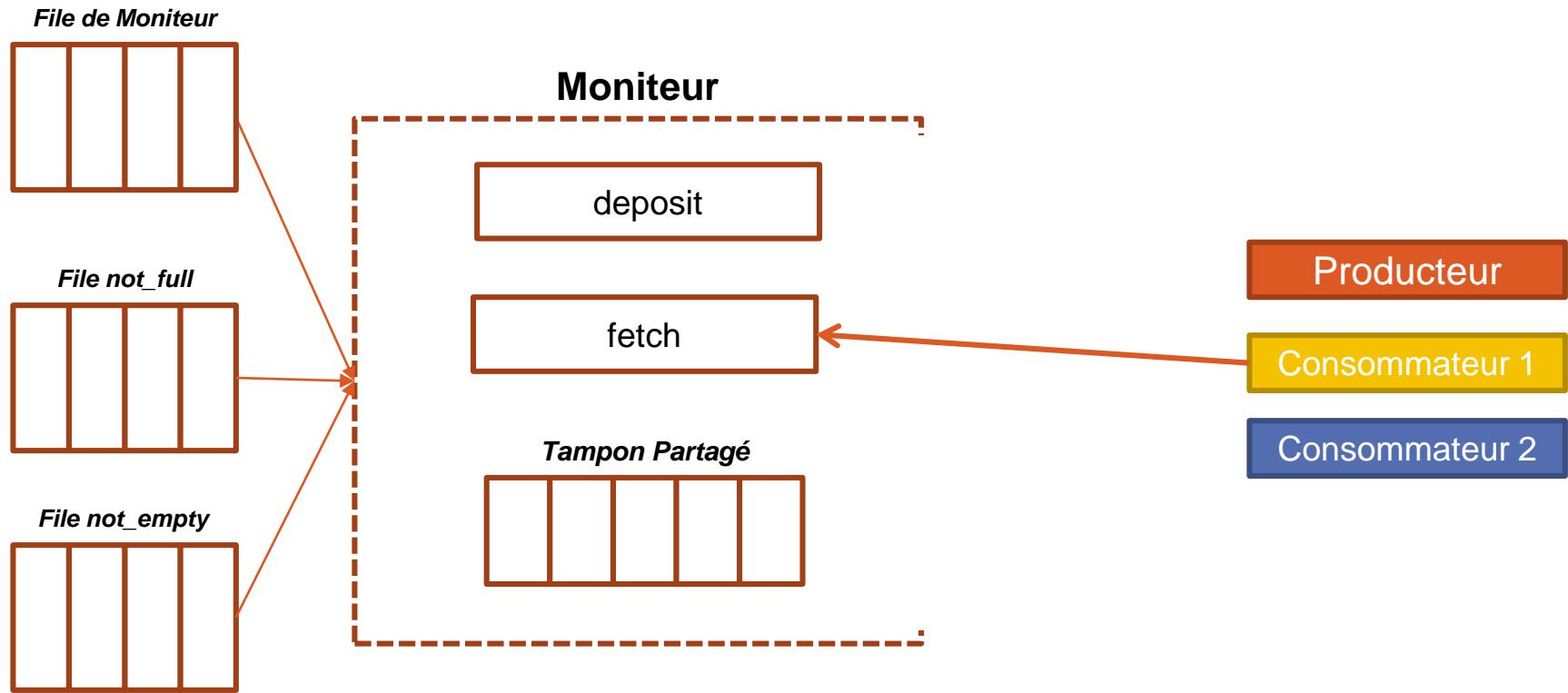
    filled = filled - 1


    signal(not_full) // free a task that has been waiting on not_full


return item

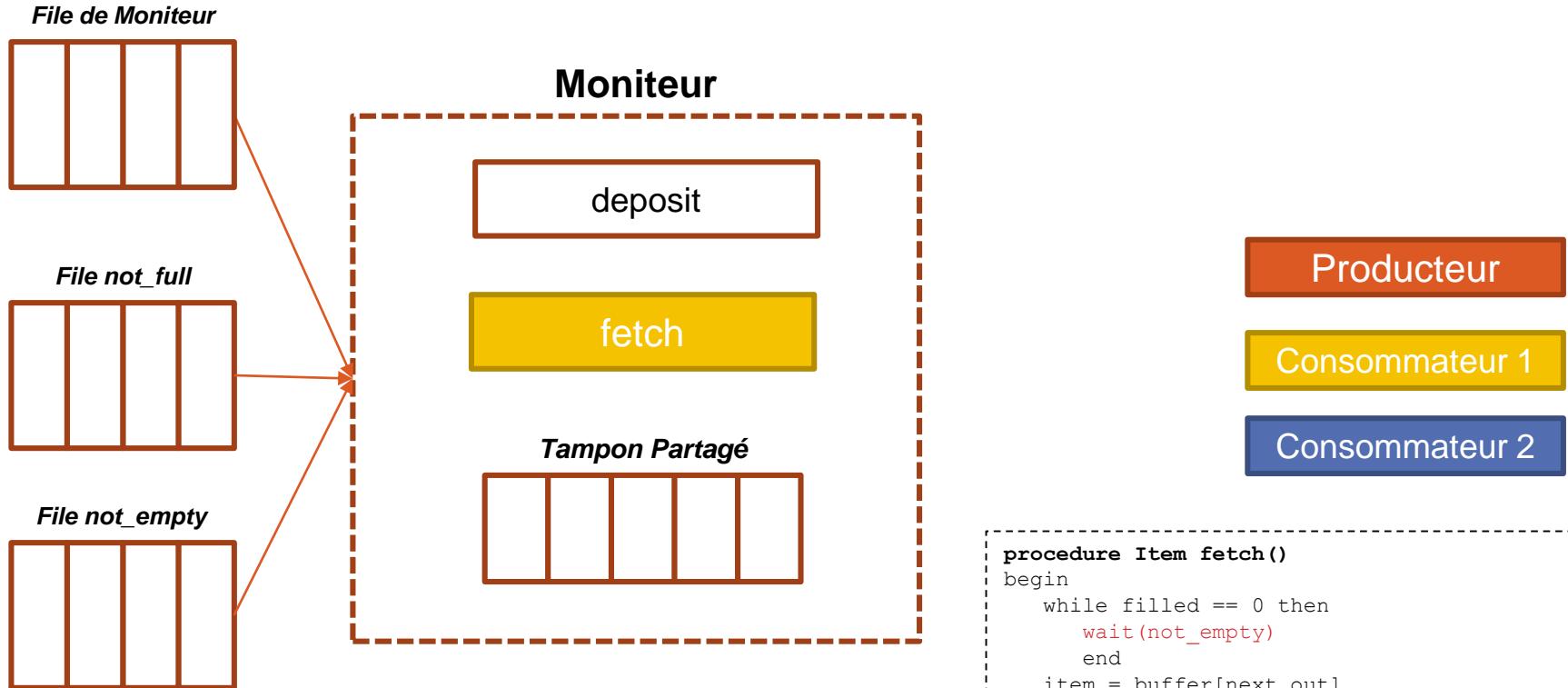
end procedure
```

EXEMPLE: PRODUCTEUR CONSUMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



```

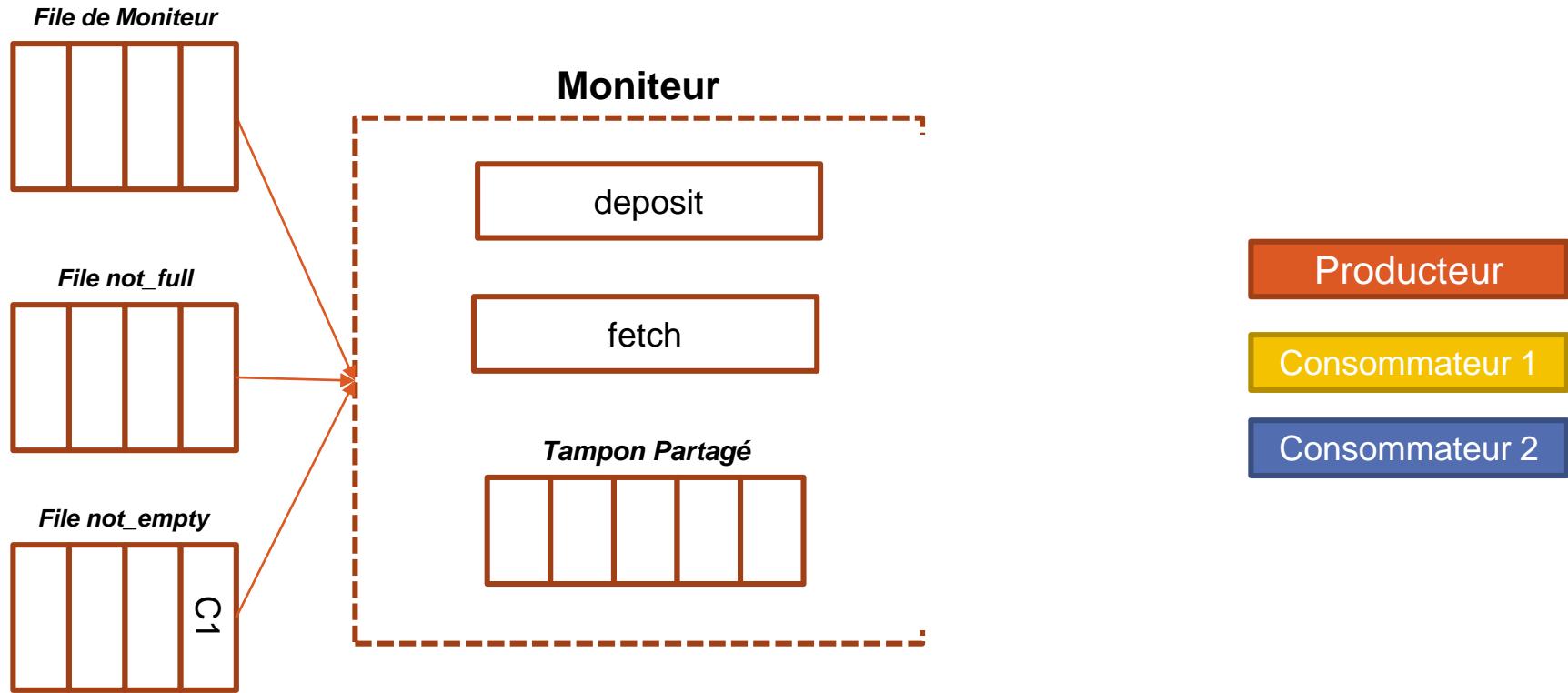
procedure Item fetch()
begin
    while filled == 0 then
        wait(not_empty)
    end
    item = buffer[next_out]
    next_out = (next_out + 1) mod bufferSize
    filled = filled - 1

    signal(not_full)
    return item
end procedure

```

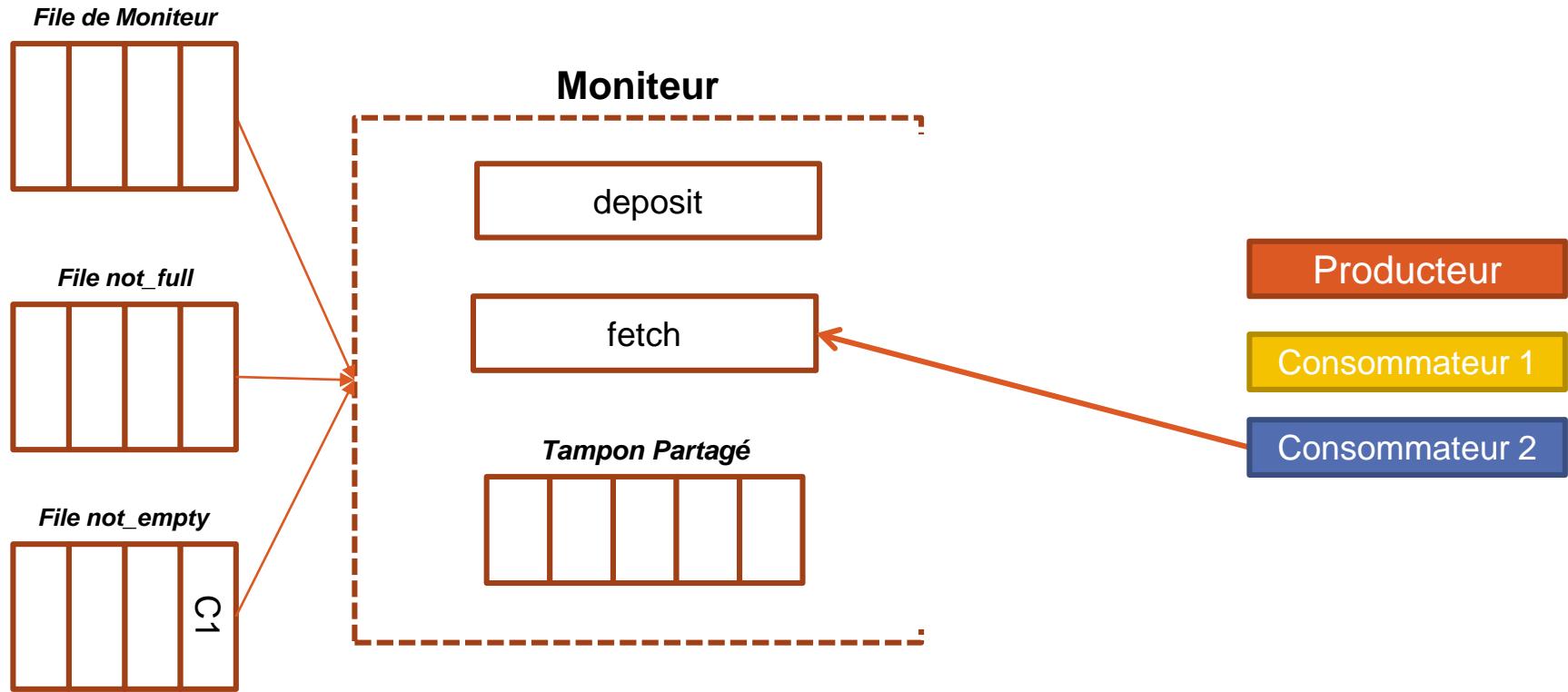
Propriétaire du Moniteur: Consommateur 1

EXEMPLE: PRODUCTEUR CONSUMMATEUR



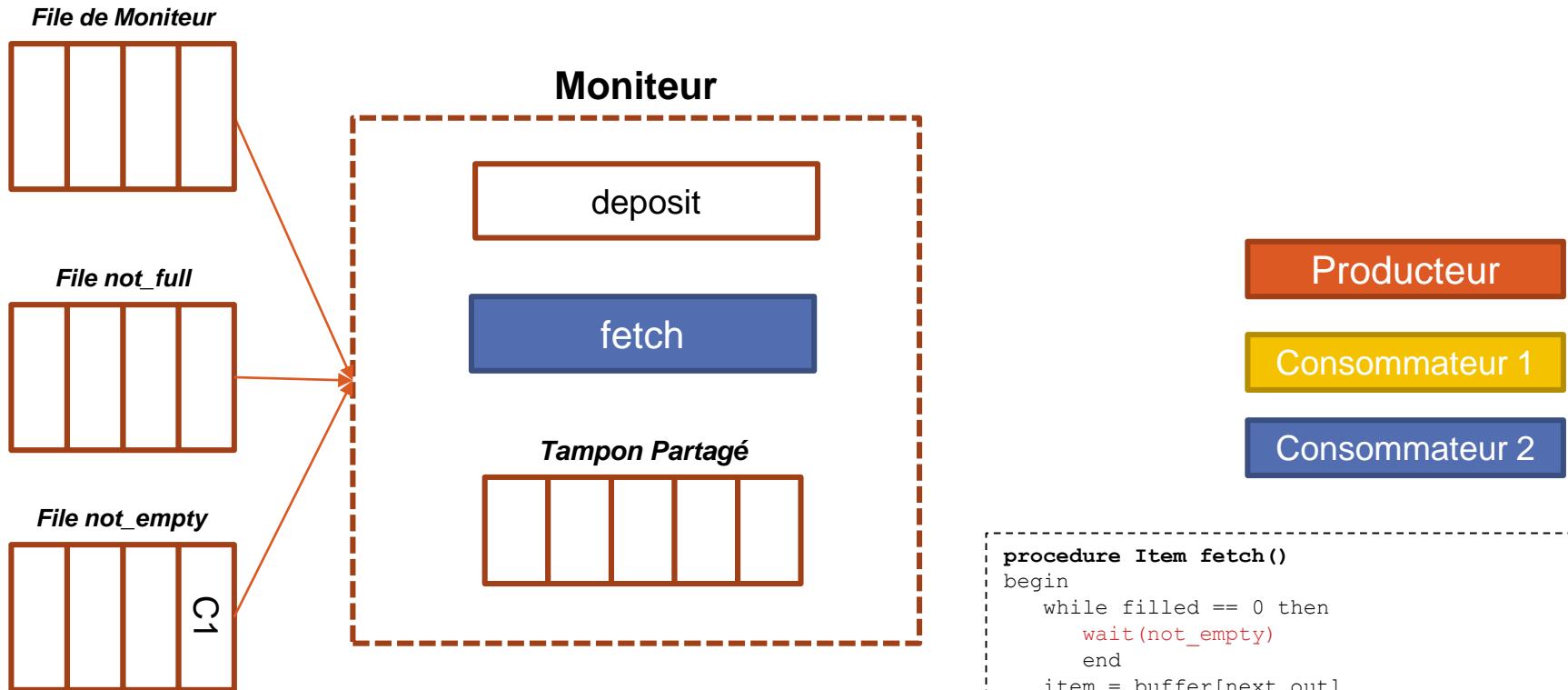
Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSUMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



```

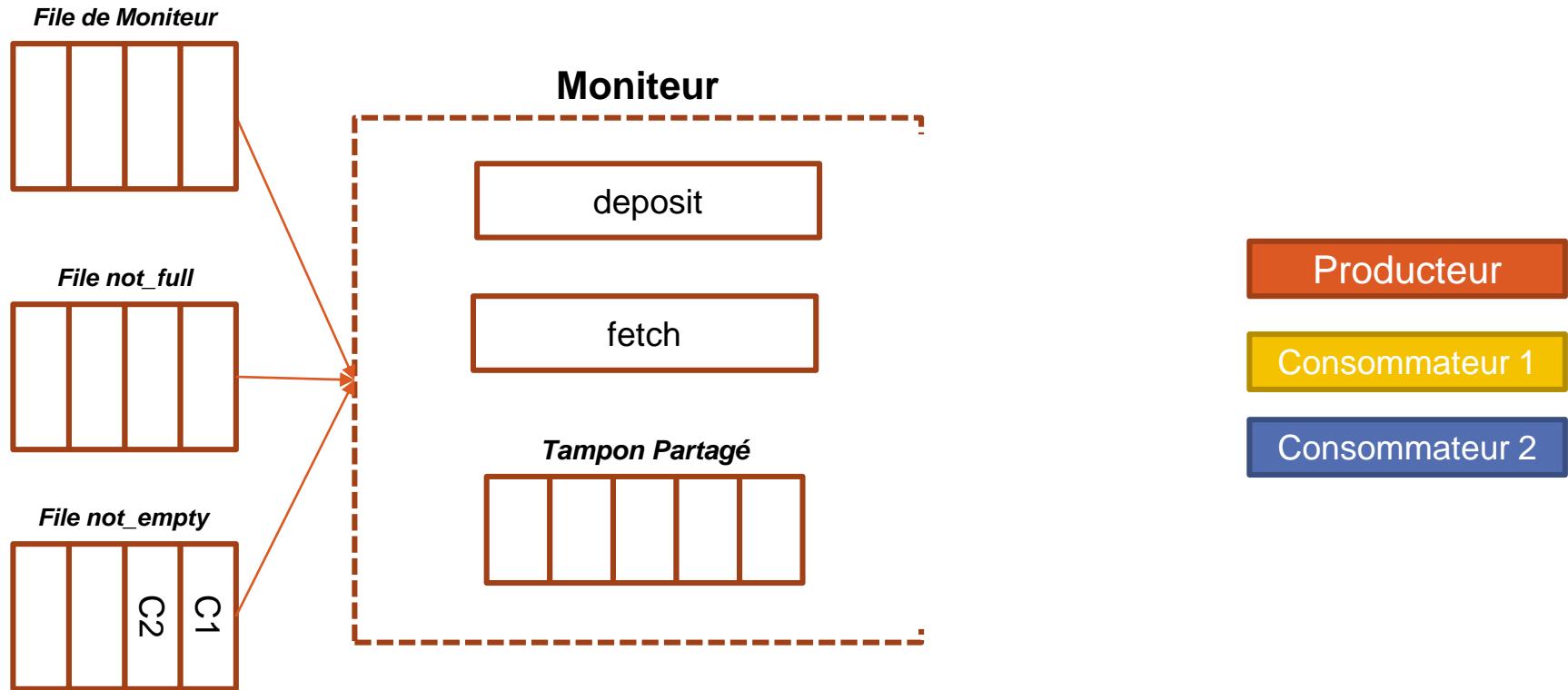
procedure Item fetch()
begin
    while filled == 0 then
        wait(not_empty)
    end
    item = buffer[next_out]
    next_out = (next_out + 1) mod bufferSize
    filled = filled - 1

    signal(not_full)
    return item
end procedure

```

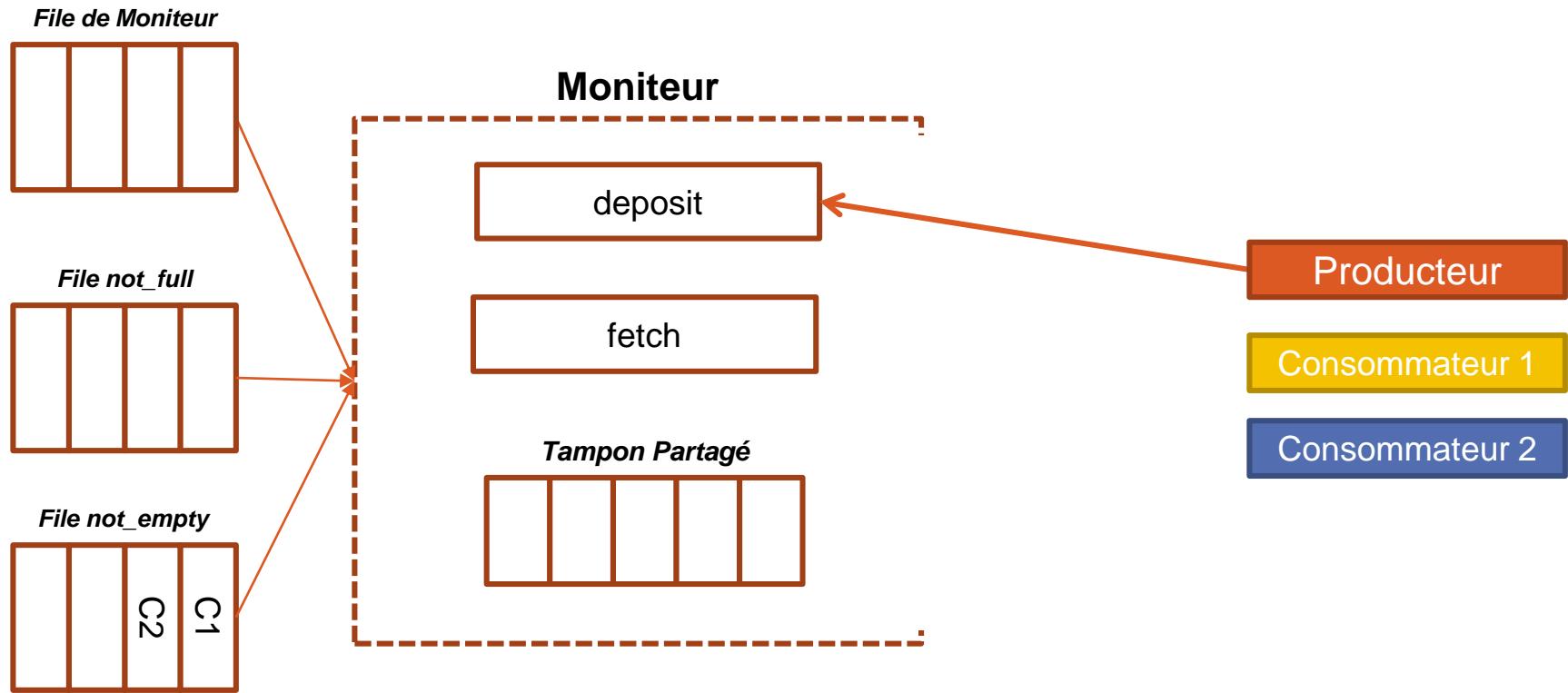
Propriétaire du Moniteur: Consommateur 2

EXEMPLE: PRODUCTEUR CONSUMMATEUR



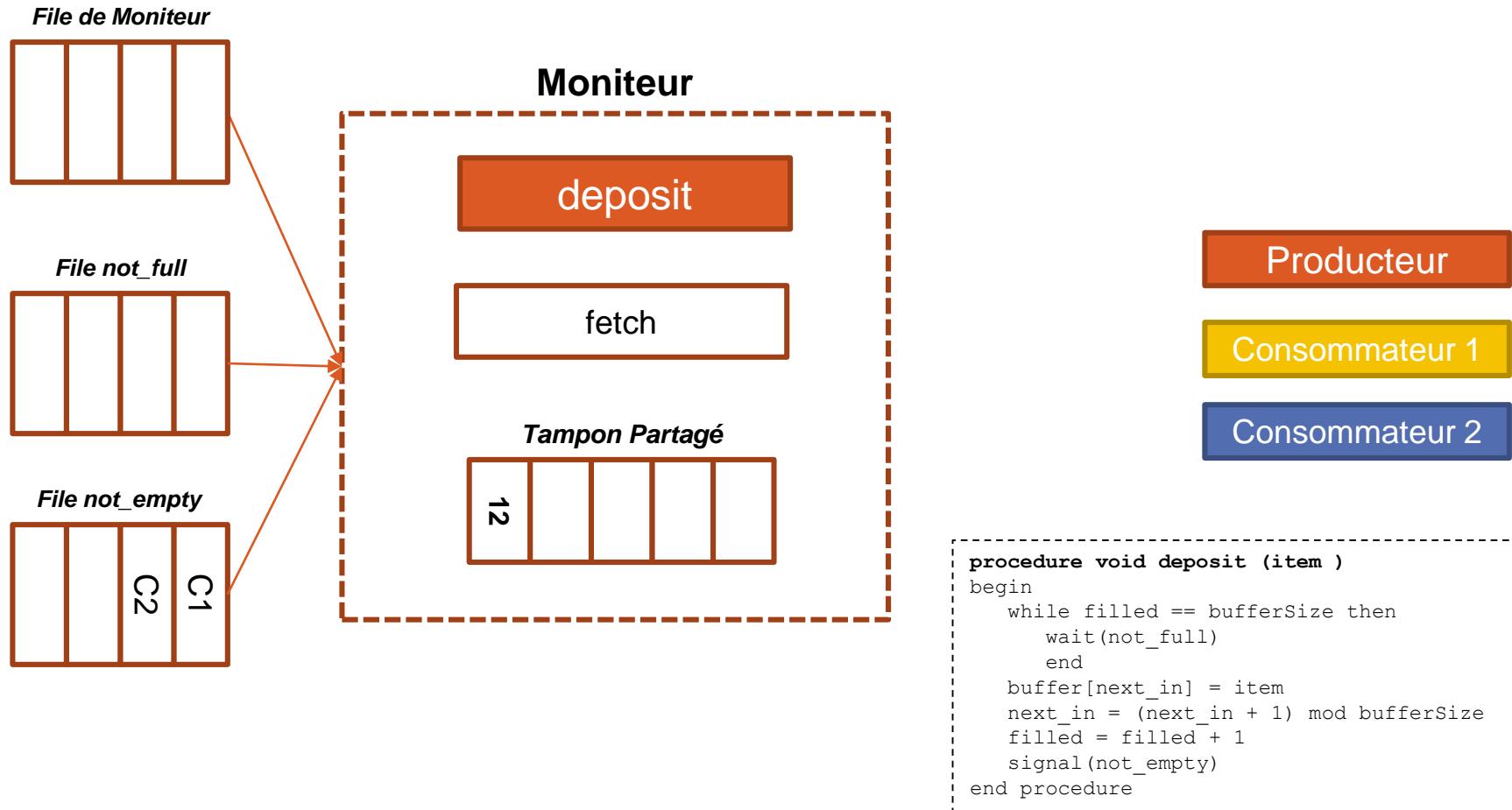
Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSUMMATEUR



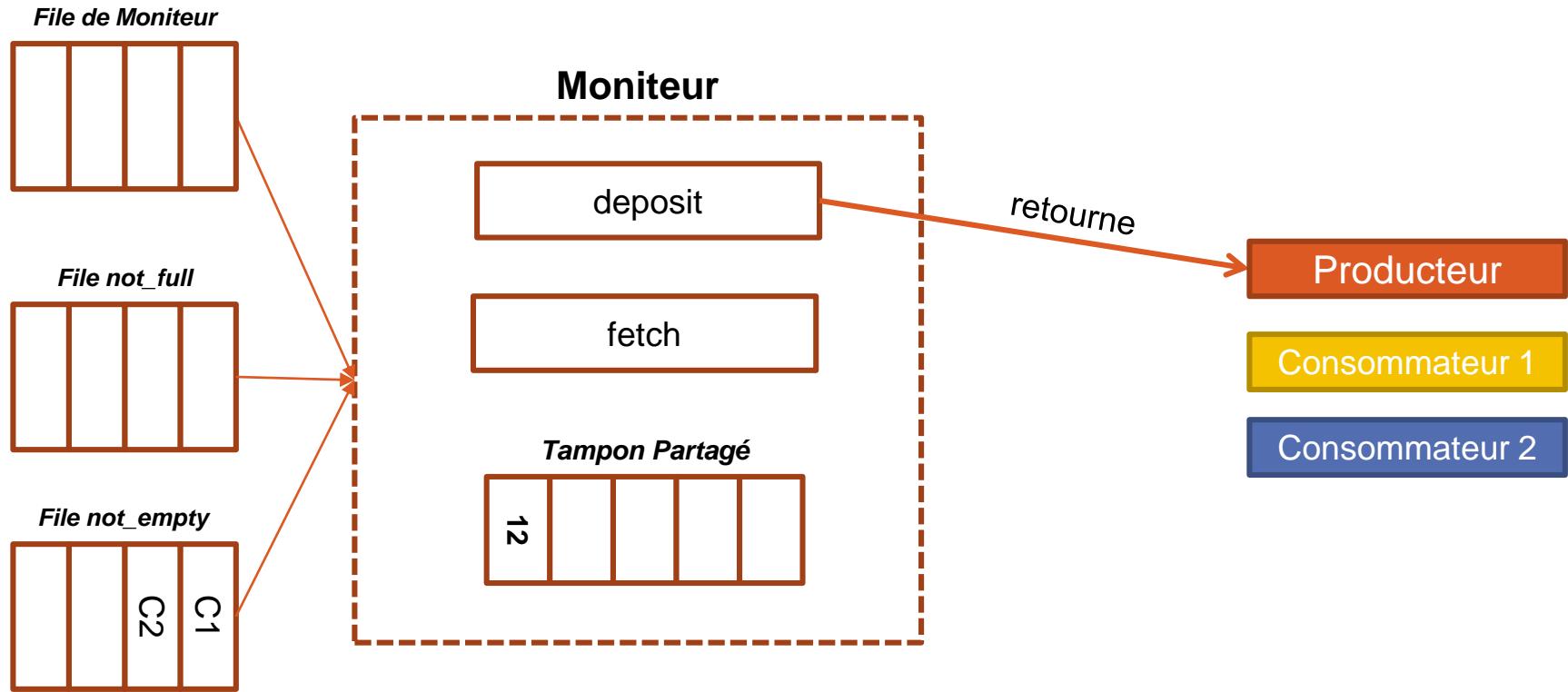
Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



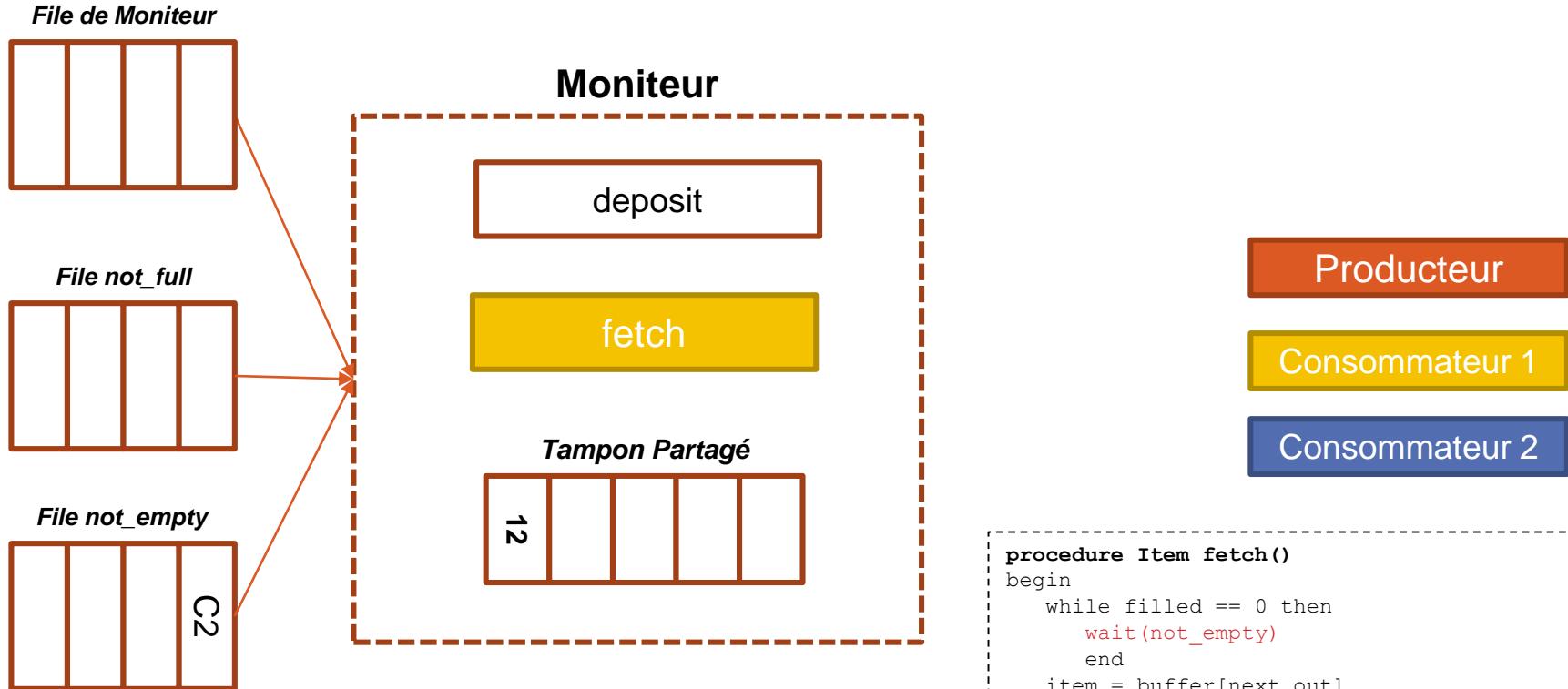
Propriétaire du Moniteur: Producteur

EXEMPLE: PRODUCTEUR CONSUMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSOMMATEUR



```

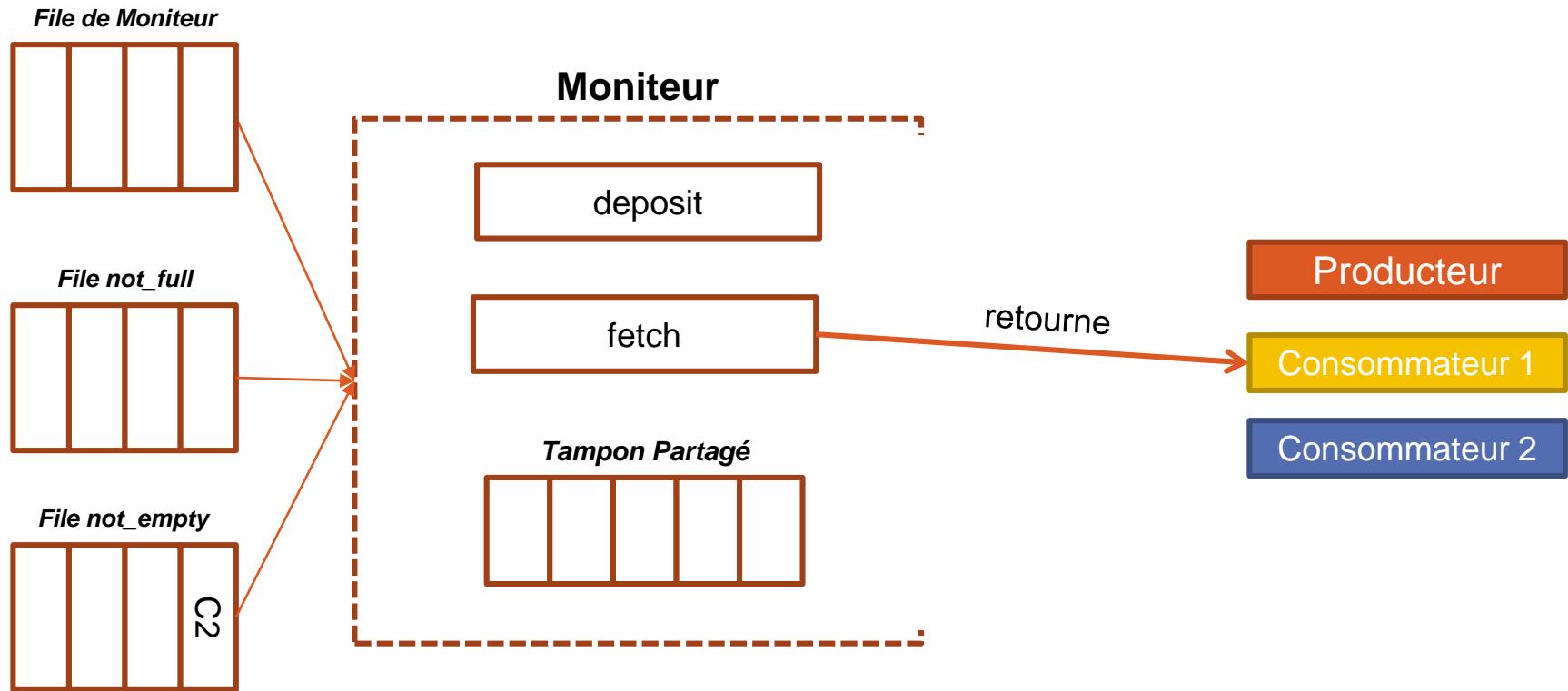
procedure Item fetch()
begin
    while filled == 0 then
        wait(not_empty)
    end
    item = buffer[next_out]
    next_out = (next_out + 1) mod bufferSize
    filled = filled - 1

    signal(not_full)
    return item
end procedure

```

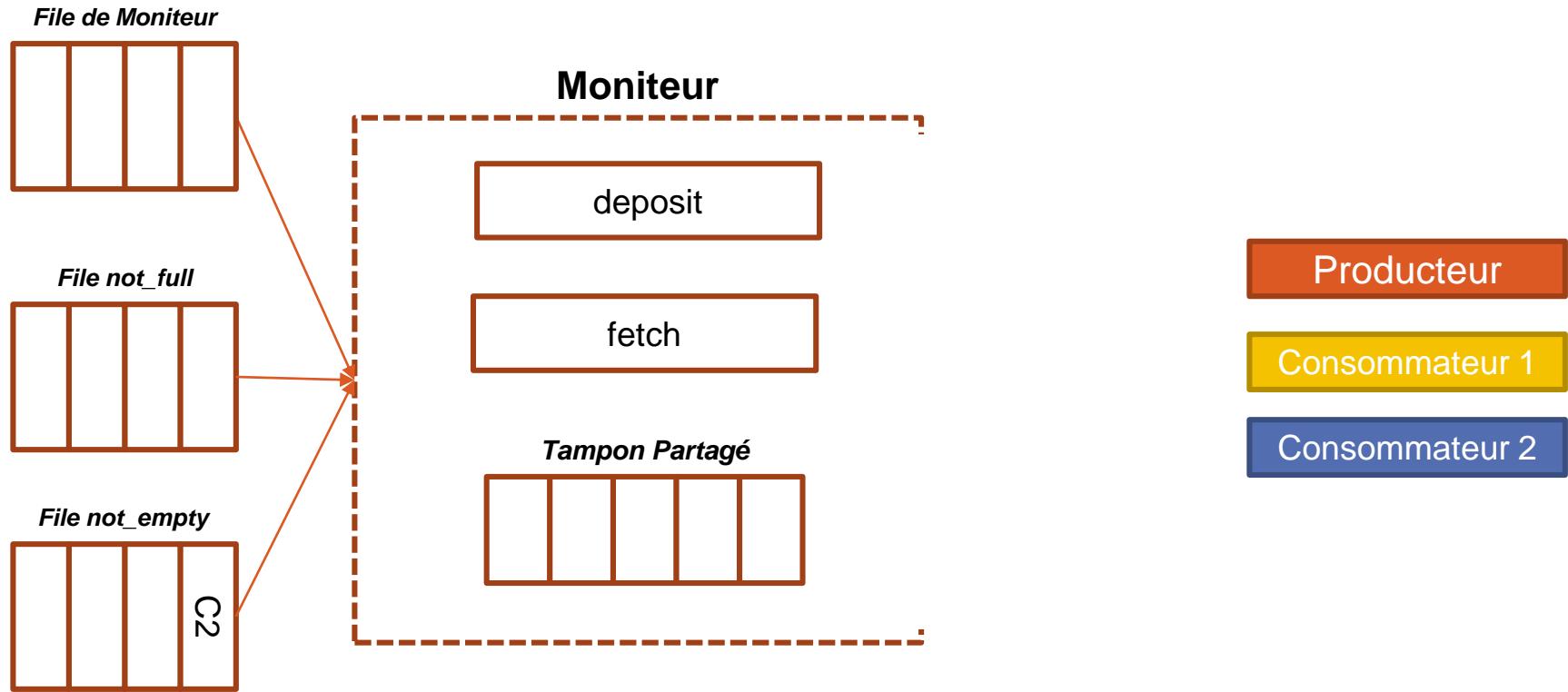
Propriétaire du Moniteur: Consommateur 1

EXEMPLE: PRODUCTEUR CONSUMMATEUR



Propriétaire du Moniteur: Aucun

EXEMPLE: PRODUCTEUR CONSUMMATEUR



Propriétaire du Moniteur: Aucun

MERCI!

QUESTIONS?

SÉANCE 16

CONCURRENCE JAVA



uOttawa

L'Université canadienne
Canada's university

SUJETS

Encore plus sur les Principes de Base des Threads Java

- Sleep
- Join
- Interrupt

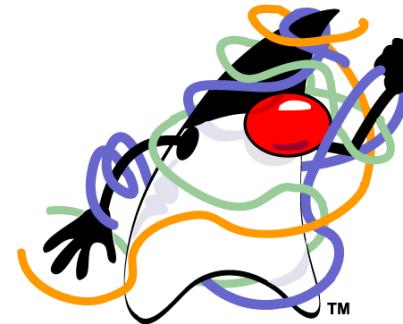
Verrous intrinsèques de Java

Moniteurs Implémentés avec des Verrous intrinsèques

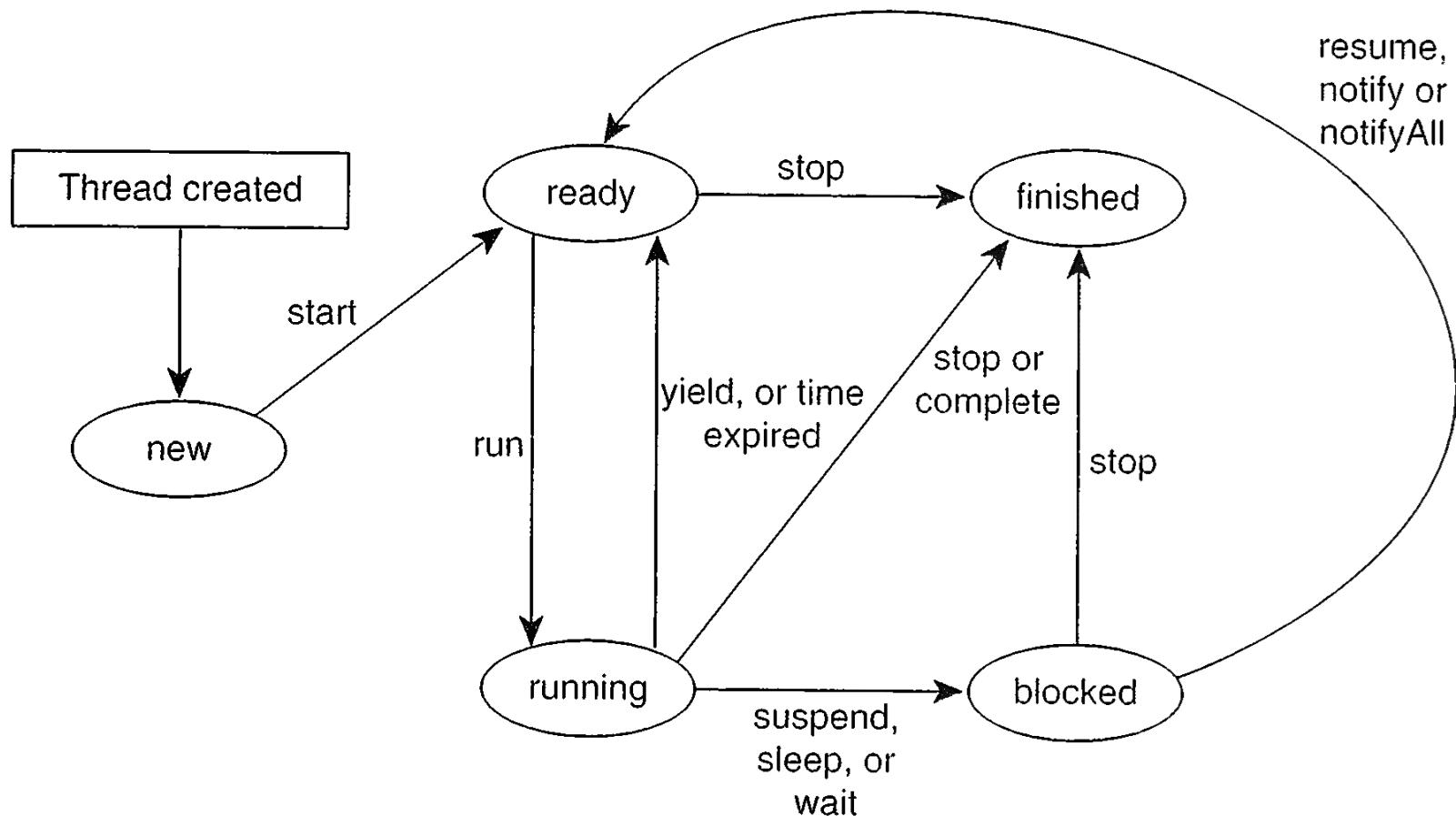
Moniteurs Implémentés avec des Interfaces Lock et Condition

Sémaphores Java

Variables Atomiques



ÉTATS THREAD





uOttawa

L'Université canadienne
Canada's university

CRÉER DES THREADS EN ÉTENDANT LA CLASSE THREAD

```
// Custom thread class
public class CustomThread extends Thread
{
    ...
    public CustomThread(...)
    {
        ...
    }
    ...

    // Override the run method in Thread
    public void run()
    {
        // Tell system how to run custom thread
        ...
    }
    ...
}
```

```
// Client class
public class Client
{
    ...
    public someMethod()
    {
        ...
        // Create a thread
        CustomThread thread = new CustomThread(...);

        // Start a thread
        thread.start();
        ...
    }
    ...
}
```

CRÉER DES THREADS EN IMPLÉMENTANT L'INTERFACE EXÉCUTABLE

```
// Custom thread class
public class CustomThread
    implements Runnable
{
    ...
    public CustomThread(...)
    {
        ...
    }

    // Implement the run method in Runnable
    public void run()
    {
        // Tell system how to run custom thread
        ...
    }
    ...
}
```

```
// Client class
public class Client
{
    ...
    public someMethod()
    {
        ...
        // Create an instance of CustomThread
        CustomThread customThread
            = new CustomThread(...);

        // Create a thread
        Thread thread = new Thread(customThread);

        // Start a thread
        thread.start();
        ...
    }
    ...
}
```



uOttawa

L'Université canadienne
Canada's university

THREAD GROUPS

Un thread group est un ensemble de threads

Certains programmes contiennent plusieurs threads avec des fonctions similaires

- On peut les regrouper ensemble et effectuer des opérations sur le groupe entier

Ex., on peut suspendre ou résumer tous les threads dans un groupe en même temps.



uOttawa

L'Université canadienne
Canada's university

UTILISATION DES GROUPES DE THREAD

Construisez un thread group:

```
ThreadGroup g = new ThreadGroup("thread  
group");
```

Placez un thread dans le thread group:

```
Thread t = new Thread(g, new ThreadClass());
```

Trouvez combien de threads dans un groupe sont couramment en train d'exécuter:

```
System.out.println("the number of runnable  
threads in the group " + g.activeCount());
```

Trouvez à quel groupe le thread appartient-il:

```
theGroup = t.getThreadGroup();
```



uOttawa

L'Université canadienne
Canada's university

PRIORITÉS

Les priorités des threads n'ont pas besoin d'être identiques

La priorité du thread par défaut est:

- NORM_PRIORITY(5)

La priorité est un nombre entier entre 0 et 10, où:

- MAX_PRIORITY(10)
- MIN_PRIORITY(0)

Vous pouvez utiliser:

- `setPriority(int)`: changer la priorité de ce thread
- `getPriority()`: retourner la priorité de ce thread



uOttawa

L'Université canadienne
Canada's university

SLEEP

Faire un thread dormir (**sleep**) pour plusieurs millisecondes

- Très populaire avec les applications de jeux (animation 2D ou 3D)

```
Thread.sleep(1000);
```

```
Thread.sleep(1000, 1000); (la précision dépend du système)
```

- Notez que ces méthodes sont statiques
- Elles génèrent un InterruptedException



uOttawa

L'Université canadienne
Canada's university

JOIN

Si vous créez plusieurs threads, chacun est responsable pour des calculs

Vous pouvez attendre que le thread meurt (die**)**

- Avant de rassembler les résultats de ces threads

Afin d'achever ceci, on utilise la méthode « join » définie dans la classe Thread

```
try {thread.join();}  
  
catch (InterruptedException e) {e.printStackTrace();}
```

L'Exception est générée si un autre thread a interrompu le thread courant



uOttawa

L'Université canadienne
Canada's university

EXAMPLE JOIN

```
public class JoinThread {  
  
    public static void main(String[] args) {  
  
        Thread thread2 = new Thread(new WaitRunnable());  
  
        Thread thread3 = new Thread(new WaitRunnable());  
  
  
        thread2.start();  
  
        try {thread2.join();} catch (InterruptedException e) {e.printStackTrace();}  
  
  
        thread3.start();  
  
        try {thread3.join(1000);} catch (InterruptedException e) {e.printStackTrace();}  
    }  
}
```

INTERRUPTION DES THREADS

Dans Java, on n'a aucune façon de forcer un Thread d'arrêter

- Si le Thread n'est pas implémenté correctement, il peut continuer son exécution indéfiniment

Mais on peut interrompre un Thread avec la méthode `interrupt()`

- Si le Thread dort (sleep) ou joint (join) un autre Thread, un `InterruptedException` est généré
- Dans ce cas, le statut interrompu du thread est remis à faux (`false`)



uOttawa

L'Université canadienne
Canada's university

EXAMPLE INTERRUPT

```
public class InterruptThread {  
  
    public static void main(String[] args) {  
  
        Thread thread1 = new Thread(new WaitRunnable());  
        thread1.start();  
  
        try {Thread.sleep(1000);}  
        catch (InterruptedException e){e.printStackTrace();}  
        thread1.interrupt();  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

EXAMPLE INTERRUPT

```
private static class WaitRunnable implements Runnable {  
  
    @Override  
  
    public void run() {  
  
        System.out.println("Current time millis: " + System.currentTimeMillis());  
  
        try {Thread.sleep(5000);}  
  
        catch (InterruptedException e) {  
  
            System.out.println("The thread has been interrupted");  
  
        }  
  
        System.out.println("Current time millis: " + System.currentTimeMillis());  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

VERROUS INTRINSÈQUES

Tout le code qui peut être modifié simultanément par plusieurs threads doit être **Thread Safe**

Considérez le code simple suivant:

```
public int getNextCount () {  
    return ++counter;  
}
```

Un incrément comme ceci n'est pas une action **atomique**, il implique:

- Lire la valeur actuelle de `counter`
- Ajouter un (1) à sa valeur actuelle
- Stocker le résultat dans la mémoire



uOttawa

L'Université canadienne
Canada's university

VERROUS INTRINSÈQUES

Si on a deux threads qui invoquent `getNextCount()`, la séquence suivante d'événements peut se passer (parmi plusieurs scénarios possibles):

- | | |
|---|---|
| <p>1 <i>Thread 1 : reads counter, gets 0, adds 1, so counter = 1</i></p> <p>3 <i>Thread 1 : writes 1 to the counter field and returns 1</i></p> | <p>2 <i>Thread 2 : reads counter, gets 0, adds 1, so counter = 1</i></p> <p>4 <i>Thread 2 : writes 1 to the counter field and returns 1</i></p> |
|---|---|

Conséquemment, on doit utiliser un verrou sur l'accès à “counter”

On peut ajouter un tel verrou à une méthode en utilisant simplement le mot-clé: **synchronized**

```
public synchronized int getNextCount() {  
    return ++counter;  
}
```

Ceci garantit que seulement un thread exécute la méthode

Si on a plusieurs méthodes avec le mot-clé synchronisé, seulement une méthode peut être exécutée à la fois

- Ceci s'appelle un **Verrou intrinsèque**



uOttawa

L'Université canadienne
Canada's university

VERROUS INTRINSÈQUES

Chaque objet Java possède un verrou intrinsèque associé avec lui (souvent appelé tout simplement moniteur)

Dans le dernier exemple, on a utilisé ce verrou pour synchroniser l'accès à une méthode

- Au lieu, on peut choisir de synchroniser l'accès à un bloc (ou segment) de code

```
public int getNextValue() {  
    synchronized (this) {return value++;}  
}
```

- Alternativement, on peut utiliser le verrou d'un autre objet

```
public int getNextValue() {  
    synchronized (lock) {return value++;}  
}
```

- Ce dernier est utile puisqu'il nous permet d'utiliser plusieurs verrous pour la sécurité du thread dans une classe simple



uOttawa

L'Université canadienne
Canada's university

VERROUS INTRINSÈQUES

Alors, on a mentionné que chaque objet Java possède un verrou intrinsèque associé avec lui

Qu'en est-il des méthodes statiques qui ne sont pas associées avec un objet particulier?

- Il existe aussi un verrou intrinsèque associé avec la classe
- Utilisé seulement pour les méthodes à classe synchronisée (statique)



uOttawa

L'Université canadienne
Canada's university

MONITEURS JAVA

Mauvaises nouvelles: Dans Java, il n'y a pas un mot-clé pour créer directement un moniteur

Bonnes nouvelles: il existe plusieurs mécanismes pour créer des moniteurs

- Le plus simple, utilise les connaissances qu'on a déjà accumulées concernant les **verrous intrinsèques**



uOttawa

L'Université canadienne
Canada's university

MONITEURS JAVA

Les verrous intrinsèques peuvent être utilisés effectivement pour l'exclusion mutuelle (synchronisation de compétition**)**

On a besoin d'un mécanisme pour implémenter la synchronisation de coopération

- En particulier, on a besoin de permettre aux threads de se suspendre si une condition empêche leur exécution dans un moniteur
- Ceci est fait en utilisant les méthodes `wait()` et `notify()`

Ces deux méthodes sont très importantes qu'elles ont été définies dans la classe Object...



uOttawa

L'Université canadienne
Canada's university

OPÉRATIONS “WAIT”

`wait()`

Permet au thread actuel d'abandonner le moniteur et attendre (**wait**) jusqu'à ce qu'un autre thread entre dans le même moniteur et fait appel à **notify()** ou **notifyAll()**

`wait(long timeout)`

Permet au thread actuel d'attendre (**wait**) jusqu'à ce qu'un autre thread invoque la méthode **notify()** ou **notifyAll()** , ou bien que le temps spécifié est terminé



uOttawa

L'Université canadienne
Canada's university

OPÉRATIONS “NOTIFY”

notify()

Réveille un seule thread qui attend sur le moniteur de cet objet (verrou intrinsèque). Si plus qu'un seul thread attendent, le choix est arbitraire (est-ce que c'est juste?)

Le thread réveillé ne pourra pas procéder jusqu'à ce que le thread actuel quitte le verrou.

notifyAll()

Réveille tous les threads qui attendent le moniteur de cet objet.

Le thread réveillé ne pourra pas procéder jusqu'à ce que le thread actuel quitte le verrou.

Le thread suivant qui verrouille ce moniteur est aussi choisi au hasard

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

```
public class BufferMonitor{

    int [] buffer = new int [5];

    int next_in = 0, next_out = 0, filled = 0;

    public synchronized void deposit (int item) throws InterruptedException{
        while (buffer.length == filled){
            wait(); // blocks thread
        }

        buffer[next_in] = item;
        next_in = (next_in + 1) % buffer.length;
        filled++;

        notify(); // free a task that has been waiting on a condition
    }
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

```
public synchronized int fetch() throws InterruptedException{
    while (filled == 0) {
        wait(); // block thread
    }

    int item = buffer[next_out];
    next_out = (next_out + 1) % buffer.length;
    filled--;
}

notify(); // free a task that has been waiting on a condition

return item;
}
```

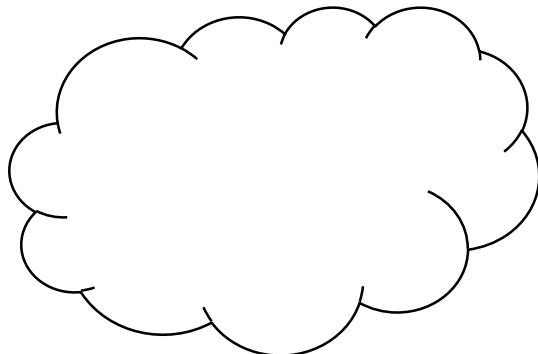
EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

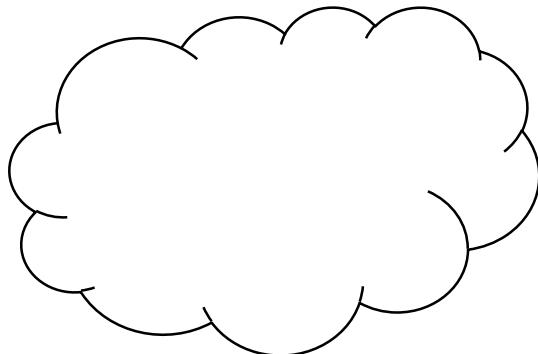


Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Espace d'attente

Producer
Consumer1
Consumer2

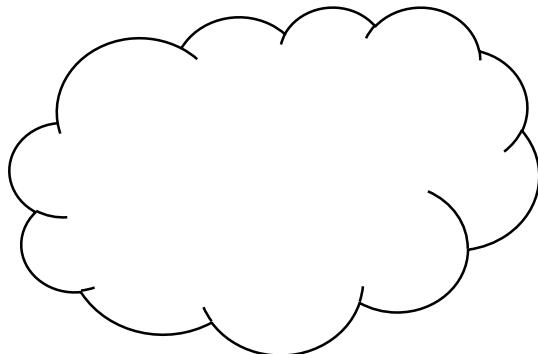
Consumer2 appelle **fetch**

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Espace d'attente

Producer
Consumer1
Consumer2

Consumer2 appelle **fetch**

```
public synchronized int fetch()  
throws InterruptedException{  
  
    → while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

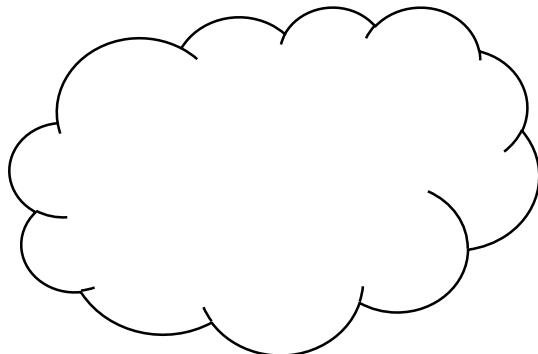


uOttawa

L'Université canadienne
Canada's university

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Espace d'attente

Producer
Consumer1
Consumer2

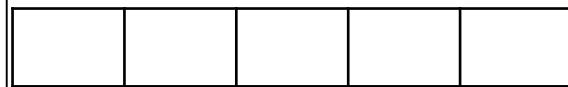
Consumer2 appelle **fetch**

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        →      wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer2 lâche le verrou

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 appelle **fetch**

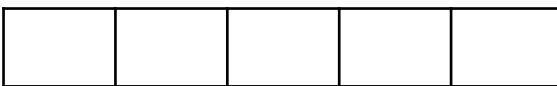
```
public synchronized int fetch()  
throws InterruptedException{  
  
    → while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 appelle **fetch**

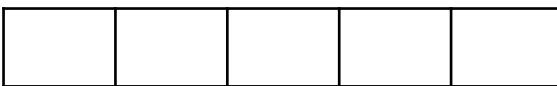
```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        →      wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 lâche le verrou

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

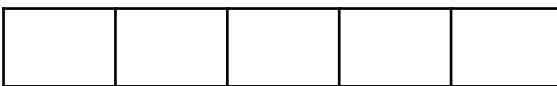


Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Producer appelle **deposit (12)**

```
public synchronized void deposit (int item )  
    throws InterruptedException {  
  
    → while (buffer.length == filled) {  
        wait();  
    }  
  
    buffer[next_in] = item;  
    next_in = (next_in + 1) % buffer.length;  
    filled++;  
  
    notify();  
}
```

Consumer1
Consumer2

Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 0  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Producer appelle **deposit (12)**

```
public synchronized void deposit (int item )  
throws InterruptedException {  
  
    while (buffer.length == filled) {  
        wait();  
    }  
  
    → buffer[next_in] = item;  
    next_in = (next_in + 1) % buffer.length;  
    filled++;  
  
    notify();  
}
```

Consumer1
Consumer2

Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 0  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Producer appelle **deposit (12)**

```
public synchronized void deposit (int item )  
throws InterruptedException {  
  
    while (buffer.length == filled) {  
        wait();  
    }  
  
    buffer[next_in] = item;  
    next_in = (next_in + 1) % buffer.length;  
    filled++;  
  
    notify();  
}
```



EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 0  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Producer appelle **deposit (12)**

```
public synchronized void deposit (int item )  
throws InterruptedException {  
  
    while (buffer.length == filled) {  
        wait();  
    }  
  
    buffer[next_in] = item;  
    next_in = (next_in + 1) % buffer.length;  
    filled++;  
  
    notify();  
}
```

Consumer1
Consumer2

Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 0  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Producer appelle **deposit (12)**

```
public synchronized void deposit (int item )  
throws InterruptedException {  
  
    while (buffer.length == filled) {  
        wait();  
    }  
  
    buffer[next_in] = item;  
    next_in = (next_in + 1) % buffer.length;  
    filled++;  
  
    → notify();  
}
```

Consumer1
Consumer2

Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

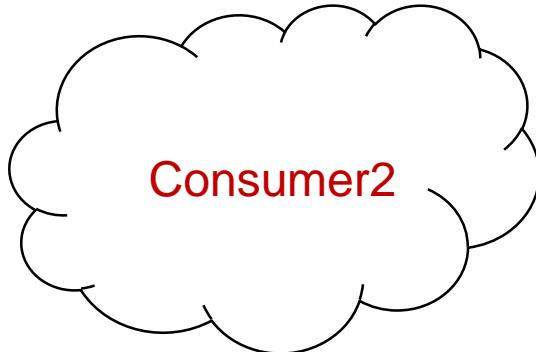
```
next_in = 1  
  
next_out = 0  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        → wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 0  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    → while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 0  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    → int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 1;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    → int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    → notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE



uOttawa

L'Université canadienne
Canada's university

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    → notify();  
    return item;  
}
```



EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer1 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    → notify();  
    return item;  
}
```



Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE



uOttawa

L'Université canadienne
Canada's university

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Producer
Consumer1
Consumer2

fetch retourne 12

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    → return item;  
}
```

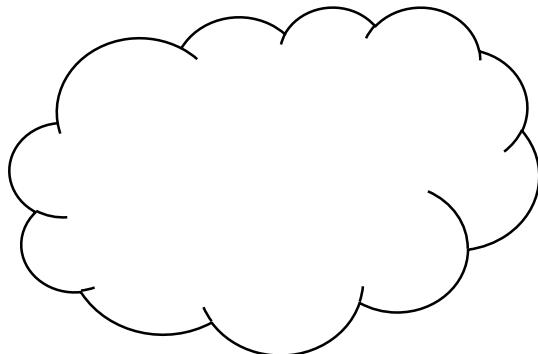


Espace d'attente

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Espace d'attente

Producer
Consumer1
Consumer2

Consumer2 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

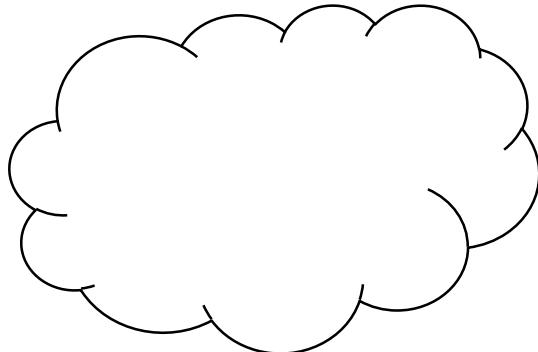


uOttawa

L'Université canadienne
Canada's university

BufferMonitor:

```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Espace d'attente

Producer
Consumer1
Consumer2

Consumer2 est débloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    → while (filled == 0){  
        wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```

EXEMPLE DE MONITEUR BASÉ SUR UN VERROU INTRINSÈQUE

BufferMonitor:

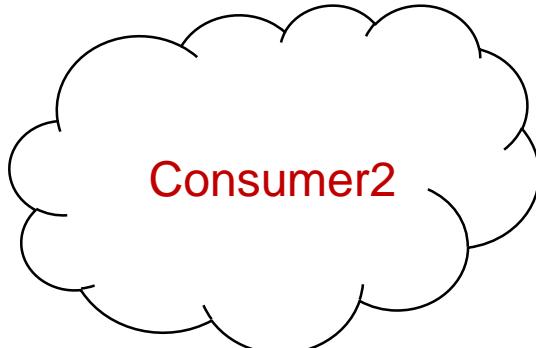
```
next_in = 1  
  
next_out = 1  
  
filled = 0;
```



Producer
Consumer1
Consumer2

Consumer2 est bloqué

```
public synchronized int fetch()  
throws InterruptedException{  
  
    while (filled == 0){  
        →    wait();  
    }  
  
    int item = buffer[next_out];  
    next_out = (next_out + 1) % buffer.length;  
    filled--;  
  
    notify();  
    return item;  
}
```



Espace d'attente

UN AUTRE MÉCANISME POUR CRÉER DES MONITEURS

On peut aussi créer un moniteur en utilisant l'interface Java `Lock`

- `ReentrantLock` est l'implémentation la plus populaire de `Lock`

`ReentrantLock` définit deux constructeurs:

- Constructeur de Défaut
- Constructeur qui prend un Booléen (spécifiant si le verrou est juste)

Dans une situation de verrou juste, les threads vont avoir accès au verrou dans le même ordre qu'ils l'ont demandé (FIFO)

- Sinon, le verrou ne garantit aucun ordre particulier
- La justesse exige plus de computations (en termes de traitement), et donc, doit être seulement utilisé si requis

Afin d'acquérir le verrou, vous devez simplement utiliser la méthode `lock`, et pour le lâcher, faites appel à `unlock`



uOttawa

L'Université canadienne
Canada's university

EXAMPLE “LOCK”

```
public class SimpleMonitor {  
  
    private final Lock lock = new ReentrantLock();  
  
    public void testA() {  
  
        lock.lock();  
  
        try { //Some code}  
  
        finally {lock.unlock();}  
  
    }  
  
    public int testB() {  
  
        lock.lock();  
  
        try {return 1;}  
  
        finally {lock.unlock();}  
  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

QUESTION??

*Pourquoi devons-nous avoir un bloc try-
finally dans l'exemple précédent?*





UN AUTRE MÉCANISME POUR CRÉER DES MONITEURS

Comment implémenter les conditions?

- Sans pouvoir attendre une condition, les moniteurs seront inutiles...
 - *La Coopération n'est pas possible*

Il existe une classe spécifique qui a été développée juste pour cet effet: **Classe Condition**

- On crée un exemple de **Condition** en utilisant la méthode **newCondition()** définie dans l'interface **Lock**



uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE MONITEUR TAMPON (ENCORE)

```
public class BufferMonitor {  
  
    int [] buffer = new int [100];  
    int next_in = 0, next_out = 0, filled = 0;  
  
    private final Lock lock = new ReentrantLock(true);  
    private final Condition notFull = lock.newCondition();  
    private final Condition notEmpty = lock.newCondition();
```

EXEMPLE DE MONITEUR TAMPON (ENCORE)

```
public void deposit (int item ) throws InterruptedException{  
    lock.lock(); // Lock to ensure mutually exclusive access  
    try{  
        while (buffer.length == filled) {  
            notFull.await(); // blocks thread (wait on condition)  
        }  
        buffer[next_in] = item;  
        next_in = (next_in + 1) % buffer.length;  
        filled++;  
        notEmpty.signal(); // signal thread waiting on the empty condition  
    }  
    finally{  
        lock.unlock(); // Whenever you lock, you must unlock  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE MONITEUR TAMPON (ENCORE)

```
public void fetch () throws InterruptedException{  
    lock.lock(); // Lock to ensure mutually exclusive access  
    try{  
        while (filled == 0) {  
            notEmpty.await(); // blocks thread (wait on condition)  
        }  
        int item = buffer[next_out];  
        next_out = (next_out + 1) % buffer.length;  
        filled--;  
        notFull.signal(); // signal thread waiting on the full condition  
    }  
    finally{  
        lock.unlock(); // Whenever you lock, you must unlock  
    }  
    return item;  
}
```



uOttawa

L'Université canadienne
Canada's university

“LOCK” VS “SYNCHRONIZED”

Les moniteurs implémentés avec des classes **Lock** et **Condition** ont quelques avantages comparés avec l'implémentation basée sur le verrou *intrinsèque*:

1. La capacité d'avoir plus qu'une variable de condition par moniteur (voir exemple précédent)
2. La capacité d'avoir un verrou juste (souvenez-vous que les blocs de code ou méthodes synchronisés ne garantissent pas la justesse)
3. La capacité de vérifier si le verrou est actuellement occupé(en faisant appel à la méthode **isLocked()**)
 - Alternativement, on peut faire appel à **tryLock()** qui acquiert le verrou seulement s'il n'est pas occupé par un autre thread
4. La capacité d'obtenir la liste de threads qui attendent le verrou (en faisant appel à la méthode **getQueuedThreads()**)

La liste ci-dessus n'est pas exhaustive...
SEG2506



uOttawa

L'Université canadienne
Canada's university

“VERROU” VS “SYNCHRONISÉ”

Les désavantages de Lock et Condition :

1. Besoin d'ajouter le code d'acquisition et relâchement de lock
2. Besoin d'ajouter un bloc try-finally



uOttawa

L'Université canadienne
Canada's university

SÉMAPHORES JAVA

Java définit une classe de sémaphores:

`java.util.concurrent.Semaphore`

Créer un sémafore comptant:

```
Semaphore available = new Semaphore(100);
```

Créer un sémafore binaire:

```
Semaphore available = new Semaphore(1);
```

On implémentera notre propre classe de sémaphores plus tard



uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE SÉMAPHORE

```
public class Example {  
  
    private int counter= 0;  
  
  
    private final Semaphore mutex = new Semaphore(1)  
  
  
    public int getNextCount() throws InterruptedException {  
        mutex.acquire();  
  
        try {  
            return ++counter;  
        } finally {  
            mutex.release();  
        }  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

EXERCICE DE SÉMAPHORE

Malgré qu'une classe de sémaphore soit incluse dans la librairie standard de Java, cependant, avec les connaissances que vous avez accumulé jusqu'à présent,

Pouvez-vous créer une classe de SémaPhores Comptants en utilisant des Verrous intrinsèques?



uOttawa

L'Université canadienne
Canada's university

EXERCICE DE SÉMAPHORE

```
public class Semaphore{  
  
    private int count;  
  
    public Semaphore () {  
  
        count = 0;  
    }  
  
    public synchronized void acquire() {  
  
        while (count <=0) {  
  
            try {  
  
                wait();  
            }  
  
            catch (InterruptedException e) {}  
        }  
  
        count--;  
    }  
}
```

```
    public synchronized void release() {  
  
        ++count;  
  
        notify();  
    }  
}
```



uOttawa

L'Université canadienne
Canada's university

VARIABLES ATOMIQUES

Si on a besoin de la synchronisation pour une seule variable dans notre classe, on peut utiliser une classe atomique pour la rendre thread safe:

- AtomicInteger
- AtomicLong
- AtomicBoolean
- AtomicReference

Ces classes utilisent des mécanismes de “hardware” à bas niveau pour assurer la synchronisation

- Ceci résulte en une meilleure performance



uOttawa

L'Université canadienne
Canada's university

EXEMPLES DE VARIABLES ATOMIQUES

```
public class AtomicCounter {  
  
    private final AtomicInteger value = new AtomicInteger(0);  
  
    public int getValue(){  
  
        return value.get();  
    }  
  
    public int getNextValue(){  
  
        return value.incrementAndGet();  
    }  
  
    public int getPreviousValue(){  
  
        return value.decrementAndGet();  
    }  
}
```

Autres opérations possibles:

`getAndIncrement()` , `getAndAdd(int x)` , `addAndGet(int x)` ...

MERCI!

QUESTIONS?

SÉANCE 17

**COMMUNICATION
INTERPROCESSUS ET
ORDONNANCEMENT DES
PROCESSUS**



uOttawa

L'Université canadienne
Canada's university

SUJETS

Unicast et Multicast

Passage de Message (Synchrone et Asynchrone)

Ordonnancement de CPU

- Ordonnancement « First Come First Serve » (FCFS) (premier arrivé premier servi)
- Ordonnancement Round Robin (RR)

CIP – UNICAST ET MULTICAST

L'informatique distribuée implique deux ou plusieurs processus engagés dans la CIP

- Elle utilise un protocole prédéfini

Un processus peut agir comme expéditeur (sender) à un moment et récepteur (receiver) à un autre

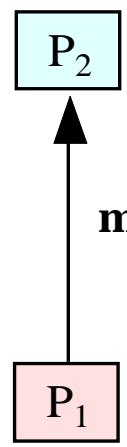
Unicast: communication entre deux processus

- Ex., communication de socket

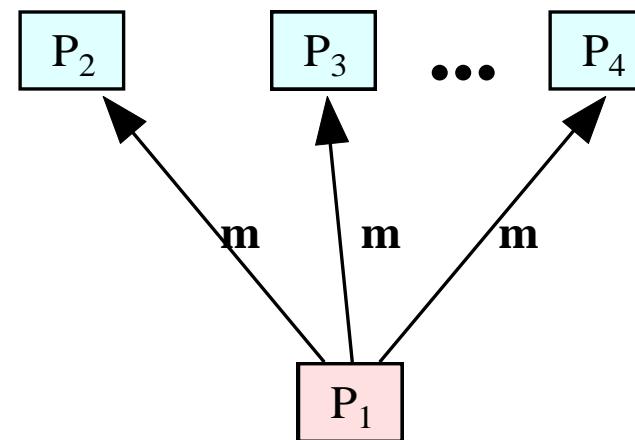
Multicast: communication entre un processus et un groupe de processus

- Ex., modèle de communication publisher/subscriber

UNICAST VS. MULTICAST



unicast



multicast



uOttawa

L'Université canadienne
Canada's university

PASSAGE DE MESSAGE

Un processus envoie un message à un autre et continue son exécution interne

- Le message peut prendre du temps afin d'atteindre l'autre processus

Le message peut être stocké dans la file de données du processus de destination

- Si ce dernier n'est pas immédiatement prêt à recevoir le message

•Deux types de passage de message

- Passage de message asynchrone
- Passage de message synchrone

PASSAGE DE MESSAGE ASYNCHRONE

Les opérations d'envoi et de réception bloquantes:

- Un récepteur est bloqué s'il atteint un point où il est capable de recevoir des messages mais qu'il n'y a pas de messages en attente
- Un expéditeur est bloqué s'il n'y a pas d'espace dans la file de données entre l'expéditeur et le récepteur
 - Cependant, dans plusieurs cas, on s'attend à des files arbitrairement longues, ce qui veut dire que l'expéditeur ne sera **presque** jamais bloqué

PASSAGE DE MESSAGE ASYNCHRONE

Les opérations d'envoi et de réception non-bloquantes:

- Les opérations d'envoi et de réception retournent immédiatement
 - Elles retournent une valeur de statut qui peut indiquer qu'aucun message est arrivé au récepteur
- Le récepteur peut tester si un message est en attente et il peut possiblement effectuer d'autres traitements
 - Il peut optionnellement être notifié par le système lorsqu'un message est reçu

PASSAGE DE MESSAGE SYNCHRONE

On suppose que l'envoi et réception se passent en même temps

- Souvent, on n'a pas besoin d'un tampon intermédiaire

Ceci est aussi appelé un rendez-vous et implique une synchronisation plus proche:

- Les opérations d'envoi et de réception peuvent seulement se passer si les deux partis sont prêts

L'expéditeur doit attendre le récepteur ou le récepteur doit attendre l'expéditeur

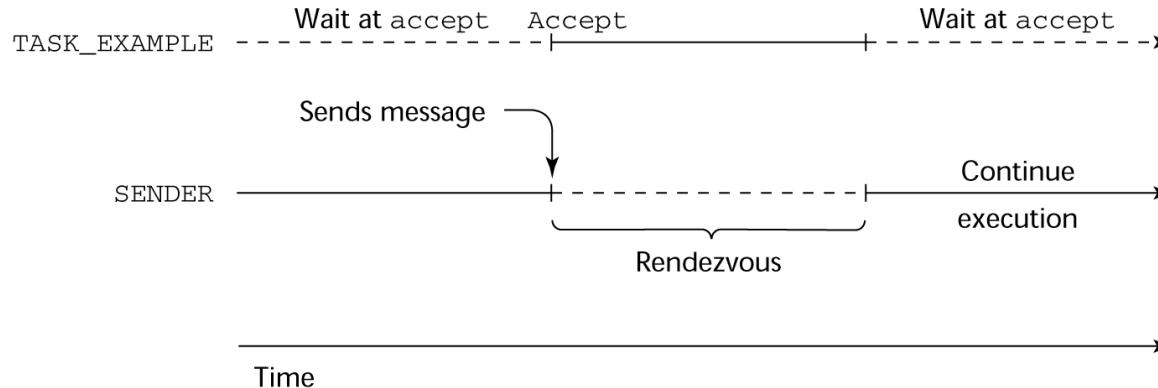


uOttawa

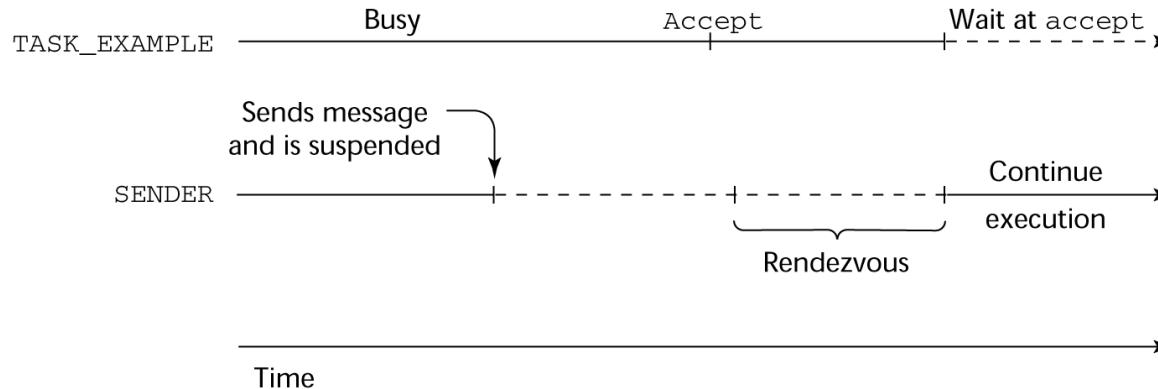
L'Université canadienne
Canada's university

RENDEZ-VOUS

(PAS AUSSI ROMANTIQUE QU'IL PARAIT!)



(a) TASK_EXAMPLE waits for SENDER



(b) SENDER waits for TASK_EXAMPLE



uOttawa

L'Université canadienne
Canada's university

ASSUMPTIONS CONCERNANT L'ORDONNANCEMENT

Comment le système d'exploitation décide-t-il, quel processus, parmi plusieurs, de retirer de la file d'attente des processus prêt (ready queue)?

Ordonnancement: décider quels processus ont accès à des ressources d'un moment à un autre

ASSOMPTION: BURSTS DU CPU

Modèle d'exécution: les processus alternent entre les bursts de CPU et les opérations E / S

- Le processus utilise généralement la CPU pendant une certaine période de temps, puis il fait des opérations E / S, puis utilise le CPU de nouveau
- Chaque décision d'ordonnancement concerne quel processus à attribuer au CPU pour l'exécuter lors du prochain burst de CPU
- Avec le découpage temporel, le processus peut être forcé d'abandonner le CPU avant de terminer son burst d'execution

QU'EST-CE QUI EST IMPORTANT POUR UN ALGORITHME D'ORDONNANCEMENT?



QU'EST-CE QUI EST IMPORTANT POUR UN ALGORITHME D'ORDONNANCEMENT?

Minimiser le temps de réponse

- Temps passé pour effectuer une opération (travail)
- Le temps de réponse que l'utilisateur observe
 - Le temps pour refléter la frappe dans un éditeur
 - Le temps pour compiler un programme
 - Tâches en temps réel: doit respecter les délais imposés par l'environnement

QU'EST-CE QUI EST IMPORTANT POUR UN ALGORITHME D'ORDONNANCEMENT?

Maximiser le throughput (débit)

- Travaux par secondes
- Throughput est relié au temps de réponse, mais non identique
 - Minimiser le temps de réponse cause plus de commutation de contexte que si vous maximiser le throughput seulement
- Minimiser le overhead (temps de commutation de contexte) en plus de l'utilisation efficace des ressources (CPU, disque, mémoire, etc.)

QU'EST-CE QUI EST IMPORTANT DANS UN ALGORITHME D'ORDONNANCEMENT?

Justesse

- Partager le CPU entre les processus d'une manière équitable
- Ne pas seulement minimiser le temps de réponse

ALGORITHMES D'ORDONNANCEMENT: FIRST-COME, FIRST-SERVED (FCFS)

“Run until Done:” algorithme FIFO

Au début, ceci voulait dire qu'un processus courre d'une façon non-préentive jusqu'à ce qu'il soit terminé

- Incluant tout processus bloqué dans les opérations I/O

Maintenant, FCFS veut dire qu'un processus garde le CPU jusqu'à ce qu'il complète son burst

Exemple: Trois processus arrivent dans l'ordre P1, P2, P3.

- Temps de burst de P1: 24
- Temps de burst de P2: 3
- Temps de burst de P3: 3

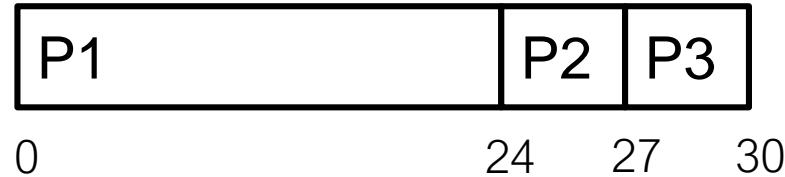
Tracez le diagramme Gantt et calculez la temps moyen d'attente et le temps moyen d'achèvement.

ALGORITHMES D'ORDONNANCEMENT: FIRST-COME, FIRST-SERVED (FCFS)

Exemple: Trois processus arrivent dans l'ordre P1, P2, P3.

- Temps de burst de P1: 24
- Temps de burst de P2: 3
- Temps de burst de P3: 3

Période d'attente:



- P1: 0
- P2: 24
- P3: 27

Temps d'achèvement:

- P1: 24
- P2: 27
- P3: 30

Période moyenne d'attente: $(0+24+27)/3 = 17$

Temps moyen d'achèvement: $(24+27+30)/3 = 27$

ALGORITHMES D'ORDONNANCEMENT: FIRST-COME, FIRST-SERVED (FCFS)

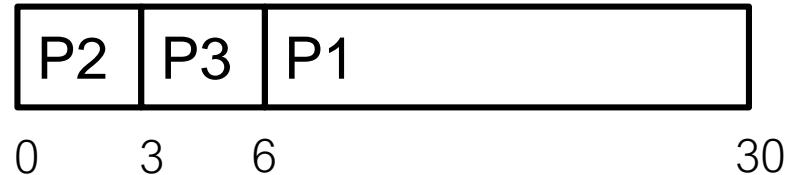
Qu'est ce qui arrive si l'ordre est P2, P3, P1?

- Temps de burst de P1: 24
- Temps de burst de P2: 3
- Temps de burst de P3: 3

ALGORITHMES D'ORDONNANCEMENT: FIRST-COME, FIRST-SERVED (FCFS)

Qu'est ce qui arrive si l'ordre est P2, P3, P1?

- Temps de burst de P1: 24
- Temps de burst de P2: 3
- Temps de burst de P3: 3



Période d'attente:

- P2: 0
- P3: 3
- P1: 6

Temps d'achèvement:

- P2: 3
- P3: 6
- P1: 30

Période moyenne d'attente: $(0+3+6)/3 = 3$ (comparé à 17)

Temps moyen d'achèvement: $(3+6+30)/3 = 13$ (comparé à 27)

ALGORITHMES D'ORDONNANCEMENT: FIRST-COME, FIRST-SERVED (FCFS)

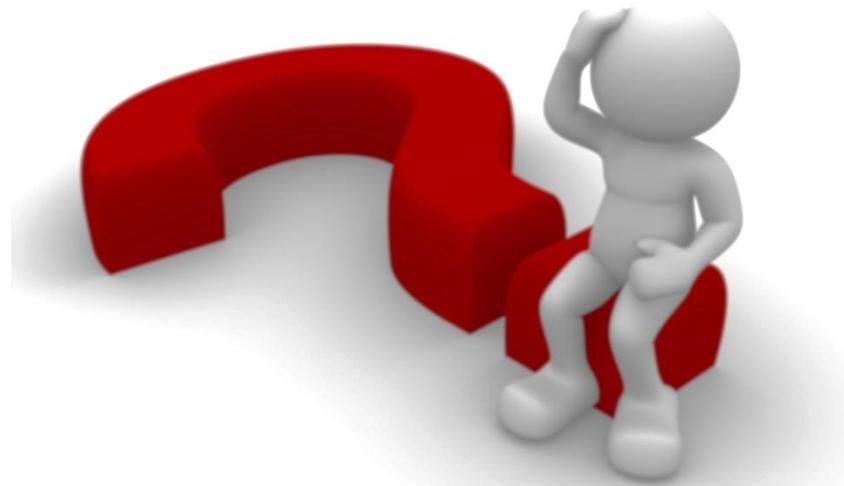
Période moyenne d'attente: $(0+3+6)/3 = 3$ (comparé à 17)

Temps moyen d'achèvement: $(3+6+30)/3 = 13$ (comparé à 27)

Avantages et Désavantages de FCFS:

- Simple (+)
- Les travaux courts se coincent derrière les travaux longs (-)
- La performance est dépendante de l'ordre dans lequel les travaux arrivent (-)

COMMENT PEUT-ON AMÉLIORER CECI?





uOttawa

L'Université canadienne
Canada's university

ORDONNANCEMENT ROUND ROBIN (RR)

Méthode Round Robin

- Chaque processus obtient une petite unité de temps de CPU (quantum de temps)
 - Habituellement 10-100 ms
- Après l'expiration du quantum, le processus est préempté et ajouté à la fin de la file des « processus prêts »
- Soit N processus dans la file des « processus prêts » et le quantum de temps est Q ms:
 - Chaque processus reçoit $1/N$ du temps de CPU
 - Quel est le temps d'attente maximum pour chaque processus ?
 - ***Pas d'attentes de processus plus que $(N-1)Q$ unités de temps***

ORDONNANCEMENT ROUND ROBIN (RR)

La performance dépend de la valeur de Q

- Petite valeur de Q → beaucoup de overhead
- Grande valeur de Q est comme **FCFS**
- Q doit être grand comparé **au temps de commutation de contexte**, sinon le overhead est trop haut
 - Vous dépensez la plupart de votre temps en faisant la commutation de contexte!



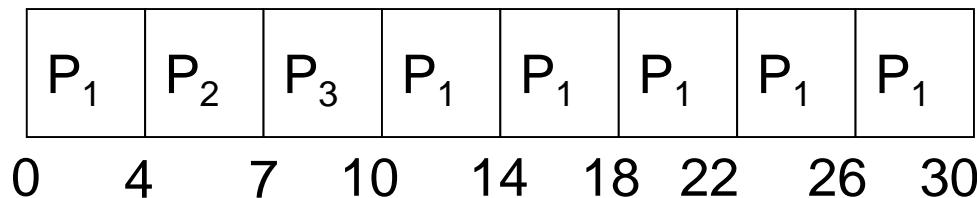
uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 4

<u>Processus</u>	<u>Temps de Burst</u>
P_1	24
P_2	3
P_3	3

Le diagramme Gantt est:





uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 4

Processus

P_1

P_2

P_3

Temps de Burst

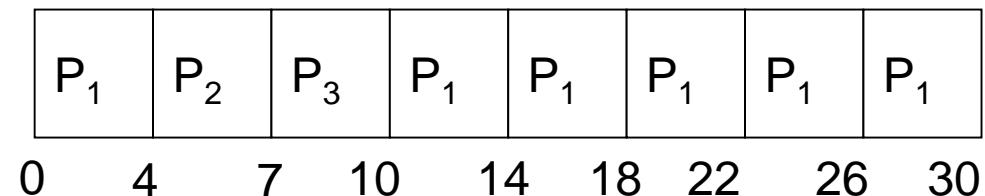
24

3

3

Période d'attente:

- P1: $(10-4) = 6$
- P2: $(4-0) = 4$
- P3: $(7-0) = 7$



Temps d'achèvement:

- P1: 30
- P2: 7
- P3: 10

Période moyenne d'attente: $(6 + 4 + 7)/3 = 5.67$

Temps moyen d'achèvement: $(30+7+10)/3=15.67$

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

<u>Processus</u>	<u>Temps de Burst</u>
P_1	53
P_2	8
P_3	68
P_4	24

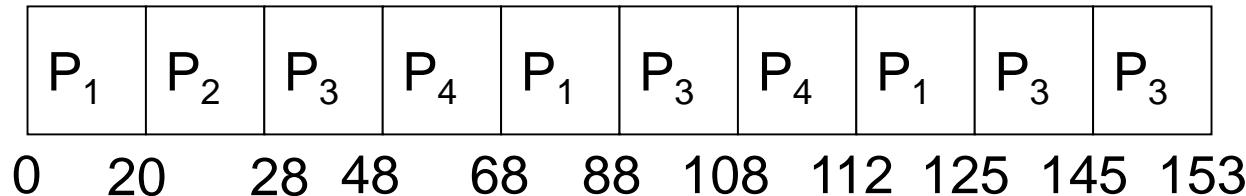


uOttawa

L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

<u>Process</u>	<u>Burst Time</u>
P_1	53 33 13 0
P_2	8 0
P_3	68 48 28 8 0
P_4	24 4 0





uOttawa

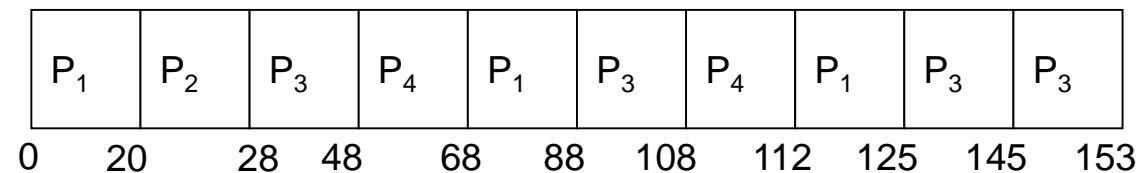
L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

Temps d'achèvement:

- P1: 125
- P2: 28
- P3: 153
- P4: 112

Process	Burst Time
P1	53
P2	8
P3	68
P4	24



Temps moyen d'achèvement: $(125+28+153+112)/4 = 104.5$



uOttawa

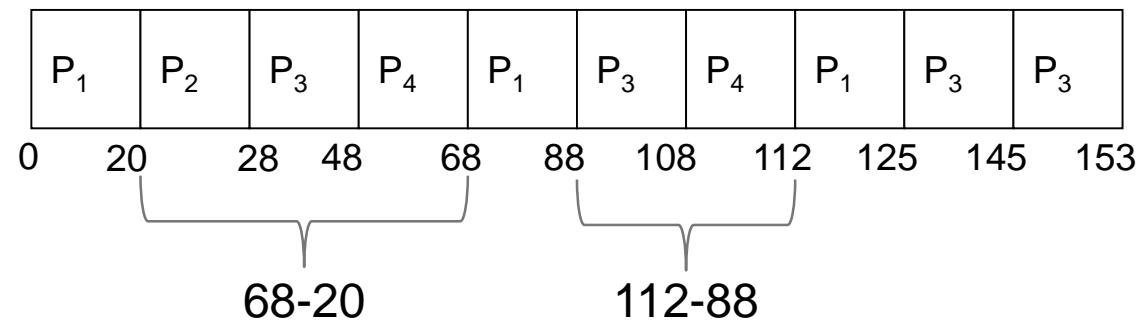
L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

Période d'attente:

- For P1:
 - $(68-20)+(112-88) = 72$

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	8
P_3	68
P_4	24





uOttawa

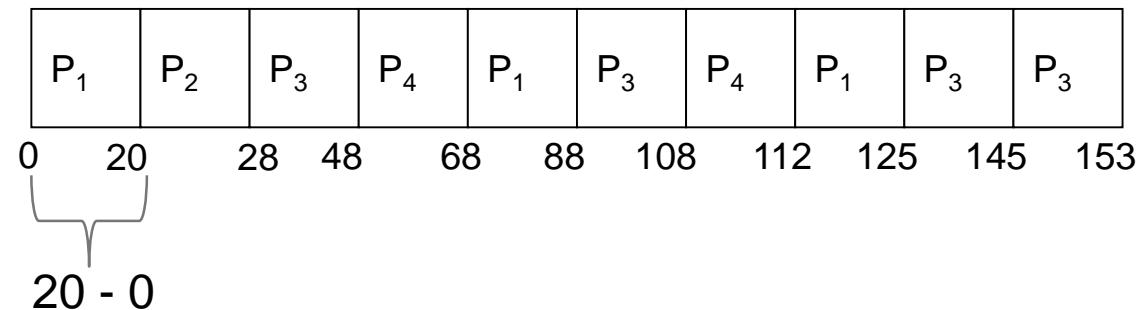
L'Université canadienne
Canada's university

EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

Période d'attente:

- For P1:
 - $(68-20)+(112-88) = 72$
- For P2:
 - $20-0 = 20$

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	8
P_3	68
P_4	24



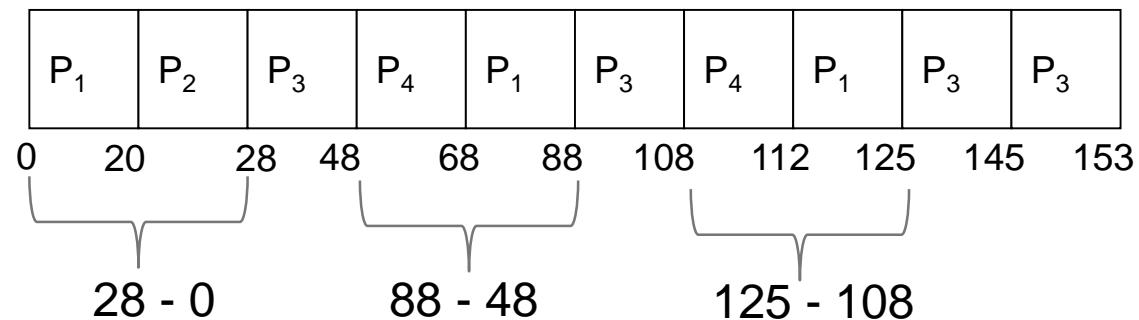


EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

Période d'attente:

- For P1:
 - $(68-20)+(112-88) = 72$
- For P2:
 - $20-0 = 20$
- For P3
 - $(28-0)+(88-48)+(125-108)= 85$

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	8
P_3	68
P_4	24



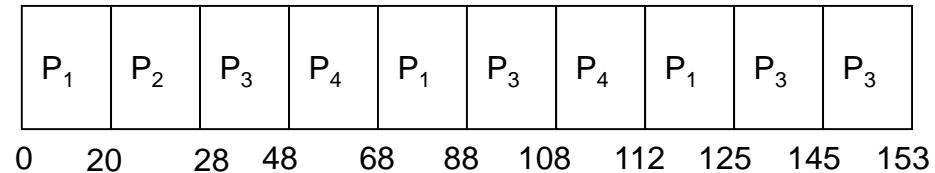


EXEMPLE DE RR AVEC QUANTUM DE TEMPS = 20

Période d'attente:

- For P1:
 - $(68-20)+(112-88) = 72$
- For P2:
 - $20-0 = 20$
- For P3
 - $(28-0)+(88-48)+(125-108)= 85$
- For P4
 - $(48-0)+(108-68) = 88$

Process	Burst Time
P1	53
P2	8
P3	68
P4	24



Période moyenne d'attente : $(72+20+85+88)/4 = 66.25$



uOttawa

L'Université canadienne
Canada's university

SOMMAIRE DE RR

Avantages et Désavantages:

- Meilleur pour les travaux courts (+)
- Juste (+)
- Le temps de commutation de contexte augmente pour les travaux longs (-)

Si le quantum choisi est

- Trop grand, le temps de réponse souffre
- Infini, la performance est la même que FCFS
- Trop petit, le throughput souffre et le pourcentage de overhead grandit

MERCI!

QUESTIONS?