Penser, modéliser, et maitriser le calcul informatique.
Cours de Gérard Berry au Collège de France du 6 Janvier 2010.

Le join-calcul, un calcul pour la programmation répartie

Cédric Fournet
Microsoft Research, Cambridge
MSR-INRIA, Orsay
http://research.microsoft.com/~fournet

avec
Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, Didier Rémy, Fabrice le Fessant, Martin Abadi

Programmer des systèmes répartis

- Un système réparti comporte plusieurs machines interconnectées. Chaque machine exécute un fragment d'un même programme, de manière concertée.

- Ces fragments de programme communiquent des valeurs, du code exécutable, voire une partie du programme en cours d'exécution.

- La localisation est importante!

Programmer des systèmes répartis

- Caractère asynchrone
  - Il n'y a pas d'opérations globales
  - Chaque opération prend un temps imprévisible
  - Les débits augmentent, la latence reste :
    Registres << caches L1 L2 L3… << mémoire << disque, réseau …

- Parallélisme, entre machines et sur chaque machine.

- Environnement hétérogène incertain
  - La structure du réseau évolue
  - Certaines machines peuvent tomber en panne
  - Certaines machines peuvent tricher.

Calculs de processus (CSP, CCS, pi-calcul)

- Avantages
  - Simplicité formelle
  - Beaux outils (vérification, types, équivalences)
  - Succès en modélisation
- Ces calculs ne sont pas adaptés à la programmation répartie
  leurs mécanismes sont trop abstraits ;
  leur implantation est difficile et inefficace
- Le join-calcul: un petit calcul pour la programmation asynchrone

  Chaque étape du calcul s’implante par (au plus)
  un message asynchrone d’une machine a une autre

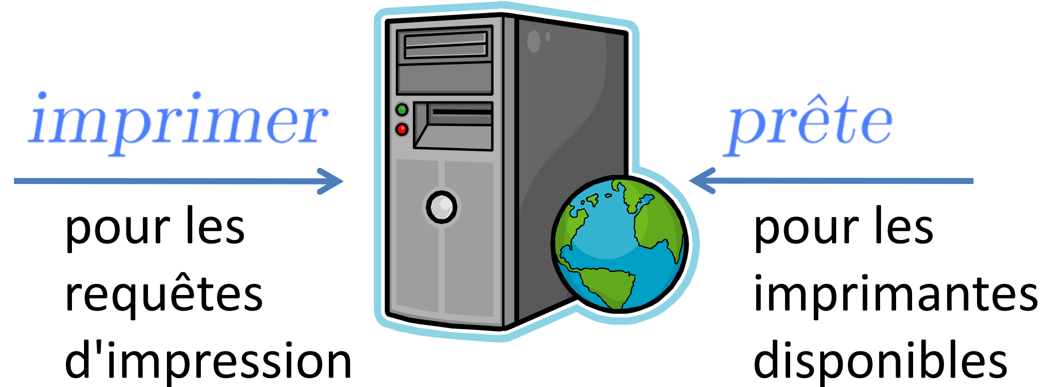
Un calcul nominal (comme le pi-calcul)

- Les seules valeurs sont des noms x, y, $imprimer$, $\ldots$
- Un message $x\langle y_1, \ldots, y_n \rangle$ utilise les noms de deux manières
  - x est l'adresse du message
  - $y_1, \ldots, y_n$ est son contenu

- Un processus peut
  - créer des noms
  - envoyer des messages avec les noms qu'il connait
  - recevoir des messages sur les noms qu'il a créés.

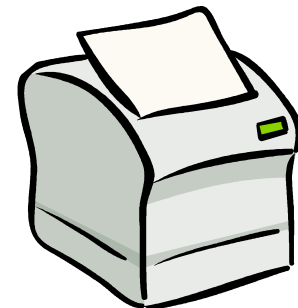
Exemple : un serveur d'impression

- Nous modélisons l'interface du serveur d'impression par deux noms:

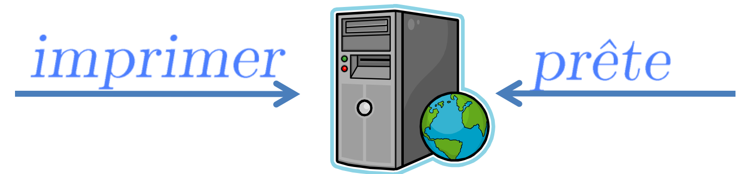


- Plusieurs messages peuvent être mis en parallèle; ils représentent l'état du système

$$imprimer\langle 1\rangle \mid imprimer\langle 2\rangle \mid prête\langle laser\rangle$$



La synchronisation ("join")



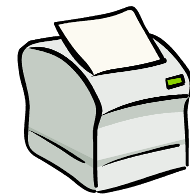
- Pour chaque nom, le traitement des messages est fixé par des règles de réaction, de la forme $J \triangleright P$
  - J est un filtre, indiquant quels messages consommer
  - P est un processus qui remplace alors ces messages

- Pour le serveur d'impression, nous utilisons la règle

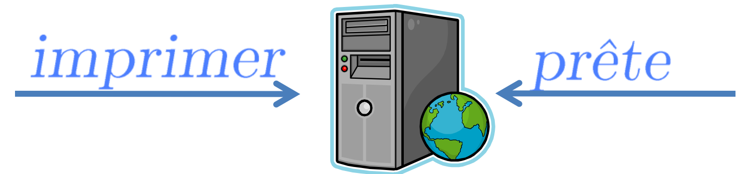
$$D \stackrel{\text{def}}{=} \mathit{imprimer}\langle \mathit{fichier}\rangle \mid \mathit{prête}\langle \mathit{imprimante}\rangle \triangleright \mathit{imprimante}\langle \mathit{fichier}\rangle$$

- Chaque processus peut définir des règles locales:

$$P \stackrel{\text{def}}{=} \text{def } D \text{ in } \mathit{imprimer}\langle 1\rangle \mid \mathit{imprimer}\langle 2\rangle \mid \mathit{prête}\langle \mathit{laser}\rangle$$



La synchronisation ("join")



- Pour chaque nom, le traitement des messages est fixé par des règles de réaction, de la forme $J \triangleright P$
  - J est un filtre, indiquant quels messages consommer
  - P est un processus qui remplace alors ces messages

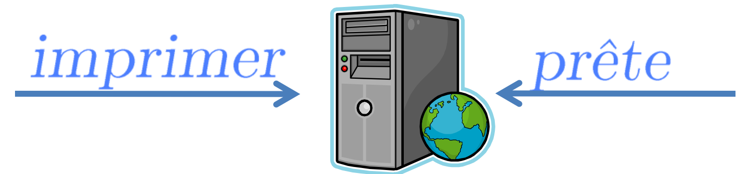
- Pour le serveur d'impression, nous utilisons la règle

$$D \stackrel{\text{def}}{=} \mathit{imprimer}\langle \mathit{fichier} \rangle \mid \mathit{prête}\langle \mathit{imprimante} \rangle \triangleright \mathit{imprimante}\langle \mathit{fichier} \rangle$$

- Chaque processus peut définir des règles locales:

$$P \stackrel{\text{def}}{=} \text{def } D \text{ in } \mathit{imprimer}\langle 1 \rangle \mid \mathit{imprimer}\langle 2 \rangle \mid \mathit{prête}\langle \mathit{laser} \rangle$$

La synchronisation ("join")



- Pour chaque nom, le traitement des messages est fixé par des règles de réaction, de la forme $J \triangleright P$
  - J est un filtre, indiquant quels messages consommer
  - P est un processus qui remplace alors ces messages

- Pour le serveur d'impression, nous utilisons la règle

$$D \stackrel{\text{def}}{=} \mathit{imprimer}\langle \mathit{fichier} \rangle \mid \mathit{prête}\langle \mathit{imprimante} \rangle \triangleright \mathit{imprimante}\langle \mathit{fichier} \rangle$$

- Chaque processus peut définir des règles locales:

$$P \stackrel{\text{def}}{=} \text{def } D \text{ in } \mathit{imprimer}\langle 1 \rangle \mid \mathit{imprimer}\langle 2 \rangle \mid \mathit{prête}\langle \mathit{laser} \rangle$$

$$\rightarrow \text{def } D \text{ in } \mathit{imprimer}\langle 1 \rangle \mid \mathit{laser}\langle 2 \rangle$$

Syntaxe du join-calcul

$$
\begin{array}{lllll}
P & ::= & & & \textbf{processus} \\
& & x\langle v_1, \ldots, v_n \rangle & & \text{message asynchrone} \\
& | & \mathsf{def}\ D\ \mathsf{in}\ P & & \text{définition locale} \\
& | & P \mid P' & & \text{exécution parallèle} \\
D & ::= & & & \textbf{définitions} \\
& & J \triangleright P & & \text{règle de réaction} \\
& | & D \wedge D' & & \text{composition de définitions} \\
J & ::= & & & \textbf{filtres} \\
& & x\langle y_1, \ldots, y_n \rangle & & \text{message à recevoir} \\
& | & J \mid J' & & \text{synchronisation entre messages}
\end{array}
$$

Un peu de plomberie

- On peut agréger, filtrer, dupliquer des messages

$$\text{def } x\langle y\rangle \triangleright x_1\langle y\rangle \mid x_2\langle y\rangle \text{ in } P$$

$$\text{def } x_1\langle y\rangle \triangleright x\langle y\rangle \wedge x_2\langle y\rangle \triangleright x\langle y\rangle \text{ in } P$$

- La présence d'un relais ne change rien:

$$P \quad \approx \quad \text{def } x'\langle y\rangle \triangleright x\langle y\rangle \text{ in } P\{x'/x\}$$

C'est une loi "asynchrone" (fausse en pi-calcul)

UNE MACHINE RÉPARTIE?

La machine chimique (rappel)

- Calculer, c’est réécrire des multi-ensembles (Banâtre & Le Metayer)
  - L’état du programme est un multi-ensemble de molécules
  - Des réactions chimiques transforment ces molécules
  - Ces transformations sont locales, indépendantes du reste de la solution

- Berry et Boudol reformulent la sémantique des processus par deux groupes de règles:
  - L'équivalence structurelle (notée $\rightleftharpoons$) décrit comment réarranger les processus. Ces étapes sont réversibles
  - La réduction chimique (notée $\rightarrow$) décrit l’interaction locale des processus.

Une machine chimique réflexive

Pour garantir une implantation répartie du join-calcul:

1. On ne fait pas d'hypothèses sur la répartition des processus:
   Il y a de nombreuses règles de réaction simples
   et indépendantes (parallélisme)
2. Toutes les règles qui peuvent consommer un message donné se trouvent sur un même site (localité)
3. Au cours du calcul, de nouvelles molécules apparaissent avec leurs règles de réactions (réflexivité)

Une machine chimique réflexive

- Notée $\mathcal{D} \vdash \mathcal{P}$, une solution représente l'état du calcul par deux multi-ensembles
  - Les processus en cours d'exécution $\mathcal{P} = P_1 \mid P_2 \mid \ldots$
  - Les règles de réaction actives $\mathcal{D} = D_1 \wedge D_2 \wedge \ldots$
- La solution évolue localement comme suit:

Activation d'une définition $\qquad \vdash \text{def } D \text{ in } P \quad \rightleftharpoons \quad D\sigma \vdash P\sigma$

Réception de messages $\qquad J \triangleright P \vdash J\sigma \quad \rightarrow \quad J \triangleright P \vdash P\sigma$

- σ remplace les noms définis par D par des noms frais (si nécessaire)
- σ remplace les noms reçus par J par ceux envoyés dans les messages

Vers une implantation répartie

- On partitionne la solution chimique en plusieurs sites

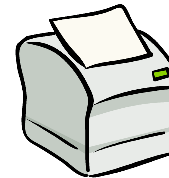
$$\mathcal{D}_1 \vdash \mathcal{P}_1 \ \parallel \ \mathcal{D}_2 \vdash \mathcal{P}_2 \ \parallel \ \ldots$$

avec l'invariant: chaque nom est défini sur un seul site

- On utilise les mêmes règles qu'avant pour chaque site
- On ajoute une règle de routage globale (déterministe):

$$\begin{array}{ll} & \vdash x\langle y_1, \ldots, y_n\rangle \ \parallel \ J \triangleright P \vdash \\ \rightarrow & \vdash \qquad\qquad\qquad\quad \parallel \ J \triangleright P \vdash x\langle y_1, \ldots, y_n\rangle \end{array}$$

Exemple réparti (trois sites)



$$
\begin{array}{llll}
 & \vdash \mathit{imprimer}\langle 1\rangle & \parallel D \vdash & \parallel \vdash \mathsf{def}\ \mathit{laser}\langle f\rangle \triangleright P\ \mathsf{in}\ \mathit{prête}\langle \mathit{laser}\rangle \\
\rightleftharpoons & \vdash \mathit{imprimer}\langle 1\rangle & \parallel D \vdash & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash \mathit{prête}\langle \mathit{laser}\rangle \\
\rightarrow & \vdash & \parallel D \vdash \mathit{imprimer}\langle 1\rangle & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash \mathit{prête}\langle \mathit{laser}\rangle \\
\rightarrow & \vdash & \parallel D \vdash \mathit{prête}\langle \mathit{laser}\rangle, \mathit{imprimer}\langle 1\rangle & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash \\
\rightarrow & \vdash & \parallel D \vdash \mathit{laser}\langle 1\rangle & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash \\
\rightarrow & \vdash & \parallel D \vdash & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash \mathit{laser}\langle 1\rangle \\
\rightarrow & \vdash & \parallel D \vdash & \parallel \mathit{laser}\langle f\rangle \triangleright P \vdash P[1/f]
\end{array}
$$

avec $D \stackrel{\text{def}}{=} \mathit{imprimer}\langle \mathit{fichier}\rangle \mid \mathit{prête}\langle \mathit{imprimante}\rangle \triangleright \mathit{imprimante}\langle \mathit{fichier}\rangle$

Expressivité :

QUE PEUT-ON PROGRAMMER EN JOIN-CALCUL ?

Programmation impérative

- On peut définir un allocateur de mémoire mutable:

$$cellule\langle v_0, c_0\rangle \triangleright \left(\begin{array}{lrl} \mathsf{def} & lire\langle c\rangle \mid valeur\langle v\rangle \triangleright & c\langle v\rangle \mid valeur\langle v\rangle \\ & \wedge\ écrire\langle u, c\rangle \mid valeur\langle v\rangle \triangleright & c\langle\rangle \mid valeur\langle u\rangle \\ \mathsf{in} & c_0\langle lire, écrire\rangle \mid valeur\langle v_0\rangle & \end{array} \right)$$

- On peut faire de la programmation objet
(avec un nom par méthode pour chaque objet)

Programmation fonctionnelle

- Le contrôle séquentiel reste très utile;
  il s'exprime par des canaux de continuation
  pour renvoyer les résultats intermédiaires.

- Pour certains canaux dits « synchrones »,
  on peut rendre ces continuations implicites

$\mathit{lire}\langle c\rangle \mid \mathit{valeur}(v) \triangleright c\langle z\rangle \mid \mathit{valeur}\langle v\rangle$ devient $\mathrm{lire}() \mid \mathit{valeur}\langle v\rangle \triangleright \text{reply } v \mid \mathit{valeur}\langle v\rangle$

$\text{def } c\langle v\rangle \triangleright \mathit{écrire}\langle v+1, \cdots\rangle \text{ in } \mathit{lire}\langle c\rangle$ devient $\mathrm{écrire}(\mathrm{lire}() + 1); \ldots$

- On retrouve une extension parallèle de ML
  avec ses fonctions, ses références, et même ses types

Du lambda-calcul au join-calcul

- Chaque valeur (une fonction) devient l'adresse (un nom) d'un serveur (un processus) qui répond aux requêtes d'évaluation
- Le protocole dépend de la stratégie d'évaluation.
  En appel par valeur (parallèle):

$$
\begin{aligned}
[\![x]\!]_c &\stackrel{\text{def}}{=} c\langle x \rangle \\
[\![\lambda x.T]\!]_c &\stackrel{\text{def}}{=} \text{def } y\langle x, c' \rangle \triangleright [\![T]\!]_{c'} \text{ in } c\langle y \rangle \\
[\![T\ U]\!]_c &\stackrel{\text{def}}{=} \text{def } c_x\langle x \rangle \mid c_y\langle y \rangle \triangleright x\langle y, c \rangle \text{ in } [\![T]\!]_{c_x} \mid [\![U]\!]_{c_y}
\end{aligned}
$$

Implantations

- JoCaml http://join.inria.fr
  - Compilation efficace (filtrage local)
  - Support pour la programmation répartie
  - Mobilité de noms, de machines

- C#, C-omega http://research.microsoft.com/comega/
  - Intégration dans un langage impératif
  - Programmation asynchrone locale (.NET)

PI-CALCUL & JOIN-CALCUL

Pi-calcul: la communication dans l’éther

- En pi-calcul, la communication a lieu “dans l’éther” (Milner) par rendez-vous entre émetteur et récepteur
  - Un problème pour l’implémentation: Où se donner rendez-vous? Comment router les message?

$$\overline{a}\langle 1\rangle \mid \overline{a}\langle 2\rangle \;\|\; a(x).Q_1 \;\|\; a(x).Q_2$$

$$\equiv 0 \;\|\; \overline{a}\langle 1\rangle \mid \overline{a}\langle 2\rangle \mid a(x).Q_1 \;\|\; a(x).Q_2$$

$$\equiv 0 \;\|\; a(x).Q_1 \;\|\; \overline{a}\langle 1\rangle \mid \overline{a}\langle 2\rangle \mid a(x).Q_2$$

$$0 \;\|\; \overline{a}\langle 1\rangle \mid \overline{a}\langle 2\rangle \mid a(x).Q_1 \;\|\; a(x).Q_2$$

Du pi-calcul (asynchrone) au join-calcul

- Pour coder un canal du pi-calcul en join-calcul, on sépare les capacités "envoyer" et "recevoir"

$$
\begin{aligned}
[\![\nu a.P]\!] &\stackrel{\text{def}}{=} \text{def } \overline{a}\langle \overline{x}, x\rangle \mid a\langle c\rangle \triangleright c\langle \overline{x}, x\rangle \text{ in } [\![P]\!] \\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!] \\
[\![\overline{a}\langle x\rangle]\!] &\stackrel{\text{def}}{=} \overline{a}\langle \overline{x}, x\rangle \\
[\![a(x).P]\!] &\stackrel{\text{def}}{=} \text{def } c\langle \overline{x}, x\rangle \triangleright [\![P]\!] \text{ in } a\langle c\rangle
\end{aligned}
$$

- Il faut choisir quel site définit chaque canal
- Cette traduction est correcte; (il existe aussi
  - une traduction correcte & complète
  - une traduction inverse correcte & complète)

UN PEU DE SÉCURITÉ

Un exemple de programme réparti

$$\text{def} \quad bulletin\langle a, n\rangle \,|\, ouvert\langle\rangle \quad \triangleright \; gagnant\langle n\rangle \,|\, a\langle \texttt{prix}\rangle$$
$$\wedge \quad bulletin\langle a, n\rangle \,|\, gagnant\langle m\rangle \; \triangleright \; gagnant\langle m\rangle \,|\, a\langle m + \texttt{a gagné}\rangle \text{ in}$$

$$ouvert\langle\rangle \,|\, participant_1\langle bulletin\rangle \,|\, \ldots \,|\, participant_k\langle bulletin\rangle$$

L'organisateur du concours crée un canal *bulletin* et le diffuse. Le premier participant qui envoie son nom gagne le concours.

Un exemple de programme réparti

$$
\begin{array}{lll}
\text{def} & bulletin\langle a,n\rangle \mid ouvert\langle\rangle & \triangleright\ gagnant\langle n\rangle \mid a\langle \texttt{prix}\rangle \\
\wedge & bulletin\langle a,n\rangle \mid gagnant\langle m\rangle & \triangleright\ gagnant\langle m\rangle \mid a\langle m + \texttt{a gagné}\rangle\ \text{in}
\end{array}
$$

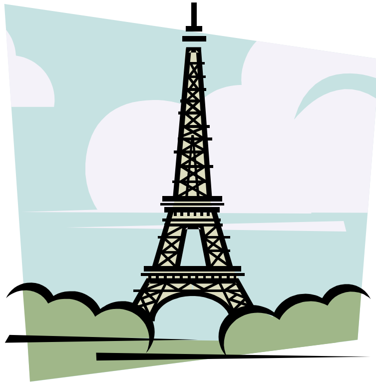
$$
ouvert\langle\rangle \mid participant_1\langle bulletin\rangle \mid \ldots \mid participant_k\langle bulletin\rangle
$$

De nombreuses propriétés de sécurité s'expriment
dans le join-calcul par des équivalences (faciles à vérifier).

Contrôle d'accès : Chaque participant a reçu le nom *bulletin*.

Intégrité : Tous les participants reçoivent le nom du gagnant.

Anonymité: Le nom des perdants reste secret.



Peut-on croire la sémantique du calcul?

Un joueur à Paris

$bulletin\langle x, 42\rangle$

la Manche

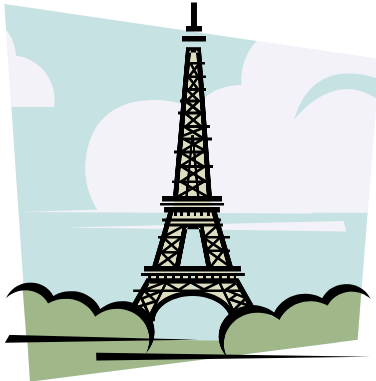
un serveur à Cambridge

$bulletin\langle a, n\rangle \triangleright \ldots$

Un joueur qui triche peut observer/intercepter le bulletin:

Les messages sur des canaux privés sont ici trop abstraits

Peut-on croire la sémantique du calcul?



Un joueur à Paris

$emit\langle\{a^+, 42\}_{bulletin^+}\rangle$

la Manche

un serveur à Cambridge

$recv\langle m\rangle \triangleright$
decrypt m using $bulletin^-$
to a^+, n in $\ldots$

Un joueur qui triche peut observer/intercepter le bulletin:

Les messages sur des canaux privés sont ici trop abstraits

- On peut les remplacer par des messages cryptographiques sur des canaux publics (en passant des clés à la place des canaux)
- On peut montrer que ce genre d'implantation est sure (pour tout processus, contre tout adversaire actif)

Le join-calcul (résumé)

- La programmation répartie est délicate
  - La localité est importante:
    ailleurs il y a des pannes, des adversaires
- On peut programmer avec des messages asynchrones
  - Mobilité des noms (pour les canaux, les clés)
  - Séparation entre routage (global, déterministe)
    et synchronisation (local, statiquement défini)
  - Codages pour la programmation fonctionnelle, impérative
- Implantations
- Formalisation
- Extensions pour la sécurité