

Wiki aime la paix 2026

Partagez les histoires de
paix du monde.
Téléversez vos photos
dès maintenant !



Aidez-nous à traduire !

Pi-calcul

10 langues

[Article](#) [Discussion](#)

[Lire](#) [Modifier](#) [Modifier le code](#) [Voir l'historique](#)

Le **Pi-calcul** (ou ***π-calcul***) est un [langage de programmation](#) théorique inventé par [Robin Milner](#). Ce langage occupe dans le domaine de l'informatique parallèle et distribuée un rôle similaire à celui du [λ-calcul](#) dans l'informatique classique.

La problématique [[modifier](#) | [modifier le code](#)]

En tant que [langage de programmation](#) théorique (ou [langage formel](#)), le π -calcul ne vise pas à permettre de construire des programmes exécutables. L'objectif du π -calcul est de permettre d'étudier les concepts importants en programmation parallèle et distribuée, afin :

- d'isoler les mécanismes essentiels
- de reformuler les mécanismes redondants sous la forme de macros construites à partir des mécanismes essentiels
- de définir des notions de haut niveau telles que
 - comportement d'un programme (utile pour fournir des garanties sur l'absence de certaines erreurs)
 - égalité de deux programmes (utile notamment pour écrire des [compilateurs](#) ou des [ramasse-miettes](#))
 - [politique de sécurité](#)
 - confidentialité d'une information
 - …

Afin de construire un programme exécutable, il est nécessaire de passer par un [langage de programmation](#) concret dérivé (ou inspiré) du π -calcul, tel que [JoCaml](#), [Acute](#) ou [Nomadic Pict](#).

Définition informelle [[modifier](#) | [modifier le code](#)]

Les entités du π-calcul [[modifier](#) | [modifier le code](#)]

En pratique, un programme écrit en π -calcul (ou *terme*) décrit systématiquement un processus, généralement noté *P*, *Q*, *R*… Les processus représentent aussi bien les [processus légers](#) en [anglais](#) : *threads* ou [processus lourds](#) et aucune hypothèse n'est faite sur l'emplacement du processus : à ce niveau d'abstraction, celui-ci peut être exécuté sur n'importe quel composant ou n'importe quel ordinateur. Un langage de programmation concret ajoutera un compilateur ou un environnement d'exécution intelligent ou nécessitera plus de précisions pour savoir où placer le processus.

Plutôt que des données explicites, les programmes écrits en π -calcul manipulent des **noms**, le plus souvent notés *a*, *b*, *c*… *x*, *y*, *z*. Un nom est une construction abstraite qui peut, à ce niveau, représenter n'importe quel type d'information. Les noms sont utilisés comme des canaux : tout processus qui connaît le nom *a* peut écrire sur *a* ou lire depuis *a*. Les canaux diffèrent des variables et des constantes au sens où chaque lecture doit correspondre à une seule et unique écriture -- le processus qui écrit sur *a* envoie donc une information au processus qui lit sur *a*. Les informations, de nouveau, sont des noms. Un langage de programmation concret ajoutera des types de base (entiers, booléens) et des structures de données traditionnelles.

En particulier, un système est décrit uniquement sous la forme d'un terme : en π -calcul, il n'y a pas de différence, syntaxique ou sémantique, entre le code et les données.

Les constructions du π-calcul [[modifier](#) | [modifier le code](#)]

Note : il existe plusieurs variantes canoniques du π -calcul. Cet article présente le *π-calcul polyadique asynchrone à réplication gardée avec comparaison de noms*.

Processus terminé [[modifier](#) | [modifier le code](#)]

Le processus le plus simple est le processus *nil*, noté *0*. Ce terme ne fait rien – il s'agit d'un processus qui a terminé de s'exécuter.

Émission [[modifier](#) | [modifier le code](#)]

Un terme plus compliqué est l'émission (asynchrone) d'un message : le processus ***a*⟨*b*⟩** émet sur le canal *a* l'information *b*. L'émission est asynchrone au sens où le processus n'attend pas de réponse et continue son exécution même si aucun processus n'a reçu l'information. Comme il est très facile d'écrire une macro pour la communication synchrone à partir de cette primitive de communication asynchrone, il ne s'agit pas d'une limitation.

Réception [[modifier](#) | [modifier le code](#)]

La réception d'un message se note ***a*(*x*).*P*** et se comporte de la manière suivante : si un processus de la forme ***a*⟨*b*⟩** est présent, il y a communication entre les processus. Le processus récepteur continue alors sous la forme ***P*{*x* ← *b*}**, c'est-à-dire comme le processus *P* dans lequel le nom *x* représente le message *b*. Ceci est similaire à un appel de fonction dans un langage traditionnel. Les différences principales sont les suivantes :

- s'il n'existe aucun processus récepteur sur le canal *a*, émettre sur le canal *a* ne constitue pas une erreur -- le message est simplement mis en attente jusqu'à ce qu'un récepteur soit prêt à écouter sur le canal *a*
- il peut y avoir plusieurs processus récepteurs prêts à écouter sur le canal *a* à un moment donné -- en cas de communication, l'un d'entre eux est choisi arbitrairement (dans le π -calcul, ceci peut servir à modéliser plusieurs agents prêts à fournir le même service)
- la fonction est « effacée » après avoir été appelée et ne renvoie jamais de valeur -- à ce sens, la communication ressemble plus à un `goto` avec paramètres ou à un `join` avec paramètres qu'à un appel de fonction proprement dit.

Composition parallèle [[modifier](#) | [modifier le code](#)]

Si *P* et *Q* sont deux processus, ***P*|*Q*** est la composition parallèle de *P* et *Q*, qui représente le fait que *P* et *Q* sont exécutés simultanément.

Création d'un nom [[modifier](#) | [modifier le code](#)]

Un processus peut créer un nouveau nom. Comme tous les noms, celui-ci sera a priori un canal de communication. Ainsi, le processus **(*νx*)*P*** (prononcer *new x P*) crée un nouveau canal, qui sera désigné sous le nom *x* dans *P*. Contrairement à l'opération `new x` ou dans de nombreux langages, **(*νx*)** produit *une constante* *x*. De plus, comme l'état entier du système est représenté dans la syntaxe du terme (sans mémoire), **(*νx*)*P*** représente aussi le fait que *x* a été créé à un moment ou à un autre et que *P* est susceptible d'utiliser *x*. Par la suite de l'exécution, si **(*νx*)*P*** est en parallèle avec le processus *Q* (ce que nous noterons **(*νx*)*P*|*Q***) et si *P* communique la valeur de *x* à *Q*, le terme deviendra **(*νx*)(*P*|*Q*)** pour représenter le fait que *P* et *Q* peuvent tous deux utiliser *x*.

Test d'égalité [[modifier](#) | [modifier le code](#)]

Un processus peut tester l'égalité entre deux noms. Ainsi, si *a* et *b* représentent le même nom, **if *a* = *b* then *P* else *Q*** réagira comme *P*, sinon comme *Q*.

Processus dual [[modifier](#) | [modifier le code](#)]

La notation *P* + *Q* dénote un processus qui se comporte tout aussi bien comme *P* que comme *Q*.

Infinité de processus [[modifier](#) | [modifier le code](#)]

Il est possible de faire référence à une infinité de copies d'un processus *P* par la notation *!P*. Cette syntaxe n'a donc pas la signification usuelle *not P* utilisée par la plupart des langages de programmation !

Exemple : un gestionnaire d'impression [[modifier](#) | [modifier le code](#)]

Dans cet exemple, nous construisons progressivement un gestionnaire d'impression (*print spooler*).

Informellement [[modifier](#) | [modifier le code](#)]

Le rôle du gestionnaire d'impression est de recevoir des requêtes d'impression, de localiser une imprimante disponible et de transmettre la requête à l'imprimante. Si aucune imprimante n'est disponible, la requête est mise en attente. Lorsqu'une imprimante a terminé un travail, elle est considérée comme de nouveau en attente.

Plus formellement, une imprimante est représentée par un canal ***printer**₁*, ***printer**₂* ... Effectuer une impression sur une imprimante ***printer**_i* revient à envoyer un message sur ce canal composé

- du document à imprimer
- d'un canal à employer pour prévenir que l'impression est terminée (technique dite de *callback*, souvent employée pour la programmation asynchrone).

Pour cet exemple, la réserve d'imprimantes peut être représentée comme un ensemble de messages émis sur le canal ***spool***, chacun contenant le nom d'un canal ***printer**_i*. Localiser une imprimante disponible revient donc à recevoir un nom d'imprimante sur le canal ***spool***. Remettre l'imprimante dans la réserve revient à réémettre le nom de l'imprimante sur ce même canal. Pour « protéger » la réserve d'imprimante, le nom du canal ***spool*** sera considéré comme local, c'est-à-dire que le gestionnaire sera préfixé par (***νspool***).

Pour cet exemple, faire appel au gestionnaire d'impression consiste à envoyer un message sur le canal ***spooler***, composé

- du document à imprimer
- d'un canal à employer pour prévenir que l'impression est terminée.

La réception sur ***spool*** est répliquée car le service peut être invoqué un nombre arbitraire de fois.

Formellement [[modifier](#) | [modifier le code](#)]

Ainsi, le gestionnaire d'impression peut être défini par

$$\begin{aligned} \mathbf{SPOOLER} = (\nu\mathbf{spool})(\\ & \mathbf{spool}\langle\mathbf{printer}_1\rangle|\mathbf{spool}\langle\mathbf{printer}_2\rangle|\mathbf{spool}\langle\mathbf{printer}_3\rangle|\cdots|\mathbf{spool}\langle\mathbf{printer}_n\rangle \\ & |!\mathbf{spooler}(\mathbf{task}, \mathbf{callback}). \mathbf{spool}(p). (\nu\mathbf{w})(p\langle\mathbf{task}, \mathbf{w}\rangle|\mathbf{w}(). (\mathbf{callback}\langle\rangle|\mathbf{spool}\langle p\rangle)) \\ &) \end{aligned}$$

Dans ce qui précède

- ***task*** est le nom local donné au document à imprimer
- ***callback*** est le nom local donné au canal sur lequel émettre un message pour prévenir que le gestionnaire d'impression a terminé son travail
- ***spool**(p)* attend qu'une imprimante ***p*** soit disponible
- (***νw***) définit un canal sur lequel l'imprimante doit émettre un message pour prévenir le gestionnaire d'impression qu'elle a terminé son travail
- ***p**(task, w)* demande à l'imprimante d'imprimer le document ***task***
- ***w**()* attend que l'imprimante ait émis son message de fin de travail
- ***callback**⟨⟩* permet au gestionnaire d'impression d'émettre son message de fin de travail
- ***spool**⟨p⟩* permet de remplacer ***p*** dans la liste des imprimantes disponibles

Par suite, invoquer le gestionnaire d'impression consistera à placer en présence du terme précédent un terme tel que

$$\mathbf{CLIENT} = (\nu\mathbf{cb})(\mathbf{spooler}\langle\mathbf{document}, \mathbf{cb}\rangle|\mathbf{cb}(). P)$$

Ce terme émet une requête d'impression et attend que la requête soit achevée pour exécuter ***P***.

Exécution du gestionnaire d'impression [[modifier](#) | [modifier le code](#)]

Note : cette section s'apparente à observer la trace d'un programme à l'aide d'un débogueur et est donc, de manière prévisible, peu lisible.

Mis en présence, le processus ***CLIENT***, le processus ***SPOOLER*** et un ensemble d'imprimantes ***PRINTERS*** réagissent de la manière suivante :

$$\begin{aligned} \mathbf{CLIENT}|\mathbf{SPOOLER}|\mathbf{PRINTERS} = (\nu\mathbf{cb})(\mathbf{spooler}\langle\mathbf{document}, \mathbf{cb}\rangle|\mathbf{cb}(). P) \\ |(\nu\mathbf{spool})(\\ & \mathbf{spool}\langle\mathbf{printer}_1\rangle|\mathbf{spool}\langle\mathbf{printer}_2\rangle|\mathbf{spool}\langle\mathbf{printer}_3\rangle|\cdots|\mathbf{spool}\langle\mathbf{printer}_n\rangle \\ & |!\mathbf{spooler}(\mathbf{task}, \mathbf{callback}). \mathbf{spool}(p). (\nu\mathbf{w})(\\ & \quad p\langle\mathbf{task}, \mathbf{w}\rangle|\mathbf{w}(). (\mathbf{callback}\langle\rangle|\mathbf{spool}\langle p\rangle) \\ &) \\ &) \\ & |\mathbf{PRINTERS} \end{aligned}$$

$$\begin{aligned}
CLIENT|SPOOLER|PRINTERS \longrightarrow & (\nu cb)(\\
& cb().P|(\nu spool)(\\
& \quad spool\langle printer_1 \rangle | spool\langle printer_2 \rangle | spool\langle printer_3 \rangle | \dots | spool\langle printer_n \rangle \\
& \quad | !spooler(task, callback).spool(p).(\nu w)(p\langle task, w \rangle | w().(callback\langle \rangle | spool\langle p \rangle)) \\
& \quad | spool(p).(\nu w)(p\langle document, w \rangle | w().(cb\langle \rangle | spool\langle p \rangle)) \\
& \quad) \\
&) \\
& |PRINTERS
\end{aligned}$$

$$\begin{aligned}
CLIENT|SPOOLER|PRINTERS \longrightarrow & (\nu cb)(\\
& cb().P|(\nu spool)(\\
& \quad spool\langle printer_2 \rangle | spool\langle printer_3 \rangle | \dots | spool\langle printer_n \rangle \\
& \quad | !spooler(task, callback).spool(p).(\nu w)(p\langle task, w \rangle | w().(callback\langle \rangle | spool\langle p \rangle)) \\
& \quad | (\nu w)(printer_1\langle document, w \rangle | w().(cb\langle \rangle | spool\langle printer_1 \rangle)) \\
& \quad) \\
&) \\
& |PRINTERS
\end{aligned}$$

$$\begin{aligned}
CLIENT|SPOOLER|PRINTERS \longrightarrow^* & (\nu cb)(\\
& cb().P|(\nu spool)(\\
& \quad spool\langle printer_2 \rangle | spool\langle printer_3 \rangle | \dots | spool\langle printer_n \rangle \\
& \quad | !spooler(task, callback).spool(p).(\nu w)(p\langle task, w \rangle | w().(callback\langle \rangle | spool\langle p \rangle)) \\
& \quad | cb\langle \rangle | spool\langle printer_1 \rangle \\
& \quad) \\
&) \\
& |PRINTERS
\end{aligned}$$

$$\begin{aligned}
CLIENT|SPOOLER|PRINTERS \longrightarrow & P|(\nu spool)(\\
& \quad spool\langle printer_2 \rangle | spool\langle printer_3 \rangle | \dots | spool\langle printer_n \rangle \\
& \quad | !spooler(task, callback).spool(p).(\nu w)(p\langle task, w \rangle | w().(callback\langle \rangle | spool\langle p \rangle)) \\
& \quad | spool\langle printer_1 \rangle \\
& \quad) \\
& |PRINTERS
\end{aligned}$$

$$CLIENT|SPOOLER|PRINTERS \equiv P|SPOOLER|PRINTERS$$

Propriétés du gestionnaire d'impression [\[modifier | modifier le code \]](#)

À l'aide des divers outils du π -calcul, il est possible de caractériser le gestionnaire d'impression par ses propriétés.

Format des messages [\[modifier | modifier le code \]](#)

Tout comme les [fonctions](#) et [variables](#) dans un langage de programmation traditionnel, les canaux du π -calcul peuvent être [typés](#) afin de certifier que les messages qui transitent sur ces canaux ont toujours le même format. Dans le cas du gestionnaire d'impression :

- ***spooler*** permet de communiquer des couples composés d'un document et d'un nom de canal sur lequel aucune information ne transite
- chaque ***printer_i*** permet de communiquer des couples composés d'un document et d'un nom de canal sur lequel aucune information ne transite
- ***spool_i*** permet de communiquer des noms de canaux qui respectent la même politique que ***printer_i***
- ***task*** est un nom de document
- ***callback*** et ***w*** sont des noms de canaux sur lesquels aucune information ne transite

Un système de types simple permet d'associer à chaque nom de canal le type des informations qui peuvent circuler sur ce canal. Une manière de noter cela est

- $\Gamma \vdash \textit{callback} : \textit{Channel}()$ (soit Γ , l'ensemble des informations de typage connues en ce point du programme, spécifie que *callback* est un nom canal sur lequel aucune information ne transite)
- $\Gamma \vdash \textit{spooler} : \textit{Channel}(\textit{Document}, \textit{Channel}())$
- $\Gamma \vdash \textit{printer}_i : \textit{Channel}(\textit{Document}, \textit{Channel}())$
- ...

Ces informations peuvent aisément être vérifiées statiquement (i.e. au cours de la compilation) par un logiciel approprié.

Autorisations de lecture/d'écriture [[modifier](#) | [modifier le code](#)]

Tout comme les [modules](#) et les [classes](#) dans de nombreux langages de programmation introduisent des notions de [méthodes](#) et de [champs](#) publics ou privés, les canaux du π -calcul peuvent être [typés](#) afin de certifier que seuls certains [agents](#) peuvent écrire sur un canal ou que seuls certains agents peuvent lire sur un canal. Ainsi, dans le cadre du gestionnaire d'impression, on souhaitera probablement s'assurer que

- seul le gestionnaire d'impression est autorisé à recevoir des requêtes sur le canal *spool*, pour des raisons de sécurité
- pour chaque canal *printer_i*, seule l'imprimante correspondante est autorisée à recevoir des requêtes sur ce canal
- seuls les gestionnaires d'impression sont autorisés à émettre des requêtes sur les canaux *printer_i*
- le gestionnaire d'impression n'est pas autorisé à écrire sur le document
- ...

Typiquement, afin d'assurer le respect de cette politique de sécurité, un système de types examinera de quelle manière les noms de ces canaux sont distribués (étape assimilable à la liaison dynamique dans les langages traditionnels).

Ainsi, les informations de type du gestionnaire d'impression pourront être complétées en $\Gamma \vdash \textit{spooler} : \textit{Channel}(\textit{Document}[\textit{read}], \textit{Channel}()[\textit{write}])([\textit{read}, \textit{write}])$. En adoptant ce type, le gestionnaire d'impression s'engage à respecter la politique suivante : *spooler* est un canal qui permet d'émettre ou de recevoir. Sur le canal transitent deux informations : un Document, qui est un canal en lecture seule, et un canal, en écriture seule.

Note : cette garantie de sécurité, tout comme le typage public/privé/protégé/... ou [interface/mise en œuvre](#) procède d'une [analyse statique](#). Un composant inconnu et non vérifié peut donc violer la politique de sécurité. Pour assurer que seuls des composants vérifiés sont exécutés sur une machine, il peut être nécessaire de faire appel à des techniques telles que le [Code Porteur de Preuves](#) ou/et des vérifications dynamiques de typage de code reçu de tierces parties.

Gestion des ressources [[modifier](#) | [modifier le code](#)]

Cette section est vide, insuffisamment détaillée ou incomplète. [Votre aide](#) est la bienvenue ! [Comment faire ?](#)

Équivalence avec un autre gestionnaire d'impression [[modifier](#) | [modifier le code](#)]

Cette section est vide, insuffisamment détaillée ou incomplète. [Votre aide](#) est la bienvenue ! [Comment faire ?](#)

Implémentations [[modifier](#) | [modifier le code](#)]

Les langages de programmation suivants sont des implémentations du π -calcul ou de ses variantes :

- [Acute](#)
- [Business Process Modeling Language](#) (BPML)
- [Nomadic Pict](#)
- [jumel](#) [archive] : un compilateur pour un dialecte de ML assorti de primitives du π -calcul pour le passage de messages
- [occam- \$\pi\$](#)
- [Pict](#)
- [JoCaml](#) (basé sur le [Join-Calcul](#))
- [Funnel](#) (une implémentation du join-calcul compatible avec JRE)
- [CubeVM](#) [archive] (une implémentation sans pile)
- Le langage [SpiCO](#) [archive] : un π -calcul stochastique pour les objets concurrents
- [BioSPI](#) [archive] et [SPiM](#) [archive]: simulateurs pour le π -calcul stochastique
- [Ateji PX](#) : une extension de Java avec des primitives de parallélisme inspirées du π -calcul

La dernière modification de cette page a été faite le 26 mars 2023 à 05:21. La page a été rendue avec [Parsoid](#).

Droit d'auteur : les textes sont disponibles sous [licence Creative Commons attribution, partage dans les mêmes conditions](#) ; d'autres conditions peuvent s'appliquer. Voyez les [conditions d'utilisation](#) pour plus de détails, ainsi que les [crédits graphiques](#). En cas de réutilisation des textes de cette page, voyez [comment citer les auteurs et mentionner la licence](#).
Wikipedia® est une marque déposée de la [Wikimedia Foundation, Inc.](#), organisation de bienfaisance régie par le paragraphe [501\(c\)\(3\)](#) du code fiscal des États-Unis.

[Politique de confidentialité](#) [À propos de Wikipédia](#) [Avertissements](#) [Contact](#) [Contacts juridiques & sécurité](#) [Code de conduite](#) [Développement](#) [Statistiques](#)

[Déclaration sur les témoins \(cookies\)](#) [Version mobile](#)

