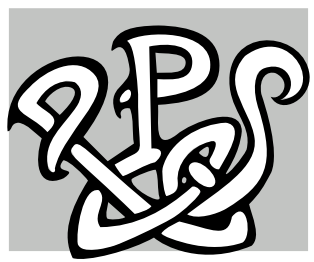
De la webradio lambda à la λ-webdradio

Samuel Mimram

et David Baelde, Romain Beauxis, ...

Groupe de Travail Programmation

20 novembre 2008



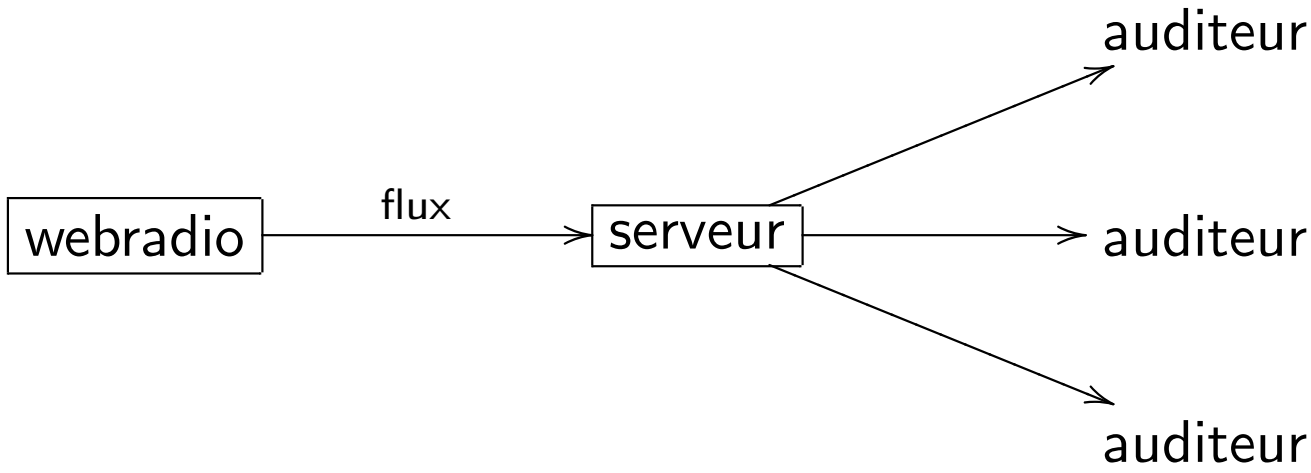
De la radio à la webradio

- La plupart des radios hertziennes diffusent aussi leur contenu sur internet.

De la radio à la webradio

- La plupart des radios hertziennes diffusent aussi leur contenu sur internet.
- De plus en plus de **webradios** naissent, ne diffusant *que* par internet.

Une webradio, c'est simple



(Liquidsoap) *(Icecast)*

Ce que l'on veut

- Divers formats et moyens de stockage des fichiers
  (en particulier émission d'un flux dans plusieurs formats)
- Gestion des listes de lecture
  (critères, horaire, décrochage, jingles, . . .)
- Traitement du son
  (normalisation, enchaînement, détection de blanc, . . .)

Outils préexistants

- **Outils légers**
  (Ices, EZStream, Darkice)
  - en console
  - flux à partir d'une playlist ou d'une carte son

Outils préexistants

- **Outils légers**
  (Ices, EZStream, Darkice)
  - en console
  - flux à partir d'une playlist ou d'une carte son
- **Outils plus complets**
  (Rivendell, Master Control, WinRadio, Open Radio)
  - interface graphique
  - génération de playlists
  - transitions
  - quelques traitements audio

Outils préexistants

- **Outils légers**
  (Ices, EZStream, Darkice)
  - en console
  - flux à partir d'une playlist ou d'une carte son
- **Outils plus complets**
  (Rivendell, Master Control, WinRadio, Open Radio)
  - interface graphique
  - génération de playlists
  - transitions
  - quelques traitements audio
- λ-**outils**
  (Liquidsoap)
  - scriptable
  - extensible

Qui utilise Liquidsoap ?

- Des webradios (Dolebraï, . . .)
- Des hôpitaux et des supermarchés
- Des générateurs de webradios (radionomy)
- D'autres gens (transcodage, number radios, morse, . . .)

Plan

1. **Conception** : on veut un langage
   - pour manipuler des flux
   - le plus générique possible
   - le plus sûr possible
   - abordable pour des *non*-informaticiens

Plan

1. **Conception** : on veut un langage
   - pour manipuler des flux
   - le plus générique possible
   - le plus sûr possible
   - abordable pour des *non*-informaticiens
2. **Implémentation** : il faut que ça soit
   - rapide ($\leq$ temps réel)
   - modulaire

Première partie I

Conception

Un programme typique

1. Décodage à partir d'une ou plusieurs sources
2. Traitement du son
3. Encodage et émission

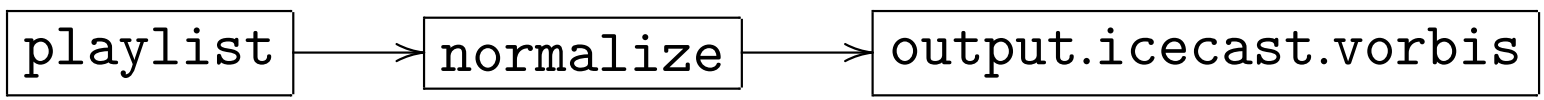
Gestion de flux

Une instance de Liquidsoap produit un ou plusieurs **flux** à partir d'un graphe d'**opérateurs** décrit par un script.

Gestion de flux

Une instance de Liquidsoap produit un ou plusieurs **flux** à partir d'un graphe d'**opérateurs** décrit par un script.

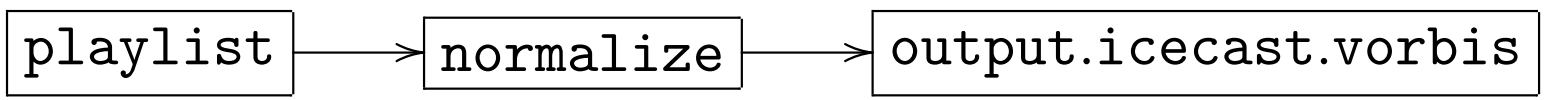
Enchaînement linéaire d'opérateurs



Gestion de flux

Une instance de Liquidsoap produit un ou plusieurs **flux** à partir d'un graphe d'**opérateurs** décrit par un script.

Enchaînement linéaire d'opérateurs



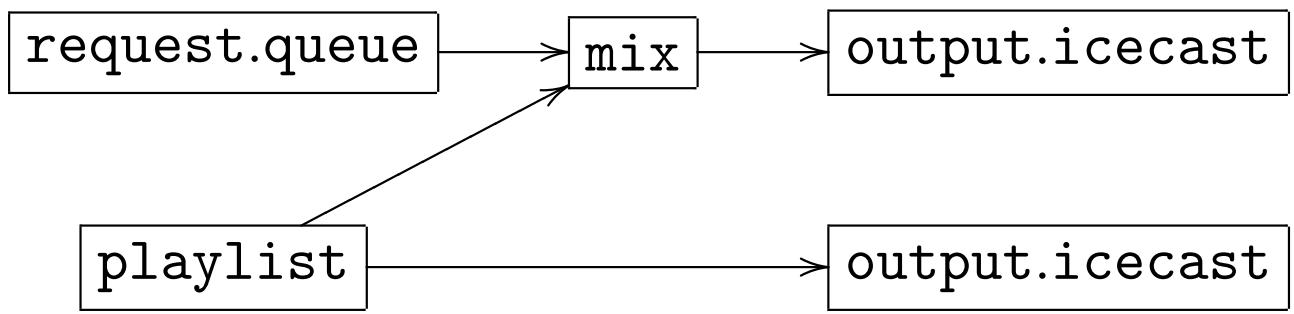

```
#!/usr/bin/liquidsoap

s = playlist("/home/smimram/liste_des_chansons")
s = normalize(s)
s = output.icecast.vorbis(host="localhost",s)
```

Gestion de flux

Une instance de Liquidsoap produit un ou plusieurs **flux** à partir d'un graphe d'**opérateurs** décrit par un script.

Un exemple plus complexe



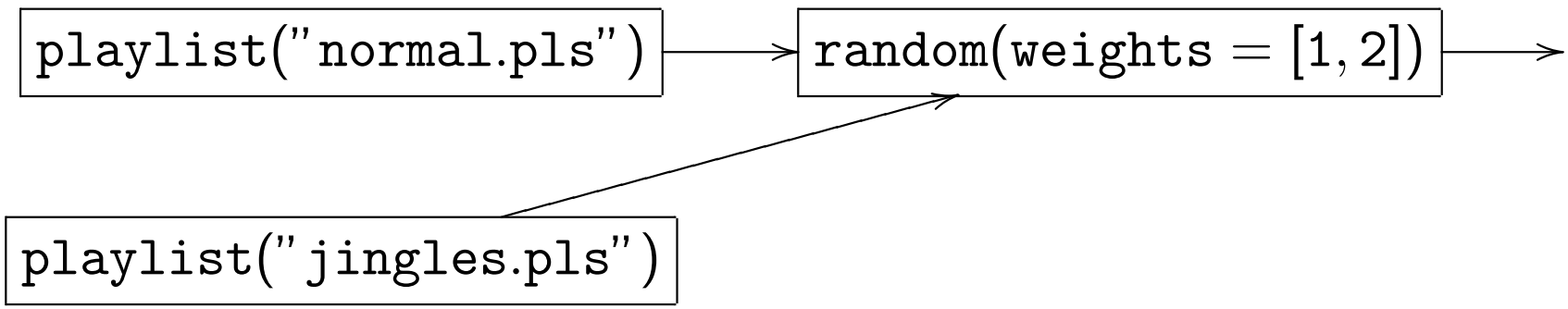
Le modèle de flux

Les flux sont :

- des suites de **données** (échantillons, . . .)
- regroupés en **pistes**
- avec des **métadonnées**
- qui peuvent être temporairement **indisponibles**.

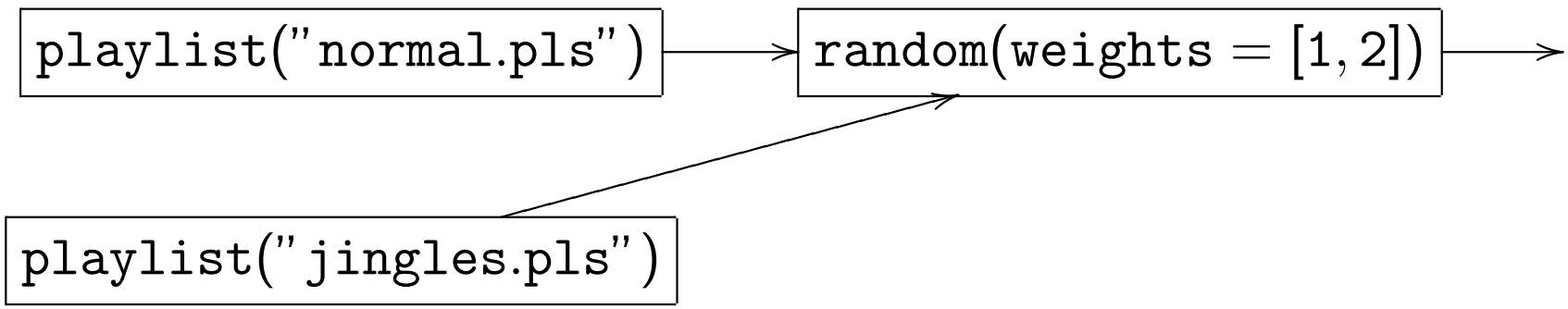
Le modèle de flux

Exemple : un *jingle* tous les deux morceaux

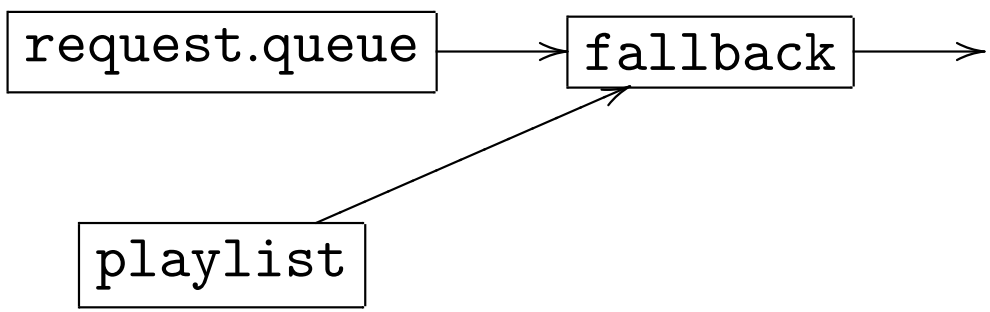


Le modèle de flux

Exemple : un *jingle* tous les deux morceaux



Exemple : requêtes d'auditeurs



Les transitions

Pour une écoute agréable, il faut des *transitions*

- entre deux pistes consécutives d'une même source,
- ou quand on passe d'une source à l'autre (e.g. `fallback`).

Les transitions

Pour une écoute agréable, il faut des *transitions*

- entre deux pistes consécutives d'une même source,
- ou quand on passe d'une source à l'autre (e.g. `fallback`).

Différents types de transitions.

- Aucune : les pistes sont juxtaposées,
- Fondu : durée du fondu, en entrée ou en sortie.
- Fondu croisé (*crossfade*) :
  - Durées des fondus, durée du recouvrement.
  - Ajuster les volumes ?
  - Insérer un jingle ?
- etc.

Les transitions

Pour une écoute agréable, il faut des *transitions*

- entre deux pistes consécutives d'une même source,
- ou quand on passe d'une source à l'autre (e.g. `fallback`).

Différents types de transitions.

- Aucune : les pistes sont juxtaposées,
- Fondu : durée du fondu, en entrée ou en sortie.
- Fondu croisé (*crossfade*) :
  - Durées des fondus, durée du recouvrement.
  - Ajuster les volumes ?
  - Insérer un jingle ?
- etc.

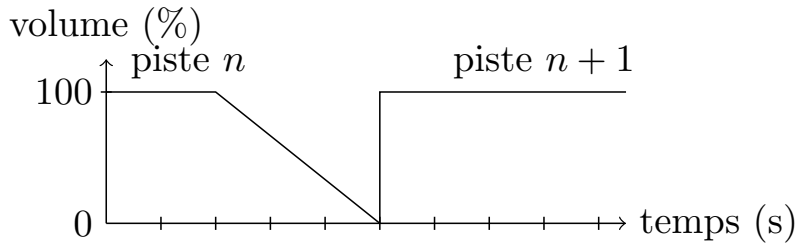
Solution générique :

$$\text{transition} = \texttt{source} \times \texttt{source} \longrightarrow \texttt{source}.$$

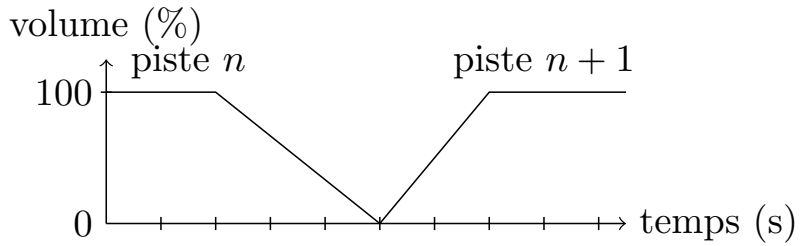
Exemple typique de transition

Réalisons un fondu croisé sur la source `s` :

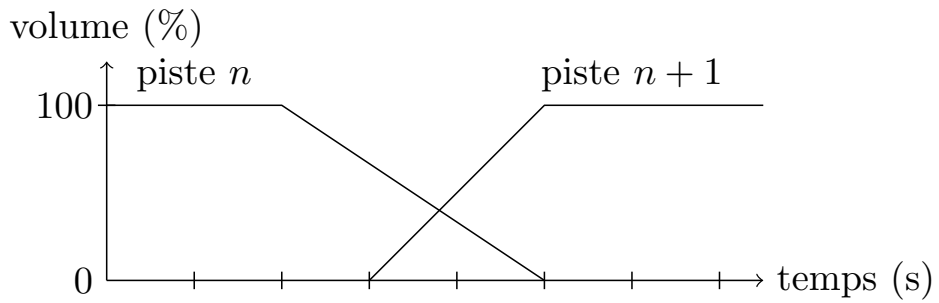
1. `s = fade.out(duration=3., s)`



2. `s = fade.in(duration=2., s)`



3. `cross(duration=2., (fun (a,b) -> add([a,b])), s)`



Le langage de script

- Un langage fonctionnel (pour les transitions).
- Des arguments étiquetés et optionnels (pour le confort).
- Besoin d'analyse statique, intégration de la documentation, simplicité d'utilisation.

$\Rightarrow$ Création d'un nouveau langage typé et d'un interpréteur.

```
$ liquidsoap -h cross
Generic cross operator, allowing the composition of the
[duration] last seconds of a track with the beginning of the
next track.
Type:
 (?duration:float, ((src, src)->src), src)->src
Parameters:
* duration    :: float (default 5.)
* (unlabeled) :: (src, src)->src    Transition function.
* (unlabeled) :: src
```

```
% liquidsoap -h output.icecast.vorbis
Output the source stream as an Ogg Vorbis stream to an Icecast-c
Type:
 (?id:string, ?samplerate:int, ?stereo:bool, ?skeleton:bool,
  ?start:bool, ?restart:bool, ?restart_delay:int, ?host:string,
  ?port:int, ?user:string, ?password:string, ?genre:string,
  ?url:string, ?description:string, ?public:bool,
  ?multicast_ip:string, ?sync:bool, ?dumpfile:string,
  ?mount:string, ?name:string, source, ?quality:float)->source
Parameters:
* id :: string (default "")                Force the value of the s
* samplerate :: int (default 44100)
* stereo :: bool (default true)
* skeleton :: bool (default false)         Add an ogg skeleton to t
* start :: bool (default true)             Start output threads on
* restart :: bool (default false)          Restart output after a f
* restart_delay :: int (default 3)         Delay, in seconds, befor
* host :: string (default "localhost")
* port :: int (default 8000)
* user :: string (default "source")        User for shout source co
* password :: string (default "hackme")
  ...
```

Comment gérer les étiquettes ?

Les étiquettes d'OCaml

Les étiquettes d'OCaml sont conçues pour être éliminées à la compilation, ce qui entraîne quelques limitations :

```
# let app f = f ~a:1 ~b:2 ;;
val app : (a:int -> b:int -> 'a) -> 'a = <fun>
# app (fun ~b ~a -> a+b) ;;
This function should have type a:int -> b:int -> 'a.
```

Les étiquettes d'OCaml

Les étiquettes d'OCaml sont conçues pour être éliminées à la compilation, ce qui entraîne quelques limitations :

```
# let app f = f ~a:1 ~b:2 ;;
val app : (a:int -> b:int -> 'a) -> 'a = <fun>
# app (fun ~b ~a -> a+b) ;;
This function should have type a:int -> b:int -> 'a.
```

On a besoin d'une multi-application, mais celle-ci ne peut être 0-aire en OCaml (d'où les arguments ineffaçables).

```
# let f ?(a=false) () = a ;;
val f : ?a:bool -> unit -> bool = <fun>
# f () ~a:true ;;
- : bool = true
# (f ()) ~a:true ;;
This expression is not a function, it cannot be applied.
```

- Dans notre cas, la simplicité passe avant l'efficacité.
- On a besoin de formaliser un λ-calcul avec
  - étiquettes,
  - arguments optionnels,
  - multi-abstraction et multi-application.

Termes

Les termes sont générés par les constructions suivantes :

$$\begin{array}{rcl} M & ::= & v \\ & | & x \\ & | & \texttt{let}\ x = M\ \texttt{in}\ M \\ & | & \lambda\{\ldots, l_i : x_i, \ldots, l_j : x_j = M, \ldots\}.M \\ & | & M\{l_1 = M, \ldots, l_n = M\} \end{array}$$

Modulo permutation d'étiquettes distinctes :

$$\begin{array}{rcl} \lambda\{\Gamma, l : x, l' : x', \Delta\}.M & \equiv & \lambda\{\Gamma, l' : x', l : x, \Delta\}.M \\ \lambda\{\Gamma, l : x = N, l' : x', \Delta\}.M & \equiv & \lambda\{\Gamma, l' : x', l : x = N, \Delta\}.M \\ \lambda\{\Gamma, l : x = N, l' : x' = N', \Delta\}.M & \equiv & \lambda\{\Gamma, l' : x' = N', l : x = N, \Delta\}.M \end{array}$$

(si $l \neq l'$).

Réduction

On a trois règles de réduction :

$$\texttt{let } x = M \texttt{ in } N \rightsquigarrow N[M/x]$$

L'application peut être *partielle*, si Γ est *ineffaçable* :

$$(\lambda\{\overrightarrow{l_i : x_i, l_j : x_j = M_j}, \Gamma\}.M)\{\overrightarrow{l_i = N_i}\} \rightsquigarrow \lambda\{\Gamma\}.(M[\overrightarrow{N_i/x_i}])$$

ou *totale* sinon :

$$(\lambda\{\overrightarrow{l_i : x_i, l_j : x_j = P_j}, \overrightarrow{l'_k : y_k = M_k}\}.M)\{\overrightarrow{l_i = N_i}\} \rightsquigarrow M[\overrightarrow{N_i/x_i}, \overrightarrow{M_k/y_k}]$$

Exemple

- $F := \lambda\{l_1 : x, l_2 : y = 12, l_3 : z = 13\}.M$
- $F\{l_3 = 3\} \rightsquigarrow \lambda\{l_1 : x, l_2 : y = 12\}.M[3/z]$
- $F\{l_3 = 3\}\{l_1 = 1\} \rightsquigarrow M[1/x, 12/y, 3/z]$

Typage

Les types et schémas de types sont sans surprise :

$$t \quad ::= \quad \iota \quad | \quad \alpha \quad | \quad \{\ldots, l_i : t_i, \ldots, ?l_j : t_j, \ldots\} \to t$$

$$\sigma \quad ::= \quad t \quad | \quad \forall \alpha.\, \sigma$$

On considère naturellement des règles comme

$$\frac{\Gamma \vdash M : \{\ldots, l_i : t_i \ldots, ?l_j : t_j, \ldots\} \to t \qquad \Gamma \vdash N_1 : t_1 \qquad \cdots \qquad \Gamma \vdash N_n : t_n}{\Gamma \vdash M\{\ldots, l_i = N_i, \ldots, l_j = N_j, \ldots\} : t}(\mathrm{App})$$

ou encore

$$\frac{\Gamma, \ldots, x_i : t_i, \ldots, x_j : t_j, \ldots \vdash M : t \qquad \cdots \qquad \Gamma \vdash M_j : t_j \qquad \cdots}{\Gamma \vdash \lambda\{\ldots, l_i : x_i, \ldots, l_j : x_j = M_j, \ldots\}.M : \{\ldots, l_i : t_i, \ldots, ?l_j : t_j, \ldots\} \to t}(\mathrm{Abs})$$

Sous-typage et application partielle

La réduction permet l'application partielle. Le système de type, pas encore : une fonction de type $\{l_1 : t_1, l_2 : t_2\} \to t$ doit pouvoir être considérée comme une fonction de type $\{l_1 : t_1\} \to \{l_2 : t_2\} \to t$.

$$\frac{\Delta \text{ ineffaçable}}{\{\Gamma, \Delta\} \to t \leq \{\Gamma\} \to \{\Delta\} \to t} \qquad \frac{\Delta \text{ effaçable}}{\{\Gamma, \Delta\} \to t \leq \{\Gamma\} \to t}$$

Sous-typage et application partielle

La réduction permet l'application partielle. Le système de type, pas encore : une fonction de type $\{l_1 : t_1, l_2 : t_2\} \to t$ doit pouvoir être considérée comme une fonction de type $\{l_1 : t_1\} \to \{l_2 : t_2\} \to t$.

$$\frac{\Delta \text{ ineffaçable}}{\{\Gamma, \Delta\} \to t \leq \{\Gamma\} \to \{\Delta\} \to t} \qquad \frac{\Delta \text{ effaçable}}{\{\Gamma, \Delta\} \to t \leq \{\Gamma\} \to t}$$

Quelques équations invalides :

- $\{l_1 : t_1\} \to \{l_2 : t_2\} \to t \not\leq \{l_1 : t_1, l_2 : t_2\} \to t$
- $\{\Gamma, ?l : t, \Delta\} \to t' \not\leq \{\Gamma, l : t, \Delta\} \to t'$

$$\{\Gamma, ?l : t, \Delta\} \to t' \not\leq \{\Gamma, l : t, \Delta\} \to t'$$

Par exemple, considérons le terme f suivant

$$\lambda\{l : x = 5\}.x \quad : \quad \{?l : int\} \to int$$

On ne peut pas calculer

$$f\{\}\{l = 6\}$$

donc on n'a pas

$$\lambda\{l : x = 5\}.x \quad : \quad \{l : int\} \to int$$

Quelques propriétés du calcul

- Le calcul pur est confluent.
- Subject reduction.
- Les termes typables sont terminants.

Inférence

- Une variante de l'algorithme W de Damas-Milner.

Inférence

- Une variante de l'algorithme W de Damas-Milner.
- Il n'y a pas de type principal : $\lambda\{l : g\}.g\{l' = 42\}$ admet

$$\forall\alpha.\ \{l : \{l' : \mathit{int}\} \to \alpha\} \to \alpha$$

et

$$\forall\alpha.\ \{l : \{?l' : \mathit{int}\} \to \alpha\} \to \alpha$$

comme types.

Inférence

- Une variante de l'algorithme W de Damas-Milner.
- Il n'y a pas de type principal : $\lambda\{l : g\}.g\{l' = 42\}$ admet

$$\forall\alpha.\ \{l : \{l' : int\} \to \alpha\} \to \alpha$$

et

$$\forall\alpha.\ \{l : \{?l' : int\} \to \alpha\} \to \alpha$$

comme types.

- Cependant, on peut plonger les termes

$$f \quad : \quad \{?l : int\} \to \alpha$$

dans

$$\lambda\{l : x\}.f\{l = x\} \quad : \quad \{l : int\} \to \alpha$$

Remarque : variante avec types principaux

Décomposer la multi-application en

- une multi-application partielle

$$(\lambda\{\overrightarrow{l_i : x_i, l_j : x_j = M_j}, \Gamma\}.M)\{\overrightarrow{l_i = N_i}\} \rightsquigarrow \lambda\{\Gamma\}.(M[\overrightarrow{N_i/x_i}])$$

- une destruction des multi-abstractions effaçables

$$(\lambda\{\overrightarrow{l_i : x_i = M_i}\}.M)\bullet \rightsquigarrow M[\overrightarrow{M_i/x_i}]$$

Deuxième partie II

Implémentation

Le code

- Implémenté en OCaml (et en C)
- Savonet (OCaml : 60 kLOC, C : 15 kLOC)
  Liquidsoap (OCaml : 35 kLOC, C : 1.8 kLOC)
- Plus de 30 librairies :
  - formats audio (mad+lame, ogg/vorbis, aac, speex, ... + tags)
  - carte son (alsa, ao, pulseaudio, portaudio, ...)
  - effets audio (ladspa, samplerate, ...)
  - transmission audio (jack, shout)
  - autres (lastfm, irc)

Les flux

- Pas d'accès direct depuis le langage.
- Découpés en **buffers**
  (perfs vs realtime)
- de type `float array`
  (endianness, précis, non boxé, tout en OCaml)
- Il faut gérer les buffers partiels
- On partage et réutilise les buffers pour éviter les copies
  (mais problèmes de temporalités)

Interaction avec l'extérieur

- Playlists dynamiques
- Telnet (ou socket)
  - queues de requêtes
  - variables
- Métadonnées (ex : replaygain, transitions)

Tout est envisageable : sites web, bots IRC, etc.

Gestion des “fichiers”

Il faut en particulier

- télécharger les fichiers distants en avance,
- gérer les flux web,
- protocoles extensibles avec plusieurs résolutions (`say://`, `replaygain://`, `strider://`, ...)

Le présent

La version SVN de Liquidsoap a le support de la **vidéo**.

- Bon point : ça rentre très facilement dans notre cadre
- De nouveaux problèmes :
  - la synchronisation entre les pistes du flux
  - il faut *vraiment* être efficace :

$$640 \times 480 \times 4 \times 24 \quad \approx \quad 30 \text{ Mo/s}$$

(en particulier encodage et YUV$\rightarrow$RGB)
$\Rightarrow$ codé en C, gcc avec optimisations

Le futur

- Gérer les flux hétérogènes de façon typée :
  - des flux que audio ou que vidéo (canaux, résolution, . . .)
  - des flux compressés
  - etc.
- Gérer finement les partages (avec des boîtes dans le typage ?)

Conclusion

- Une variante originale du λ-calcul.

Conclusion

- Une variante originale du λ-calcul.
- Beaucoup de composants réutilisables.

Conclusion

- Une variante originale du λ-calcul.
- Beaucoup de composants réutilisables.
- Un langage
  - purement fonctionnel
  - statiquement typé
  - programmé en OCaml

  qui commence à être utilisé (par des non-programmeurs).

Conclusion

- Une variante originale du λ-calcul.
- Beaucoup de composants réutilisables.
- Un langage
  - purement fonctionnel
  - statiquement typé
  - programmé en OCaml

  qui commence à être utilisé (par des non-programmeurs).
- Démo par David. . .