

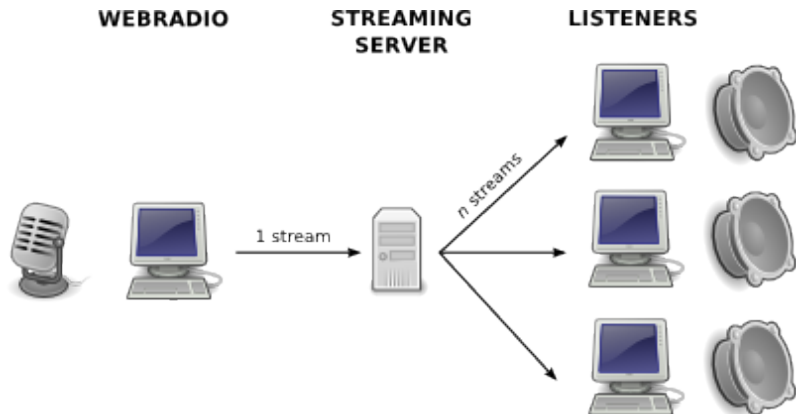
De la webradio lambda à la λ -webradio

David Baelde & Samuel Mimram

LIX, École Polytechnique, INRIA & PPS, Université Paris VII, CNRS

JFLA, 28 janvier 2008, Étretat

Qu'est-ce qu'une webradio ?



Liquidsoap est un **générateur de flux** audio, qui agit notamment en **client source** auprès du **serveur de diffusion** (Iccast).

La webradio lambda

- ▶ Outils UNIX, **légers** et **scriptables** : Ices, EZStream, ou Darkice. Diffusion d'un flux à partir d'une **suite de fichiers** ou d'une **carte son**.
- ▶ Outils professionnels : Rivendell, Master Control, WinRadio, Open Radio, etc. Interface **graphique** intégrant gestion de la playlist, des **transitions**, quelques **traitements audio**, le décrochage sur **une émission en direct**.

Dans les deux cas : un cadre prédéfini, une chaîne fixée de génération et traitement du son.

La λ -webradio

- ▶ Un **modèle** de flux audio, permettant de définir les opérations souhaitées
- ▶ Un **langage** de programmation pour composer ces opérations

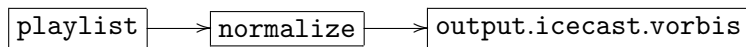
Le projet Savonet en chiffres

- ▶ 20 homme-années
- ▶ 50 000 lignes de code,
dont 40 000 en OCaml (25 000 dans Liquidsoap)
- ▶ une vingtaine de bibliothèques OCaml, natives ou bindings C

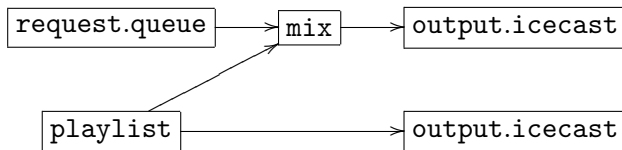
Graphe des sources

Une instance de Liquidsoap produit **un ou plusieurs flux** à partir d'un graphe constituant sa configuration.

Un exemple simple



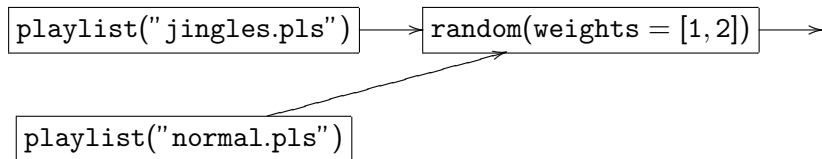
Exemple plus complexe



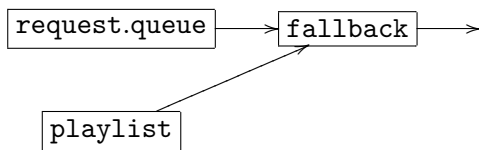
Qu'est-ce qui sort d'une source ?

Un **flux d'échantillons**, regroupés en **pistes**, qui peut être temporairement **indisponible**.

Exemple : un **jingle** tous les 3 morceaux



Exemple : requêtes d'auditeurs



Les transitions

Pour une écoute agréable, on met en place des **transitions** :

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

Types de transitions :

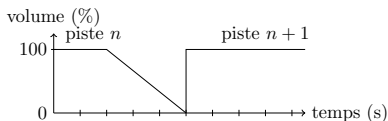
- ▶ Aucune, les pistes sont juxtaposées.
- ▶ Fondu : durée du fondu, en entrée ou en sortie.
- ▶ Fondu croisé (**crossfade**) :
 - ▶ Durées des fondus, durée du recouvrement.
 - ▶ Ajuster les volumes ?
 - ▶ Insérer un jingle ?
- ▶ etc.

Solution générique : une transition est une fonction de $\text{source} \times \text{source} \rightarrow \text{source}$.

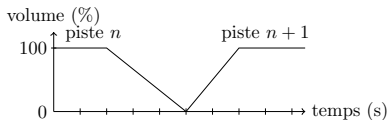
Exemple typique de transition

Réalisons un fondu croisé sur la source `s` :

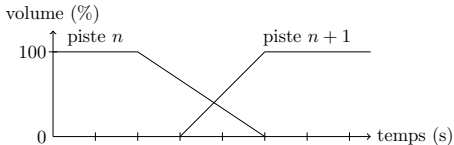
1. `s = fade.out(duration=3., s)`



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



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



Jusque là :

- ▶ Un modèle naturel
- ▶ Subtilités techniques : sources partagées, efficacité, etc.
- ▶ La notion de fonction est étroitement liée à notre modèle
- ▶ Besoin d'embarquer un langage fonctionnel

Le langage de script

Les besoins :

- ▶ Un langage fonctionnel, pour les transitions.
- ▶ Typage statique et inféré.
- ▶ Des arguments **étiquetés et optionnels** pour le confort.
- ▶ ⇒ Conception et implémentation d'un nouveau langage.

```
dbaelde@snow jfla08 $ liquidsoap -h output.icecast.vorbis
*** One entry in scripting values:
Output the source stream as an Ogg Vorbis stream to an Icecast-compatible server
in Variable BitRate mode.
Category: Source / Output
Type: (?id:string, ?samplerate:int, ?stereo: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, ?mount:string, ?name:string,
      ?quality:float, source)->source
Parameters:
* id :: string (default "")
  Force the value of the source ID.
* samplerate :: int (default 44100)
* stereo :: bool (default true)
* start :: bool (default true)
  Start output threads on operator initialization.
* restart :: bool (default false)
  Restart output after a failure. By default, liquidsoap will stop
  if the output failed.
* restart_delay :: int (default 3)
  Delay, in seconds, before attempting new connection, if restart is enabled.
* host :: string (default "localhost")
```

Les étiquettes d'OCaml

```
# let f ~a ~b = a+b ;;  
val f : a:int -> b:int -> int = <fun>  
# f ~b:3 ~a:1 ;;  
- : int = 4  
# f ~b:12 ;;  
- : a:int -> int = <fun>
```

```
# let f ?(a=1) x = x+a ;;  
val f : ?a:int -> int -> int = <fun>  
# f 2 ~a:2 ;;  
- : int = 4  
# f 2 ;;  
- : int = 3
```

Du sucre syntaxique typé

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=1) = a+1 ;;  
Warning X: this optional argument cannot be erased.  
val f : ?a:int -> int = <fun>
```

Termes

Les termes sont générés par les constructions suivantes :

$$\begin{aligned} M & ::= v \\ & | x \\ & | \text{let } x = M \text{ in } M \\ & | M\{l_1 = M, \dots, l_n = M\} \\ & | \lambda\{\dots, l_i : x_i, \dots, l_j : x_j = M, \dots\}.M \end{aligned}$$

Modulo permutation d'étiquettes distinctes $l \neq l'$:

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

Réduction

On a trois règles de réduction :

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

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

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

et **totale** sinon :

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

Exemple

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

Typage

Les **types** et **schémas** de types sont sans surprise :

$$t ::= \iota \mid \alpha \mid \{\dots, l_i : t_i, \dots, ?l_j : t_j, \dots\} \rightarrow t$$

$$\sigma ::= t \mid \forall \alpha. \sigma$$

On considère naturellement des règles comme (**App**) :

$$\frac{\Gamma \vdash M : \{\dots, l_i : t_i, \dots, ?l_j : t_j, \dots\} \rightarrow t \quad \Gamma \vdash N_1 : t_1 \quad \dots \quad \Gamma \vdash N_n : t_n}{\Gamma \vdash M\{\dots, l_i = N_i, \dots, l_j = N_j, \dots\} : t}$$

ou encore (**Abs**) :

$$\frac{\Gamma, \dots, x_i : t_i, \dots, x_j : t_j, \dots \vdash M : t \quad \dots \quad \Gamma \vdash M_j : t_j \quad \dots}{\Gamma \vdash \lambda\{\dots, l_i : x_i, \dots, l_j : x_j = M_j, \dots\}.M : \{\dots, l_i : t_i, \dots, ?l_j : t_j, \dots\} \rightarrow t}$$

Sous-typage et application partielle

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

$$\frac{\Delta \text{ ineffaçable}}{\{\Gamma, \Delta\} \rightarrow t \leq \{\Gamma\} \rightarrow \{\Delta\} \rightarrow t}$$

$$\frac{\Delta \text{ effaçable}}{\{\Gamma, \Delta\} \rightarrow t \leq \{\Gamma\} \rightarrow t}$$

Sous-typage et application partielle

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

$$\frac{\Delta \text{ ineffaçable}}{\{\Gamma, \Delta\} \rightarrow t \leq \{\Gamma\} \rightarrow \{\Delta\} \rightarrow t}$$

$$\frac{\Delta \text{ effaçable}}{\{\Gamma, \Delta\} \rightarrow t \leq \{\Gamma\} \rightarrow t}$$

Une équation invalide :

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

Sous-typage et application partielle

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

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

Une équation invalide :

► $\{\Gamma, ?l : t, \Delta\} \rightarrow t' \not\leq \{\Gamma, l : t, \Delta\} \rightarrow t'$
 $((\lambda\{l : x = 13\}.x)\{\})\{l = 1\}$

Sous-typage et application partielle

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

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

Une équation invalide :

$$\begin{aligned} \blacktriangleright \quad & \{\Gamma, ?l : t, \Delta\} \rightarrow t' \not\leq \{\Gamma, l : t, \Delta\} \rightarrow t' \\ & ((\lambda\{l : x = 13\}.x)\{\})\{l = 1\} \rightsquigarrow 13\{l = 1\} \not\rightsquigarrow \end{aligned}$$

Inférence

Pas de **type principal**, e.g. pour $\lambda\{\epsilon : f\}.(f\{l = 3\})$:

- ▶ $\forall\alpha. \{\epsilon : \{l : int\} \rightarrow \alpha\} \rightarrow \alpha$
- ▶ $\forall\alpha. \{\epsilon : \{?l : int\} \rightarrow \alpha\} \rightarrow \alpha$

Aucun n'est plus général au sens de la relation de **sous-typage**.
Le type inféré par Liquidsoap est le premier.

On peut **plonger** un f de type $\{?l : int\} \rightarrow \alpha$ dans $\{l : int\} \rightarrow \alpha$:

$$\lambda\{l : x\}.(f\{l = x\})$$

Conclusion

Contributions

- ▶ Vulgarisation :
 - ▶ Une application **OCaml** de plus **pour la “vraie vie”**,
 - ▶ un langage fonctionnel, avec types statiques inférés, pour l'utilisateur lambda.
- ▶ Beaucoup de composants réutilisables.
- ▶ Une variante originale du λ -calcul.

Conclusion

Travaux futurs

- ▶ Format de flux hétérogènes.
- ▶ Vérification statique de l'**exclusivité** requise notamment par **cross**, de la **vivacité** d'un flux.
- ▶ Variante du langage avec application partielle explicitée, qui pourrait restaurer la principalité ?

Variante : la bullet

On peut envisager d'expliciter les applications totales, en ajoutant la construction \bullet pour changer la réduction :

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

On gagne :

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

On perd (Δ effaçable) :

$$\{\Gamma, \Delta\} \rightarrow t \leq \{\Gamma\} \rightarrow \{\Delta\} \rightarrow t$$

Exemple

$$\lambda\{l : x\}.(x\{l' = 1\}) \quad :: \quad \forall \Gamma. \forall \alpha. \{\Gamma, l : \{l' : int, \Gamma\} \rightarrow \alpha\} \rightarrow \{\Gamma\} \rightarrow \alpha$$