Analyse formelle et implémentation du langage Faust
Présentation de 1re année de doctorat

Imré Frotier de la Messelière[1]
Directeur de thèse : Pierre Jouvelot[1]
Co-directeur de thèse : Jean-Pierre Talpin[2]

[1]MINES ParisTech, CRI

[2]INRIA, IRISA

Lundi 26 mai 2014

Plan de présentation

Introduction et sujet de thèse

Rappel sur Faust : Functional Audio Stream

- DSL dédié au traitement du signal.
- Paradigme de programmation strictement fonctionnel.
- Programme Faust = Description d'un processeur de signal.
- Liste de signaux d'entrée $\longrightarrow$ Liste de signaux de sortie.
- Programmes intégralement compilés (langage cible : C++).
- Utilisation d'une représentation en diagramme de blocs.

Première analyse formelle

- Définition de la sémantique des instructions de coeur du langage.
- Première caractérisation des propriétés sémantiques clés du langage.

Introduction et sujet de thèse

Objectif général

- Étendre les définitions formelles existantes du coeur du langage à l'ensemble du langage.
- Étendre et généraliser les théorèmes existants concernant les propriétés mathématiques de Faust.
- Implémenter ces analyses au sein du compilateur de Faust afin de fournir des implémentations encore plus performantes de programmes Faust.

Pistes d'étude

- Optimisation du compilateur de Faust.
- Création d'un système de typage statique pour :
  - trouver des erreurs de type durant la compilation,
  - obtenir un code compilé qui s'exécute plus rapidement,
  - et ainsi rendre Faust plus fiable et efficace.
- $\Longrightarrow$ Algorithme d'inférence de type.

Introduction et sujet de thèse

FEEVER : “Faust Environment Everyware”

Création de technologies musicales innovantes et ubiquitaires qui soient :

- portables,
- aisément programmables,
- capables de gérer de multiples plateformes,
- efficaces en terme de temps d’exécution et d’énergie,
- sécurisées.

Organisation

- Tâche 1 : Modèles.
- Tâche 2 : Compilateur.
- Tâche 3 : Ubiquité.
- Tâche 4 : Éducation.

Introduction et sujet de thèse

Intérêt et enjeux pour Faust

- Optimisation du compilateur de Faust, utilisé dans de nombreux domaines partout dans le monde.
- Typage statique $\Longrightarrow$ Sûreté.
  - Approximations pour les cas d'indécidabilité lors du type checking.
  - Gestion des intervalles, actuellement simplifiés lors des cas de récurrence.
- Preuve de propriétés actuellement conjecturées des programmes Faust.
- Gestion du problème de macro-expansion pour Faust multirate.

Contributions scientifiques générales

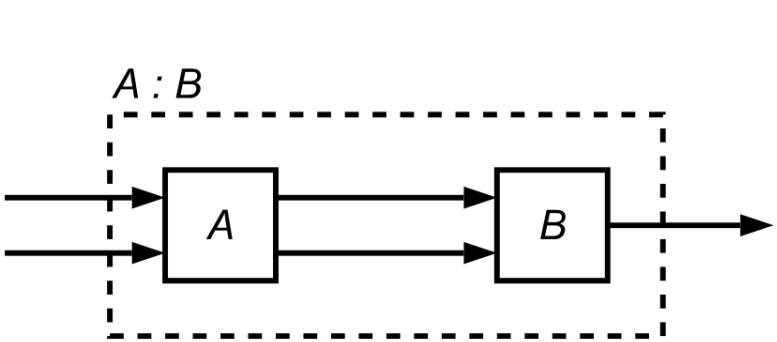
- Introduction d'un typage statique $\Longrightarrow$ Inférence de types.
- Gestion de l'indécidabilité $\Longrightarrow$ Contraintes.
- Gestion des intervalles $\Longrightarrow$ Interprétation abstraite.
- Système synchrone combinant inférence de type, contraintes et interprétation abstraite.

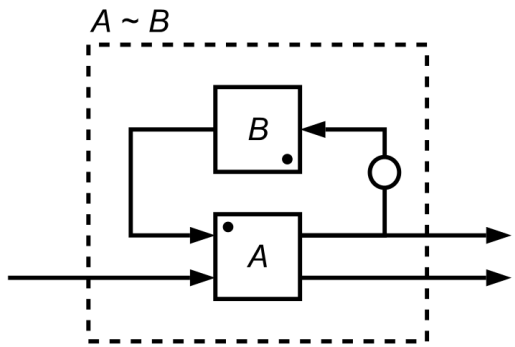
Vue d'ensemble de Faust

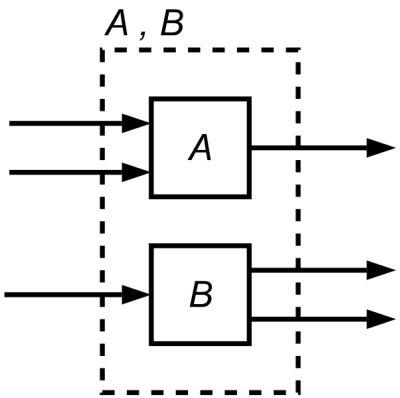
Syntaxe

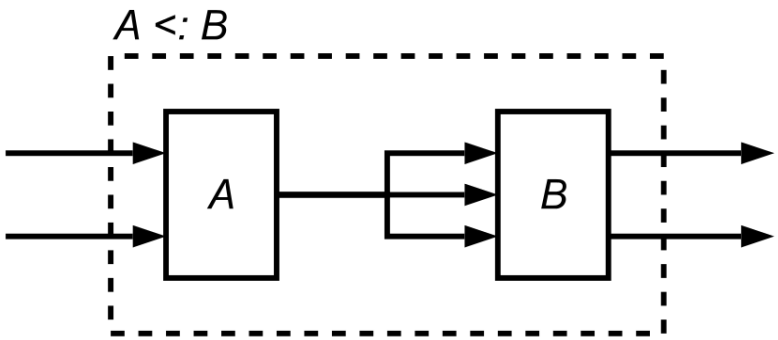
e : expression Faust, i : identifiant

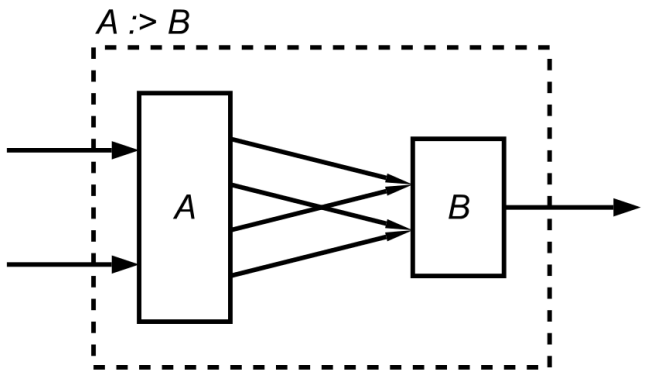
$$e ::= i \mid e_1 : e_2 \mid e_1, e_2 \mid e_1 <: e_2 \mid e_1 :> e_2 \mid e_1 \sim e_2$$











Vue d'ensemble de Faust

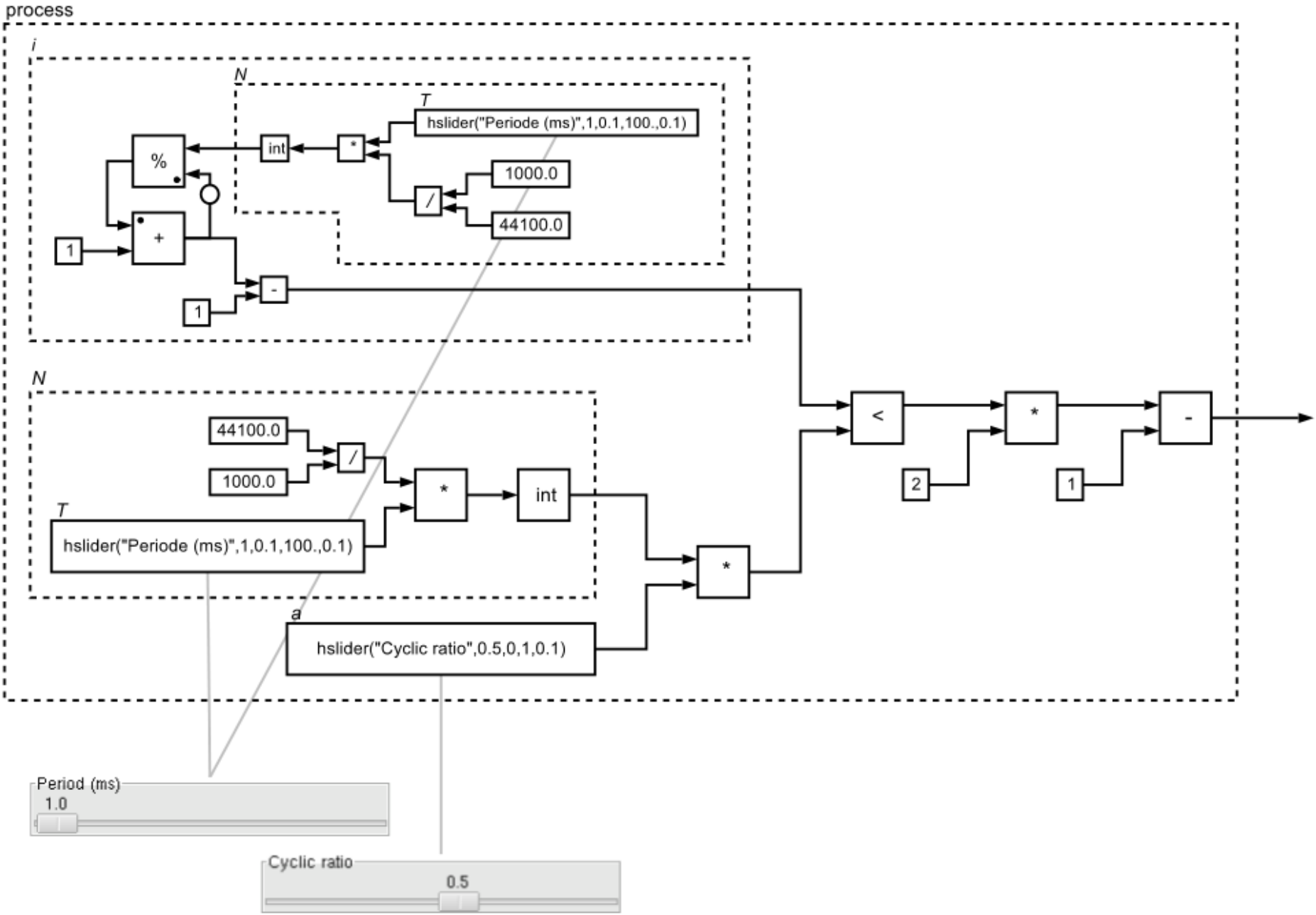
Exemple de programme Faust :

```
//-----------------------------------
//     A square wave oscillator
//-----------------------------------

T = hslider("Period",1,0.1,100.,0.1); // Period (ms)
N = 44100./1000.*T:int; // The period in samples
a = hslider("Cyclic ratio",0.5,0,1,0.1); // Cyclic ratio
i = +(1)~%(N):-(1); // 0,1,2...,n

process = i,N*a : < : *(2) : -(1) ;
```

Vue d'ensemble de Faust



Vue d'ensemble de Faust

Type d'une expression Faust :

B $\in \{$int,float$\}$, $\alpha, \beta \in \mathbb{R}$

$$\text{B } [\alpha, \beta]$$

Exemples de types :

1 : int [-2,2]

2 : int [0,3]

+ : ((int [-20,20], int [-20,20]), (int [-40,40]))

1 + 2 : int [-40,40]

Vue d'ensemble de Faust

Règles de typage :

$$\frac{\begin{array}{rcl} T(\mathtt{I}) & = & \Lambda l.(z, z') \\ \forall (x, S) \in l & . & l'(x) \in S \end{array}}{T \vdash \mathtt{I} : (z, z')[l'/l]} \qquad \frac{\begin{array}{rcl} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_1', z_2') \end{array}}{T \vdash \mathtt{E}_1 : \mathtt{E}_2 : (z_1, z_2')} \qquad \frac{\begin{array}{rcl} T & \vdash & \mathtt{E} : (z, z') \\ z' & \subset & z_1' \\ z_1 & \subset & z \end{array}}{T \vdash \mathtt{E} : (z_1, z_1')}$$

$$\frac{\begin{array}{rcl} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \end{array}}{T \vdash \mathtt{E}_1, \mathtt{E}_2 : (z_1 \| z_2, z_1' \| z_2')} \qquad \frac{\begin{array}{rcl} T & \vdash & \mathtt{E}_1 : (z_1, z') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_2 & = & z'[1, |z_2|] \\ z_2' & = & z_1[1, |z_2'|] \end{array}}{T \vdash \mathtt{E}_1 \sim \mathtt{E}_2 : (z_1[|z_2'| + 1, |z_1|], \widehat{z'})}$$

Vue d'ensemble de Faust

$$\frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_1' & \prec & z_2 \end{array}}{T \vdash \mathtt{E}_1 <: \mathtt{E}_2 : (z_1, z_2')}$$

$$\begin{array}{rcl} z_1' \prec z_2 & = & d_1' d_2 \neq 0 \text{ and} \\ & & \mathrm{mod}(d_2, d_1') = 0 \text{ and} \\ & & \|_{1, d_2, d_1'} \lambda i. z_1' = z_2 \end{array}$$

$$\frac{\begin{array}{ccc} T & \vdash & \mathtt{E}_1 : (z_1, z_1') \\ T & \vdash & \mathtt{E}_2 : (z_2, z_2') \\ z_1' & \succ & z_2 \end{array}}{T \vdash \mathtt{E}_1 :> \mathtt{E}_2 : (z_1, z_2')}$$

$$\begin{array}{rcl} z_1' \succ z_2 & = & d_1' d_2 \neq 0 \text{ and} \\ & & \mathrm{mod}(d_1', d_2) = 0 \text{ and} \\ & & \displaystyle\sum_{i \in [0, d_1'/d_2 - 1]} z_1[1 + i d_2, (i+1) d_2] = z_2 \end{array}$$

État de l'art sur le sujet

Références liées à Faust

- Faust : Yann Orlarey, Dominique Fober, Stephane Letz [10].
- Sémantique : Pierre Jouvelot, Yann Orlarey [6].
- Version multirate de Faust : Pierre Jouvelot, Yann Orlarey [7].
- Faustine : Karim Barkati, Haisheng Wang, Pierre Jouvelot [1].

Systèmes de types

- Systèmes de types et inférence de types : Luis Damas, Robin Milner [3].
- Reconstruction algébrique : Pierre Jouvelot, David Gifford [5].
- Types dépendants : Hongwei Xi [16].
  - Types de raffinement : Andrew D. Gordon, Cédric Fournet [4].
  - Types liquides : Patrick M. Rondon, Ming Kawaguchi, Ranjit Jhala [11].
- Contraintes : Olivier Tardieu, Nathaniel Nystrom, Igor Peshansky, Vijay Saraswat [13].

État de l'art sur le sujet

Solveurs

- Z3 : Leonardo de Moura, Nikolaj Bjorner [9].
- Gecode : Christian Schulte, Mikael Lagerkvist, Guido Tack.
- veriT : Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, Pascal Fontaine.
- Redlog : Andreas Dolzmann, Thomas Sturm.
- Langage smt-lib : Clark Barrett, Aaron Stump, Cesare Tinelli.

Interprétation abstraite

- Présentation : Patrick Cousot, Radhia Cousot [2].
- Système de types et effets par interprétation abstraite : Jérôme Vouillon, Pierre Jouvelot [15].
- Raffinements de types abstraits : Niki Vazou, Patrick M. Rondon, Ranjit Jhala [14].

État de l'art sur le sujet

Synchronicité

- Polychronicité :
  - Paul Le Guernic, Jean-Pierre Talpin, Jean-Christophe Le Lann [8].
  - Bin Xue, Sandeep K. Shukla.
- Constructivité séquentielle :
  Jean-Pierre Talpin, Jens Brandt, Mike Gemünde, Klaus Schneider, Sandeep Shukla [12].
- "End-to-end latency" :
  Rémy Wyss, Frédéric Boniol, Claire Pagetti, Julien Forget.
- Concurrence synchrone :
  Reinhard von Hanxleden, Michael Mendler, Joaquin Aguado, Bjorn Duderstadt, Insa Fuhrmann, Christian Motika, Stephen Mercer, Owen O'Brien.

Algorithme de typage de Faust

Vue d'ensemble

Algorithme en deux parties :

- Algorithme classique d'inférence de type, couplé avec une génération de contraintes.
- Solveur pour déterminer si le système de contraintes obtenu est décidable et fournir un mapping des variables d'unification vers des valeurs de types.

Implémentation

- Premier prototype en OCaml.
- Traduction en C++ $\Longrightarrow$ Migration vers le compilateur de Faust.
- Basé sur l'algorithme de typage :
  - Entrée : expression Faust.
  - Sortie : type de l'expression Faust ou "fail"(plusieurs cas "fail" différents).
  - Utilisation de la structure de classes introduite dans l'algorithme.
- Utilisation des bases d'implémentation fournies par Faustine, un interpréteur pour le langage Faust codé en OCaml.

Algorithme de typage de Faust

Prédicats

- Prédicat sur x : p : $x \longrightarrow$ true, false
- x = comparaison entre :
  - types de base (types),
  - bornes d'intervalles (valeurs numériques).
- Dans l'environnement n,
  (eval p) $\in$ { satisfiable, insatisfiable, satisfiabilité non déterminable }

Contraintes

Soit $c \in$ C

$c = (\rho, \iota)$ où $\rho \in \mathcal{P}(\mathsf{P})$ et $\iota \in \mathcal{P}(\mathsf{I})$

Algorithme de typage de Faust

Types contraints

- Soit $k \in \mathrm{K} = \mathrm{T} \times \mathrm{C}$
  $k = (t,c)$ où $t \in \mathrm{T}$ et $c \in \mathrm{C}$
- Type contraint = Résultat de la première partie de l'algorithme.
- Non requis par le solveur :
  - Réutilisé après l'exécution du solveur afin que le mapping fourni par le solveur y soit appliqué.
  - Application du mapping sur le type contraint $\Longrightarrow$ Type (résultat global de l'algorithme).
- Concept ayant des différences avec les "constrained types" déjà existant.

Algorithme de typage de Faust

Signatures des fonctions principales :

R : { environnements }, E : { expressions Faust },
C : { contraintes }, T : { types }, K : { types contraints },
M : { mappings de variables d'unification vers des valeurs de type }.

- type_checking : $R \longrightarrow E \longrightarrow C \longrightarrow T$ + fail
- constrained_type : $R \times C \longrightarrow E \longrightarrow K$ + fail
- solve : $C \longrightarrow M$ + fail

Niveau supérieur de l'algorithme :

type_checking *r e c* =
let (t,c') = constrained_type (r,c) *e*
let m = solve c'
m t

Algorithme de typage de Faust

“Soundess” de l’algorithme de typage :

Théorème

Soient $e \in E$, $r \in R$ *et* $c \in C$

Si *constrained_type (r,c) e* = (t, c')

et *solve* c' = m

alors $r \vdash e : m\, t$

“Completeness” de l’algorithme de typage :

Théorème

Soient $e \in E$, $r \in R$, $c \in C$ *et* $t \in T$

Si $r \vdash e : t$ *et* *solve* $c \neq$ *fail*

alors $\exists\, m \in M$ *tel que*

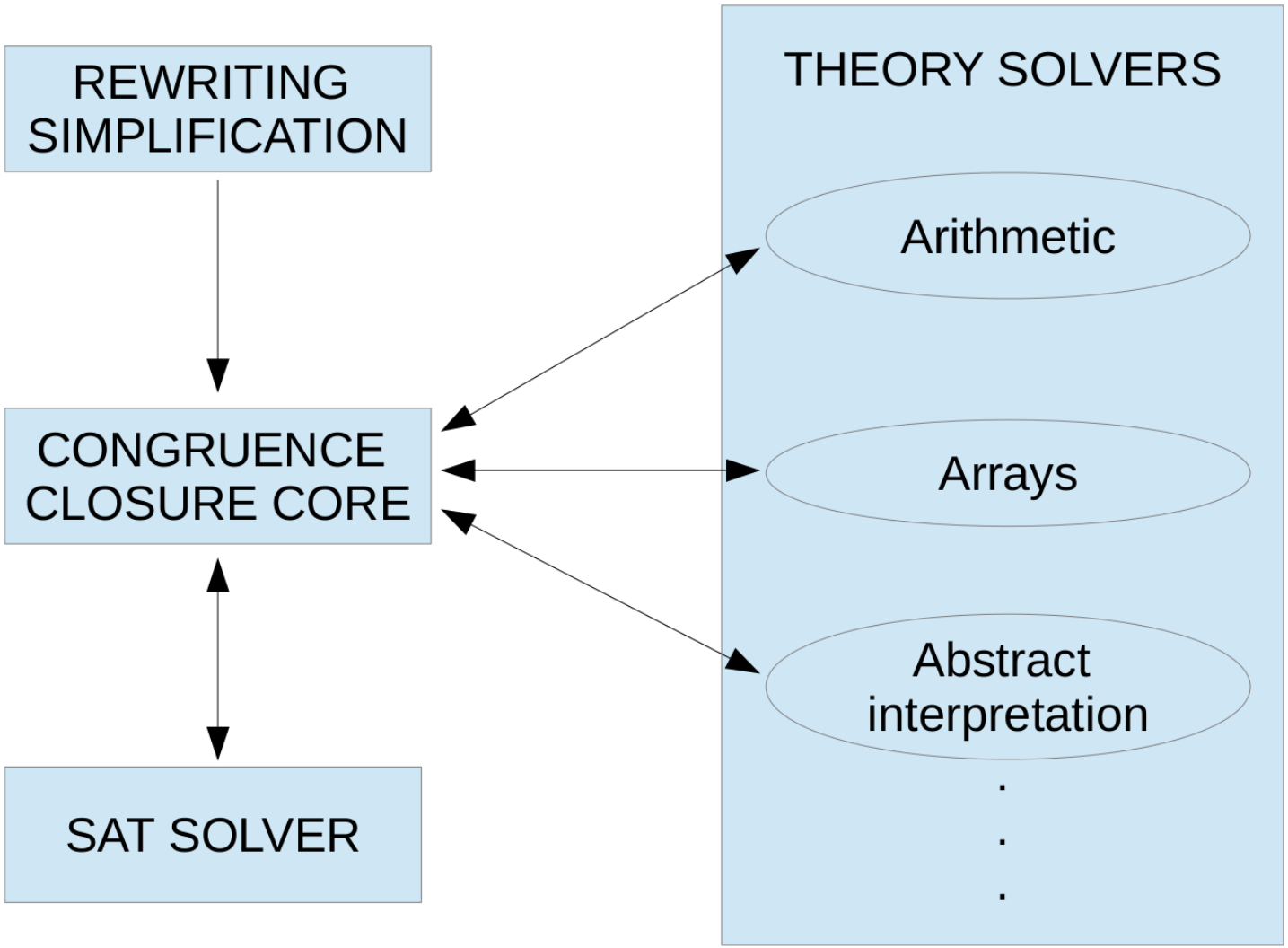
constrained_type (r,c) e = (t', c'), *solve* c' = m *et* $t \supset m\, t'$

Travail en cours et perspectives d'études

Solveur

- En cours de conception :
  - basé principalement sur Z3,
  - mais aussi sur Gecode, veriT et Redlog.
- Conception d'un solveur plus léger, utilisant uniquement les théories qui sont impliquées dans l'algorithme.
- Utilisation du "congruence closure core", par lequel le solveur SAT appelle les théories nécessaires.
- Partie solveur de l'algorithme traitée extérieurement au départ.
  - Implémentation globale opérationnelle.
  - Implémentation rendue autonome dans un second temps.
  - Conversion des contraintes en smt-lib.

Travail en cours et perspectives d'études



Travail en cours et perspectives d'études

Preuves des théorèmes

- Induction sur l'expression d'entrée.
- Admettre pour le moment la validité de la partie solveur de l'algorithme.

  $\Longrightarrow$ Fournir des preuves liées à la partie de génération de contraintes.
- Conception de la partie solveur de l'algorithme.

  $\Longrightarrow$ Possible preuve de la validité du cas d'un solveur spécifique.

Extension des théorèmes

Améliorer la fiabilité de l'algorithme de typage en évitant de mettre de côté des solutions qui sont acceptables en réalité.

- Liaison de types équivalents par substitutions.
- Considération de types différents du type précis fourni par les règles de typage, par exemple des sous-types.

Travail en cours et perspectives d'études

Version multirate de Faust

- Prise en charge de la version monorate de Faust par l'algorithme.
- Extension de l'algorithme à la version multirate de Faust.
- Considération de contraintes additionnelles sur les fréquences et les structures de vecteurs.

Implémentation

- Basé sur l'algorithme de typage, et utilisant la même infrastructure de classes.
  - Entrée : expression Faust.
  - Sortie : type de l'expression ou "fail".
- Version monorate $\Longrightarrow$ Version multirate.
- Prototype en OCaml $\Longrightarrow$ Compilateur de Faust en C++.

Travail en cours et perspectives d'études

Interprétation abstraite

- Optimisation de la gestion du cas de boucle dans la syntaxe de Faust.
- Utilisation d'une approximation basée sur l'interprétation abstraite au lieu de l'approximation à l'infini actuelle.
- Définition d'un opérateur de point fixe pour gérer ce cas.
- Insertion de cette construction dans les règles de typage.
- Deux approches additionelles différentes :
  - Insertion de cette construction hors des règles de typage lors de la génération des contraintes.
  - Insertion d'une théorie d'interprétation abstraite dans le module de théories du solveur.
- Étude de travaux existants sur des sémantiques abstraites pour des systèmes de type, notamment dans le cas de systèmes de types statiques.

Travail en cours et perspectives d'études

Synchronicité

- Système à horloges multiples $\Longrightarrow$ Extension multirate de Faust.
- Branches additionelles d'étude :
  - "end-to-end latency",
  - constructivité séquentielle,
  - concurrence synchrone.
- Constructivité séquentielle :
  - Mise en place d'une horloge interne
    $\Longrightarrow$ Gestion de flux d'entrée de fréquences différentes.
  - Outils pour créer une sémantique constructive pour Faust via la Concurrence Séquentiellement Constructive (SCC).

Conclusion et planning prévisionnel

Etat actuel de la thèse

- Contribution scientifique principale actuelle : algorithme de typage de Faust.
- Version monorate $\Longrightarrow$ Version multirate.
- Travail immédiat : complétion de l'implémentation du prototype en OCaml.
- Perspectives à moyen terme :
  - Utilisation de l'interprétation abstraite.
  - Traitement d'aspects synchrones.
  - Intégration dans le compilateur en C++ de Faust.
- Exposés supplémentaires :
  - Présentation de thèse au kick-off du projet FEEVER.
  - Présentation de thèse au centre IRISA de l'INRIA.
  - Présentation sur les types liquides dans le cadre des séminaires mensuels du CRI.

Conclusion et planning prévisionnel

Contribution des formations doctorales

- Cours scientifiques :
  - “Programmation fonctionnelle et systèmes de types“, Master parisien de recherche en informatique (MPRI). (48h)
  - “Le temps élargi : horloges multiples, temps discrets et temps continu”, Collège de France. (12h)
- Cours professionnalisants :
  - “La publication scientifique : stratégies, outils et optimisation de sa recherche.” (13h)
  - “Point de départ.” (14h)
  - “Eléments de droit pratique pour la vie en entreprise.” (30h)
- Niveau d’anglais :
  - Scores au Toefl IBT : 103/120 (2009), 107/120 (2011).
  - Master of Science in Computer Science, Columbia University in the City of New York (2013).

Conclusion et planning prévisionnel

Planning prévisionnel :

- Étape 1 : 2014
  - Version finale de l'algorithme de typage, prenant en charge la version multirate de Faust et les vecteurs, avec une implémentation complète en OCaml.
  - Présentation pour des workshops.
- Étape 2 : 2014 - 2015
  - Raffinement de l'approximation d'intervalles grâce à l'interprétation abstraite.
  - Insertion de nouveaux développements synchrones.
  - Article pour des conférences.
- Étape 3 : 2015 - 2016
  - Intégration dans le compilateur en C++ de Faust.
  - Remise de la thèse de doctorat.

Bibliographie principale I


Karim Barkati, Haisheng Wang, and Pierre Jouvelot.
Faustine : a vector faust interpreter test bed for multimedia signal processing.
In *Twelfth International Symposium on Functional and Logic Programming (FLOPS 2014)*, 2014.


Patrick Cousot and Radhia Cousot.
Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977.


Luis Damas and Robin Milner.
Principal type-schemes for functional programs.
In *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1982.


Andrew D Gordon and Cédric Fournet.
Principles and applications of refinement types.
Logics and Languages for Reliability and Security, 25, 2010.

Bibliographie principale II


Pierre Jouvelot and David Gifford.
Algebraic reconstruction of types and effects.
In *Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1991.


Pierre Jouvelot and Yann Orlarey.
Semantics for multirate faust.
New Computational Paradigms for Computer Music-Editions Delatour France, 2009.


Pierre Jouvelot and Yann Orlarey.
Dependent vector types for data structuring in multirate faust.
Computer Languages, Systems & Structures, 37(3), 2011.


Paul Le Guernic, Jean-Pierre Talpin, and Jean-Christophe Le Lann.
Polychrony for system design.
Journal of Circuits, Systems, and Computers, 12(03), 2003.

Bibliographie principale III


Leonardo de Moura and Nikolaj Bjorner.
Z3 : An efficient SMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems*, number 4963 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008.


Yann Orlarey, Dominique Fober, and Stephane Letz.
FAUST : an efficient functional approach to DSP programming.
New Computational Paradigms for Computer Music, 2009.


Patrick M. Rondon, Ming Kawaguchi, and Ranjit Jhala.
Liquid types.
In *ACM SIGPLAN Notices*, volume 43. ACM, 2008.


Jean-Pierre Talpin, Jens Brandt, Mike Gemunde, Klaus Schneider, and Sandeep Shukla.
Constructive polychronous systems.
In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, number 7734 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, January 2013.

Bibliographie principale IV


Olivier Tardieu, Nathaniel Nystrom, Igor Peshansky, and Vijay Saraswat.
Constrained kinds.
In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, New York, NY, USA, 2012. ACM.


Niki Vazou, Patrick M. Rondon, and Ranjit Jhala.
Abstract refinement types.
In *Programming Languages and Systems*. Springer, 2013.


Jerome Vouillon and Pierre Jouvelot.
Type and Effect Systems via Abstract Interpretation.
Ecole des mines de Paris, Centre de recherche en informatique, 1995.


Hongwei Xi.
Dependent ML an approach to practical programming with dependent types.
Journal of Functional Programming, 17(02), 2007.

Analyse formelle et implémentation du langage Faust
Présentation de 1re année de doctorat

Imré Frotier de la Messelière[1]
Directeur de thèse : Pierre Jouvelot[1]
Co-directeur de thèse : Jean-Pierre Talpin[2]

[1]MINES ParisTech, CRI

[2]INRIA, IRISA

Lundi 26 mai 2014

Annexes - Algorithme de typage de Faust

constrained_type (r,c) e = **case** e **in**

i =>
  if $i \in$ dom n **then**
    let (t,l) = r i
    let l' = map new_unification_variable l
    let t' = map $(\lambda xy$ (substitution x y $t))$ l l'
    $(t', (\varnothing, l') \cup c)$
  else fail

e_1 : e_2 =>
  let ($((z_1,z_1'),c_1')$, $((z_2,z_2'),c_2')$) = map (constrained_type (r,c)) (e_1,e_2)
  if $|z_1'|$ = $|z_2|$ **then**
    subtype $(c_1' \cup c_2')$ (z_1',z_2)
  else fail

e_1 , e_2 =>
  let ($((z_1,z_1'),c_1')$, $((z_2,z_2'),c_2')$) = map (constrained_type (r,c)) (e_1,e_2)
  ((append z_1 z_2, append z_1' z_2'), $c_1' \cup c_2'$)

. . .

Annexes - Algorithme de typage de Faust

. . .

e_1 <: e_2 =>
    let ($((z_1,z_1'),c_1')$, $((z_2,z_2'),c_2')$) = map (constrained_type (r,c)) (e_1,e_2)
    if ($|z_1'||z_2| \neq 0$ & $|z_2|\%|z_1'| = 0$) **then**
        constrained_type $(r,c_1' \cup c_2')$ (split_to_seq (e_1,e_2))
    else fail

e_1 :> e_2 =>
    let ($((z_1,z_1'),c_1')$, $((z_2,z_2'),c_2')$) = map (constrained_type (r,c)) (e_1,e_2)
    if ($|z_1'||z_2| \neq 0$ & $|z_1'|\%|z_2| = 0$) **then**
        constrained_type $(r,c_1' \cup c_2')$ (merge_to_seq (e_1,e_2))
    else fail

$e_1 \sim e_2$ =>
    let ($((z_1,z_1'),c_1')$, $((z_2,z_2'),c_2')$) = map (constrained_type (r,c)) (e_1,e_2)
    if ($|z_1| \geq |z_2'|$ & $|z_1'| \geq |z_2|$) **then**
        loop_approximation $(c_1' \cup c_2')$ $((z_1,z_1'),(z_2,z_2'))$
    else fail

else fail