Types liquides

Patrick M. Rondon, Ming Kawaguchi, Ranjit Jhala[1]

PLDI'08

Présentation par Imré Frotier de la Messelière[2]

[1]University of California, San Diego

[2]MINES ParisTech

Séminaire mensuel du CRI : Lundi 5 Mai 2014

Plan de présentation

Introduction - Présentation générale

Types liquides $\longleftrightarrow$ “Logically qualified data types”

- Type auquel est associée une conjonction de qualificateurs logiques.
- Qualificateur logique = prédicat q, fonction à valeur booléenne sur
  - une variable muette ν, distincte des variables du programme,
  - une variable tampon $\star$, instanciable par une variable du programme.
- Ensemble des qualificateurs logiques :

$$\mathrm{Q} = \{\ 0 \leq \nu,\ \star \leq \nu,\ \nu < \star,\ \nu < \mathrm{len}\ \star\ \}$$

Types liquides $\subset$ Types dépendants

- Type dépendant : type qui traduit une dépendance sur la valeur de l’instance concernée.
- Exemples de types dépendants : types de raffinement, types appliqués.

Introduction - Présentation générale

Notation : $\{ \nu : B \mid q \}$

- ν : variable muette n'apparaissant pas dans le programme,
- B : type de base,
- q : qualificateur logique :
  - expression à valeur booléenne contraignant ν,
  - qualificateur logique de base : ensemble des valeurs c du type de base B telles que l'évaluation de $[c/\nu]q$ renvoie $true$.

Exemples de type liquide

- $\{ \nu : \text{int} \mid 0 < \nu \}$
- $\{ \nu : \text{int} \mid \nu \leq n \}$
- $\{ \nu : B \mid true \}$
- $\{ \nu : \text{int} \mid 1 \leq \nu \wedge \nu \leq 99 \}$

Introduction - Intérêt

Intérêt propre

- Typage statique fort, qui détecte des erreurs au moment de la compilation.
- Inférence de types liquides, qui permet d'alléger les annotations d'information de type du programme :
  - Inférence de type de Hindley-Milner + abstraction par prédicat $\Longrightarrow$ types dépendants.
  - Implémentation :

    Entrée : p programme en Ocaml, $q \subset$ Q.
    Sortie : types dépendants pour les expressions dans p.

Lien avec ma thèse "Analyse formelle et implémentation du langage Faust"

- Approche parallèle à mon algorithme de typage avec une inférence de types à la Hindley-Milner suivie d'une résolution de contraintes.
- Différence de spécification des contraintes toutefois.

Système de types liquides - Syntaxe

e	$::=$		*Expressions:*
	$\mid$	x	variable
	$\mid$	$\mathtt{c}$	constant
	$\mid$	$\lambda x.e$	abstraction
	$\mid$	$e\ e$	application
	$\mid$	$\mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e$	if-then-else
	$\mid$	$\mathtt{let}\ x\ =\ e\ \mathtt{in}\ e$	let-binding
	$\mid$	$\mathtt{let\ rec}\ f\ =\ \lambda x.e\ \mathtt{in}\ e$	letrec-binding
	$\mid$	$[\Lambda\alpha]e$	type-abstraction
	$\mid$	$[\tau]e$	type-instantiation
Q	$::=$		*Liquid Refinements*
	$\mid$	$true$	true
	$\mid$	q	logical qualifier in $\mathbb{Q}^\star$
	$\mid$	$Q \wedge Q$	conjunction of qualifiers
B	$::=$		*Base Types:*
	$\mid$	$\mathtt{int}$	base type of integers
	$\mid$	$\mathtt{bool}$	base type of booleans

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Système de types liquides - Syntaxe

$\mathbb{T}(\mathbb{B})$	$::=$		*Type Skeletons:*
	$\mid$	$\{\nu : B \mid \mathbb{B}\}$	base
	$\mid$	$x : \mathbb{T}(\mathbb{B}) \rightarrow \mathbb{T}(\mathbb{B})$	function
	$\mid$	α	type variable
$\mathbb{S}(\mathbb{B})$	$::=$		*Type Schema Skeletons:*
	$\mid$	$\mathbb{T}(\mathbb{B})$	monotype
	$\mid$	$\forall \alpha . \mathbb{S}(\mathbb{B})$	polytype
τ, σ	$::=$	$\mathbb{T}(true), \mathbb{S}(true)$	*Types, Schemas*
T, S	$::=$	$\mathbb{T}(E), \mathbb{S}(E)$	*Dep. Types, Schemas*
$\hat{T}, \hat{S}$	$::=$	$\mathbb{T}(Q), \mathbb{S}(Q)$	*Liquid Types, Schemas*

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Système de types liquides - Syntaxe

Exemples de macros :

$$
\begin{array}{rcl}
\texttt{true} & :: & \{\nu : \texttt{bool} \mid \nu\} \\
\texttt{false} & :: & \{\nu : \texttt{bool} \mid \texttt{not}\ \nu\} \\
\Leftrightarrow & :: & x : \texttt{bool} \rightarrow y : \texttt{bool} \rightarrow \{\nu : \texttt{bool} \mid \nu \Leftrightarrow (x \Leftrightarrow y)\} \\
3 & :: & \{\nu : \texttt{int} \mid \nu = 3\} \\
= & :: & x : \texttt{int} \rightarrow y : \texttt{int} \rightarrow \{\nu : \texttt{bool} \mid \nu \Leftrightarrow (x = y)\} \\
+ & :: & x : \texttt{int} \rightarrow y : \texttt{int} \rightarrow \{\nu : \texttt{int} \mid \nu = x + y\} \\
\texttt{fix} & :: & \forall \alpha.(\alpha \rightarrow \alpha) \rightarrow \alpha \\
\texttt{len} & :: & \texttt{intarray} \rightarrow \{\nu : \texttt{int} \mid 0 \leq \nu\} \\
\texttt{sub} & :: & a : \texttt{intarray} \rightarrow i : \{\nu : \texttt{int} \mid 0 \leq \nu \wedge \nu < \texttt{len}\ a\} \rightarrow \texttt{int}
\end{array}
$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Système de types liquides - Règles de typage

Liquid Type Checking $\boxed{\Gamma \vdash_{\mathbb{Q}} e : S}$

$$\frac{\Gamma \vdash_{\mathbb{Q}} e : S_1 \qquad \Gamma \vdash S_1 <: S_2 \qquad \Gamma \vdash S_2}{\Gamma \vdash_{\mathbb{Q}} e : S_2} \text{ [LT-SUB]}$$

$$\frac{\Gamma(x) = \{\nu : B \mid e\}}{\Gamma \vdash_{\mathbb{Q}} x : \{\nu : B \mid \nu = x\}} \text{ [LT-VAR]} \qquad \frac{\Gamma(x) \text{ not a base type}}{\Gamma \vdash_{\mathbb{Q}} x : \Gamma(x)} \text{ [LT-VAR]}$$

$$\frac{}{\Gamma \vdash_{\mathbb{Q}} \mathsf{c} : ty(\mathsf{c})} \text{ [LT-CONST]}$$

$$\frac{\Gamma; x : \hat{T}_x \vdash_{\mathbb{Q}} e : \hat{T} \qquad \Gamma \vdash x : \hat{T}_x \rightarrow \hat{T}}{\Gamma \vdash_{\mathbb{Q}} \lambda x.e : (x : \hat{T}_x \rightarrow \hat{T})} \text{ [LT-FUN]}$$

$$\frac{\Gamma \vdash_{\mathbb{Q}} e_1 : (x : T_x \rightarrow T) \qquad \Gamma \vdash_{\mathbb{Q}} e_2 : T_x}{\Gamma \vdash_{\mathbb{Q}} e_1\ e_2 : [e_2/x]T} \text{ [LT-APP]}$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Système de types liquides - Règles de typage

$$\frac{\Gamma \vdash_{\mathbb{Q}} e_1 : \texttt{bool} \quad \Gamma; e_1 \vdash_{\mathbb{Q}} e_2 : \hat{T} \quad \Gamma; \neg e_1 \vdash_{\mathbb{Q}} e_3 : \hat{T} \quad \Gamma \vdash \hat{T}}{\Gamma \vdash_{\mathbb{Q}} \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \hat{T}} \text{ [LT-IF]}$$

$$\frac{\Gamma \vdash_{\mathbb{Q}} e_1 : S_1 \qquad \Gamma; x : S_1 \vdash_{\mathbb{Q}} e_2 : \hat{T} \qquad \Gamma \vdash \hat{T}}{\Gamma \vdash_{\mathbb{Q}} \texttt{let } x = e_1 \texttt{ in } e_2 : \hat{T}} \text{ [LT-LET]}$$

$$\frac{\Gamma \vdash_{\mathbb{Q}} e : S \qquad \alpha \text{ not free in } \Gamma}{\Gamma \vdash_{\mathbb{Q}} [\Lambda\alpha]e : \forall\alpha.S} \text{ [LT-GEN]}$$

$$\frac{\Gamma \vdash_{\mathbb{Q}} e : \forall\alpha.S \qquad \Gamma \vdash \hat{T} \qquad \mathsf{Shape}(\hat{T}) = \tau}{\Gamma \vdash_{\mathbb{Q}} [\tau]e : [\hat{T}/\alpha]S} \text{ [LT-INST]}$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Système de types liquides - Règles de typage

Decidable Subtyping $\boxed{\Gamma \vdash S_1 <: S_2}$

$$\frac{\mathsf{Valid}(\llbracket\Gamma\rrbracket \wedge \llbracket e_1\rrbracket \Rightarrow \llbracket e_2\rrbracket)}{\Gamma \vdash \{\nu : B \mid e_1\} <: \{\nu : B \mid e_2\}} \text{ [DEC-<:-BASE]}$$

$$\frac{\Gamma \vdash T'_x <: T_x \qquad \Gamma; x : T'_x \vdash T <: T'}{\Gamma \vdash x : T_x \rightarrow T <: x : T'_x \rightarrow T'} \text{ [DEC-<:-FUN]}$$

$$\frac{}{\Gamma \vdash \alpha <: \alpha} \text{ [<:-VAR]} \qquad \frac{\Gamma \vdash S_1 <: S_2}{\Gamma \vdash \forall\alpha.S_1 <: \forall\alpha.S_2} \text{ [<:-POLY]}$$

Well-Formed Types $\boxed{\Gamma \vdash S}$

$$\frac{\Gamma; \nu : B \vdash e : \texttt{bool}}{\Gamma \vdash \{\nu : B \mid e\}} \text{ [WT-BASE]} \qquad \frac{}{\Gamma \vdash \alpha} \text{ [WT-VAR]}$$

$$\frac{\Gamma; x : T_x \vdash T}{\Gamma \vdash x : T_x \rightarrow T} \text{ [WT-FUN]} \qquad \frac{\Gamma \vdash S}{\Gamma \vdash \forall\alpha.S} \text{ [WT-POLY]}$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Inférence de types liquides - Algorithme global

$$
\begin{array}{l}
\mathsf{Infer}(\Gamma, e, \mathbb{Q}) = \\
\quad \textbf{let } (F, C) = \mathsf{Cons}(\Gamma, e) \textbf{ in} \\
\quad \textbf{let } A = \mathsf{Solve}(\mathsf{Split}(C), \lambda\kappa.\mathsf{Inst}(\Gamma, e, \mathbb{Q})) \textbf{ in} \\
\quad A(F)
\end{array}
$$

Algorithme organisé en 2 parties :

- Etape 1 - Inférence de type de Hindley-Milner
  +
  Etape 2 - Génération de contraintes
- Etape 3 - Résolution de contraintes

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Inférence de types liquides - Génération de contraintes

Etape 1 - Inférence de type de Hindley-Milner

- Appel de l'algorithme de Hindley-Milner pour déduire les types de chaque sous-expression.
- Utilisation des types ML générés pour assigner un modèle à chaque sous-expression.
- Modèle :
  - Type dépendant avec la même structure que le type ML déduit,
  - Variables de types liquides représentant les valeurs de type inconnues.

Inférence de types liquides - Génération de contraintes

Etape 2 - Génération de contraintes liquides

- Utilisation des règles de typage liquide :
  - Génération d'un système de contraintes,
  - Capture des relations de sous-typage entre les modèles obtenus dans l'étape 1.
- Respect des relations $\Longrightarrow$ Existence des types fournis en sortie

Inférence de types liquides - Génération de contraintes

$$
\begin{array}{l}
\mathsf{Cons}(\Gamma, e) = \\
\quad \textbf{match } e \textbf{ with} \\
\quad \mid\ x \longrightarrow \\
\qquad \textbf{if } \mathsf{HM}(\mathsf{Shape}(\Gamma), e) = B \\
\qquad \textbf{then } (\{\nu : B \mid \nu = x\}, \emptyset) \\
\qquad \textbf{else } (\Gamma(x), \emptyset) \\
\quad \mid\ \mathsf{c} \longrightarrow \\
\qquad (ty(\mathsf{c}), \emptyset) \\
\quad \mid\ e_1\ e_2 \longrightarrow \\
\qquad \textbf{let } (x : F_x \rightarrow F, C_1) = \mathsf{Cons}(\Gamma, e_1) \textbf{ in} \\
\qquad \textbf{let } (F'_x, C_2) = \mathsf{Cons}(\Gamma, e_2) \textbf{ in} \\
\qquad ([e_2/x]F, C_1 \cup C_2 \cup \{\Gamma \vdash F'_x <: F_x\}) \\
\quad \mid\ \lambda x.e \longrightarrow \\
\qquad \textbf{let } (x : F_x \rightarrow F) = \mathsf{Fresh}(\mathsf{HM}(\mathsf{Shape}(\Gamma), \lambda x.e)) \textbf{ in} \\
\qquad \textbf{let } (F', C) = \mathsf{Cons}(\Gamma; x : F_x, e) \textbf{ in} \\
\qquad (x : F_x \rightarrow F, C \cup \{\Gamma \vdash x : F_x \rightarrow F\} \cup \{\Gamma; x : F_x \vdash F' <: F\})
\end{array}
$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Inférence de types liquides - Génération de contraintes

$$
\begin{array}{l}
\mid \ \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \longrightarrow \\
\quad \textbf{let}\ F = \mathsf{Fresh}(\mathsf{HM}(\mathsf{Shape}(\Gamma), e))\ \textbf{in} \\
\quad \textbf{let}\ (\_, C_1) = \mathsf{Cons}(\Gamma, e_1)\ \textbf{in} \\
\quad \textbf{let}\ (F_2, C_2) = \mathsf{Cons}(\Gamma; e_1, e_2)\ \textbf{in} \\
\quad \textbf{let}\ (F_3, C_3) = \mathsf{Cons}(\Gamma; \neg e_1, e_3)\ \textbf{in} \\
\quad (F, C_1 \cup C_2 \cup C_3 \cup \{\Gamma \vdash F\} \cup \\
\qquad \{\Gamma; e_1 \vdash F_2 <: F\} \cup \{\Gamma; \neg e_1 \vdash F_3 <: F\}) \\
\mid \ \texttt{let}\ x\ =\ e_1\ \texttt{in}\ e_2 \longrightarrow \\
\quad \textbf{let}\ F = \mathsf{Fresh}(\mathsf{HM}(\mathsf{Shape}(\Gamma), e))\ \textbf{in} \\
\quad \textbf{let}\ (F_1, C_1) = \mathsf{Cons}(\Gamma, e_1)\ \textbf{in} \\
\quad \textbf{let}\ (F_2, C_2) = \mathsf{Cons}(\Gamma; x : F_1, e_2)\ \textbf{in} \\
\quad (F, C_1 \cup C_2 \cup \{\Gamma \vdash F\} \cup \{\Gamma; x : F_1 \vdash F_2 <: F\}) \\
\mid \ [\Lambda\alpha]e \longrightarrow \\
\quad \textbf{let}\ (F, C) = \mathsf{Cons}(\Gamma, e)\ \textbf{in} \\
\quad (\forall\alpha.F, C) \\
\mid \ [\tau]e \longrightarrow \\
\quad \textbf{let}\ F = \mathsf{Fresh}(\tau)\ \textbf{in} \\
\quad \textbf{let}\ (\forall\alpha.F', C) = \mathsf{Cons}(\Gamma, e)\ \textbf{in} \\
\quad ([F/\alpha]F', C \cup \{\Gamma \vdash F\})
\end{array}
$$

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Inférence de types liquides - Résolution de contraintes

Etape 3 - Résolution des contraintes liquides

- Utilisation des règles de sous-typage pour scinder les contraintes complexes du modèle en contraintes simples sur les variables de type liquide.
- Résolution des contraintes simples en utilisant un calcul de point fixe inspiré de l'abstraction par prédicat.
- Obtention, pour chaque variable de type liquide, de la plus forte conjonction de qualificateurs de Q* qui satisfasse toutes les contraintes.

Inférence de types liquides - Résolution de contraintes

- Satisfaction des assignations liquides :
  - Assignation liquide sur Q : mapping A des variables de type liquide vers des ensembles de qualificateurs issus de Q*.
  - A satisfait une contrainte c si A(c) est valide.
- Etape 1 : Subdivision en contraintes simples :

$$\textbf{let } A = \mathsf{Solve}(\mathsf{Split}(C), \lambda\kappa.\mathsf{Inst}(\Gamma, e, \mathbb{Q})) \textbf{ in}$$

- Etape 2 : Affaiblissement iteratif :

$$\begin{array}{l}
\mathsf{Solve}(C, A) = \\
\quad \textbf{if } \text{exists } c \in C \text{ such that } A(c) \text{ is not valid} \\
\quad \textbf{then } \mathsf{Solve}(C, \mathsf{Weaken}(c, A)) \textbf{ else } A \\
\\
\mathsf{Weaken}(c, A) = \\
\quad \textbf{match } c \textbf{ with} \\
\quad \mid \Gamma \vdash \{\nu : B \mid \theta \cdot \kappa\} \longrightarrow \\
\qquad A[\kappa \mapsto \{q \mid q \in A(\kappa) \textbf{ and } \mathsf{Shape}(\Gamma); \nu : B \vdash \theta \cdot q : \texttt{bool}\}] \\
\quad \mid \Gamma \vdash \{\nu : B \mid \rho\} <: \{\nu : B \mid \theta \cdot \kappa\} \longrightarrow \\
\qquad A[\kappa \mapsto \{q \mid q \in A(\kappa) \textbf{ and } [\![A(\Gamma)]\!] \wedge [\![A(\rho)]\!] \Rightarrow [\![\theta \cdot q]\!]\}] \\
\quad \mid _ \longrightarrow \textbf{Failure}
\end{array}$$

Inférence de types liquides - Théorèmes

THEOREM 1. **[Liquid Type Safety]**

1. *(Overapproximation) If* $\Gamma \vdash_{\mathbb{Q}} e : S$ *then* $\Gamma \vdash e : S$.
2. *(Preservation) If* $\Gamma \vdash e : S$ *and* $e \hookrightarrow e'$ *then* $\Gamma \vdash e' : S$.
3. *(Progress) If* $\emptyset \vdash e : S$ *and* e *is not a value then there exists an* e', $e \hookrightarrow e'$.

THEOREM 2. **[Liquid Type Inference]**

1. $\mathsf{Infer}(\Gamma, e, \mathbb{Q})$ *terminates,*
2. *If* $\mathsf{Infer}(\Gamma, e, \mathbb{Q}) = S$ *then* $\Gamma \vdash_{\mathbb{Q}} e : S$, *and,*
3. *If* $\mathsf{Infer}(\Gamma, e, \mathbb{Q}) =$ **Failure** *then there is no* S *s.t.* $\Gamma \vdash_{\mathbb{Q}} e : S$.

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Résultats expérimentaux

Ensemble de benchmarks en OCaml annotés avec des types dépendants dans le projet DML [9].

$\Longrightarrow$ Montrer que les types liquides réduisent le nombre d'annotations manuelles requises pour prouver la sûreté des accès de tableaux.

Intégration de types dépendants dans ML

- Utilisation de DML.
- Récupération de garanties statiques sur la sûreté des accès de tableaux $\Longrightarrow$ inférence automatique.
- Annotations manuelles dont le type checker a besoin pour prouver la sûreté dans les benchmarks utilisés : environ 31% du code.

Résultats expérimentaux

Implémentation avec DSolve

- Entrée :
  - programme en OCaml,
  - ensemble de qualificateurs.
- Sortie :
  - types dépendants inférés pour les expressions du programme,
  - ensemble des éléments dont la sûreté n'a pu être prouvée.

Résultats

- Réduction de la quantité d'annotations requises :

  31% du code $\longrightarrow$ moins de 1% du code

- Pas besoin d'annotations manuelles pour la plupart des programmes.

Résultats expérimentaux

	Size		DML		DSOLVE		
Program	**Line**	**Char**	**Line**	**Char**	**Line**	**Char**	**Time (s)**
dotprod	7	158	3 (30%)	110 (41%)	0 (0%)	0 (0%)	0.31
bcopy	8	199	3 (27%)	172 (46%)	0 (0%)	0 (0%)	0.15
bsearch	24	486	3 (11%)	157 (24%)	0 (0%)	0 (0%)	0.46
queen	30	886	7 (19%)	309 (26%)	0 (0%)	0 (0%)	0.70
isort	33	899	12 (27%)	702 (44%)	0 (0%)	0 (0%)	0.88
tower	36	927	8 (18%)	348 (27%)	1 (2%)	26 (2%)	3.33
matmult	43	797	10 (19%)	454 (36%)	0 (0%)	0 (0%)	1.79
heapsort	84	1414	11 (12%)	433 (23%)	0 (0%)	0 (0%)	0.53
fft	107	3279	13 (11%)	790 (19%)	1 (1%)	25 (1%)	9.13
simplex	118	2712	33 (22%)	1913 (41%)	0 (0%)	0 (0%)	7.73
gauss	142	2431	22 (13%)	999 (29%)	1 (1%)	67 (2%)	3.17
TOTAL	633	14188	125 (17%)	6387 (31%)	3(1%)	93(1%)	
qsort-o	62	1585			0 (0%)	0 (0%)	1.89
qsort-d	112	2735			5 (5%)	63 (2%)	18.28
bitv	426	10757			65 (15%)	3138 (29%)	63.11

- Légende :
  - Size : taille du code (sans annotation) après suppression des espaces et des commentaires.
  - DML : quantité d'annotations manuelles requises par DML.
  - DSolve : quantité d'annotations manuelles requises par DSolve.
  - Time : temps d'exécution de l'inférence de type avec DSolve.

Source : Patrick M. Rondon , Ming Kawaguchi , Ranjit Jhala. Liquid Types. University of California , San Diego.

Travaux connexes - Principaux domaines d'étude

- Types dépendants pour améliorer l'expressivité des systèmes de types : [4, 9]
- Abstraction par prédicats : [2, 5]
- Analyse de programmes basée sur des contraintes : [7, 8]
  - Qualificateurs de types : [3]
  - Assignation d'une sémantique aux qualificateurs : [1]
- Raffinements de types : [4, 6, 10]

Travaux connexes - Bibliographie I


Brian Chin, Shane Markstrum, Todd Millstein, and Jens Palsberg.
Inference of user-defined type qualifiers and qualifier rules.
In *Programming Languages and Systems*. Springer, 2006.


Cormac Flanagan and Shaz Qadeer.
Predicate abstraction for software verification.
ACM SIGPLAN Notices, 37(1), 2002.


Jeffrey Scott Foster.
Type qualifiers : lightweight specifications to improve software quality.
PhD thesis, UNIVERSITY OF CALIFORNIA, 2002.


Tim Freeman and Frank Pfenning.
Refinement types for ML, volume 26.
ACM, 1991.


Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan.
Abstractions from proofs.
In *ACM SIGPLAN Notices*, volume 39. ACM, 2004.

Travaux connexes - Bibliographie II


Kenneth Knowles and Cormac Flanagan.
Type reconstruction for general refinement types.
In *Programming Languages and Systems*. Springer, 2007.


Patrick Lincoln and John C. Mitchell.
Algorithmic aspects of type inference with subtypes.
In *Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1992.


Martin Odersky, Martin Sulzmann, and Martin Wehr.
Type inference with constrained types.
TAPOS, 5(1), 1999.


Hongwei Xi and Frank Pfenning.
Eliminating array bound checking through dependent types.
ACM SIGPLAN Notices, 33(5), 1998.

Travaux connexes - Bibliographie III

Hongwei Xi and Frank Pfenning.
Dependent types in practical programming.
In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1999.

Conclusion et travaux ultérieurs

- Contributions scientifiques :
  - Système de types dépendants : les types liquides,
  - Outil DSolve pour inférer des types liquides,
  - Expériences montrant que les types liquides sont une optimisation des types dépendants usuels.
- Pistes d'étude :
  - Raffinements pour les variables de types et les types de données récursifs,
  - Rapportage d'erreurs,
  - Raisonnement sur les fonctionnalités impératives.
- Articles ultérieurs :
  - Implémentation et application directe des types liquides.
- $\Longrightarrow$ Démarche parente de celle de l'élaboration de l'algorithme de typage de Faust, mais sans gestion du typage multirate ou de l'interprétation abstraite.

Types liquides

Patrick M. Rondon, Ming Kawaguchi, Ranjit Jhala[1]

PLDI'08

Présentation par Imré Frotier de la Messelière[2]

[1]University of California, San Diego

[2]MINES ParisTech

Séminaire mensuel du CRI : Lundi 5 Mai 2014