Typage et programmation en logique avec contraintes

Emmanuel COQUERY

Projet Contraintes – INRIA Rocquencourt

Plan

1. Introduction
2. Système de types pour
   la programmation en logique avec contraintes
3. Résolution de contraintes de sous-typage
4. Le logiciel TCLP
5. Conclusion

Programmation logique avec contraintes

- Termes : a, `1+X`, `[]`, `[Head|Tail]`
- Contraintes : `=/2`, `#</2`
- Prédicats : `length/2`, `arg/3`, `same_length/2`
- Buts : `length(L1,X), length(L2,X)`
- Clauses :
  `same_length(L1,L2)`
  ← `length(L1,X), length(L2,X)`

Objectifs

- Détection d'erreurs de programmation
- Système de types (prescriptif)
- Utilisable avec les langages logiques avec contraintes actuels (ex : GNU-Prolog, SICStus, etc)
- Conserver les capacités de ces langages :
  - Méta-programmation :

    `arg/3`, `call/1`, `assert/1`

Objectifs

- Détection d'erreurs de programmation
- Système de types (prescriptif)
- Utilisable avec les langages logiques avec contraintes actuels (ex : GNU-Prolog, SICStus, etc)
- Conserver les capacités de ces langages :
  - Méta-programmation
  - Utilisation conjointe de plusieurs domaines de contraintes :

Contrainte (disjonctive) d'exclusion mutuelle en ordonnancement

```
(X #> Y + DY) #<=> B1,
(Y #> X + DX) #<=> B2,
B1+B2 #= 1
```

coercition : booléens vu comme des entiers

Objectifs

- Détection d'erreurs de programmation
- Système de types (prescriptif)
- Utilisable avec les langages logiques avec contraintes actuels (ex : GNU-Prolog, SICStus, etc)
- Conserver les capacités de ces langages :
  - Méta-programmation
  - Utilisation conjointe de plusieurs domaines de contraintes
  - Symboles utilisés de différentes manières:

    `2-3` opération arithmétique
    `a-[b,c,d]` paire
- Typer des bibliothèques de programmes existants

Systèmes de type existants

- [Mycroft, O'Keefe '84] : adaptation du système de type de ML à Prolog, implanté dans Gödel [Lloyd, Hill '92]
  - Polymorphisme paramétrique
  - Ne supporte pas la méta-programmation
- [Smolka '89,Hanus '91,...] : système avec sous-typage
  - Modèle d'exécution typé non standard
  - Sous-typage pas assez puissant pour la méta-programmation : $\mathrm{list}(\alpha) \not< \mathrm{term}$
- [Fages-Paltrinieri '97] : système avec sous-typage
  - Supporte la méta-programmation: $\mathrm{list}(\alpha) < \mathrm{term}$
  - Algorithme de vérification, mais pas d'inférence de type
  - Pas d'implantation

Trois formes de polymorphisme

- Polymorphisme paramétrique :
  $\mathrm{list}(\alpha)$, $\mathrm{pair}(\alpha, \beta)$

- Sous-typage non structurel non homogène :
  $\mathrm{boolean} < \mathrm{int}$, $\mathrm{list}(\alpha) < \mathrm{term}$

- Surcharge (polymorphisme *ad-hoc*) :
  `-/2`: $\mathrm{int} \times \mathrm{int} \rightarrow \mathrm{int}$ et `-/2`: $\alpha \times \beta \rightarrow \mathrm{pair}(\alpha, \beta)$

Polymorphisme paramétrique

- Structures de données :
  listes : `[]/0`: $\mathrm{list}(\alpha)$,
  `[|]/2`: $\alpha \times \mathrm{list}(\alpha) \rightarrow \mathrm{list}(\alpha)$
  paires : `-/2`: $\alpha \times \beta \rightarrow \mathrm{pair}(\alpha, \beta)$

- Certaines contraintes ou certains prédicats :
  `=/2`: $\alpha \times \alpha \rightarrow \mathrm{pred}$
  `append/3`: $\mathrm{list}(\alpha) \times \mathrm{list}(\alpha) \times \mathrm{list}(\alpha) \rightarrow \mathrm{pred}$

Sous-typage

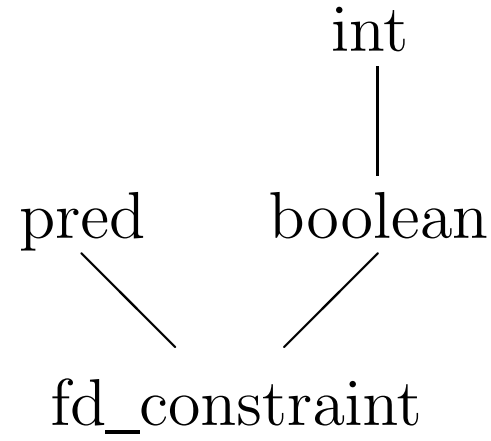
- Méta-programmation
  - Décomposition de termes, avec le supertype term :
    `arg/3`: $\mathrm{int} \times \mathrm{term} \times \mathrm{term} \rightarrow \mathrm{pred}$
    `=../2`: $\mathrm{term} \times \mathrm{list}(\mathrm{term}) \rightarrow \mathrm{pred}$
  - Prédicats dynamiques, avec $\mathrm{pred} < \mathrm{clause} < \mathrm{term}$:
    `assert/1`: $\mathrm{clause} \rightarrow \mathrm{pred}$
- Coercition entre domaines de contraintes :

  Exemple dans CLP($\mathcal{FD}, \mathcal{B}$) :

  `#<=>/2`: $\mathrm{boolean} \times \mathrm{boolean} \rightarrow \mathrm{pred}$

  `#</2`: $\mathrm{int} \times \mathrm{int} \rightarrow \mathrm{fd\_constraint}$

  `#=/2`: $\mathrm{int} \times \mathrm{int} \rightarrow \mathrm{fd\_constraint}$




```
(X #> Y + DY) #<=> B1, (Y #> X + DX) #<=> B2,
B1+B2 #= 1
```

Surcharge

- L'opérateur `-/2` :

  $\text{int} \times \text{int} \rightarrow \text{int}$ $\quad\quad$ $\text{int} \times \text{float} \rightarrow \text{float}$

  $\text{float} \times \text{float} \rightarrow \text{float}$ $\quad\quad$ $\text{float} \times \text{int} \rightarrow \text{float}$

  $\alpha \times \beta \rightarrow \text{pair}(\alpha, \beta)$

- Le symbole `[|]/2` :

  $\alpha \times \text{list}(\alpha) \rightarrow \text{list}(\alpha)$

  $\text{atom} \times \text{list}(\text{atom}) \rightarrow \text{pred}$ (alias pour `consult/1`)

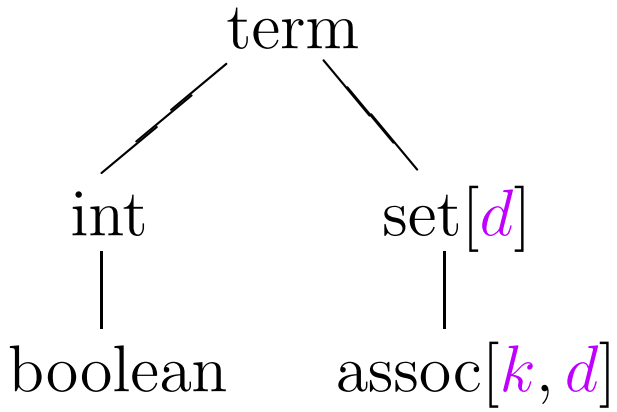
Plan

1. Introduction
2. Système de types pour
   la programmation en logique avec contraintes
3. Résolution de contraintes de sous-typage
4. Le logiciel TCLP
5. Conclusion

Algèbre de types

- Constructeurs : $\mathrm{boolean}$, int, $\mathrm{set}[d]$, $\mathrm{assoc}[k, d]$, term
- Types : $\mathrm{boolean}$, $\mathrm{set}(\mathrm{int})$, $\mathrm{set}(\alpha)$, $\mathrm{assoc}(\mathrm{boolean}, \mathrm{set}(\mathrm{boolean}))$, $\mathrm{set}(\mathrm{set}(\mathrm{set}(\ldots)))$

Ordre partiel quelconque $\leq_{\mathcal{K}}$ sur les constructeurs :



Relation d'ordre induite sur les types :

$$\mathrm{boolean} \leq \mathrm{int}$$

$$\mathrm{set}(\mathrm{boolean}) \leq \mathrm{set}(\mathrm{int})$$

$$\mathrm{assoc}(\mathrm{set}(\mathrm{boolean}), \mathrm{boolean}) \leq \mathrm{set}(\mathrm{int})$$

$$\mathrm{assoc}(\tau_1, \tau_2) \leq \mathrm{set}(\tau_3) \text{ si } \tau_2 \leq \tau_3$$

Arguments contravariants possibles

Contraintes de sous-typage : $\bigwedge_i \tau_i \leq \tau'_i$

Système de type

(Sub) $$\frac{\Gamma \vdash t : \tau, \tau \leq \tau'}{\Gamma \vdash t : \tau'}$$

(Var) $$\Gamma \vdash X : \Gamma(X)$$

(Func) $$\frac{\Gamma \vdash t_1 : \rho(\tau_1) \ , \ldots , \ \Gamma \vdash t_n : \rho(\tau_n)}{\Gamma \vdash f(t_1, \ldots, t_n) : \rho(\tau)}$$

ρ est une substitution de types

$$\tau_1 \times \ldots \times \tau_n \rightarrow \tau \in types(f/n)$$

(Query) $$\frac{\Gamma \vdash A_1 : \mathrm{pred} \ , \ldots , \ \Gamma \vdash A_n : \mathrm{pred}}{\Gamma \vdash A_1, \ldots, A_n \ Query}$$

Système de type - suite

(Head) $$\frac{\Gamma \vdash t_1 : \rho(\tau_1) \ , \ldots , \ \Gamma \vdash t_n : \rho(\tau_n)}{\Gamma \vdash p(t_1, \ldots, t_n) \ Head_{p:\tau_1,\ldots,\tau_n}}$$

ρ est un renommage

$\tau_1 \times \ldots \times \tau_n \rightarrow \mathrm{pred} \in types(p/n)$

[Lakshman, Reddy '91]

(Clause) $$\frac{\Gamma \vdash H \ Head_{p:\tau_1,\ldots,\tau_n} \ , \ \Gamma \vdash Q \ Query}{\Gamma \vdash H \leftarrow Q \ Clause_{p:\tau_1,\ldots,\tau_n}}$$

Cohérence au modèle d'exécution CSLD

Résolution CSLD :

$$\frac{p(t) \leftarrow c' | A_1, \ldots, A_n \in P\theta, \quad \mathcal{X} \models \exists(c \wedge s = t \wedge c')}{(c|Q, p(s), Q') \rightarrow^{P}_{CSLD} (c, s = t, c' \mid Q, A_1, \ldots, A_n, Q')}$$

où θ est un renommage du programme P avec de nouvelles variables

Théorème :

Soit P un programme bien typé.
Soit Q un but bien typé, $\Gamma \vdash Q\ Query$.

Si $Q \rightarrow^{P}_{CSLD} Q'$ alors il existe Γ' tel que $\Gamma \cup \Gamma' \vdash Q'\ Query$.

Exemple d'erreur: inversion d'argument

total_length/3 : $\text{list}(\alpha) \times \text{list}(\alpha) \times \text{int} \rightarrow \text{pred}$

append/3: $\text{list}(\alpha) \times \text{list}(\alpha) \times \text{list}(\alpha) \rightarrow \text{pred}$

length/2: $\text{list}(\alpha) \times \text{int} \rightarrow \text{pred}$

$$\begin{array}{c} \text{set}[d] \\ | \\ \text{int} \\ | \\ \text{list}[d] \end{array}$$

Programme mal typé:

```
total_length(L1,L2,N) :-
      append(L1,L2,L), length(N,L).
```

But bien typé :

```
total_length(T1,T2,3)
```

$\longrightarrow^{P}_{CSLD}$

```
T1=L1, T2=L2, 3=N, append(L1,L2,L), length(N,L)
```

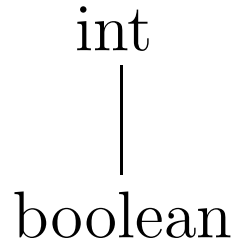
Modèle d'exécution avec substitutions

Soit le programme P suivant :

```
p(X) :- X = 3.
q(Y) :- Y #<=> 1.
```

`p/1`: int → pred

`q/1`: boolean → pred



$$\texttt{p(Z),q(Z)} \longrightarrow^{P}_{CSLD} \texttt{Z=X,X=3,q(Z)} \longrightarrow_{subst} \texttt{Z=3,X=3,q(3)}$$

$\Rightarrow$ Contraintes de type sur les variables à l'exécution

Modèle d'exécution typé

Contraintes de types $t : \tau$ dans le domaine $\mathcal{X} = \mathcal{Y} \cup 2^{\mathcal{Y}}$:

- $[[\, u \,]] \in 2^{\mathcal{Y}}, \quad [[\, u[e_1, \ldots, e_n] \,]] : 2^{\mathcal{Y}} \times \ldots \times 2^{\mathcal{Y}} \rightarrow 2^{\mathcal{Y}}$
- $[[\, . \,]]$ compatible avec $\leq$ et avec les déclarations de types
- $t : \tau$ satisfiable dans $\mathcal{Y} \cup 2^{\mathcal{Y}}$ si $\exists \rho$ t.q. $\rho(t) \in \rho(\tau)$

Le but TCLP associé à un but Q bien typé, $\Gamma \vdash Q \ Query$, est le but Q auquel on ajoute les contraintes de type correspondant à Γ.

Théorème : *Soit Q un but TCLP, $\Gamma \vdash Q \ Query$.*
Si Q contient une contrainte `X=t` *alors $\Gamma \vdash Q[X/t] \ Query$.*

`Z : boolean, p(Z), q(Z)` $\not\rightarrow_{CSLD}$

car `Z : boolean, Z=X, X=3` n'est pas satisfiable

Vérification des types

- Incorporation de (Sub) dans (Func), (Atom) et (Head)

(Func') $$\frac{\Gamma \vdash t_1 : \tau'_1 \leq \rho(\tau_1) \ , \ldots , \ \Gamma \vdash t_n : \tau'_n \leq \rho(\tau_n)}{\Gamma \vdash f(t_1, \ldots, t_n) : \rho(\tau)}$$

ρ est une substitution de types

$$\tau_1 \times \ldots \times \tau_n \rightarrow \tau \in types(f/n)$$

- Inférence du type des variables par résolution des contraintes de type collectées lors de la dérivation
- Résolution de la surcharge par backtracking en retardant au maximum les choix

Inférence de type pour les prédicats

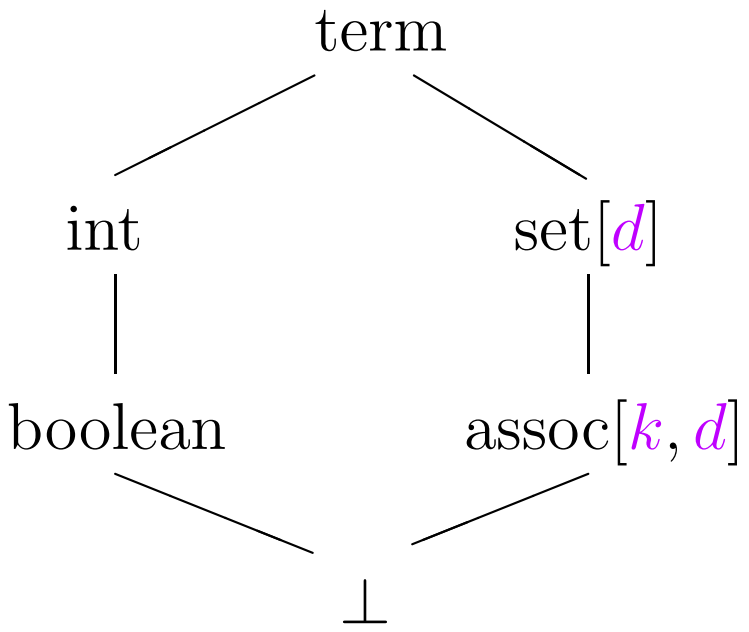
- Type $\mathrm{term} \times \ldots \times \mathrm{term} \rightarrow \mathrm{pred}$ toujours possible
  $\Rightarrow$ inférence heuristique
  - Ordre des contructeurs complété en treillis avec $\bot$ et $\top$
  - Calcul d'un type minimal τ_1 comme la borne supérieure des types trouvés dans les têtes des différentes clauses
  - Calcul d'un type heuristique τ_2 basé sur τ_1 et sur le type des variables apparaissant dans les têtes des clauses
  - Le type heuristique inféré est la généralisation de τ_2
- Surcharge $\Rightarrow$ plusieurs types possibles

Plan

1. Introduction
2. Système de types pour
   la programmation en logique avec contraintes
3. Résolution de contraintes de sous-typage
4. Le logiciel TCLP
5. Conclusion

Contraintes de sous-typage non structurel non homogène

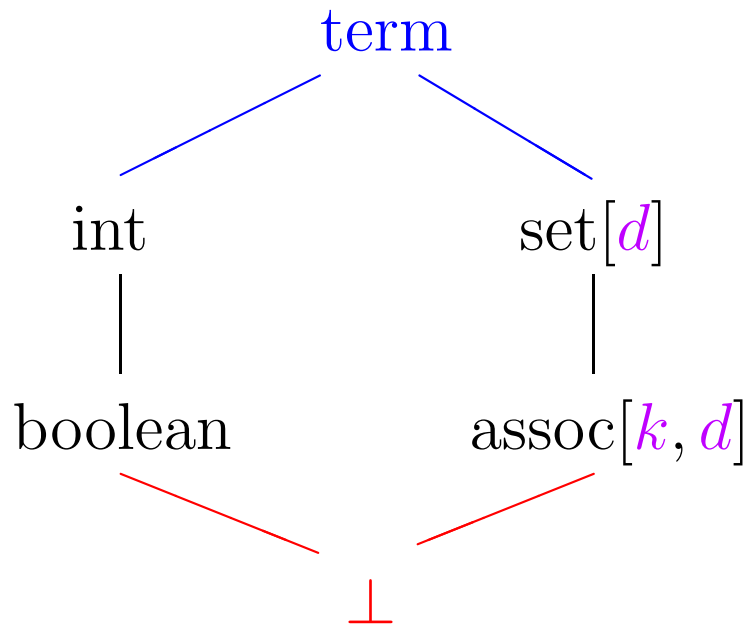
Cas des treillis



- Satisfiabilité : Trifonov & Smith (’96)
- Calcul de solutions : Pottier (’98)

Contraintes de sous-typage non structurel non homogène

Cas des treillis



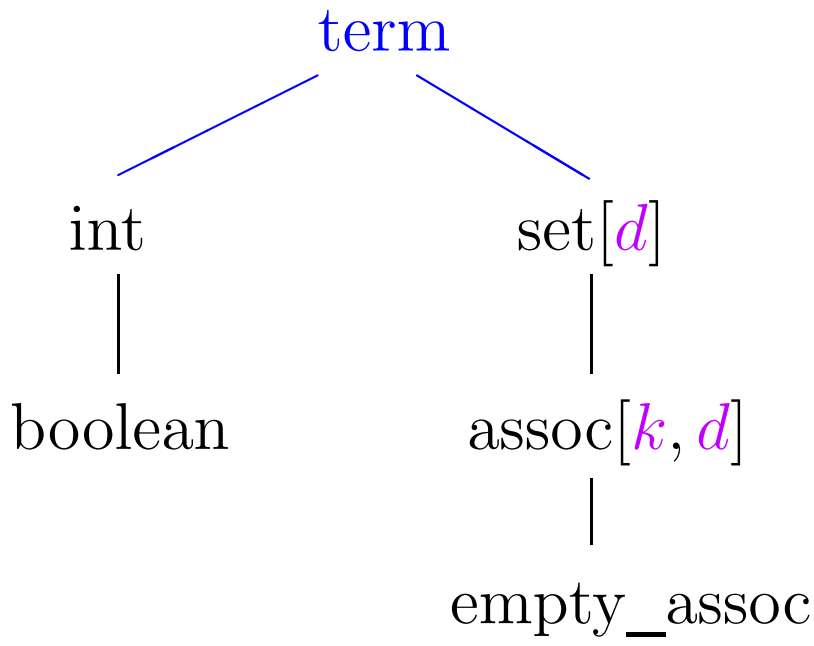
- Satisfiabilité : Trifonov & Smith (’96)
- Calcul de solutions : Pottier (’98)
- Problème : présence obligatoire de $\bot$, sans signification, et de term, non souhaité en l’absence de méta-programmation

Contraintes de sous-typage non structurel non homogène

Cas des quasi-treillis

borne inférieure ssi minorant

borne supérieure ssi majorant



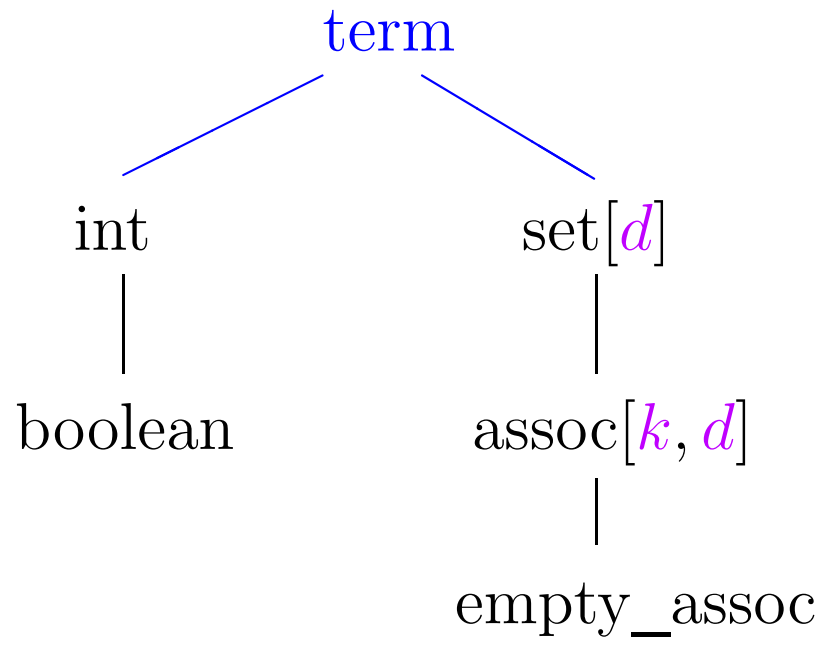
- **Théorème :** *Si* $(\mathcal{K}, \leq_{\mathcal{K}})$ *est un quasi-treillis complet, alors* $(\mathcal{T}, \leq)$ *est un quasi-treillis complet*

Contraintes de sous-typage non structurel non homogène

Cas des quasi-treillis

borne inférieure ssi minorant

borne supérieure ssi majorant



- Décidabilité de la satisfiabilité dans les quasi-treillis ? conjecture de Smolka (’88)
- **Théorème :** *Dans les quasi-treillis ayant un nombre fini d’extrema tous constants et dans lesquels tout élément est encadré par deux extrema, le problème de la satisfiabilité est décidable et NP-Complet.*

Décomposition de contraintes

Trifonov et Smith ('96)

(Trans) $C, \tau \leq \alpha, \alpha \leq \tau' \rightarrow C, \tau \leq \alpha, \alpha \leq \tau', \tau \leq \tau'$

(Fail) $C, \mathrm{t}(\tau_1, \ldots, \tau_m) \leq \mathrm{u}(\tau'_1, \ldots, \tau'_n) \rightarrow$ faux si $\mathrm{t} \not\leq_{\mathcal{K}} \mathrm{u}$

(Dec) $C, \mathrm{t}(\tau_1, \ldots, \tau_m) \leq \mathrm{u}(\tau'_1, \ldots, \tau'_n) \rightarrow$
$C, \mathrm{t}(\tau_1, \ldots, \tau_m) \leq \mathrm{u}(\tau'_1, \ldots, \tau'_n) \cup \{\tau_i \leq \tau'_j \mid e_i = e'_j\}$
si $\mathrm{t}[e_1, \ldots, e_m] \leq_{\mathcal{K}} \mathrm{u}[e'_1, \ldots, e'_n]$

exemple :

$\mathrm{set}[d]$
|
$\mathrm{assoc}[k, d]$

$\mathrm{assoc}(\tau_1, \tau_2) \leq \mathrm{set}(\tau_3) \rightarrow$
$\mathrm{assoc}(\tau_1, \tau_2) \leq \mathrm{set}(\tau_3), \tau_2 \leq \tau_3$

Satisfiabilité dans les quasi-treillis

- Système C pré-clos : Toutes les variables de C sont bornées ($\forall \alpha \in V(C), \exists \tau, \tau' \notin V(C), \tau \leq \alpha, \alpha \leq \tau' \in C$)

- **Théorème :** *Dans un quasi-treillis, les systèmes de contraintes pré-clos et clos par $\rightarrow$ sont satisfiables.*

$\Rightarrow$ Algorithme de test de satisfiabilité dans les quasi-treillis pour les systèmes pré-clos par calcul de clôture en $O(n^3)$

Pré-clôture

Hypothèses :

- Le quasi-treillis admet un nombre fini M d'extrema.
- Ces extrema sont des constantes.
- Tout constructeur de type est majoré par un élément maximal et minoré par un élément minimal.

$\Rightarrow$ Pour un système C non pré-clos, on peut énumérer les bornes possibles pour chaque variable non bornée de C.

$\Rightarrow$ Algorithme NP pour tester la satisfiabilité des systèmes non pré-clos en $O(n^3 M^v)$.

- NP-Complet pour les "n-crowns" [Pratt, Tiuryn '96]

Calcul de bornes

Généralisation de l'algorithme de Pottier ('98) aux quasi-treillis

- Calcul de solutions explicites
  par factorisation des bornes des variables.
  - Phénomène d'oubli d'argument

- Obtention d'un ensemble de solutions maximales.

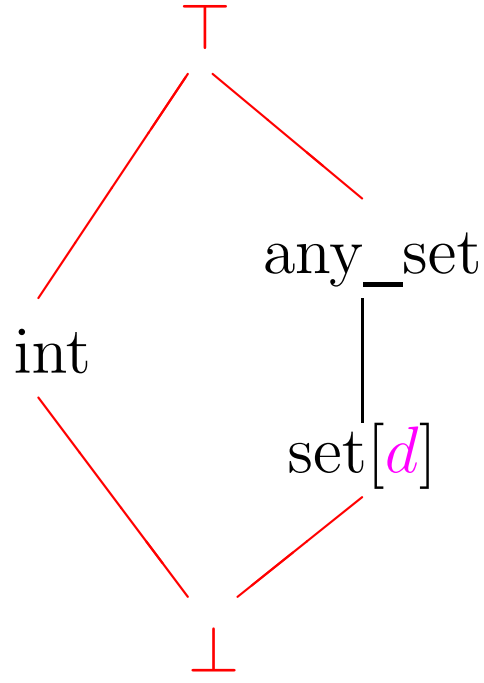
- Pas d'hypothèse sur les minima du quasi-treillis.
  Le type $\mathrm{empty\_assoc}$ n'est pas nécessaire.

Hypothèse: le système C de départ est constitué de types plats,
i.e. des variables ou des types de la forme $\mathrm{t}(\alpha_1, \ldots, \alpha_n)$.

Calcul de bornes dans les treillis

Généralisation de l'algorithme de Pottier ('98) aux quasi-treillis

- Phénomène d'oubli d'argument

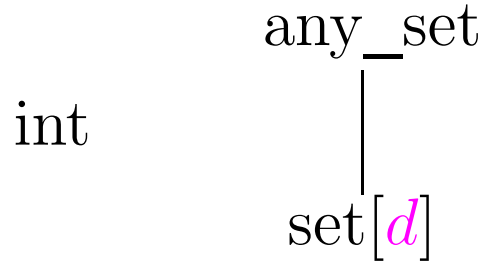


$$\mathrm{set}(\mathrm{int}) \leq \alpha,$$
$$\mathrm{set}(\mathrm{any\_set}) \leq \alpha$$
$$\rightarrow$$
$$\mathbf{set}(\top) \leq \alpha$$

Calcul de bornes dans les quasi-treillis

Généralisation de l'algorithme de Pottier ('98) aux quasi-treillis

- Phénomène d'oubli d'argument



$$\mathrm{set}(\mathrm{int}) \leq \alpha,$$
$$\mathrm{set}(\mathrm{any\_set}) \leq \alpha$$
$$\rightarrow$$
$$\mathbf{any\_set} \leq \alpha$$

Algorithme

- Plongement du quasi-treillis des contructeurs de types dans un treillis par ajout des constructeurs $\bot$ et $\top$.
- Completion du système C en un système C':
  Pour chaque sous-ensemble A de $V(C)$, deux variables sont introduites dans C' à des fins de factorisation :
  - λ_A qui représente la borne supérieure de A,
  - γ_A qui représente la borne inférieure de A,
  - Les variables de C sont dites originelles, les autres sont dites introduites.
- Application d'un système de réécriture $\rightarrow^{\uparrow}$ non déterministe sur C'

Système $\rightarrow^{\uparrow}$

Règles issues de l'algorithme de canonisation incrémentale de Pottier ('98) pour les treillis:

- Transitivité sur les variables: (Trans)
- Décomposition: (Dec) et (Clash)
- Factorisation et transitivité des bornes: (Glb), (Lub), (Glb Trans) et (Lub Trans)

Règles spécifiques aux quasi-treillis:

- Pré-clôture supérieure: (Intro $\uparrow$)
- Elimination de $\top$ et $\bot$: (Fail $\bot$), (Fail $\top$), (Up), (Down)

Règle de pré-clôture (Intro ↑)

(Intro ↑) $C \rightarrow C, \alpha \leq \mathrm{t}$

(Intro ↑) $C \rightarrow C, \top \leq \alpha$

où $\mathrm{t} \in \overline{\mathcal{K}}$ et
si $\alpha \leq \mathrm{u}(\alpha_1, \ldots, \alpha_n) \notin C$ et $\top \leq \alpha \notin C$

Non-terminaison si les maxima ne sont pas constants:

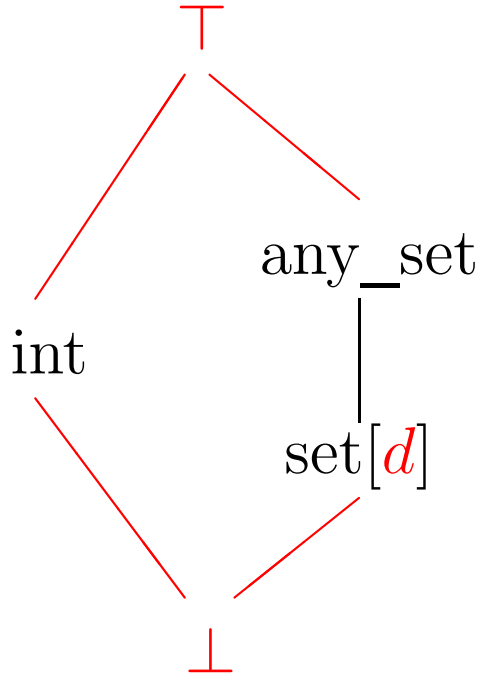
$$\begin{array}{lll} \alpha \leq \mathrm{list}(\beta) \rightarrow & \alpha \leq \mathrm{list}(\beta), \beta \leq \mathrm{set}(\delta) & \\ \rightarrow & \alpha \leq \mathrm{list}(\beta), \beta \leq \mathrm{set}(\delta), \delta \leq \mathrm{set}(\zeta) & \mathrm{int} \\ \rightarrow & \ldots & \end{array} \qquad \begin{array}{c} \mathrm{set}[d] \\ \vert \\ \mathrm{list}[d] \end{array}$$

Règles (Fail $\top$) et (Up)

(Fail $\top$) $\quad \top \leq \alpha \rightarrow$ faux

si α est originelle

Si λ_A non originelle, pas d'échec:

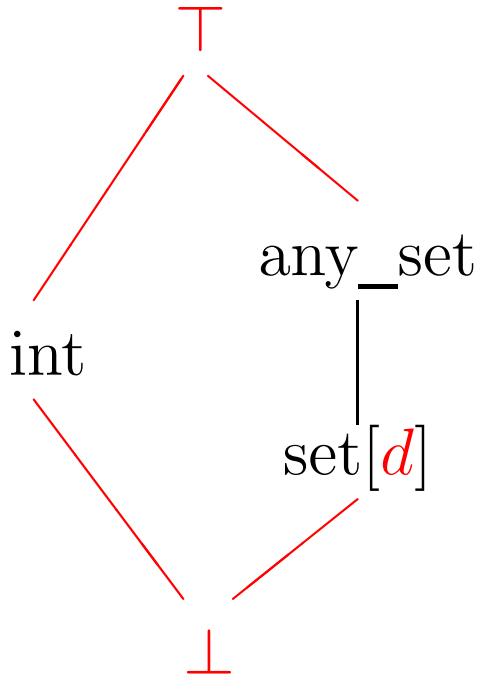


$\text{set}(\lambda_A) \leq \alpha$, $\top \leq \lambda_A$

Règles (Fail $\top$) et (Up)

(Up) $\mathrm{t}(\tau_1, \ldots, \tau_i, \ldots, \tau_m) \leq \alpha, \top \leq \lambda_A \rightarrow \mathrm{u}(\tau'_1, \ldots, \tau'_n) \leq \alpha, \top \leq \lambda_A$

si $\tau_i = \lambda_A$, si $\mathrm{u}[e'_1, \ldots, e'_n] = \sqcup \uparrow_{\{e_1,\ldots,e_m\}\setminus\{e_i\}} \mathrm{t}[e_1, \ldots, e_m]$,
et si $e'_j = e_k$ alors $\tau'_j = \tau_k$



$\mathrm{set}(\lambda_A) \leq \alpha\ ,\ \top \leq \lambda_A \rightarrow$
$\mathrm{any\_set} \leq \alpha, \top \leq \lambda_A$

Solutions maximales

Si les constructeurs de types n'ont que des arguments covariants:

Théorème: *Soit un système C, complété en C', tel que $C' \rightarrow^{\uparrow} D \not\rightarrow^{\uparrow}$. Si $D \neq$ faux alors la valuation ρ solution de l'unification de chaque variable avec sa borne supérieure dans D est:*

- *la solution maximale de D*
- *une solution de C dans le quasi-treillis*

$\Rightarrow$ Algorithme pour tester la satisfiabilité sans condition sur les minima

$\Rightarrow$ Algorithme pour obtenir un ensemble de solutions maximales

Complexité ? problème ouvert (comme dans les treillis) potentiellement exponentiel

Plan

1. Introduction
2. Système de types pour
   la programmation en logique avec contraintes
3. Résolution de contraintes de sous-typage
4. Le logiciel TCLP
5. Conclusion

Vérification et inférence de type

- Implantation en SICStus Prolog et CHR
- Gestion de la surcharge:
  - Une paire de contraintes CHR pour chaque occurence de symbole surchargé
  - Elimination des types impossibles pour occurence de symbole surchargé en utilisant son contexte
- Inférence heuristique du type des prédicats:
  - Pour éviter une explosion combinatoire due aux prédicats surchargés, seul le premier type inféré est utilisé
- Possibilité de typer des programmes CHR et (L)CC: langages Concurrent avec Contraintes (Linéraires)

Performances

Bibliothèque		Vérification		Inférence pour les prédicats			
	#lignes	temps	surcharge	#inf/#tot	temps	surcharge	exacts
arrays	96	0.8 s	22 %	6/13	1.0 s	19 %	68 %
assoc	209	2.2 s	24 %	13/31	3.9 s	22 %	91 %
atts	216	3.7 s	39 %	18/20	3.3 s	41 %	91 %
bdb	487	3.1 s	26 %	55/79	4.2 s	25 %	66 %
clpr	4286	47.0 s	61 %	396/419	142.5 s	68 %	N/A
heaps	137	1.9 s	26 %	11/21	5.5 s	27 %	97 %
lists	186	1.9 s	51 %	10/39	2.6 s	46 %	98 %
ordsets	208	2.4 s	37 %	17/35	7.3 s	50 %	97 %
sockets	211	1.8 s	44 %	12/27	2.1 s	36 %	92 %
terms	97	1.3 s	33 %	4/14	1.7 s	32 %	77 %
trees	72	0.8 s	34 %	8/16	1.2 s	27 %	75 %
ugraphs	542	14.1 s	52 %	62/90	31.9 s	35 %	67 %

Implantation des contraintes de sous-typage

Solveur implanté en CHR

- Unification dans CHR : règle de simpli£cation des égalités

Comparaison avec la bibliothèque OCaml Wallace [Pottier ’00]:

Fichier	#lignes	Véri£cation		Inférence pour les prédicats	
		OCaml	CHR	OCaml	CHR
assoc	209 l	5.3 s	6.0 s	40.1 s	13.6 s
atts	216 l	7.4 s	5.5 s	77.5 s	12.4 s
bdb	487 l	23.6 s	20.2 s	41.1 s	17.4 s
heaps	137 l	3.5 s	4.2 s	43.3 s	17.4 s
lists	186 l	3.5 s	3.8 s	16.2 s	6.6 s
ordsets	208 l	4.1 s	5.2 s	199.4 s	44.8 s
terms	97 l	2.5 s	2.6 s	308.7 s	4.3 s
trees	72 l	1.4 s	1.6 s	12.6 s	3.2 s

Distribution de TCLP

- Supporte SICStus Prolog, GNU-Prolog et ISO Prolog
- Extension aisée à d'autres dialects CLP en donnant les types des prédicats "built-in"

- Testé, entre autres, sur :
  - Des bibliothèques de SICStus Prolog
  - Une implantation de CLP($\mathcal{FD}$) en Prolog
  - Le logiciel CLPGUI (version GNU-Prolog)
  - TCLP lui-même

- Disponible:
  `http://contraintes.inria.fr/~coquery/tclp`

Conclusion

Système de types pour la programmation logique avec contraintes

- Utilise le polymorphisme paramétrique, le sous-typage et la surcharge pour typer :
  - La méta-programmation
  - Les coercitions de domaine de contraintes
  - La surcharge des opérateurs
- Inférence de type des variables et inférence de type heuristique pour les prédicats

Résolution de contraintes de sous-typage:

- Sous-typage non structurel non homogène
- Dans des quasi-treillis de types avec un nombre fini de maxima, tous constants
  (conjecture [Smolka '88] sans hypothèse sur les maxima)

Perspectives

- Approche algébrique [Frey ’03] + théorème d’auto-réduction [Coquery, Fages, Smaus ’05]
- Extension du système de types aux langages de solveurs de contraintes comme CHR, (L)CC
- Typage de langages de règles pour le web, CLP(XML)

- Contraintes de type dans des structures plus générales:
  - Les quasi-treillis dont les maxima ne sont pas des constantes
  - Les ordres partiels quelconques
    (Cas homogène: satisfiabilité PSPACE-Complet [Frey ’97])
- Application des contraintes d’ordre dans les quasi-treillis à d’autres domaines, par exemple les ontologies [Laburthe ’03]