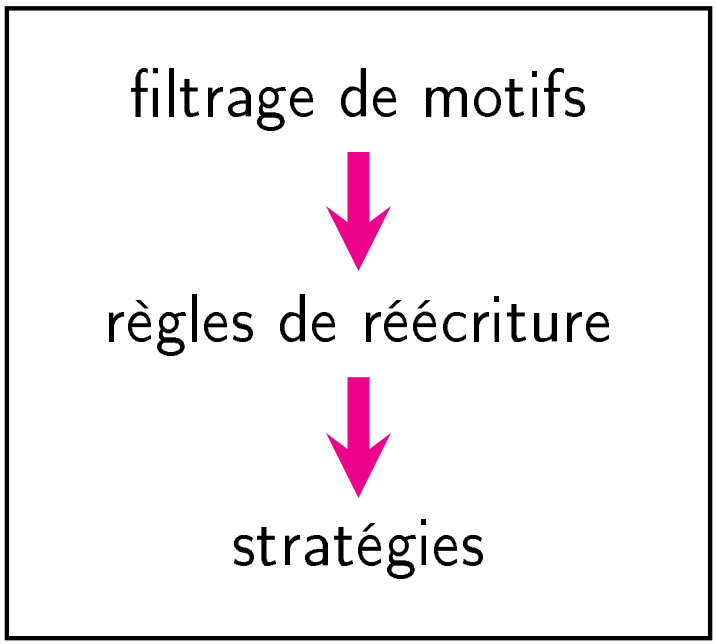
Un système de types pour la programmation par réécriture embarquée

Cláudia Tavares

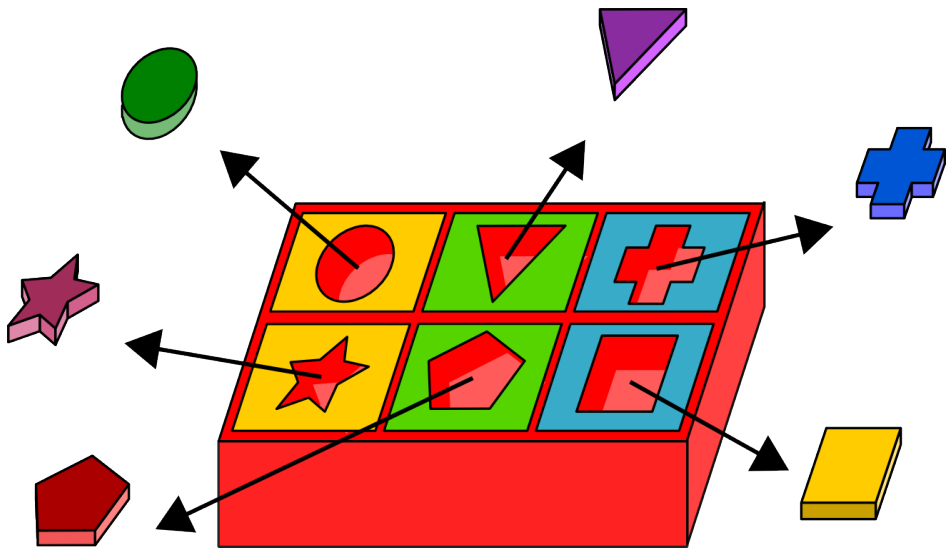
Pareo
INRIA & LORIA
Université de Lorraine & École doctorale IAEM

Le 2 mars 2012

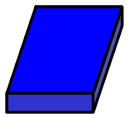
- Projet Tom [?, ?, ?]
- Intégrer la réécriture dans un langage généraliste comme Java
- Augmenter l'expressivité de Tom

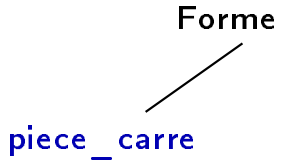

filtrage de motifs
règles de réécriture
stratégies

Recherche le motif qui correspond à un terme donné

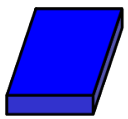


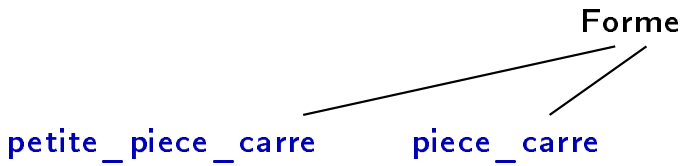
Classe les motifs par rapport à leurs types



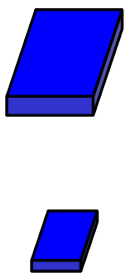


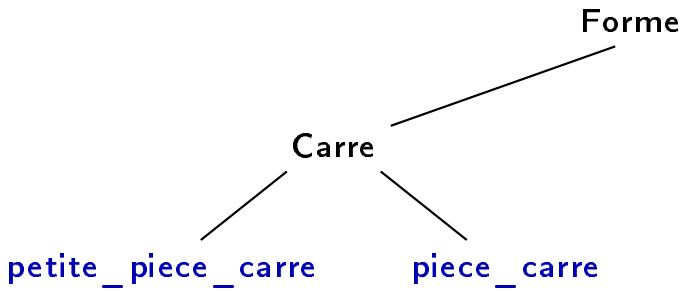
Classe les motifs par rapport à leurs types



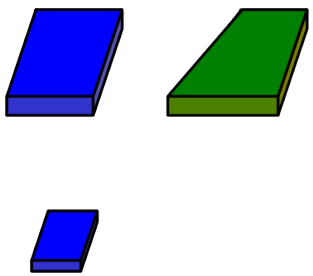


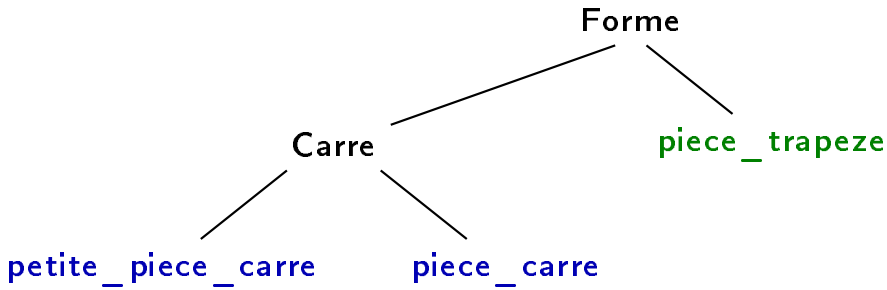
Classe les motifs par rapport à leurs types



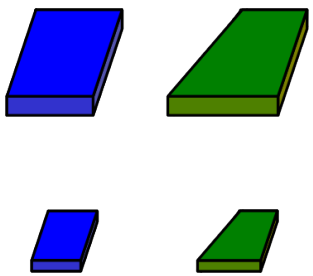


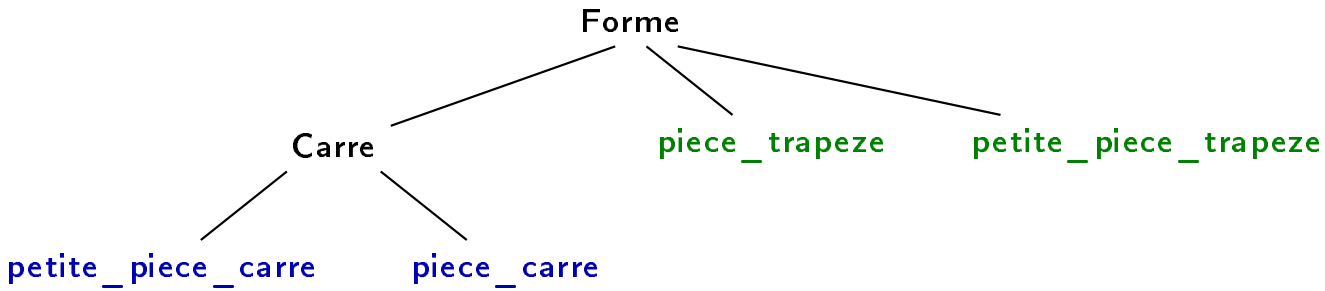
Classe les motifs par rapport à leurs types



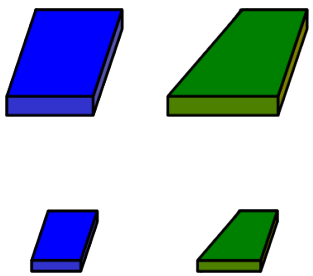


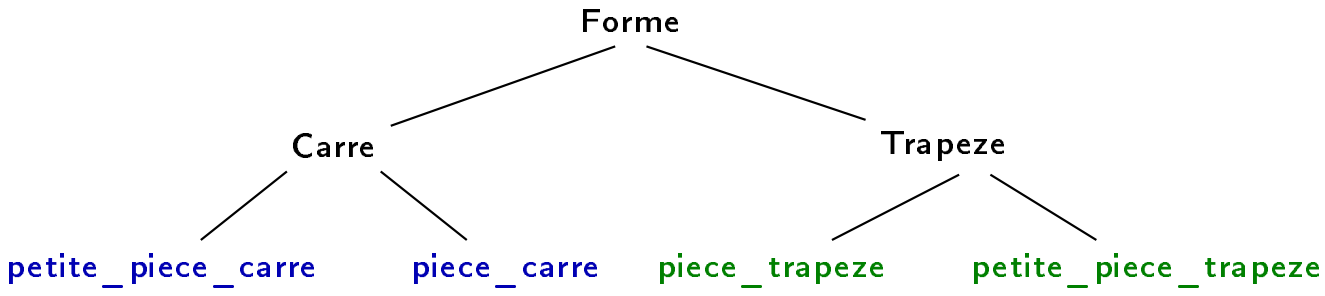
Classe les motifs par rapport à leurs types





Classe les motifs par rapport à leurs types





Une signature algébrique :

...
Piece = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur = ... | bleu() | vert()
Carre = petite_piece_carre() | piece_carre()
Trapeze = petite_piece_trapeze() | piece_trapeze()

Carre < Forme
Trapeze < Forme

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Carre      = petite_piece_carre() | piece_carre()
Trapeze    = petite_piece_trapeze() | piece_trapeze()

Carre < Forme
Trapeze < Forme
```

Terme bien-typé ?

```
piece(piece_carre(), bleu(), un_cote(1))
```

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Carre      = petite_piece_carre() | piece_carre()
Trapeze    = petite_piece_trapeze() | piece_trapeze()

Carre < Forme
Trapeze < Forme
```

Terme bien-typé ?

piece(piece_carre(), bleu(), un_cote(1)) ✔

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Carre      = petite_piece_carre() | piece_carre()
Trapeze    = petite_piece_trapeze() | piece_trapeze()

Carre < Forme
Trapeze < Forme
```

Terme bien-typé ?

piece(piece_carre(), bleu(), un_cote(1)) ✔

piece(bleu(), vert(), un_cote(1))

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Carre      = petite_piece_carre() | piece_carre()
Trapeze    = petite_piece_trapeze() | piece_trapeze()

Carre < Forme
Trapeze < Forme
```

Terme bien-typé ?

piece(piece_carre(), bleu(), un_cote(1)) ✔

piece(bleu(), vert(), un_cote(1)) ✖

- Augmenter l'expressivité de Tom par l'ajout de sous-typage [?]
- Augmenter la sûreté de Tom en proposant un algorithme de vérification et d'inférence de types
- Particularités :
  - inférence de types en présence de sous-typage [?]
  - opérateurs variadiques et associatifs avec élément neutre

- Vue d'ensemble de cette thèse
- Introduction aux caractéristiques principales de Tom
- Typage avec sous-types en Tom
- Implémentation du système de types
- Résultats expérimentaux
- Perspectives

Formalisation et implémentation d'un système de types :

- vérification de types + inférence de types
- équipé de sous-typage
- compatibilité avec Java
- support du filtrage de motifs associatif sur des termes variadiques

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Forme      = petite_piece_carre() | piece_carre() | petite_piece_trapeze() | piece_trapeze()
```

Le filtrage de motifs :

```
public JPiece peindreTrapeze(JPiece jp, JCouleur jc) {
  %match {
    piece(f,c,d) << jp &&
      (piece_trapeze|petite_piece_trapeze)[ ] << f -> { return `piece(f,jc,d); }
  }
  return jp;
}
```

Une signature algébrique :

```
...
Piece      = piece(f: Forme, c: Couleur, d: Dimensions)
Dimensions = ... | un_cote(i: int)
Couleur    = ... | bleu() | vert()
Forme      = petite_piece_carre() | piece_carre() | petite_piece_trapeze() | piece_trapeze()
ListePiece = conc(Piece*)
```

Le filtrage de motifs :

```
public void supprimerDoublons(ListePiece lp) {
  %match {
    conc(C1*,p@piece(f,_,_),C2*,piece(f,_,_),C3*) << lp -> { supprimerDoublons(`conc(p,C1,C2,C3)); }
  }
}
```

Distinction entre listes et fonctions par leurs types :

- les types sont des sortes s décorées avec des symboles de fonctions variadiques v

  conc(piece(piece_carre(),blue(),un_cote(2))) : $ListePiece^{conc}$

Augmentation de la notion de compatibilité de types :

- les sortes sont organisées selon un ordre partiel $<:$
- les types sont organisés selon un ordre partiel $<:_t$

Règles standard avec des sortes décorées :

$$\frac{\Gamma \vdash \mathsf{t}_1 : s_1^? \qquad \ldots \qquad \Gamma \vdash \mathsf{t}_n : s_n^?}{\Gamma(\mathsf{f} : s_1^?, \ldots, s_n^? \to s^?) \vdash \mathsf{f}(\mathsf{t}_1,\ldots,\mathsf{t}_n) : s^?} \text{ T-Fun}$$

$$\frac{\Gamma \vdash e : s_1^h}{\Gamma(s_1^h <:_t s^?) \vdash e : s^?} \text{ Sub}$$

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2) conc(p3,p4)

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2,conc(p3,p4))

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2,conc(p3,p4)) étrange !

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2,conc(p3,p4))

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2,conc(p3,p4))

$$\Downarrow$$

conc(p1,p2,p3,p4)

Nouvelle signature pour des listes :

$$\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListePiece^{conc}$$

conc(p1,p2,conc(p3,p4))

$$\Downarrow$$

conc(p1,p2,p3,p4)

Une règle spéciale pour des listes :

$$\frac{\Gamma \vdash \mathsf{v}(\mathsf{t}_1, \ldots, \mathsf{t}_{n-1}) : s^v \qquad \Gamma \vdash \mathsf{t}_n : s^v}{\Gamma(\mathsf{v} : (s_1^?)^{\star} \rightarrow s^v) \vdash \mathsf{v}(\mathsf{t}_1, \ldots, \mathsf{t}_n) : s^v}\ \text{T-Merge}$$

Des nouveaux types :

- des variables de types α
- un type spécial wt (well-typed)

Des règles pour générer des contraintes de types [?] :

$$\frac{\Gamma \vdash_{ct} \mathsf{t}_1 : \alpha_1 \bullet \mathcal{C}_1 \qquad \ldots \qquad \Gamma \vdash_{ct} \mathsf{t}_n : \alpha_n \bullet \mathcal{C}_n \qquad \mathcal{C} = \{\alpha_0 =_t s^?\} \bigcup_{i=1}^{n} \mathcal{C}_i \cup \{\alpha_i <:_t s_i^?\}}{\Gamma(\mathsf{f} : s_1^?, \ldots, s_n^? \to s^?) \vdash_{ct} \mathsf{f}(\mathsf{t}_1, \ldots, \mathsf{t}_n) : \alpha_0 \bullet \mathcal{C}} \text{ CT-Fun}$$

Une règle spéciale pour des listes :

$$\frac{\Gamma \vdash_{ct} \mathsf{v}(\mathsf{t}_1,\ldots,\mathsf{t}_{n-1}) : \alpha_1 \bullet \mathcal{C}_1 \qquad \Gamma \vdash_{ct} \mathsf{t}_n : \alpha_1 \bullet \mathcal{C}_2 \qquad \mathcal{C} = \{\alpha_1 =_t s^v\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}{\Gamma(\mathsf{v} : (s_1^?)^\star \to s^v) \vdash_{ct} \mathsf{v}(\mathsf{t}_1,\ldots,\mathsf{t}_n) : \alpha_1 \bullet \mathcal{C}} \text{ CT-Merge}$$

si $\mathsf{typeof}(\Gamma,\mathsf{t}_n) = s^v$ ou $\mathsf{t}_n = \mathsf{x}*$

Ex. : $\mathsf{v}(1, 2,\mathsf{v}(3))$ ou $\mathsf{v}(1, 2,\mathsf{x}*)$

Dérivation de bas en haut et génération de contraintes :

- contraintes d'égalité entre types
- contraintes de sous-typage entre types

Résolution de contraintes :

- unification de contraintes d'égalité [?]
- simplification de contraintes de sous-typage [?]
- génération d'une solution

$$\Gamma = (\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListPiece^{conc}, \mathsf{x*} : \alpha_1, \mathsf{y} : \alpha_2, \ldots, Trapeze^{?} <:_t Forme^{?})$$

$$\dfrac{\mathcal{C}_3 = \{\alpha_3 =_t ListPiece^{conc}\}}{\Gamma \vdash_{ct} \mathsf{conc()} : \alpha_3 \bullet \mathcal{C}_3}\ \text{CT-Empty}$$

$$\dfrac{\mathcal{C}_4 = \{\alpha_1 =_t \alpha_3\}}{\Gamma \vdash_{ct} \mathsf{x*} : \alpha_3 \bullet \mathcal{C}_4}\ \text{CT-SVar}$$

$$\dfrac{\mathcal{C}_1 = \{\alpha_3 =_t ListPiece^{conc}\} \cup \mathcal{C}_3 \cup \mathcal{C}_4}{\Gamma \vdash_{ct} \mathsf{conc(x*)} : \alpha_3 \bullet \mathcal{C}_1}\ \text{CT-Merge} \qquad \dfrac{\mathcal{C}_2 = \{\alpha_4 =_t \alpha_2\}}{\Gamma \vdash_{ct} \mathsf{y}\text{:}\ \alpha_4 \bullet \mathcal{C}_2}\ \text{CT-Var}$$

$$\dfrac{\mathcal{C} = \{\alpha_3 =_t ListPiece^{conc}, \alpha_4 <:_t Piece^{?}\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}{\Gamma \vdash_{ct} \mathsf{conc(x*,y)} : \alpha_3 \bullet \mathcal{C}}\ \text{CT-Elem}$$

$$\Gamma = (\mathsf{conc} : (Piece^{?})^{\star} \rightarrow ListPiece^{conc}, \mathsf{x*} : \alpha_1, \mathsf{y} : \alpha_2, \ldots, Trapeze^{?} <:_t Forme^{?})$$

$$\dfrac{\mathcal{C}_3 = \{\alpha_3 =_t ListPiece^{conc}\}}{\Gamma \vdash_{ct} \mathsf{conc()} : \alpha_3 \bullet \mathcal{C}_3}\ \text{CT-Empty}$$

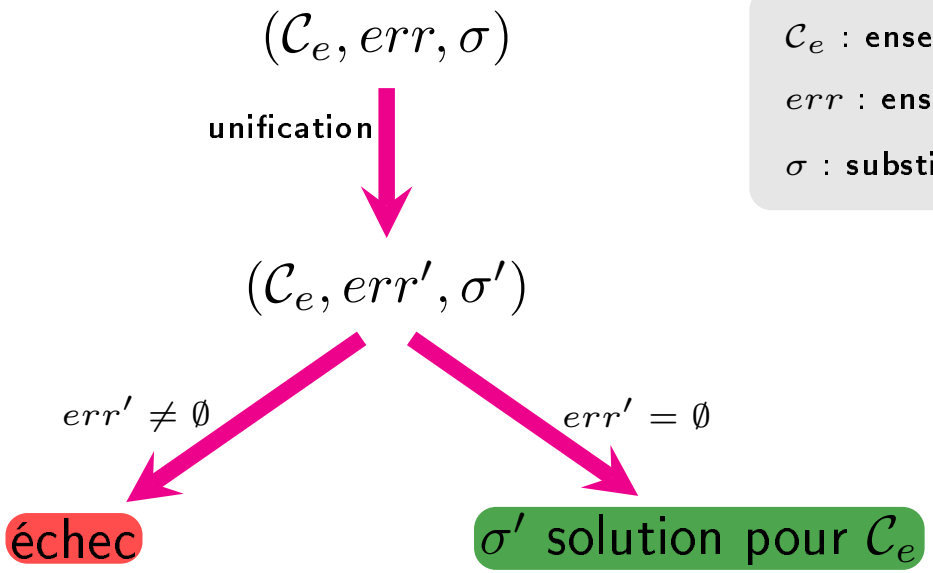
$$\dfrac{\mathcal{C}_4 = \{\alpha_1 =_t \alpha_3\}}{\Gamma \vdash_{ct} \mathsf{x*} : \alpha_3 \bullet \mathcal{C}_4}\ \text{CT-SVar}$$

$$\dfrac{\mathcal{C}_1 = \{\alpha_3 =_t ListPiece^{conc}\} \cup \mathcal{C}_3 \cup \mathcal{C}_4}{\Gamma \vdash_{ct} \mathsf{conc(x*)} : \alpha_3 \bullet \mathcal{C}_1}\ \text{CT-Merge} \qquad \dfrac{\mathcal{C}_2 = \{\alpha_4 =_t \alpha_2\}}{\Gamma \vdash_{ct} \mathsf{y}\text{:}\ \alpha_4 \bullet \mathcal{C}_2}\ \text{CT-Var}$$

$$\dfrac{\mathcal{C} = \{\alpha_3 =_t ListPiece^{conc}, \alpha_4 <:_t Piece^{?}\} \cup \mathcal{C}_1 \cup \mathcal{C}_2}{\Gamma \vdash_{ct} \mathsf{conc(x*,y)} : \alpha_3 \bullet \mathcal{C}}\ \text{CT-Elem}$$

$$\mathcal{C} = \{\alpha_3 =_t ListPiece^{conc}, \alpha_4 <:_t Piece^{?}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}$$

Résolution de contraintes d'égalité

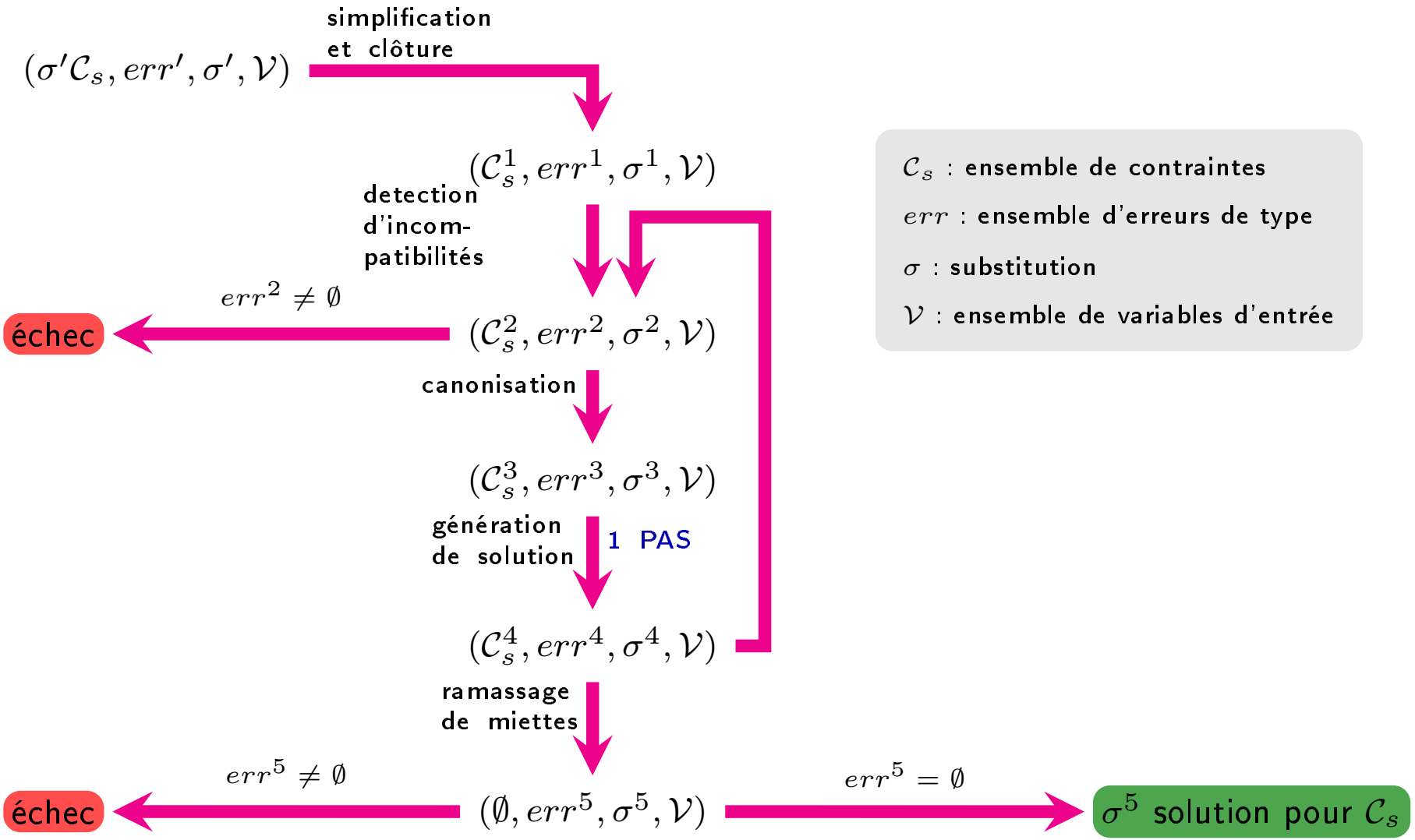


$\mathcal{C}_e$: ensemble de contraintes

err : ensemble d'erreurs de type

σ : substitution

Résolution de contraintes de sous-typage



$\mathcal{C}_s$: ensemble de contraintes

err : ensemble d'erreurs de type

σ : substitution

$\mathcal{V}$: ensemble de variables d'entrée

Considérons $\mathcal{V} = \{\alpha_1, \alpha_2\}$ et l'ensemble généré

$$\mathcal{C} = \overbrace{\{\alpha_3 =_t ListPiece^{conc}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}}^{\mathcal{C}_e} \cup \overbrace{\{\alpha_4 <:_t Piece^{?}\}}^{\mathcal{C}_s}$$

σ	$\mathcal{C}_e$
$\{\}$	$\{\alpha_3 =_t ListPiece^{conc}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}$
$\Downarrow$	
$\{\alpha_3 \mapsto ListPiece^{conc}\}$	$\{\alpha_3 =_t ListPiece^{conc}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}$
$\Downarrow$	
$\{\alpha_1 \mapsto ListPiece^{conc}, \alpha_3 \mapsto ListPiece^{conc}\}$	$\{\alpha_3 =_t ListPiece^{conc}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}$
$\Downarrow$	
$\{\alpha_4 \mapsto \alpha_2, \alpha_1 \mapsto ListPiece^{conc}, \alpha_3 \mapsto ListPiece^{conc}\}$	$\{\alpha_3 =_t ListPiece^{conc}, \alpha_1 =_t \alpha_3, \alpha_4 =_t \alpha_2\}$

σ	$\sigma\mathcal{C}_s$
$\{\alpha_4 \mapsto \alpha_2, \alpha_1 \mapsto ListPiece^{conc}, \alpha_3 \mapsto ListPiece^{conc}\}$	$\{\alpha_2 <:_t Piece^{?}\}$
$\Downarrow$	
$\{\alpha_2 \mapsto Piece^{?}, \alpha_4 \mapsto Piece^{?}, \alpha_1 \mapsto ListPiece^{conc}, \alpha_3 \mapsto ListPiece^{conc}\}$	$\{\}$

Règles standard avec du sous-typage :

$$\frac{\mathsf{x} \in \mathcal{V} \qquad \tau =_t \mathsf{typeof}([\,],\mathsf{x}) \qquad \mathsf{typeof}([\,],\dot{u}) <:_t \tau}{\rho \Vdash \mathsf{x} \ll |\tau|\ \dot{u} \Downarrow (\{\rho \lhd \{\mathsf{x} \mapsto \dot{u}\}\}, \mathsf{accept})}\ \text{E-MSubVarAcc}$$

Règles spéciales pour des listes :

$$\frac{\mathsf{x} \in \mathcal{V} \qquad \tau =_t \mathsf{typeof}([\,],\mathsf{x}) \qquad \tau =_t \mathsf{typeof}([\,],\dot{u}) \qquad symb(\dot{u}) \in \mathcal{F}^\star}{\rho \Vdash \mathsf{x}^* \ll |\tau|\ \dot{u} \Downarrow (\{\rho \lhd \{\mathsf{x} \mapsto \dot{u}\}\}, \mathsf{accept})}\ \text{E-MSVarAcc}$$

Théorème (Progrès)

Étant donnée une expression close e bien-typée : soit e est une valeur soit e fait un pas de réduction relatif aux règles d'évaluation.

Théorème (Préservation)

Si une expression e bien-typée avec type τ fait un pas d'évaluation vers e', alors e' est aussi bien-typée avec τ.

Théorème (Correction)

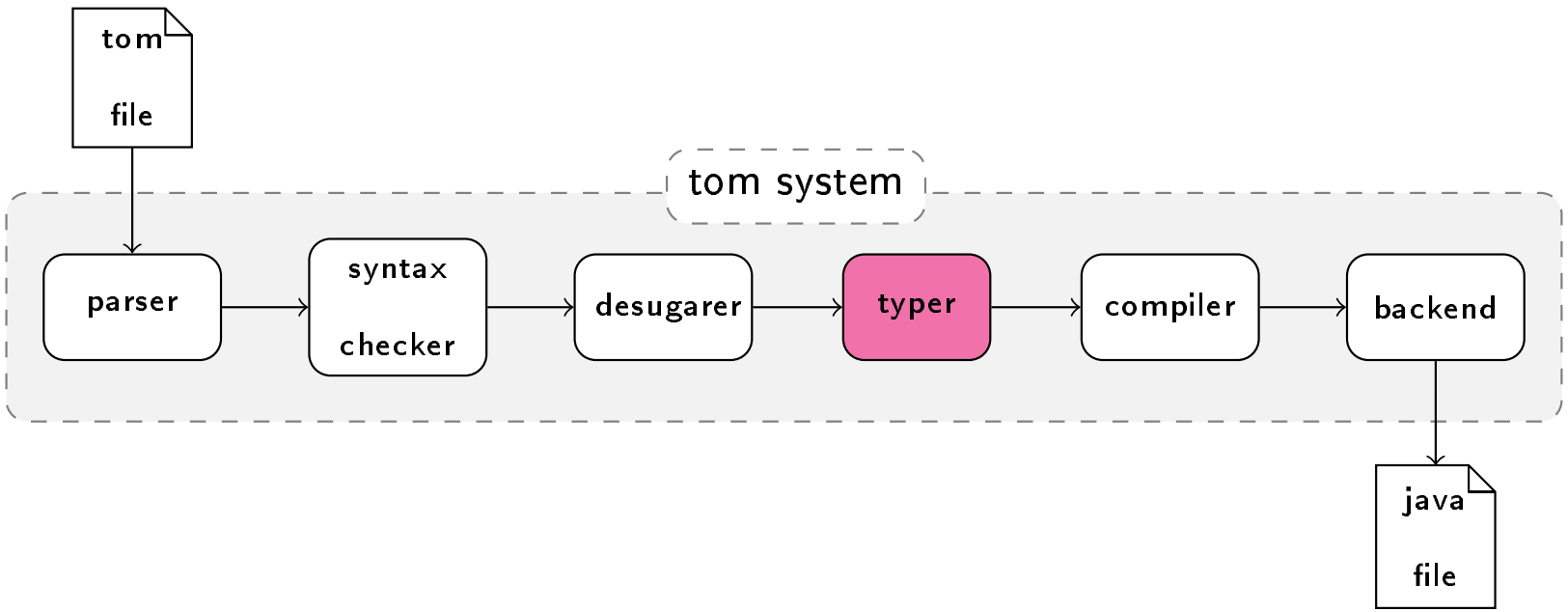
Tout jugement de typage dérivé par les règles d'inférence de types peut aussi être dérivé par les règles de vérification de types.

Théorème (Complétude)

Une solution validée par les règles de vérification de types peut être étendue à une solution proposée par les règles d'inférence de types.

Théorème (Terminaison)

L'algorithme de résolution de contraintes termine toujours soit par un échec soit par un succès avec la génération d'une solution pour l'ensemble initiale de contraintes.


tom
file
tom system
parser
syntax
checker
desugarer
typer
compiler
backend
java
file

Réorganisation des modules de Tom :

- séparation entre simplification de l'AST et l'inférence de types

Extension de la signature algébrique de Tom :

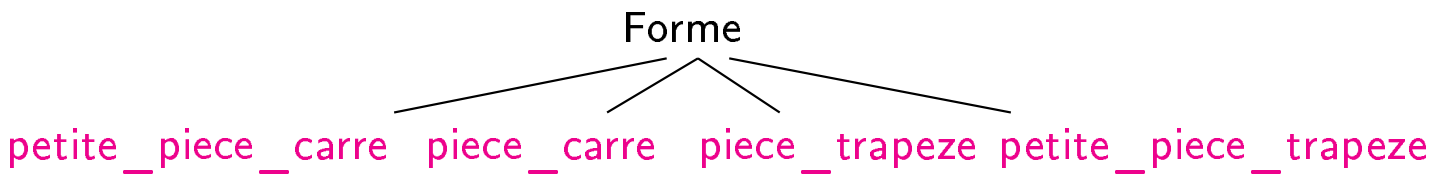
- représentation des types dans l'AST
- représentation des contraintes

Ajout de déclaration des relations de sous-typage

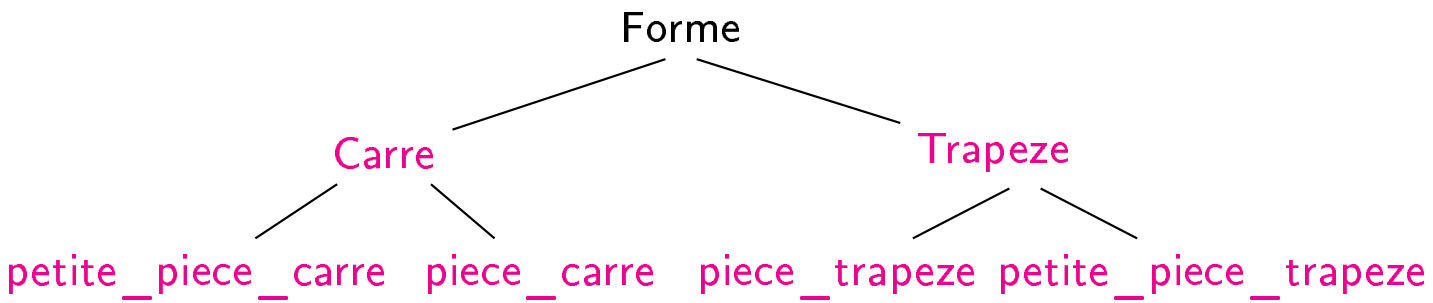
%typeterm sous-type **extends** type {...}

Implémentation et documentation des algorithmes de résolution de contraintes :

- unification par des règles de réécriture avec substitution close par transitivité
- simplification et génération d'une solution par une combinaison de règles de réécriture


Forme
petite_piece_carre
piece_carre
piece_trapeze
petite_piece_trapeze

Exemple - hiérarchie Java



```
...

%typeterm Forme {
  implement      { JForme }
}
```

```
...

%op Forme piece_carre() {
  is_fsym(s)  { (s instanceof Jpiece_carre) }
  make()      { new Jpiece_carre() }
}

%op Forme petite_piece_trapeze() {
  is_fsym(s)  { (s instanceof Jpetite_piece_trapeze) }
  make()      { new Jpetite_piece_trapeze() }
}

...
```

```
...

%typeterm Forme {
  implement      { JForme }
}

%typeterm Carre extends Forme {
  implement      { JCarre }
}

%typeterm Trapeze extends Forme {
  implement      { JTrapeze }
}

...

%op Carre piece_carre() {
  is_fsym(s)  { (s instanceof Jpiece_carre) }
  make()      { new Jpiece_carre() }
}

%op Trapeze petite_piece_trapeze() {
  is_fsym(s)  { (s instanceof Jpetite_piece_trapeze) }
  make()      { new Jpetite_piece_trapeze() }
}

...
```

```
public JPiece peindreTrapeze(JPiece jp, JCouleur jc) {
  %match {
    piece(f,c,d) << jp &&
      (piece_trapeze|petite_piece_trapeze)[ ] << f -> { return `piece(f,jc,d); }
  }
  return jp;
}
```

```
public JPiece peindreTrapeze(JPiece jp, JCouleur jc) {
  %match {
    piece(f,c,d) << jp &&
      x << Trapeze f -> { return `piece(f,jc,d); }
  }
  return jp;
}
```

Bootstrap de Tom (86 fichiers) :

	Typage	Compilation	#Contraintes	#Variables de type
version 2.9	59s	5min 40s	0	0
version 2.9 -nt	7min 55s	12min 40s	8,461,954	195,692

Augmentation du temps de typage et de compilation dûe au :

- nombre de règles $cond \rightarrow action$
- nombre de conditions de chaque *cond*
- nombre de termes algébriques dans *action*

- Développement d'un système de types avec :
  - sous-typage pour le support du filtrage de motifs associatif sur des termes algébriques variadiques
  - vérification de types et sous-types
  - inférence de types basée sur de contraintes
  - résolution des contraintes par unification et simplification
- Formalisation de la sémantique opérationelle de Tom
- Preuve de propriétés de sûreté, correction et complétude du système de types et terminaison de l'algorithme de résolution de contraintes
- Implémentation, documentation et analyse des résultats

- Optimisation de l'implémentation du typeur
- Support au sous-typage dans le générateur automatique d'ancrages
- Description d'une nouvelle approche de ramassage de miettes
- Preuve de propriétés de fiabilité du typage
- Extensions du langage Tom :
  - avec des motifs typés « x : t »
  - avec des types polymorphes

Questions ?