DYNAMO

Sergio Yovine

Verimag

http://www-verimag.imag.fr/~iosif/projects/ACI/aci-dynamo.html

Partenaires

LIAFA

Peter HABERMEHL
Ahmed BOUAJJANI
Agathe MERCERON
Antoine MEYER
Mihaela SIGHIREANU
Pierre MORO

VERIMAG

Sergio YOVINE
Radu IOSIF
Marius BOZGA
Yassine LAKHNECH
Chaker NAKHLI
Moussa AMRANI
Guillaume SALAGNAC
Karim LITIM

Motivation

Faire face à la complexité croissante des logiciels embarqués temps-réel

Structures de données dynamiques
- listes, arbres, ...

Gestion dynamique de la mémoire
- GC contraint : taille limitée, temps prédictible, ...

Structures de contrôle dynamique
- création dynamique de threads
- récursion

Problématique

Vérification

- Mémoire : absence de fuites de mémoire (memory leaks), de références "dangling"
- Contrôle : absence de "deadlocks", ...

Optimisation

- Mémoire : gestion controlée à la compilation (e.g., par régions, ...)
- Contrôle : ordonnancement, thread pools, ...

Certification

- Invariants quantitatifs : taille maximum de mémoire, nombre de threads, ...

Approche

Définition et étude de différents langages, modèles et logiques

Etude des problèmes de décidabilité

Développement d'outils d'analyse et d'optimisation

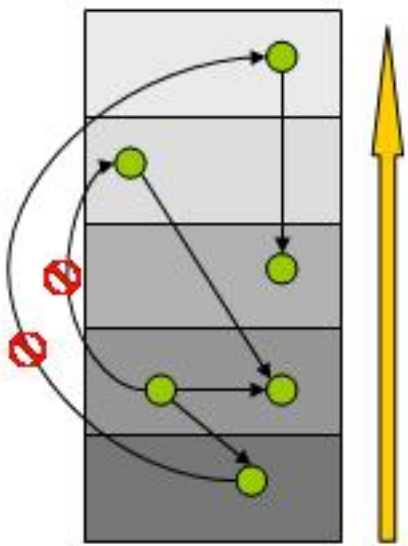
Real-Time Specification for Java (RTSJ)

Modèle de programmation

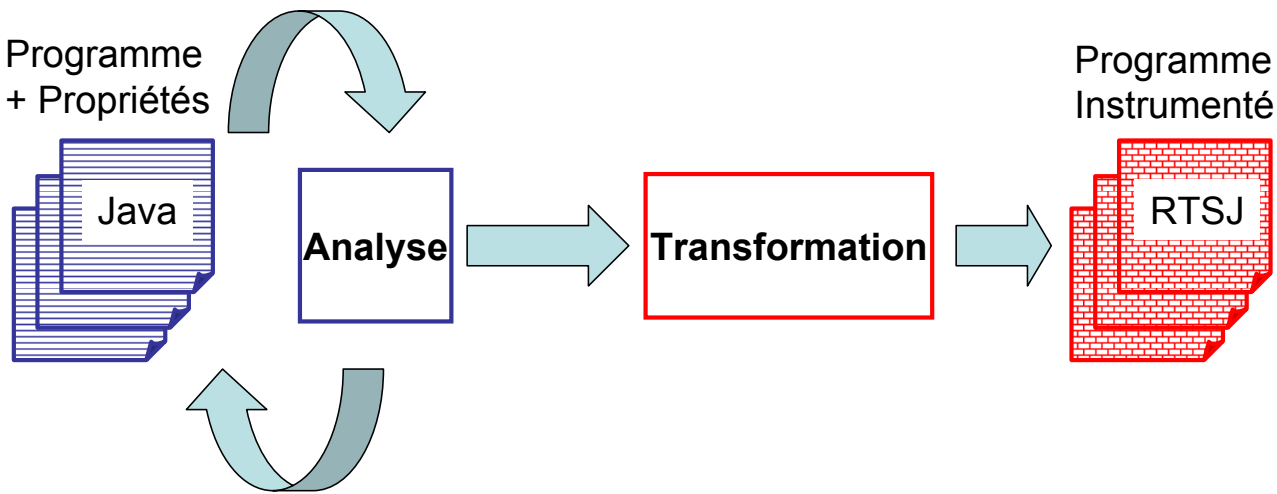
– Allocation par régions
– Règles de portée
– Multithreading, event handlers

Inconvénients

– Erreurs de programmation
– Vérification des règles de portée à l'exécution
– Taille des régions
– Analyse comportementale difficile



Cycle de développement



Approche

Définition et étude de différents langages, modèles et logiques

Etude des problèmes de décidabilité

Développement d'outils d'analyse et d'optimisation

Spécification de propriétés sur la mémoire

Logique d'Alias avec compteurs

$$
\begin{aligned}
V &= \text{alphabet de variables du programme; } u, v \in V \\
\Sigma &= \text{alphabet de selecteurs de champs; } \sigma, \tau \in \Sigma \\
X &= \text{alphabet de variables logiques; } x, y \in X \\
t &:= x \mid n \in \mathbb{N} \mid t_1 + t_2 \\
w &:= u(\sigma^t)^* \\
\varphi &:= t_1 = t_2 \mid w_1 \leq w_2 \mid w_1 \Diamond w_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \forall x\ \varphi(x)
\end{aligned}
$$

Exemples de propriétés

– Il existe un cycle σ autour de u :

$$CYCLE(u, \sigma) = \exists x\ u\sigma^x \Diamond u$$

– u et v partagent un élément :

$$SHARE(u, v, \sigma) = \exists x \exists y\ u\sigma^x \Diamond\ v\sigma^y$$

Vérification de propriétés

Décidabilité

- Graphes, DAG : SAT undécidable, MC décidable
- Arbres : SAT décidable
  automates à compteurs avec relation de transition affine
- Listes : SAT décidable
  model checking paramétré + arithmetique

Observations

1. Il est nécessaire d'étendre la logique pour raisonner sur des arbres non-bornés.
2. La décidabilité sur les arbres ne suffit pas pour vérifier des programmes sur des arbres, car la structure d'arbre peut ne pas être préservée lors d'un calcul intermédiaire.

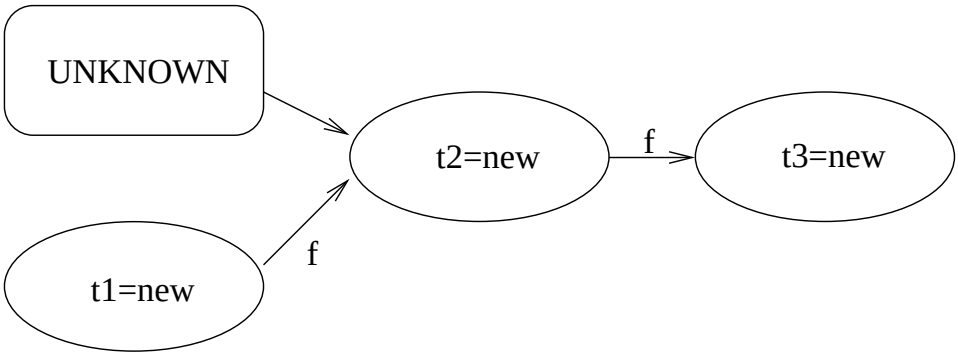
Vérification de propriétés

- Représentation des configurations par ensembles réguliers donnés par automates.
- Application des techniques du Regular-model checking pour programmes avec listes.
- Extension des automates pour tenir compte des aspects non-réguliers (contraintes de comptage) [en cours]

Analyse de programmes

Analyse de "points-to" et d'échappement

- Construit un graphe de références abstrait
- Détermine l'accessibilité d'un site d'allocation depuis un point du programme



(b) mrefs graph

Analyse de programmes - Résultats

~25% plus précis et aussi rapide que Gay et Steensgaard

~10x plus rapide que Spark (McGill) et Flex (MIT), mais moins précis

~50% des allocations en régions

Program	Lines	Allocation sites	Analysis time			INSIDE		G&S's analysis
			escape	side	total	variables	sites	*stackable* variables
bh	1128	41	9.430	23.51	32.481	34	21	23
bisort	340	10	7.876	11.509	19.385	7	7	7
em3d	462	26	8.551	15.706	24.257	13	11	11
health	562	28	8.454	19.414	27.868	18	13	10
mst	473	16	8.106	14.260	22.366	8	8	7
perimeter	745	13	11.357	23.944	35.301	7	7	7
power	765	21	3.628	1.159	4.787	9	9	5

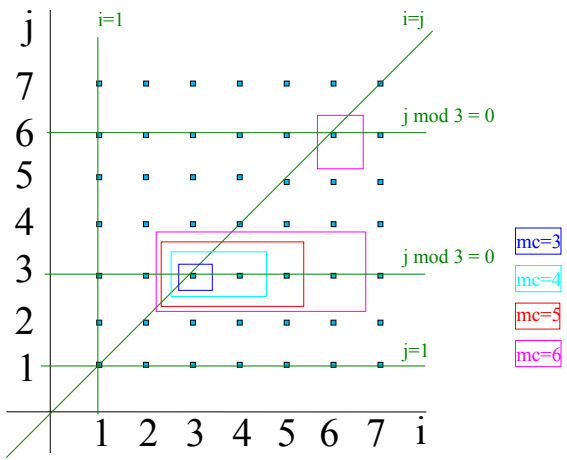
Analyse de consommation de mémoire

- Problème indécidable
- Approche "classique" : Vérification par approximations linéaires (formules de Presburger) via des annotations
- Approche DYNAMO : Synthèse des expressions non-linéaires sur des variables du programme
  - Calcul d'invariants linéaires

    $\{1 \leq j \leq i \leq mc, j \bmod 3 = 0\}$

  - Mémoire consommée = Nombre des points entiers dans l'invariant = Polinôme d'Ehrhart

    $\frac{1}{6}mc^2 - \frac{1}{6}mc + per(mc, [0, 0, -\frac{1}{3}])$



Consommation de mémoire - Résultats

Example:Class.Method	$\#CS_m$	memAlloc	Param.	#Objs	Estimation	Err%
mst:						
mst:MST.computeMST(g, nv)	1	$nv - 1$	10	9	9	0,00
			20	19	19	0,00
			100	99	99	0,00
			1000	999	999	0,00
bh:						
Tree.createTestData(nb)	23	$17nb + 26$	10	196	196	0,00
			20	366	366	0,00
			100	1726	1726	0,00
			1000	17026	17026	0,00
bisort: (recursive)						
Value.createTree(size,sd)	1	$size - 1$	10	7	9	22,22
			20	15	19	21,1
			200	127	199	36,2
			64	63	63	0,0
			128	127	127	0,0
			256	255	255	0,0
power: (recursive)						
Root.<init>	14	32622	-	32412	32622	0,64
em3d:						
Bigraph.create(nN, nD)	32	$6nD \cdot nN + 4nN + 8$	(10, 5)	348	348	0,00
			(20, 6)	808	808	0,00
			(100, 7)	4608	4608	0,00
			(1000, 8)	52008	52008	0,00

Structures de contrôle dynamique

Problèmes abordés

- Programmes avec procédures (récursives) ET
- Programmes multithreads
  - Appels parallèles de procédures :
    Le processus attend que les procédures appelées terminent (e.g., Java RMI).
  - Création dynamique de procédures avec "spawn" :
    Le processus continue à s'exécuter avec les procédures appelées (e.g., Java threads, RTSJ event handling).

Vérification de propriétés

Modèles et approche

- Automates à pile, Réseaux dynamiques d'automates à pile
- Représentation des configurations atteignables par automates d'arbre
- En général MC est indécidable :
  - Sur-approximations pour vérifier
  - Sous-approximations pour trouver des fautes

Analyse exacte de réseaux dynamiques d'automates à pile

- Modélisation d'appels de type "spawn"
- Ceci n'est pas possible avec d'autres modèles
- $post^*$ est effectivement calculable en temps polynomial

Perspectives dernière année

- Regular-model checking pour les programmes avec arbres et structures de mémoire plus générales.
- Vérification de programmes avec procédures récursives et mémoire dynamique (e.g., renversement récursif d'une liste).
- Extension de la logique d'alias pour analyser des programmes avec des operations destructives sur les arbres.
- Etude d'autres méthodes de vérification pour la logique d'alias.
- Extension et amélioration de l'analyse de "points-to" et d'échappement.
- Analyse de la consommation de mémoire d'une application industrielle.

Relations académiques et industrielles

- Projet Averiles (RNTL) :
  Verimag, Liafa, LSV, EDF, CRIL .
- Projet AnVerS (IST-FET en cours d'évaluation) :
  CFV, LIAFA, EPFL, Verimag, Stuttgart, Uppsala, Max Planck, München, Tel Aviv, Absint, FT R&D.
- Projet PIRTERAS (FP6 STREP soumis) :
  AICAS, Thalès, Verocel, Chalmers, FZI, Astrium, Verimag.
- IBM Eclipse Grant : Universidad de Buenos Aires, VERIMAG.
- Projet MADEJA (Région Rhône-Alpes) :
  Verimag, Groupe Silicomp.
- Analyse d'une application fournie par Thalès Aerospace Division (NDA en cours).