

Développement et analyses de systèmes embarqués

Séminaire LORIA le 27 février 2009

Laure Gonnord

<http://laure.gonnord.org/pro/>

Verimag/CITI/LIP
Grenoble/Villeurbanne/Lyon, France



Systèmes embarqués

Plusieurs types de besoins :

- Programmes critiques : **vérification** du logiciel avant déploiement, propriétés de **sûreté, compilation**.
 - Programmes « multimédia » : développement rapide, garanties de **Qualité de Service** (QoS) à l'exécution.
- ▶ ou une combinaison !

Contenu de l'exposé

- **Postdoc** : Maintenance des ressources à l'exécution.
- **Thèse** : Génération d'invariants pour la vérification.
- ▶ Spécification, développement, génération de code, . . .

- 1 Conception de systèmes embarqués
 - Architecture logicielle pour la QoS
 - Contributions
 - Applications

- 2 Génération d'invariants numériques
 - Analyse des Relations Linéaires
 - Contributions de ma thèse
 - Applications

Développement d'applications « resource-aware »

Le développeur a besoin d'outils pour :

- ajouter/retirer des fonctionnalités aisément (compilation/exécution) ;
- adapter les fonctionnalités à la ressource physique (modes dégradés) ;
- évaluer la performance (QoS).



Nos composants

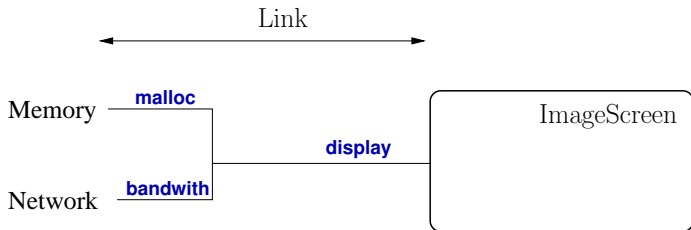
Caractéristiques

- Un composant fournit des **services** (fonctions, paramètres). Un service peut avoir des **requis** (services d'un autre composant).
- On distingue **type** de composant (mémoire,cpu,screen) et **instance** de composant.
- Notion de **liaison** entre les composants (services requis/fournis). Pas de mémoire partagée.

Composants, concepts pour la QoS (qualité de service)

Ressources variables :

- différentes implémentations des services selon la « qualité de service » (**niveau d'implémentation**)
- adaptation de ces implémentations par **contrat de liaison** (contrat = lien entre des parties).



Objectif

On veut **contractualiser des propriétés** des services :

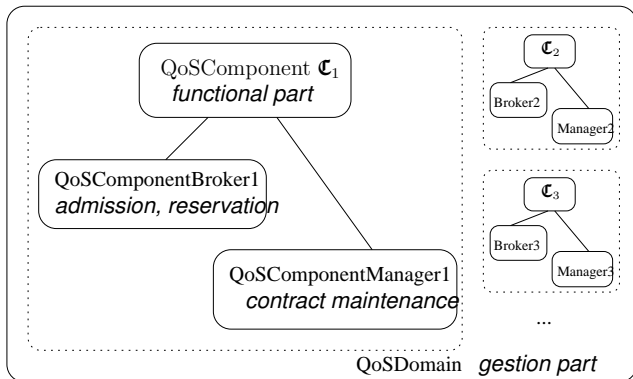
- Exprimer.
- Garantir.
- Négocier.

Les propriétés de qualité de service (QoS) portent sur les composants et les services, parlent de ressources (pas de délai pour le moment).

► Une **architecture** pour la gestion de la qualité de service (Qinna). [Babau/Tournier 2005]

Qinna en Bref - Architecture Logicielle

Une architecture logicielle dédiée à la gestion des problèmes de **Qualité de Service**. Nouveaux composants :



- 1 Conception de systèmes embarqués
 - Architecture logicielle pour la QoS
 - **Contributions**
 - Applications

- 2 Génération d'invariants numériques
 - Analyse des Relations Linéaires
 - Contributions de ma thèse
 - Applications

Travaux réalisés

- Formalisation du cadre existant, et extensions.
- Implémentation (C++).
- Génération automatique de contrôleurs de QoS à partir d'une logique à événements.
- Mise au point d'une méthode d'implémentation avec Qinna.
- "Proof of concept", développement d'une étude de cas.
- Livrable logiciel ANR.

Deux classes de propriétés

Nous avons identifié deux classes de propriétés (de QoS) :

- des propriétés sur **les services et les composants** (QoS Behavioural Constraints) : « la mémoire totale pour l'application est 4Mo », « le service `imagedisplay` est monutilisateur », « ce composant ne peut être alloué que 3 fois », ...
- des propriétés sur les **liaisons** (QoS Linking Constraints) : « pour que le service `imagedisplay` soit rendu à qualité maximale, il faut que le réseau soit rapide et que le taux de compression soit faible »...

Formules sur les **paramètres** d'appels des services, sur la suite des **occurrences** des services,

Contraintes de comportement de ressources (QBC)

- CTC : **contrainte de type** (nb total d'instances, ressource totale pour tous les services du composant, ...) :

$$CTC(C) \stackrel{def}{=} \bigwedge_i CTC_{serv}(s_i) \wedge CTC_{compo}(C)$$

- CIC : **contrainte d'instance** (idem sur l'instance) :

$$CIC(C^j) \stackrel{def}{=} \bigwedge_i CIC_{serv}(s_i^j) \wedge CIC_{instance}(C^j)$$

(C : composant qui fournit le service s , C^j et s^j instances.)

Mise en œuvre des contraintes de ressource dans Qinna++

Pour le (type) `QoSComponent S`, on crée un `QoSBrokerS`

- demandes d'initialisation/**allocation**.
- demandes de **service** à qualité donnée
- ▶ garanties des contraintes de ressource.

Génération automatique à partir d'une formule *MEDL* [Lee 99]

Exemple : $\sum_i value_1(occ(malloc, i)) \leq 1024$

Mise en œuvre des contraintes de liaison dans Qinna++

Utilisation du composant $QoSManagerS$ (pour S) .

- Expression : une **table de liaison** lie les niveaux d'implémentation des services et de leurs requis.
- Lors de la réservation d'un service à une certaine qualité objectif, le Manager utilise les infos de la table pour établir (éventuellement) un **contrat**.

Réservation d'un service à un certain niveau de QoS/d'implémentation

Mise en place du **contrat de liaison** pour le service `Screen.displayimage()` (l'image existe) d'une l'instance à une QoS objectif .

- Le Broker est déjà initialisé (CTC), et une instance créée (CIC en partie). Si non CIC(s), échec.
 - On demande le service avec le niveau de QoS == TBon (ie implLevel=1). Le Manager dit qu'il faut *20ko* de mémoire : le service `Memory.malloc(20)` est demandé : si CIC et CTC ok, mémoire allouée.
 - Contrat == `<display,1,malloc,20>`
- Réalisé **automatiquement** par les Managers de Qinna++.

Dégradation en cas d'échec

Deux notions :

- Les **niveaux d'implémentation** d'un service donné (ordonnés) : un contrat peut comporter un intervalle de niveaux d'implémentation objectifs acceptable.
 - Les appels de services sont classés par **niveau d'importance**.
- ▶ Si la négociation échoue, on dégrade la QoS objectif (ie l'implémentation) d'un service de moindre importance.

1 Conception de systèmes embarqués

- Architecture logicielle pour la QoS
- Contributions
- **Applications**

2 Génération d'invariants numériques

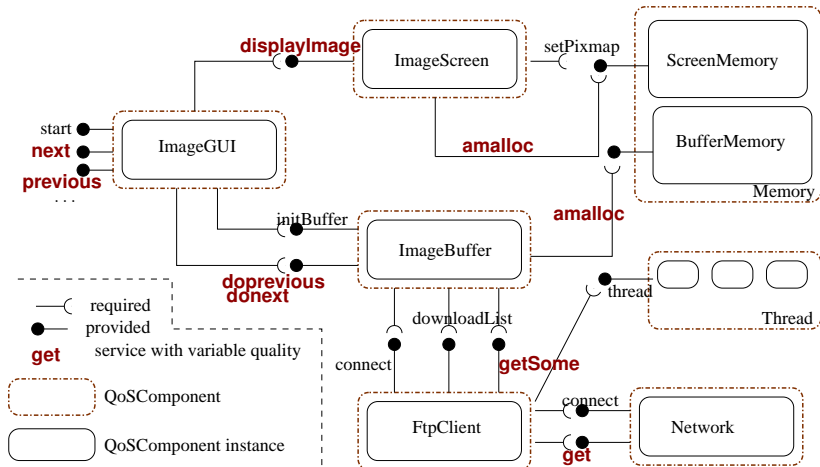
- Analyse des Relations Linéaires
- Contributions de ma thèse
- Applications

Case study : description

- Contexte : Projet ANR REVE
- Téléchargement et visualisation d'images distantes par ftp.
- Adaptation du téléchargement et de l'affichage selon :
 - la mémoire disponible
 - la bande passante
- Évaluation de différentes politiques.

Case study : qinna-isation - 1

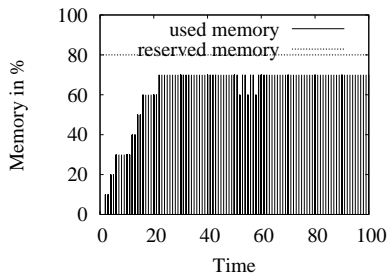
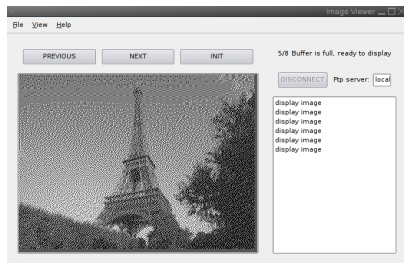
Étape 1 : Identifier les services variables.



Case study : qinna-isation - 2

- Étape 2 : Création des composants Qinna.
- Étape 3 : Codage des contraintes de liaison.
- Étape 4 : Codage des contraintes de ressources.
- Étape 5 : Initialialisation et c'est tout !

Case study : évaluation



Conclusion

Avantages de Qinna :

- ☺ Séparation des préoccupations (composants).
- ☺ Liens explicites entre les (QoS des) services (tables).
- ☺ Algorithmes de négociation et composants génériques.
- ☺ Distinctions entre classe/instance/service.
- ☺ Distinction entre contraintes numériques et contraintes de liaison.
- ☺ Mise en pratique simple.

Conclusion

Avantages de Qinna :

- ☺ Séparation des préoccupations (composants).
- ☺ Liens explicites entre les (QoS des) services (tables).
- ☺ Algorithmes de négociation et composants génériques.
- ☺ Distinctions entre classe/instance/service.
- ☺ Distinction entre contraintes numériques et contraintes de liaison.
- ☺ Mise en pratique simple.

Mais

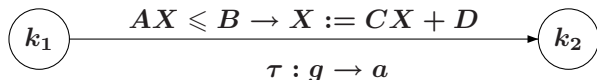
- ☹ Pas de découverte du « meilleur » vecteur de niveaux d'implémentation (synthèse de contrôleurs ?).
- ☹ Surcoût trop élevé (V1).

- 1 Conception de systèmes embarqués
 - Architecture logicielle pour la QoS
 - Contributions
 - Applications

- 2 Génération d'invariants numériques
 - **Analyse des Relations Linéaires**
 - Contributions de ma thèse
 - Applications

Modèle - Notations

Vérification de propriétés **numériques** sur des GFC avec conditions et actions **affines** :

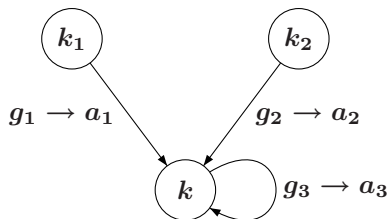


ou encore « automates interprétés », automates « à compteurs ».

- A, C matrices, B, D vecteurs.
- Sémantique « naturelle ».
- On veut des invariants pour **chaque point de contrôle**.

Formalisation du problème

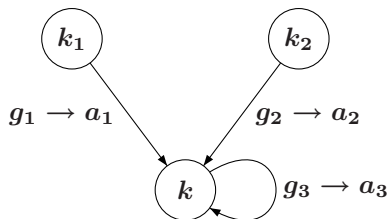
\mathcal{A}_k = ensemble des **valuations** au point k :



$$\mathcal{A}_k = a_1(\mathcal{A}_{k_1} \cap g_1) \cup a_2(\mathcal{A}_{k_2} \cap g_2) \cup a_3(\mathcal{A}_k \cap g_3)$$

Formalisation du problème

\mathcal{A}_k = ensemble des **valuations** au point k :



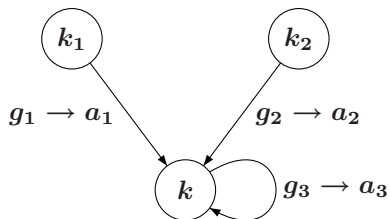
$$\mathcal{A}_k = a_1(\mathcal{A}_{k_1} \cap g_1) \cup a_2(\mathcal{A}_{k_2} \cap g_2) \cup a_3(\mathcal{A}_k \cap g_3)$$

► Système d'équations $\mathcal{A}_k = F(\mathcal{A}_k)$, **point fixe**.

- Représentation des valuations, calcul
- Convergence de la résolution.

Formalisation du problème

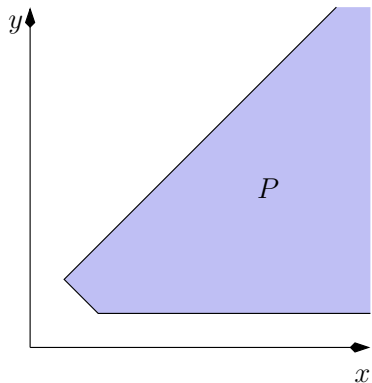
\mathcal{A}_k = ensemble des **valuations** au point k :



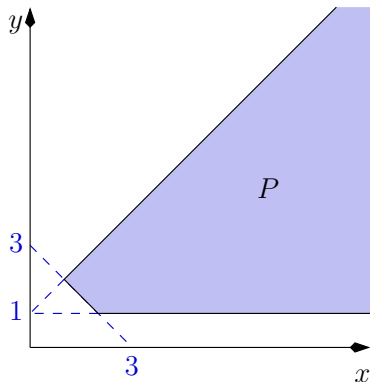
$$\mathcal{A}_k = a_1(\mathcal{A}_{k_1} \cap g_1) \cup a_2(\mathcal{A}_{k_2} \cap g_2) \cup a_3(\mathcal{A}_k \cap g_3)$$

- ▶ Système d'équations $\mathcal{A}_k = F(\mathcal{A}_k)$, **point fixe**.
- Représentation des valuations, calcul **polyèdres convexes**
- Convergence de la résolution. **opérateur d'élargissement**

Le treillis des polyèdres - Double Représentation

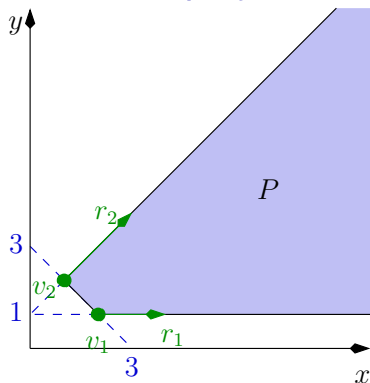


Le treillis des polyèdres - Double Représentation



$$\begin{aligned} P &= \{(x, y) \mid \\ &\quad 1 \leq y \leq x + 1 \wedge x + y \geq 3\} \\ &= \text{cons}\{AX \leq b\} \end{aligned}$$

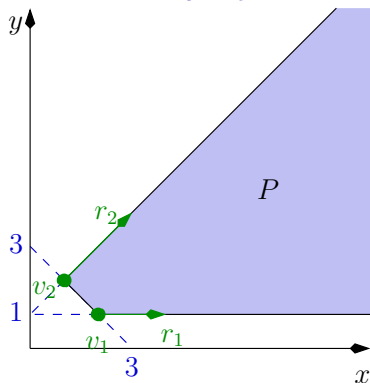
Le treillis des polyèdres - Double Représentation



$$\begin{aligned}
 P &= \{(x, y) \mid \\
 &\quad 1 \leq y \leq x + 1 \wedge x + y \geq 3\} \\
 &= \text{cons}\{AX \leq b\}
 \end{aligned}$$

$$\begin{aligned}
 P &= \{\lambda v_1 + (1 - \lambda)v_2 + \mu_1 r_1 + \mu_2 r_2 \mid \\
 &\quad \lambda \in [0, 1], \mu_1, \mu_2 \geq 0\} \\
 &= \text{gen}(V, R)
 \end{aligned}$$

Le treillis des polyèdres - Double Représentation



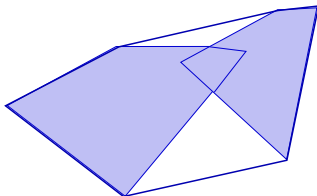
$$\begin{aligned}
 P &= \{(x, y) \mid \\
 &\quad 1 \leq y \leq x + 1 \wedge x + y \geq 3\} \\
 &= \text{cons}\{AX \leq b\}
 \end{aligned}$$

$$\begin{aligned}
 P &= \{\lambda v_1 + (1 - \lambda)v_2 + \mu_1 r_1 + \mu_2 r_2 \mid \\
 &\quad \lambda \in [0, 1], \mu_1, \mu_2 \geq 0\} \\
 &= \text{gen}(V, R)\}
 \end{aligned}$$

- Deux représentations **finies** et complémentaires (passage de l'une à l'autre).
- Algorithmique disponible.

Le treillis des polyèdres convexes (2)

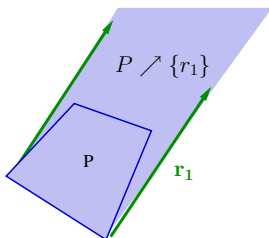
- Intersection, test du vide.
- Transformation affine : $a(P) = \{CX + D \mid X \in P\}$.
- Union convexe (perte de précision) :



Le treillis des polyèdres convexes (2)

- Intersection, test du vide.
- Transformation affine : $a(P) = \{CX + D \mid X \in P\}$.
- Union convexe (perte de précision) :
- Ajout de rayons

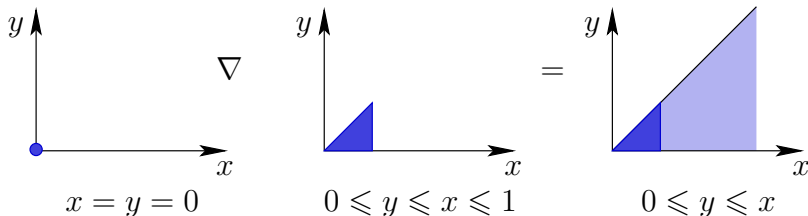
$$P \nearrow R = \left\{ X + \sum_{r_j \in R} \mu_j r_j \mid X \in P, \mu_j \in \mathbb{Q}^+ \right\}$$



Le treillis des polyèdres convexes (3)

Élargissement : $P \nabla Q$: extrapolation de la limite.

Le système de contrainte de $P \nabla Q$ est obtenu en enlevant du système de Q les contraintes non saturées par P :



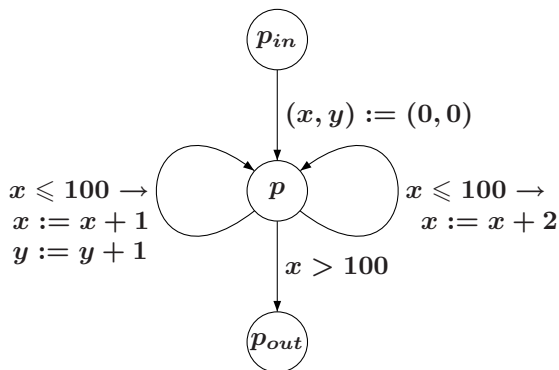
Astuce (!) : $\{x = y = 0\} = \{0 \leq y \leq x \leq 0\}$

Un exemple - 1

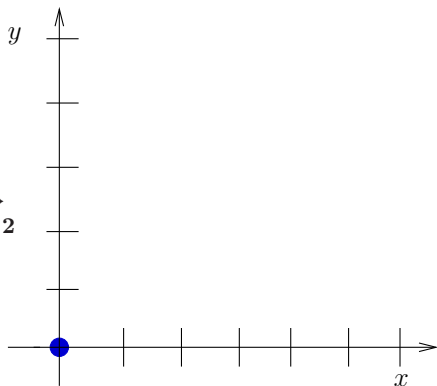
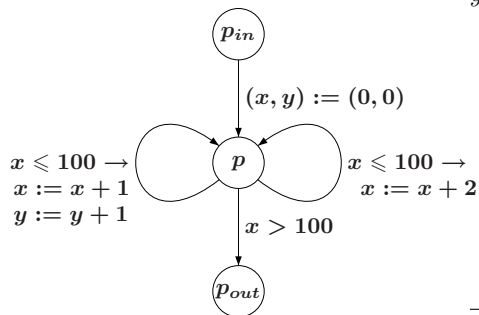
```

x:=0;y:=0
while (x<=100) do
  read(b);
  if b then
    x:=x+2
  else begin
    x:=x+1;
    y:=y+1;
  end;
endif
endwhile

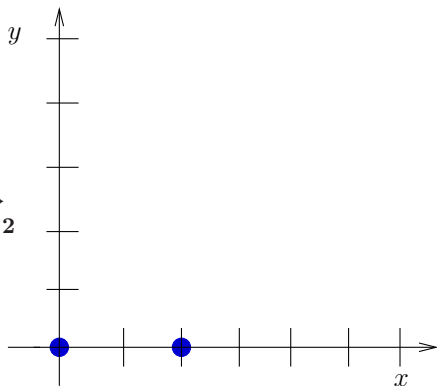
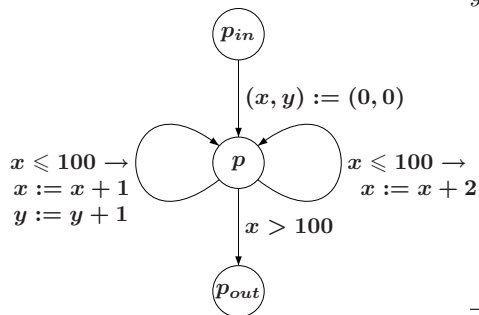
```



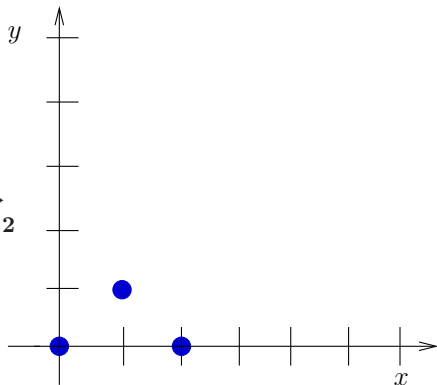
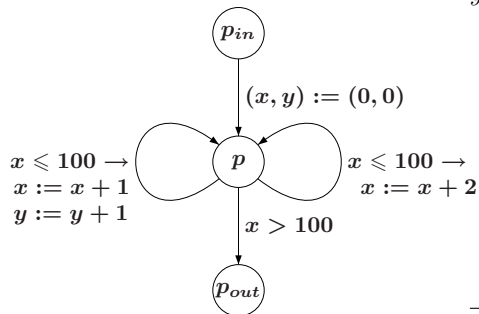
Un exemple - 2



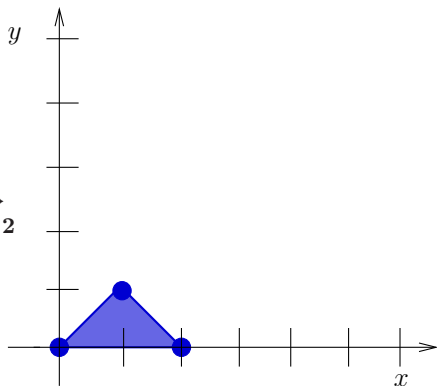
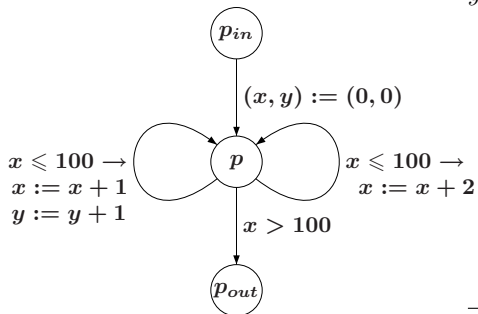
Un exemple - 2



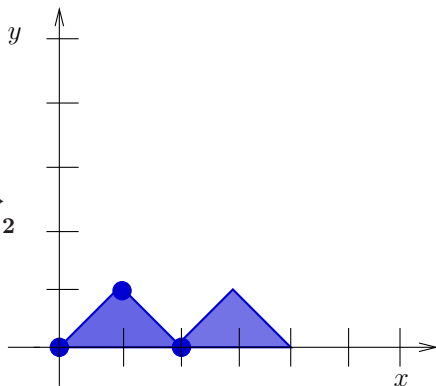
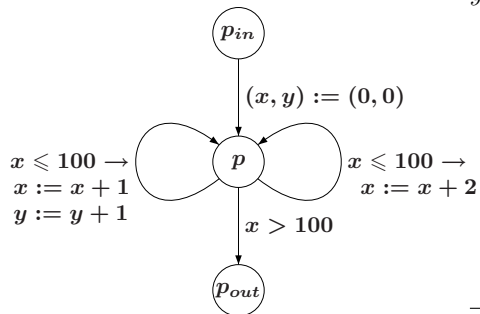
Un exemple - 2



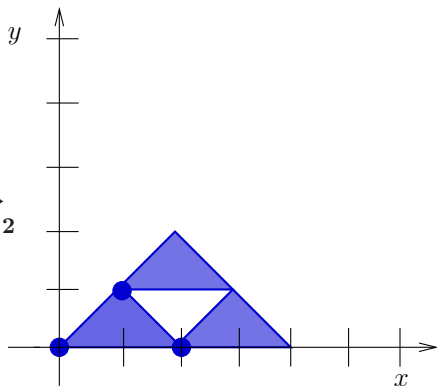
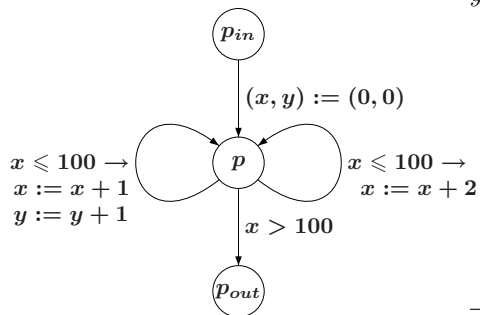
Un exemple - 2



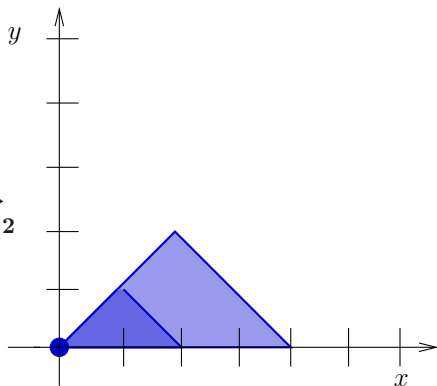
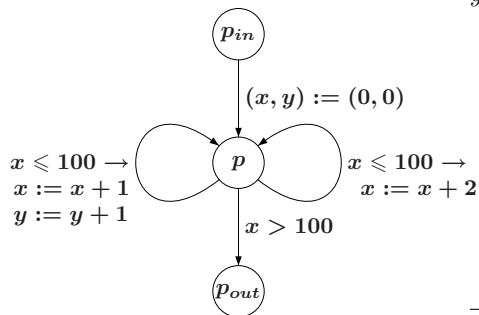
Un exemple - 2



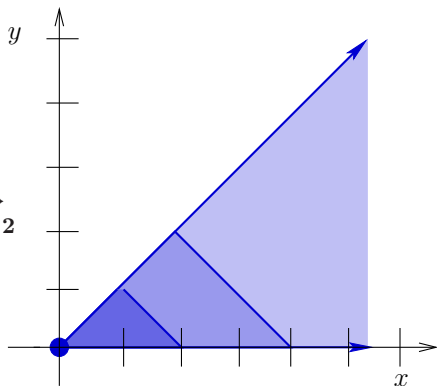
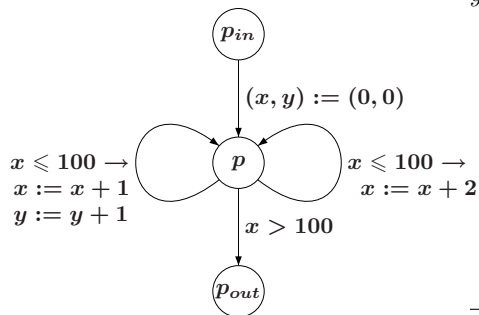
Un exemple - 2



Un exemple - 2



Un exemple - 2



- 1 Conception de systèmes embarqués
 - Architecture logicielle pour la QoS
 - Contributions
 - Applications

- 2 Génération d'invariants numériques
 - Analyse des Relations Linéaires
 - **Contributions de ma thèse**
 - Applications

Les problèmes de l'Analyse des Relations Linéaires

Sources de complexité :

- nombre de points de contrôle.
- nombre de variables numériques.

Sources d'approximation :

- Enveloppe convexe.
- **Élargissement.**

Accélération et Analyse des Relations Linéaires

Contributions théoriques et algorithmiques :

Combinaison ARL et techniques d'accération
[Finkel/Sutre/Leroux/...]

- Définition de la notion d'**accélération abstraite** qui :
 - Permet d'obtenir des approximations supérieures à faible coût.
 - Se combine bien avec l'élargissement.
- Résultats théoriques sur les boucles accélérables.
- Mise en œuvre algorithmique et **prototype**.

- 1 Conception de systèmes embarqués
 - Architecture logicielle pour la QoS
 - Contributions
 - Applications

- 2 Génération d'invariants numériques
 - Analyse des Relations Linéaires
 - Contributions de ma thèse
 - Applications

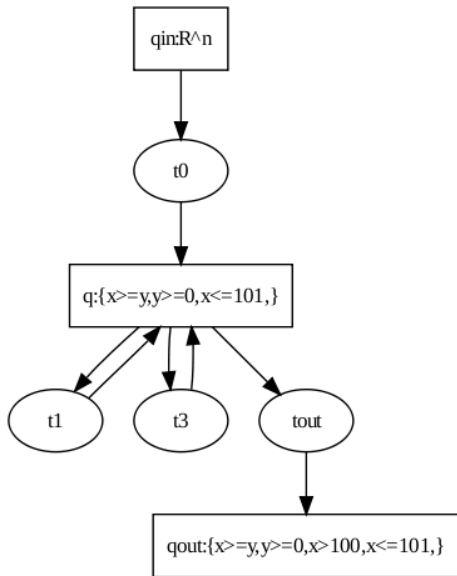
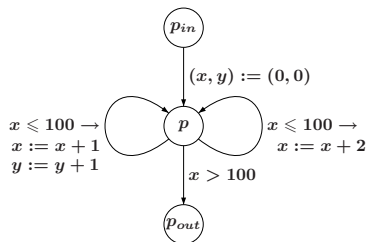
Caractéristiques d'Aspic

ASPIC : Accelerated Symbolic Polyhedral Invariant
Computation

- Un langage textuel d'automates (Fast) avec ou sans but de preuve (formule). En cours de développement, **c2aspic**.
- Calcul classique + accélérations.
- Sorties : invariants (+ diagnostic).

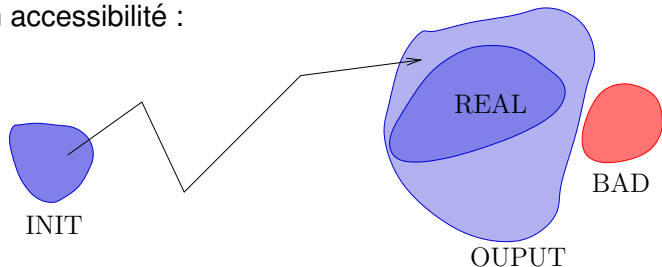
▶ <http://laure.gonnord.org/pro/aspic/aspic.html>

Invariants de l'exemple



Applications - 1

- Vérification de programmes **numériques**. On prouve la non accessibilité :



- (non) Accessibilité dans des automates à compteurs (sémantique de SystemC), une centaine de points de contrôle, J. Cornet.

Applications - 2

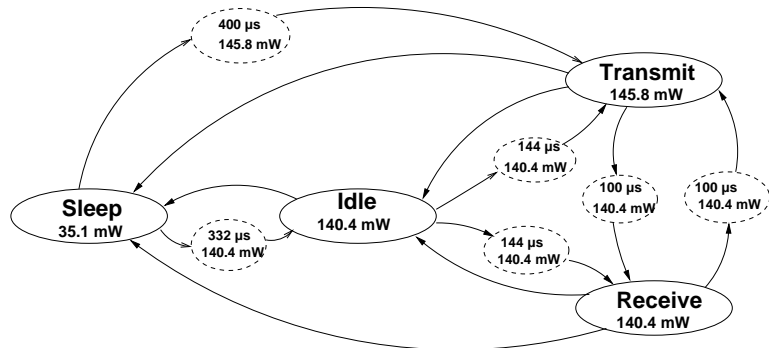
Par encodage dans des automates à compteurs :

- Vérification de programmes manipulant des **listes**, R. Iosif et S. Perarnau.
- Vérification de programmes à **pointeurs**, A. Sangnier et A. Finkel.

Applications - 3

Par encodage toujours :

- Invariants numériques d'automates modélisant une **consommation d'énergie** de (réseaux de) capteurs, L. Samper et F. Maraninchi.



Applications - 4

En cours : calcul de pire temps d'exécution de programmes :

- compilation + analyse statique
 - ordonnancement.
- ▶ Travail avec A. Darte, P. Feautrier, C. Alias.

Fin de l'exposé

Merci.