Prédiction de bornes paramétriques de consommation énergétique

Point de vue et perspectives de l'interprétation abstraite

Louis Rustenholz[1,2]

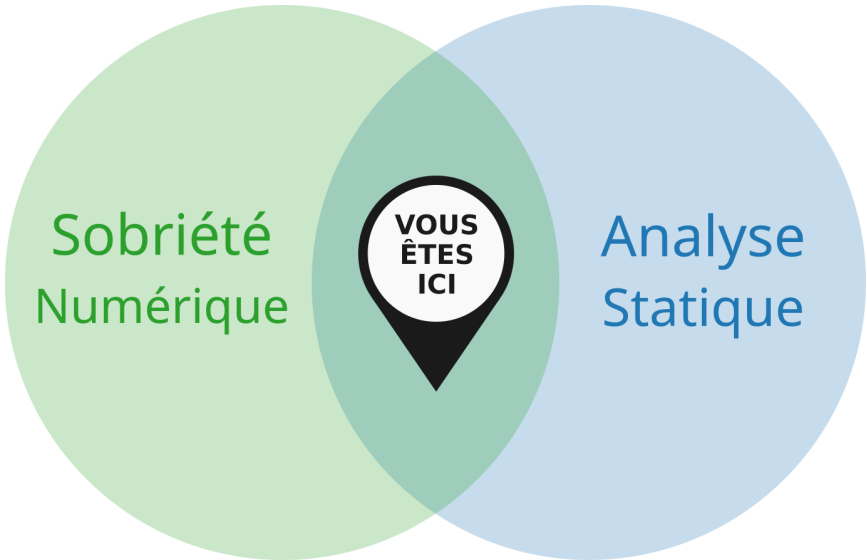
[1]Universidad Politécnica de Madrid, Espagne

[2]IMDEA Software Institute, Madrid, Espagne

16 Juin 2025

CNRS GDR GPL

Journées Nationales, Pau, France



Analyse statique

ou l'art de borner les comportements des systèmes

Analyse statique

Analyse statique de programmes

« Analyse automatique de programmes avant leur exécution »

Différentes saveurs :

- **Syntactique** (linters)

  Imposer des conventions de style, détecter des mauvaises pratiques, “code smells”, ...

➔ Haut niveau, souvent heuristique : comportements du développeur.

- **Sémantique**

  Absence de bugs, valeurs possibles des variables, flot d’information, “code summary”, ...

➔ Découverte de *propriétés formelles* (de l’ensemble infini) des *exécutions possibles*.

« Que peut-il se passer lors d’une exécution ? »

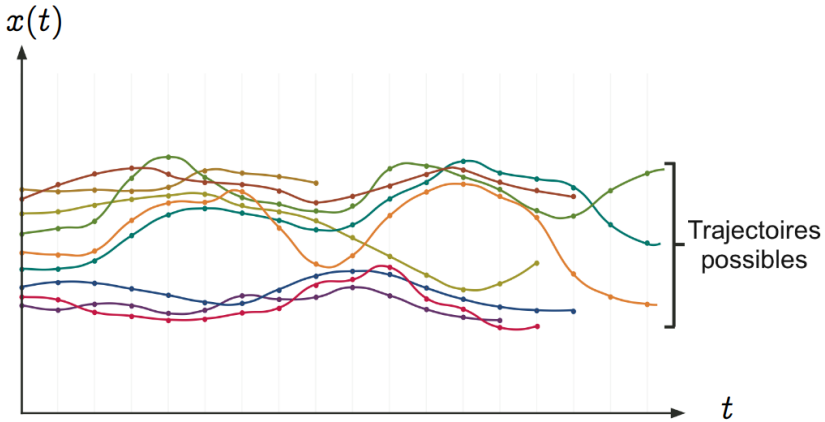
Analyse statique (sémantique)

« Que peut-il se passer lors d'une exécution ? », ou plus généralement,

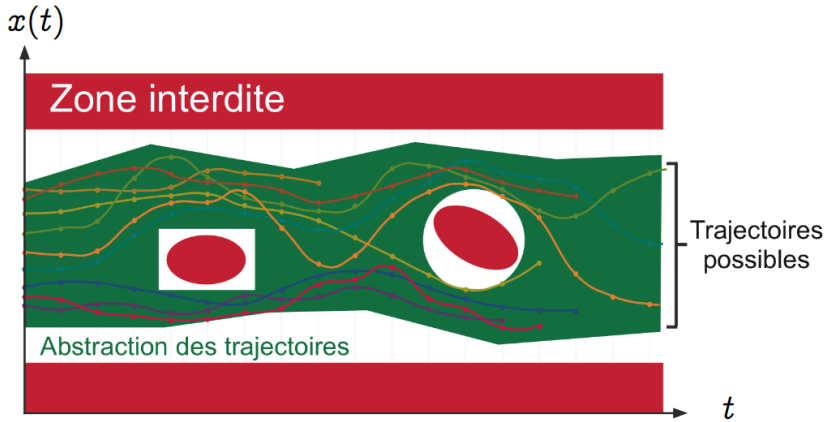
L'art de borner les comportements des systèmes

en analysant leur description (code, équations). (programmes, hardware, réseaux, cyberphysique, biochimie, ...)

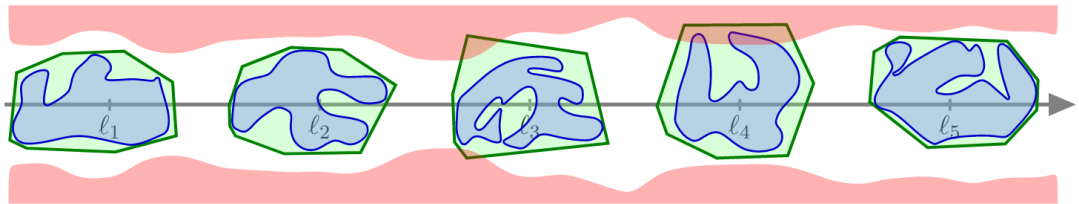
« Compréhension automatisée », « Découverte automatique de vérités »



Sémantique$[\![P]\!]$

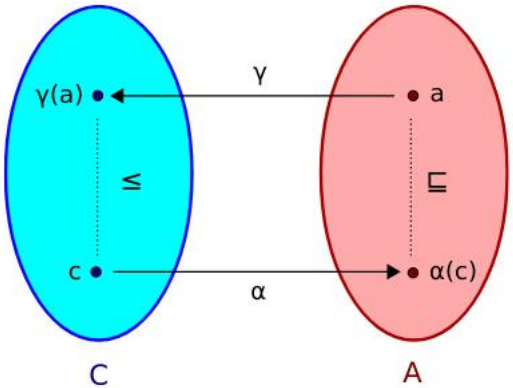


Sémantique$[\![P]\!] \subseteq$ Abstraction(Sémantique$[\![P]\!]$)

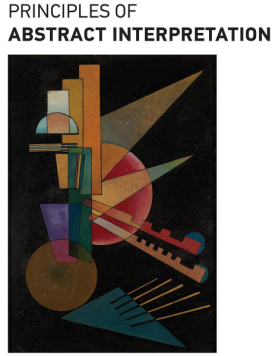


Analyse statique (sémantique) – interprétation abstraite

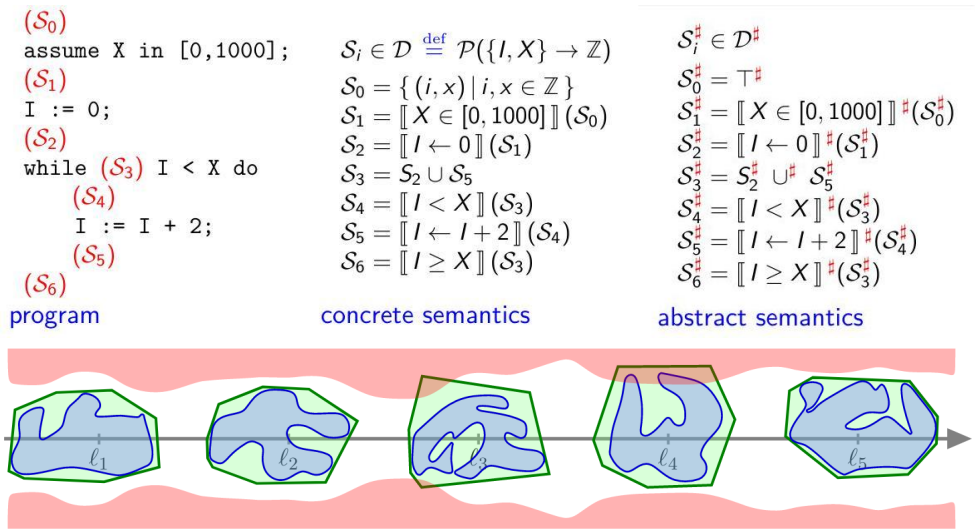
- Belle théorie mathématique (algèbre, géométrie, logique et sémantiques).
- Méthodologie de conception d'analyseurs, proche des interpréteurs/compilateurs.

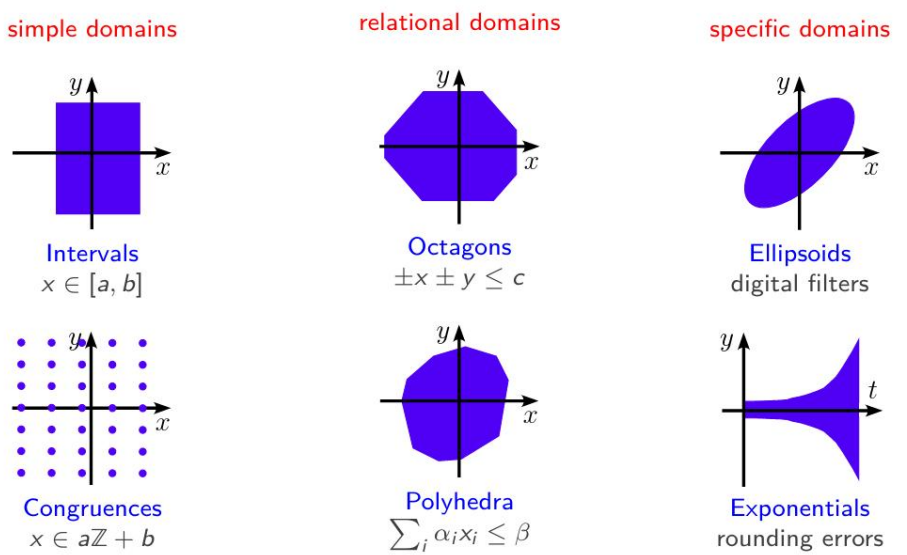


Correspondances de Galois



Cousot–Cousot



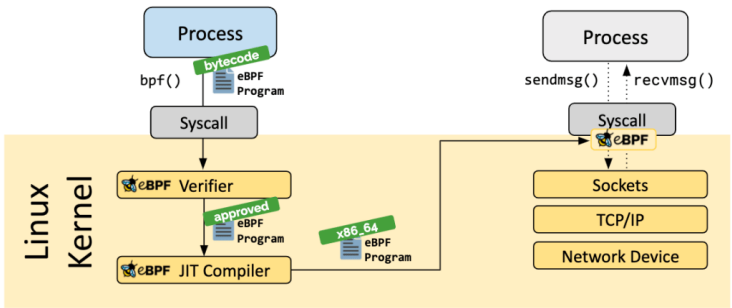


Analyse statique : applications

Sécurité des systèmes critiques

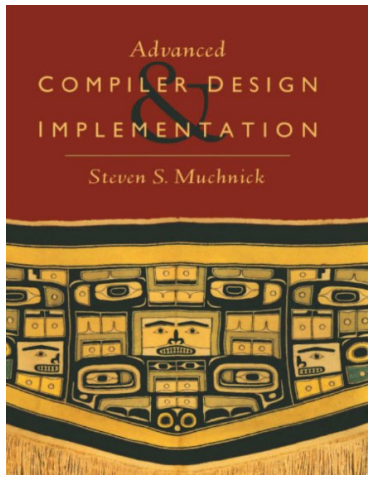
Aérospatiale, aéronautique. Crash d'Ariane 5.
Vérification dans le ferroviaire, médical, ...

En 2005 chez Airbus : preuve d'absence de bug,
100kLoC – 1MLoC, 45min – 40h



eBPF verifier ; Depuis ∼2015, interpréteurs abstraits, en direct dans le *noyau Linux*

Optimisations de compilateurs

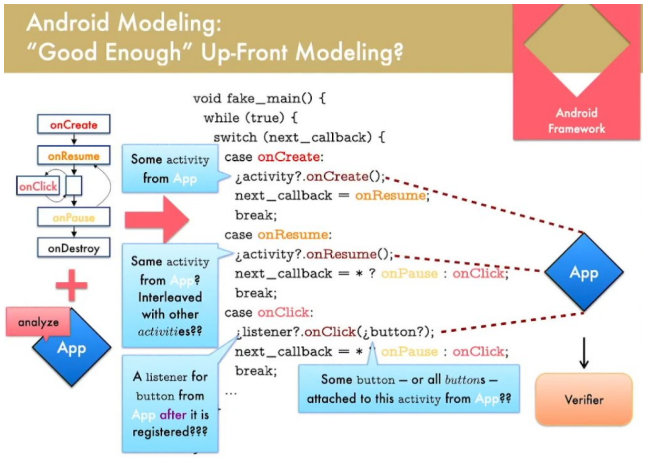



```
gcc -O3 <...>
```

Analyse statiques *efficaces* pour permettre des transformations et choix de compilation.

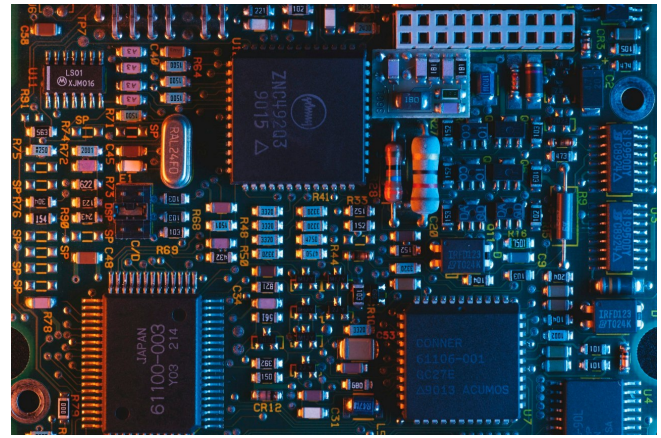
détection sémantique de code mort,
simplifications subtiles d'expressions,
réordonnancement d'instructions,
transformation de boucles, ...

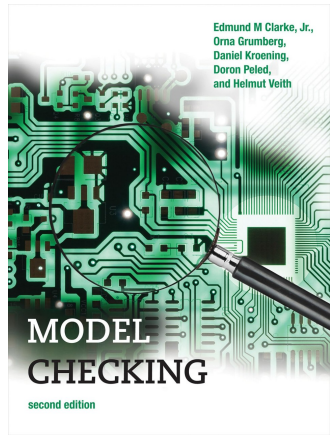
Analyse statique : applications



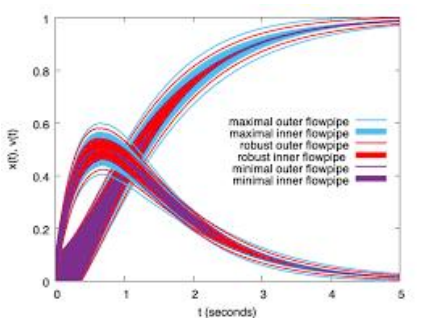
Applications Android en communication

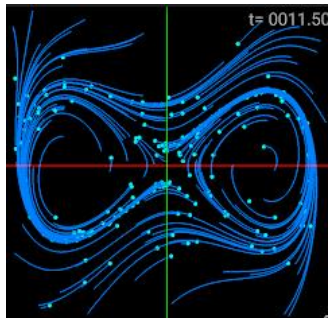
« Programmes gruyères »



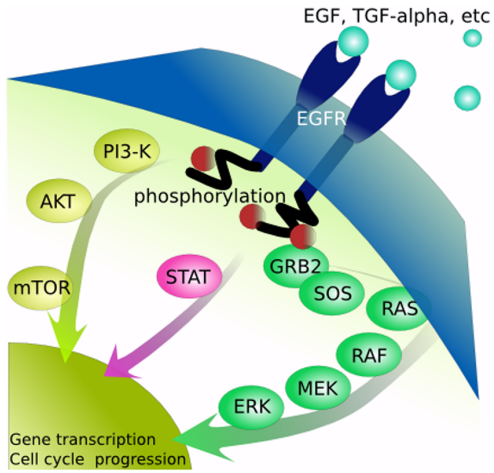


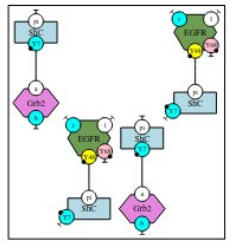
Hardware, circuits électroniques

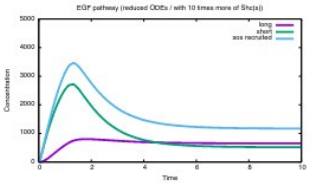




Système cyber-physiques







Réseaux d'équations (bio)chimiques

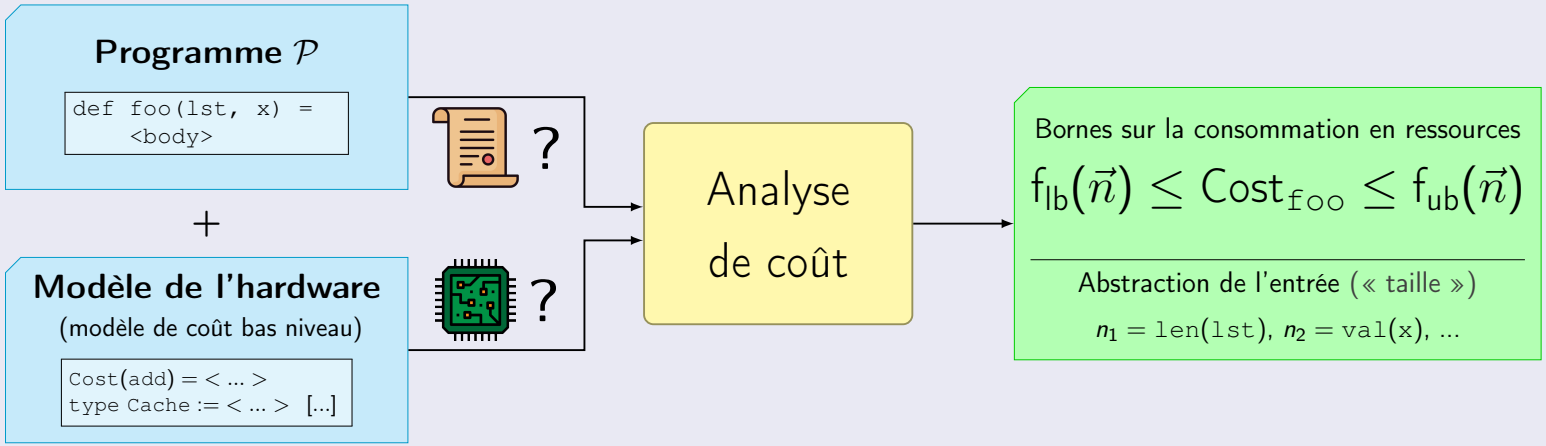
Analyse de coût

Prédiction de bornes paramétriques, et état de l'art

Qu'est-ce qu'un analyseur statique de coût ?

Prédire la consommation en ressources :
avant exécution, et sur l'ensemble des chemins possibles.

Analyse de coût : borner la consommation de ressources



Garanties de sécurité
(WCET, side-channel, ...)

Améliorations, Optimisations
(parallélisation, scheduling, ...)

Assistants de compréhension :
transparence, évaluation

Exemple : projet ENTRA, analyseur CiaoPP

CiaoPP : Analyse, Vérification et Optimisation

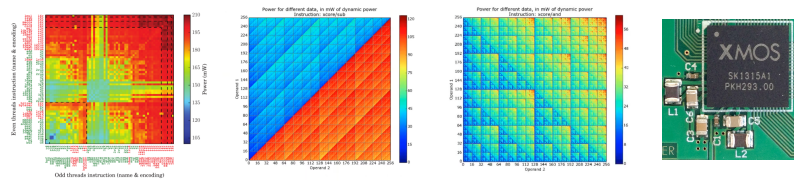
- Outil basé sur l'interprétation abstraite.
- Multilingue par conversion en clauses de Horn.

Contient un **analyseur de coût**,

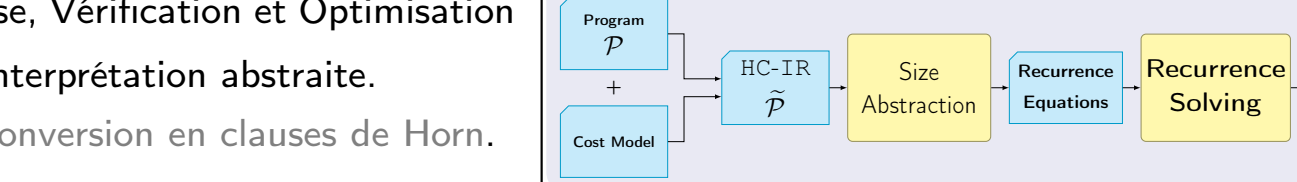
- Initialement pour la parallélisation automatique.
- Étendu et **appliqué à l'énergie** depuis ∼15 ans.
- Approche : usage d'équations de récurrences comme abstractions de programmes.

➔ Bornes paramétriques, non-linéaires, correctes.

Projet européen **ENTRA**
(**EN**ergy **TRA**nsparency)



Processeur XMOS, multicœur, multithread, sans cache




```
#pragma true fir(xn, coeffs, state, N) :
             (3347178*N + 13967829 <= energy &&
              energy <= 3347178*N + 14417829)

int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
    state[j] = state[j-1];
    {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
    ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

```
def f(n,c):
  if n>0 and c>=100:
    return f(n-1, 0) + n + 300
  if n>0 and c<100:
    return f(n-1, c+1) + n
  return c
```

$$f_{sol}(n,c) \leq \begin{cases} \frac{1}{2}n^2 + \frac{701}{202}n + \frac{30000}{101} & \text{if } n > 0 \wedge c \geq 100 \\ \frac{1}{2}n^2 + \frac{701}{202}n + \frac{300}{101}c & \text{if } n > 0 \wedge c < 100 \\ c & \text{if } n = 0 \end{cases}$$

Exemple : projet ENTRA, analyseur CiaoPP

CiaoPP : Analyse, Vérification et Optimisation

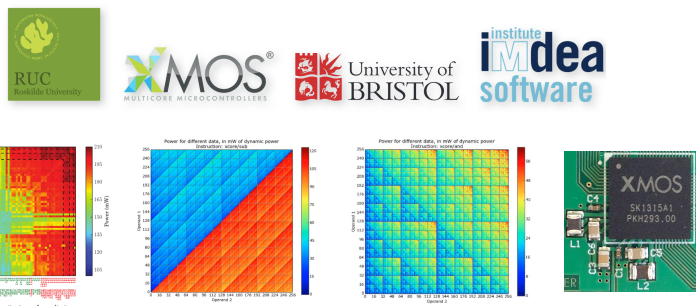
- Outil basé sur l'interprétation abstraite.
- Multilingue par conversion en clauses de Horn.

Contient un **analyseur de coût**,

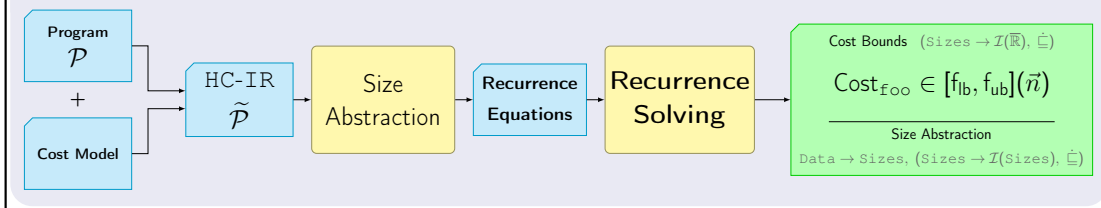
- Initialement pour la parallélisation automatique.
- Étendu et **appliqué à l'énergie** depuis ∼15 ans.
- Approche : usage d'équations de récurrences comme abstractions de programmes.

➔ Bornes paramétriques, non-linéaires, correctes.

Projet européen **ENTRA**
(**EN**ergy **TRA**nsparency)



Processeur XMOS, multicœur, multithread, sans cache



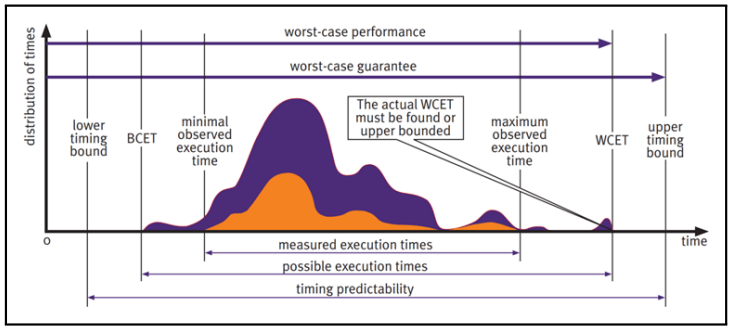

```
#pragma true fir(xn, coeffs, state, N) :
             (3347178*N + 13967829 <= energy &&
              energy <= 3347178*N + 14417829)

int fir(int xn, int coeffs[], int state[], int ELEMENTS)
{
  unsigned int ynl; int ynh;
  ynl = (1<<23); ynh = 0;
  for(int j=ELEMENTS-1; j!=0; j--) {
    state[j] = state[j-1];
    {ynh, ynl} = macs(coeffs[j], state[j], ynh, ynl);
  }
  state[0] = xn;
  {ynh, ynl} = macs(coeffs[0], xn, ynh, ynl);
  if (sext(ynh,24) == ynh) {
    ynh = (ynh << 8) | (((unsigned) ynl) >> 24);}
  else if (ynh < 0) { ynh = 0x80000000; }
  else { ynh = 0x7fffffff; }
  return ynh;
}
```

```
def f(n,c):
  if n>0 and c>=100:
    return f(n-1, 0) + n + 300
  if n>0 and c<100:
    return f(n-1, c+1) + n
  return c
```

$$f_{sol}(n,c) \leq \begin{cases} \frac{1}{2}n^2 + \frac{701}{202}n + \frac{30000}{101} & \text{if } n > 0 \wedge c \geq 100 \\ \frac{1}{2}n^2 + \frac{701}{202}n + \frac{300}{101}c & \text{if } n > 0 \wedge c < 100 \\ c & \text{if } n = 0 \end{cases}$$

Paysage de l'analyse statique de coût

État de l'art et historique

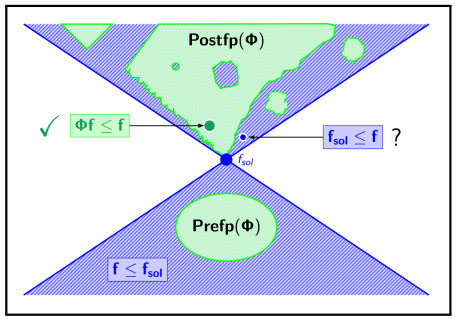


WCET, analyses bas niveau

$$f_{sol}(n,c) \leq \begin{cases} \frac{1}{2}n^2 + \frac{701}{202}n + \frac{30000}{101} & \text{if } n > 0 \wedge c \geq 100 \\ \frac{1}{2}n^2 + \frac{701}{202}n + \frac{300}{101}c & \text{if } n > 0 \wedge c < 100 \\ c & \text{if } n = 0 \end{cases}$$

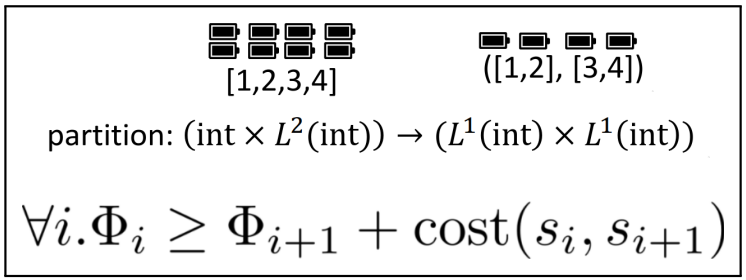
Analyses haut niveau paramétriques

Programmes naturels sauvages

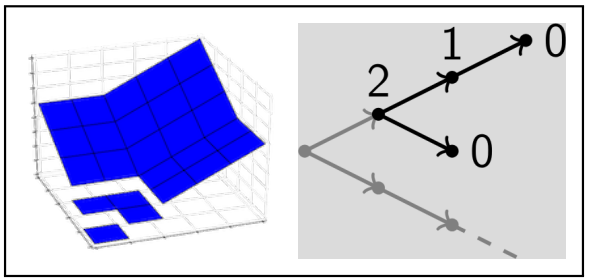


Équations de récurrences généralisées

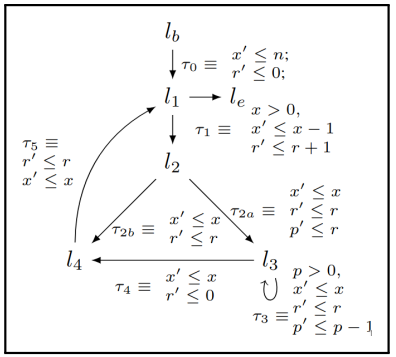
(équations-opérateurs, ...)



Types et potentiels



Fonctions de rang



Abstraction de systèmes de transition

➔ **Grâce aux progrès récents, l'analyse à grande échelle de programmes « naturels » devient possible.** Support simultané de patterns de programmation « naturels » et variés.

Mesurer ou prédire ? Sampling et analyse statique

Pourquoi faire de l'analyse statique quand on peut échantillonner et mesurer ?

- Possibilités de **garanties**,
  ou d'améliorer la confiance (modèles de coût statistiques possibles).
- **Paramétricité !** Prédictions, explicabilité et raisonnements.
  - **Prédire l'évolution du coût quand on change un paramètre :**
    tailles d'entrées, architecture, contexte d'exécution, ...
  - Pourquoi ce coût ? Comment réparer ? Optimiser ?
    Quelle option est la meilleure, dans un certain contexte, parmi des transformations
    ou pour un paramètre laissé libre par le développeur ?
  - **Modularité.** Analyses a priori, pendant le développement, de programmes à trous.
- ⚠ **Coût du sampling**, surtout quand l'espace des possibles est grand.
  + intrusivité, instrumentation de code, ...

➔ **Complémentarité**, comme physique expérimentale et physique théorique ?

- Mesure et sampling pour **calibrer les modèles, identifier les abstractions pertinentes**, ...
- Analyses et raisonnement symbolique pour **expliquer et passer à l'échelle**.
- Intermédiaire : tests abstraits. Utiliser dynamiquement des entrées abstraites
  pour un meilleure couverture de code / de l'espace des possibles.

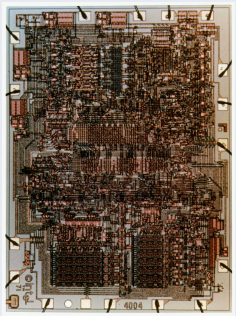
Applications à la sobriété numérique

Projets, perspectives et synergies

Bloat, « obésiciel » – du gâchis à tous les étages

Loi de Moore (1975)

« La puissance de calcul des ordinateurs double tous les deux ans [...] »

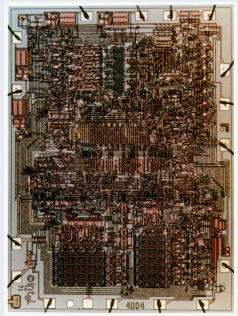


(Gordon E. Moore, cofondateur d'Intel)

Bloat, « obésiciel » – du gâchis à tous les étages

Loi de Moore (1975)

« La puissance de calcul des ordinateurs double tous les deux ans [...] »



(Gordon E. Moore, cofondateur d'Intel)

Loi de Wirth (1995)

« [...] mais les programmes ralentissent (presque) plus vite que le matériel n'accélère »

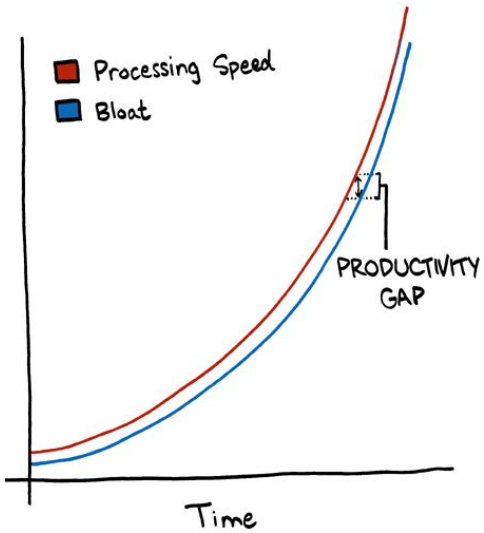
(Niklaus Wirth, prix Turing 1984)

Un peu caricatural, mais problème réel.

Situation encore pire pour l'énergie...

Du gâchis à tous les étages.

➔ Comment y faire face ?



(http://www.smbc-comics.com)

Côté compilateur : traque et amélioration automatique

Problème : beaucoup de gâchis de ressources en programmation.

➔ **Des passes de compilateur pour traquer, identifier, et éliminer certains d'entre eux ?**

- Question étudiée en temps, mais encore relativement peu pour l'énergie.
- ⚠ Énergie ≠ Temps ≠ Mémoire. (des corrélations, mais aussi des indépendances et compétitions)

Des idées simples...

- Réduction de tailles allouées pour des variables. (bitwidth des `int`, précision nécessaire de `float`, ...)

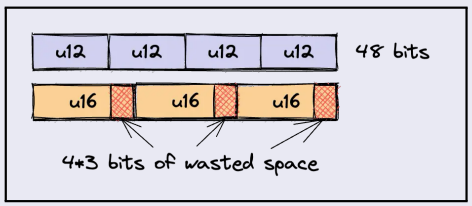
➔ On peut parfois gagner des **facteurs** 2 **en énergie** avec des optimisations aussi simples.

- Critère de choix d'option de transformation, parmi des préexistentes.
- DVFS (réduction de fréquence) sélectif, e.g. par processus/thread.

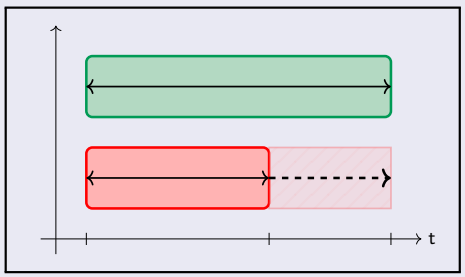
... et d'autres plus ambitieuses.

- Layout mémoire/caches/vectorisation, transformations de programmes les permettant.
- Jusqu'à la gestion locale de *températures*, « thermal-aware compilation », compromis équilibrage de charge entre composants vs désactivations partielles, ... ?

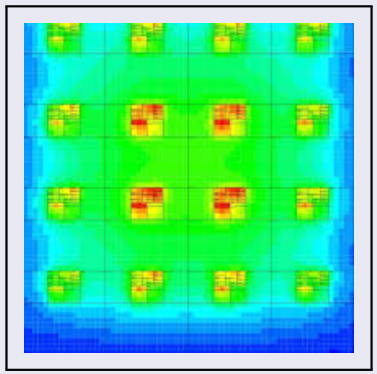
➔ Vers de la *prédiction de charge* ambitieuse à tous les niveaux ?



bitwidth



DVFS



température

Côté compilateur : traque et amélioration automatique

Problème : beaucoup de gâchis de ressources en programmation.

➔ **Des passes de compilateur pour traquer, identifier, et éliminer certains d'entre eux ?**

- Question étudiée en temps, mais encore relativement peu pour l'énergie.
- ⚠ Énergie ≠ Temps ≠ Mémoire. (des corrélations, mais aussi des indépendances et compétitions)

Des idées simples...

- Réduction de tailles allouées pour des variables. (bitwidth des `int`, précision nécessaire de `float`, ...)

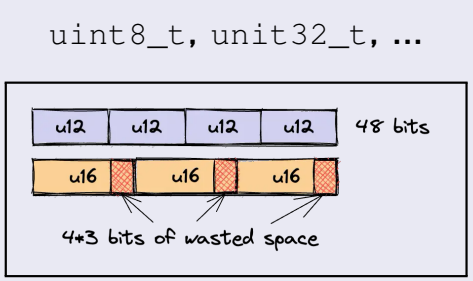
➔ On peut parfois gagner des **facteurs 2 en énergie** avec des optimisations aussi simples.

- Critère de choix d'option de transformation, parmi des préexistentes.
- DVFS (réduction de fréquence) sélectif, e.g. par processus/thread.

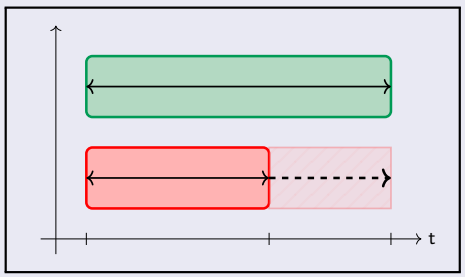
... et d'autres plus ambitieuses.

- Layout mémoire/caches/vectorisation, transformations de programmes les permettant.
- Jusqu'à la gestion locale de *températures*, « thermal-aware compilation », compromis équilibrage de charge entre composants vs désactivations partielles, ... ?

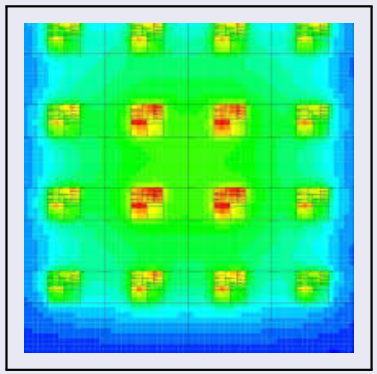
➔ Vers de la *prédiction de charge* ambitieuse à tous les niveaux ?



bitwidth



DVFS



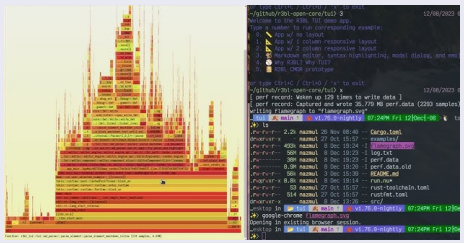
température

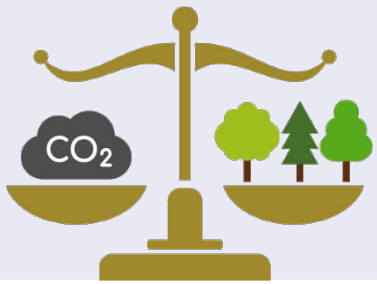
⚠ Rebond

Côté humain : transparence, assistants de compréhension

⚠ Rebond... ➔ Combiner avec **transparence énergétique** pour développeurs (et auditeurs/régulateurs) ?

- « **Profilage statique** ».
  - ***Mettre en évidence*** les coûts. (dans l'IDE, etc.)
  - Aide à la compréhension : ***expliquer*** l'origine des coûts.
    (*Localiser* les bottlenecks. Opération coûteuse ? Répétée ? Sur de grandes données ?)
  - (Aussi : aide au développement avec paramètres libres, « code à trou ».)
- Détection de « bugs de performance ». (Cas inattendus, interférences, ...)
- **Documentation automatisée et adaptative.** Résumé de bibliothèques, ...





Un peu de science-fiction ?

➔ Vers une aide à la prise en compte, grâce à des **indicateurs mis en évidence** ?

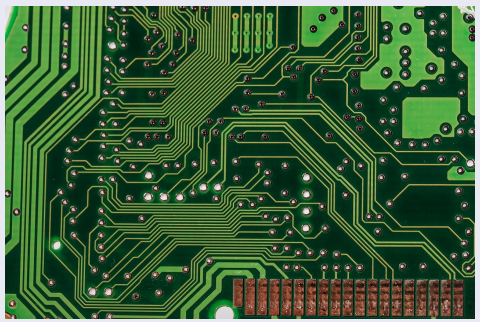
➔ Vers une **aide à l'audit**, à l'estimation de l'impact carbone *du software* ?

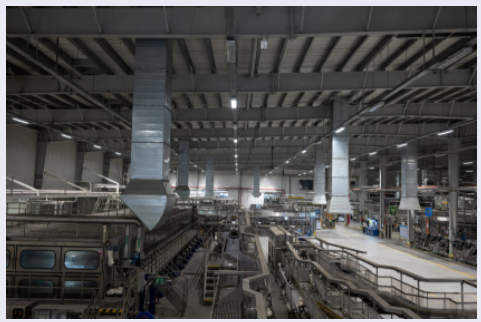
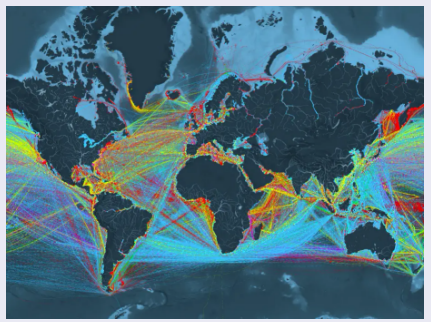
➔ Vers une **aide à la régulation** ?? (Savoir maîtriser pour pouvoir normer.)

Perspectives – autres applications possibles, au-delà du code

L'analyse statique est flexible, et peut s'appliquer aux « systèmes » dont on dispose de descriptions fiables.

- Langages de description de matériel, hardware.
- Programmes multiples en interaction. (tournant sur un serveur, OS, téléphone, ...)
- Configurations réseaux / centres de données ? Autres phénomènes réseaux ?
- « Processus opérationnels » plus généraux ??
  Chaînes de production, d'approvisionnement, ...
  ➔ Déjà quelques prototypes !
  ➔ (Lien avec la recherche opérationnelle.)



Conclusion – Synergies, et vos idées ?

Cet exposé

- **Introduction à un domaine** : qu'est-ce que l'analyse statique (de coût) ?
  Grandes familles de techniques, exemple d'analyseur.
- ➔ **Obtenir automatiquement des bornes paramétriques de la consommation énergétique de programmes.**
- Propositions d'applications, pour la sobriété numérique.
- Un peu de science-fiction (à laquelle les progrès de la recherche autorisent de croire).

Encore beaucoup de travail et d'ingénierie nécessaires pour déployer les progrès récents à grande échelle.

Complémentarité

Mesure, génie logiciel empirique. Approche expérimentale.

- Données pour produire des modèles (haut et bas niveau).
- Identifier des propriétés intéressantes à analyser.
- (Statique/dynamique) Précision, couverture, coût de mesure.
- Plus de données, d'explications causales et de prédictions.

Analyse statique version syntactique/linters, ...

- Déploiement : mêler ce qu'on sait et ne sait pas prédire.
- Identifier les « nudges » efficaces et pertinents ?
- Outils pour identification d'antipatterns plus généraux.
- Travail de l'interaction utilisateur.
- Garanties et justifications.

Au-delà des grands projets autonomes, l'**analyse statique** (sémantique) s'est aussi souvent construite et rendue utile comme un **domaine support, qui *fournit des outils***.

➔ Et vos idées ?

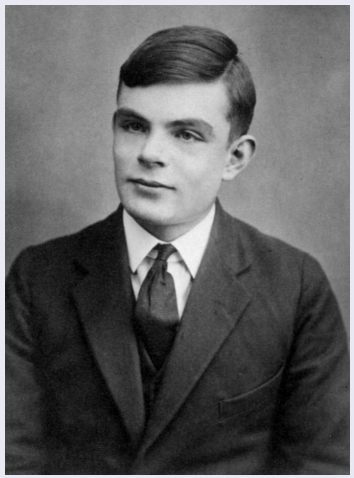
Merci pour votre attention !

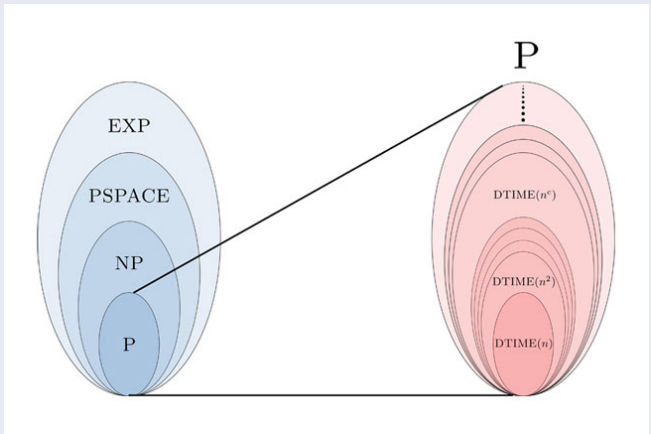
Questions ?

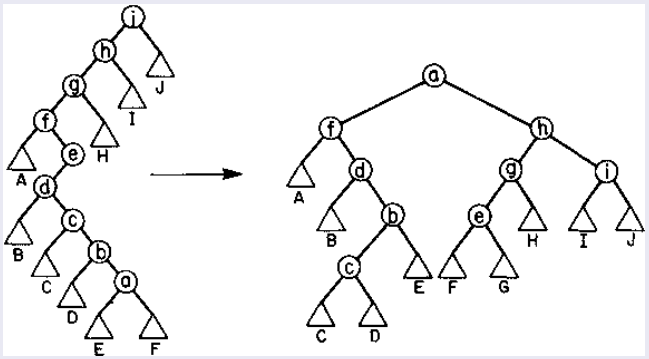
Historique — origines

Années 30–80 : Fondements théoriques des questions de coût.

- Terminaison, problème de l'arrêt (Turing 1936 ; Kleene 1952 ; ...)
- Théorie de la complexité (Hartmanis et Stearns 1965 ; Cook 1971 ; ...)
- Analyse asymptotique, popularisation du $\mathrm{O}(\cdot)$ (Knuth 1971 ; ...)
- Complexité amortie, théorie du potentiel (Tarjan 1985)







Historique — premières analyses, équations de récurrence

Analyse automatique de complexité :

méthodes par **équations de récurrences** (et **interprétation abstraite**)

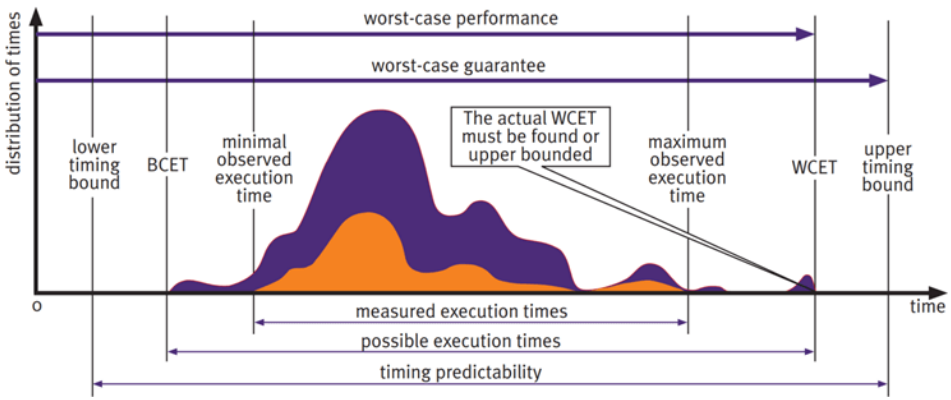
- **1975–1990** : Premières analyses automatiques.
  `Metric` (Wegbreit 1975), (Le Métayer 1988 ; Rosendahl 1989)
  - Programmes simples : limités par les outils de calculs symboliques (CAS)
    usage de méthodes à templates, n'atteint pas toujours un résultat...
  - Modèles de coût simples (proche du "`tick`").
- **1990–2007** : Maturation et démonstrateurs (projet `Ciao`, etc.)
  - Solveurs de récurrences plus spécialisés (mais encore template-based).
  - Amélioration des extractions de récurrences, représentation et analyse de plus en plus de
    - langages, types de programmes et de données,
    - flots de contrôles (non-linéaires, non-déterministes, parallèles, probabilistes, ...),
    - ressources (intrisèques au langage, temps, mémoire, communication, défini par l'utilisateur, énergie).
  - Meilleure précision, analyses auxiliaires, bornes inférieures, ...
  - Défrichement d'applications :
    - pour la compilation (parallélisation automatique, choix de transformations, ...),
    - aide au développement (static profiling, explications de coût, ...),
    - garanties de performance.
- À partir de ∼**2008** :
  Introduction de méthodes d'approximation agressives vers des équations plus complexes.

Historique — WCET, à l'attaque de l'hardware

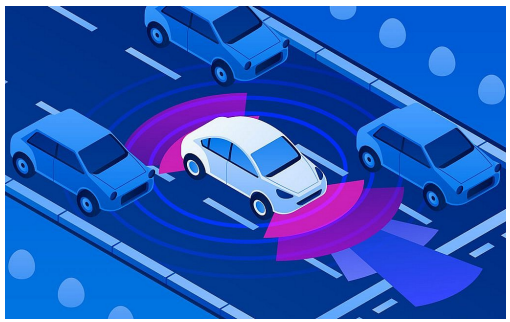
À partir des **années 90**, plus ou moins indépendamment : développement d'analyses « dans le pire des cas » (**WCET**), motivées par des applications aux **systèmes temps réel**.

➔ Pas des `ticks` mais de vraies millisecondes : traitement précautionneux de l'hardware.

- Techniques spécifiques de construction du pire cas. (arbres syntaxique, optimisation sur graphes)
  - ➔ Ok pour des bornes de coût constantes (« `873 ms` »), et des programmes très disciplinés (nombre de tours de boucle et entrées **explicitement bornés par le développeur**).
- **1995–2010** : Application à de plus en plus d'architectures.
  (analyses de caches, de pipelines, mesures de prédictabilité, ...)
- **Années 2000** : Applications industrielles marquées (e.g. logiciel aiT avec AbsInt).
- **2010–auj.** : Toujours très actif (plus encore d'architectures, un peu d'inférence de bornes de boucles, quelques bornes paramétriques, ...).



Estimations *sûres* (mesures insuffisantes)

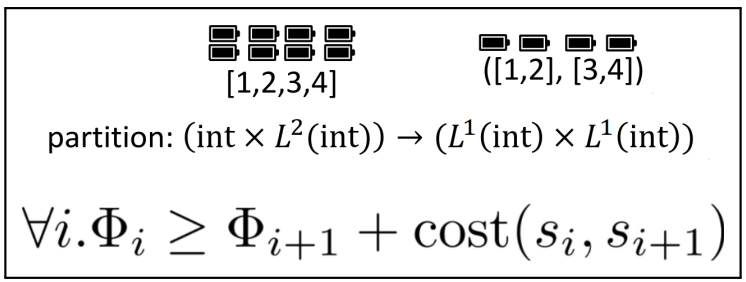


« hard deadlines »

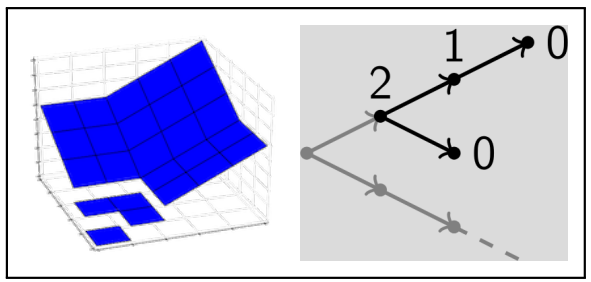
Historique — foisonnement de nouvelles méthodes

Depuis ∼**2005**, explosion de nouvelles approches haut niveau :
vers des nouveaux langages, nouvelles familles de bornes, ...

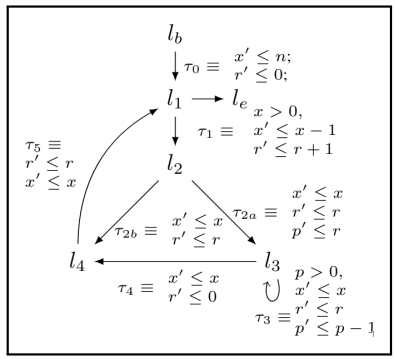
- Méthodes d'**inférence de types** pour les langages fonctionnels.
  (*sized types* en analyse de mémoire, type encodant des **potentiels** de Tarjan, ...)
- Méthodes issues de la preuve automatique de terminaison ("**ranking functions**") :
  approximations agressives, puis raffinements.
- Nouvelles **abstractions** specialisées, **de systèmes de transition**.
  ("size change graph", "difference constraints")
- Ces nouvelles méthodes évoluent pour supporter de **nouvelles classes de bornes**.
  (linéaires, polynomiales, polynomiales multivariées, un peu de exp/log...)
  Les bornes par morceaux (raisonnement disjonctif) sont cruciales.



Types et potentiels



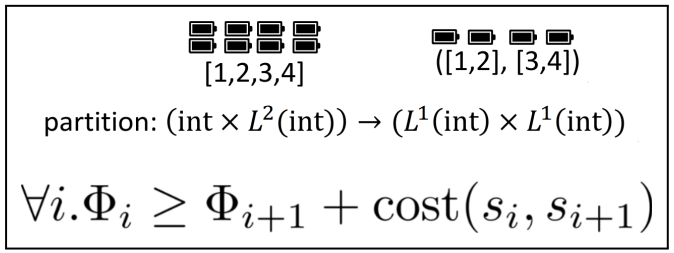
Fonctions de rang

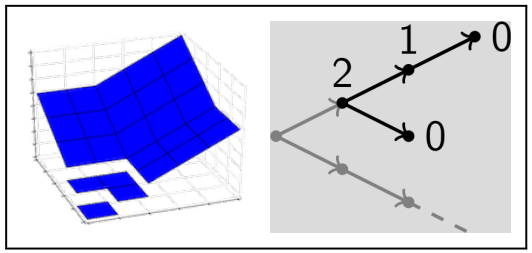


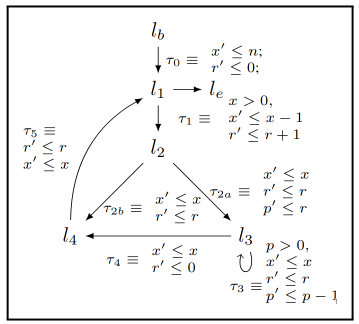
Abstraction de systèmes de transition

Historique — synthèse moderne

Depuis ∼**2005**, explosion de nouvelles approches haut niveau :
vers des nouveaux langages, nouvelles familles de bornes, ...







- Ces nouvelles méthodes évoluent pour supporter de **nouvelles classes de bornes**.
  (linéaires, polynomiales, polynomiales multivariées, un peu de exp/log...)
  Les bornes par morceaux (raisonnement disjonctif) sont cruciales.

Depuis ∼**2015** : **Synthèse** avec les méthodes à récurrences.

- Renouveau de l'engouement pour celles-ci, e.g. à travers les succès de travaux sur l'analyse non-linéaire et « accélération » de boucles plus ou moins idéalisées. (Kovács, Kincaid, Frohn, ...)
- **Récurrences généralisées** (équations fonctionnelles) **pour obtenir des bornes non-linéaires** et par morceaux, méthodes algébriques et d'**abstractions avancées**. (équations-opérateurs, ...)
- « **Fertilisation croisée** ». Côté récurrences : combinaisons flot de contrôle/récurrences, travail sur la disjonctivité, intégration sized types et ranking functions, ... De l'autre côté, réappropriation des récurrences comme outil efficace d'analyse numérique non-linéaire.

➔ **Grâce aux progrès récents, l'analyse à grande échelle de programmes « naturels » devient possible.** Support simultané de patterns de programmation « naturels » et variés.

Et maintenant ?

Quelques enjeux et travaux actuels.

- Ingénierie, passage à l'échelle de déploiement.
  - ➔ Intégration passes de compilateurs, IDE, ...
- Meilleures combinaisons hardware/software, synthèse WCET / analyses haut niveau.
  - ➔ Important pour la consommation énergétique !
- Coopérations analyses statiques / dynamiques, « machine learning ».
  - ➔ Assurer une couverture complète du code, etc.

Et l'énergie ?

- Combinaison d'informations sur les différentes ressources :
  - Énergie de tâche, puissance de fuite, coût de communication/stockage,
  - Ressources matérielles (e.g. « power gating » : coupures d'alimentation partielles).
- Modélisations automatisées (analyse VHDL, apprentissage d'automates, ...).
- Progrès en cours :
  - Modèles de coûts plus fiables, « à grain fin », sensibles à l'état mémoire et machine,
  - Meilleures combinaisons analyse hardware/software,
  - Analyses haut niveau plus générales et plus précises (pour supporter ces granularités).
- Outils, stabilisation de prototypes, intégration dans workflows, etc.