Action Concertée Incitative

SÉCURITÉ & INFORMATIQUE

APRON: Analyse de PROgrammes Numériques

Compte-rendu de la réunion du 1er février 2005: Interface commune

1 Objectifs et contexte

La réunion du 1er février 2005 a fait suite à celle du 16 décembre 2004 qui avait pour but de lancer l'effort commun de définition d'une interface commune pour domaines abstraits (numériques). La journée a été passée à prendre en compte les remarques d'Antoine Miné et de Bertrand Jeannet, à résumer les impacts que pouvaient avoir des exposés VMCAI'05 et NSAD'05, et à revisiter les décisions et les problèmes décrits dans le compte-rendu de la réunion précédente.

Ce nouveau compte-rendu reprend donc le format du compte-rendu précédent qui est disponible sur le site du projet. Il comporte d'abord la liste des décisions prises d'un commun accord, des commentaires sur la liste d'opérateurs de base proposés par Bertrand Jeannet, une liste de problèmes restant à traiter, et une étude de cas, celui du prédicat *is_ bottom*. L'étude cas n'a pas été modifié. L'annexe de Bertrand Jeannet presque pas. Comme les modifications apportées sont limitées par rapport aux reprises, elles sont indiquées en couleur.

Quand une décision est prise, nous souhaitons conserver les solutions alternatives et toutes les motivations du choix.

Bertrand Jeannet aurait souhaité que ce nouveau compte-rendu soit restructuré. Il proposera un nouveau plan lors de la prochaine réunion qui a été fixée au mardi 15 mars, probablement à Paris.

Le compte-rendu ne rend toujours pas compte de la problématique soulevée par Sebastian Pop. Elle devra être réexaminée en fonction des résultats de cette première journée de réunion pour vérifier que les besoins de Sebastian s'expriment bien en termes d'extracteurs.

Les participants étaient Corinne Ancourt, Jérôme Féret, Nicolas Halbwachs, François Irigoin, Bertrand Jeannet et Sebastian Pop.

2 Décisions

Les décisions prises ne sont pas classées par nature, mais simplement collectées chronologiquement.

1. Dans un premier temps, la problématique CRI de la gestion des liaisons entre dimensions, expressions d'adresse, *abstract locations* et identificateurs est laissé de côté pour commencer par quelque chose de plus simple.

   Peut-être au niveau 1, mais encore douteux.

   On se contente de manipuler un sous-ensemble dans un produit cartésien *et on se restreint même à la puissance* n *d'un ensemble* X, X^n: *Non*.

On admet que chaque dimension est typée. Les types reconnus sont au moins les entiers, les non-entiers, les rationnels et les flottants. La bibliothèque se chargerait de convertir les non-entiers en rationnels ou en flottants.

Les problèmes de précision et donc d'arithmétique modulaire pour les entiers 8, 16, 32 et 64 bits sont laissés à un niveau strictement supérieur à 1, voir dans l'analyseur lui-même au niveau de la sémantique concrète.

Les dimensions sont comprises en 0 et $n-1$ ou bien entre 1 et n. La dimension n de l'espace dans lequel un sous-ensemble abstrait est plongé est accessible par un opérateur.

Des arguments sont compatibles au niveau 0 s'ils ont tous la même dimension. La comptabilité des mappings entre dimensions et adresses abstraites est assurée au niveau 1... sauf s'il s'avère que ca pose un problème d'efficacité.

Ça arrangerait le CRI de pouvoir nommer les dimensions n'importe comment... Niveau 1? Peut-on fournir une implémentation de niveau 1 sans implémentation de niveau 0? Tous les opérateurs de niveau 0 sont-ils aussi présents au niveau 1?

2. La gestion des produits de domaines est laissée de côté, bien que le nommage générique puisse compliquer le problème ultérieurement.

   Chaque bibliothèque instantiant l'interface générique en C a un préfixe propre. D'autres mécanismes sont utilisés en CAML, modules, et en C++, classes. On aura par exemple, PIPS_union, Polka_union.

   Les produits cartésions pourraient être traités au niveau 1.

3. La gestion du *mapping* entre dimensions et zones de mémoire abstraites (e.g. identificateurs dans les cas simples, adresses mémoires, ensemble d'adresses mémoire) et celle du *packing*, i.e. de la définition d'un recouvrement de l'ensemble des adresses abstraites par une famille de sous-ensembles d'adresses abstraites, sont laissées de côté au niveau 1 de l'interface.

   Le niveau 2 de l'interface n'existe pas (encore). Le niveau de base est le niveau 0.

4. Le type numérique entier de base utilisé dans l'analyseur est défini à la compilation, e.g. via une option `-D`, et/ou à l'édition de lien (comme dans PIPS, type *Value, et dans Polka, type* pkint). Ces types sont définis avec un ensemble de macros à normaliser. Ces macros supportent en particulier la détection des dépassements de capacité entière (*overflow*).

   Il n'est pas prévu d'adapter dynamiquement ce type en fonction des pertes/des besoins de précision, e.g. passage en gmp.

   Le type réel,*Real*, de base reste à définir. Ainsi que les interfaces, i.e. les macros (voir au CRI *arithmetique.h* qui est aussi utilisé dans la Polylib).

5. Les signatures fonctionnelles et impératives sont toutes les deux supportées. Les objets sont recyclés ou non, bien que le recyclage ne soit pas utile pour les implémentations denses. *Il n'a pas été décidé d'utiliser le*

passage d'un paramètre supplémentaire ou bien des noms de fonctions différents pour distinguer entre ces deux appels: *Non.* Des noms différents sont utilisés.

Notes de discussion:

- impératif implique-t-il destructif?
- exemple de noms: *union* vs *union_with*;
- en mode impératif, les fonctions C renvoient-elles `void` (IRISA/VERIMAG) ou bien l'objet sur lequel un effet de bord a eu lieu (CRI)?
- il faudrait réduire la liste des doubles interfaces (Bertrand Jeannet);
- que devient le compteur de références? On a décidé de ne pas prendre en compte de compteurs de références.
- François Irigoin: suivre les conventions du package String de C et/ou les conventions du package *Set* de Pierre Jouvelot. Conventions à préciser.

6. Les implémentations partielles sont acceptées, mais elles doivent offrir toutes les signatures prévues pour permettre l'édition de lien et l'échec éventuellement en cas d'appel à une fonction non implantée.

   En particulier, le système de flags peut ne pas être implémenté (?).

   Positionnement du flag *exception* ou du flag *not_implemented* (voir les noms exacts en section 5.2).

7. Vu l'évolution prévisible des architectures des ordinateurs, l'interface permettra d'écrire des implantations compatibles avec les threads, *threadsafe*.

8. Les débordements en capacité, en temps et en espace sont détectés, ou tout au moins, les mécanismes nécessaires à leur paramétrisation et à la récupération des incidents sont définis.

   Voir la section 5.2.

9. On définira au moins une version C de l'interface.

10. Pour certains opérateurs ou même pour chaque opérateur abstrait, il est possible de sélectionner une implémentation particulière, l'implémentation par défaut, l'implémentation donnant le résultat le plus précis ou l'implémentation donnant un résultat le plus rapidement possible. La perte de généricité correspondante semble moins gênante que l'absence de cette flexibilité, au moins pour certains opérateurs particulièrement sensibles. Les trois implémentations de base peuvent être identiques: l'association numéro vers algorithme peut être surjective.

    Notes de discussion:

    - une telle initialisation n'est pas générique;
    - la sélection est-elle faite domaine par domaine?
    - la sélection est-elle faite opérateur par opérateur?

- Bertrand Jeannet propose d'avoir un champ pour chaque opération de l'interface, plutôt que d'avoir à repositionner un flag chaque fois (voir `alg_op` en section 5.2);
- cette discussion s'applique aussi à la définition du *time-out*.

11. Le retour d'un résultat mathématiquement exact peut être identifié à la demande de l'utilisateur (projection entière, union vs enveloppe convexe, test de satisfiabilité en entier,...).

12. La factorisation de l'espace, telle qu'elle est proposée par Grenoble sous la forme d'une partition de l'ensemble des dimensions, n'est pas visible à travers l'interface.

13. Le fait de garder ou non deux représentations duales, e.g. de polydèdres, n'est pas visible à travers l'interface.

14. Une notion de taille abstraite d'objet est définie.

    Un nouvel accesseur doit être définie. La taille d'un objet est un entier positif pouvant être représenté dans un entier signé de 32 bits?

15. Un contexte d'appel, *manager*, est passé explicitement en argument. Il sert à assurer la compatibilité avec les packages de *threads* et à passer toutes les informations opérationnelles, e.g. dépassement en magnitude. Ses *champs* peuvent être IN, OUT, INOUT, CONST (voir section 5.2).

    Il sert aussi à retourner les informations sur les exceptions et sur l'exactitude du résultat.

    On n'a pas à gérer une pile de contextes implicite. On n'a pas de variables globales.

    Le contexte n'est pas intégré aux objets du domaine abstrait, ce qui évite d'avoir à faire une vérification de compatibilité des contextes.

16. Le contenu du contexte d'appel n'est pas visible directement: c'est un objet opaque, fermé, avec des méthodes. Des primitives de construction et d'observation seront définies et fournies.

17. Gestion mémoire: on ne supporte pas de comptage de références.

18. Proposer un système style *varags* plutôt que des listes ou des tableaux d'arguments quand le nombre d'arguments est variable.

    Le mécanisme de *varargs* est plus léger mais moins souple que le mécanisme de tableau parce qu'il impose un nombre d'arguments fixe pour chaque site d'appel. On décide donc d'utiliser des tableaux, et de ne proposer des *varargs* que si quelqu'un peut en montrer l'utilité. La définition d'une structure de liste propre à cette interface est exclue.

    Voir section 3.2.

    Antoine Miné demande de ne pas utiliser de varargs pour passer des vecteurs.

19. Gestion des exceptions: l'interface n'expose aucune exception. Les exceptions internes doivent être traitées par l'implantation et l'information correspondante être masquée ou retournée via la structure *manager*.

20. *Il n'est pas possible de définir certaines dimensions comme entières et d'autres comme rationnelles (demande de Nicolas Halbwachs). Le typage est global (cf. point 1 sur X^n).* Non.

    Les dimensions sont typées individuellement.

3 Liste des opérateurs

La liste des opérateurs proposés par Bertrand Jeannet en décembre 2004 est donnée en annexe. Cette section concerne essentiellement les modifications demandées.

Cette section a été découpée en sous-sections pour faciliter la lecture et les références.

3.1 Confort et performance ou minimalité?

Afin d'assurer performance des implantations et confort de l'utilisateur, l'objectif n'est pas d'avoir une liste d'opérateurs minimales, mais une liste d'opérateurs comportant les combinaisons d'opérateurs de base pouvant être simplifiées ou fréquemment utilisées par un analyseur.

Bertrand Jeannet propose de ne traiter au niveau 0 que les problèmes de performance (*avantage algorithmique fort*) et de reporter au niveau 1 les problèmes de confort, sachant que les problèmes de performance ne sont pas génériques, mais dépendent des domaines abstraits. Ceci rend la séparation plus difficile à définir.

Les problèmes de mise au point, e.g. nommage symbolique des dimensions, et de validation, e.g. normalisation et forme canonique, doivent aussi être pris en compte. Une fonction de nommage des dimensions peut être passée en argument d'une ou de plusieurs des fonctions d'impression d'ensembles abstraits.

Un mécanisme de trace doit aussi être fourni. Par exemple, via une fonction de nom spécifique ou via un pointeur vers une fonction de trace utilisant des *varargs* comme *fprintf*.

3.2 Prise en compte des réductions et des nombres d'arguments variables

Par exemple, on propose un système style *varargs* pour permettre d'enchainer les enveloppes convexes, les projections,... Chaque opérateur pouvant être étendu sous forme d'une réduction, à la manière de *sum* et *prod*, l'est.

Le mécanisme de *varargs* n'est pas aussi souple que le passage d'un tableau de pointeurs et d'une longueur parce qu'il impose la connaissance statique du nombre d'arguments à chaque site d'appel. Il est proposé d'imposer d'abord l'interface par tableau de pointeurs, et, si l'utilité s'en fait sentir, d'ajouter l'interface par *varargs*. Il n'est pas souhaité définir une structure de liste propre à cette interface.

La possibilité de passer plus de deux arguments n'est pas un simple confort. Elle a aussi un impact sur la fonctionalité dans le cas où on souhaite faire une union et où on a donc besoin de savoir si l'enveloppe convexe lui est égale. La remarque est aussi valable pour les suites de projections.

3.3 Nouveaux opérateurs

Antoine Miné suggère l'ajout du *weak_ update* à la liste proposée par Bertrand Jeannet. Le CRI y est favorable. Cette opérateur permet de faire l'union entre les valeurs déjà associées à une dimension et un nouvel ensemble de valeurs. Il est utilisé pour modéliser la mise à jour d'un élément de tableau quand une unique location abstraite est définie pour tous les éléments du tableau.

Deux interprétations de *weak_ update* sont proposées par Antoine Miné:

1. une dimension correspond à plusieurs adresses abstraites (ou bien une adresse abstraite correspond à plusieurs adresses concrètes: par exemple, un tableau est représenté par une seule adresse abstraite);
2. ou bien l'analyseur ne sait pas quelle adresse abstraite est concernée et il transmet donc une liste d'adresses abstraites à l'opérateur.

Voir aussi la *summary dimension* de Reps, Gopan & al. d'après Bertrand Jeannet qui propose de placer cet opérateur au niveau 1.

L'opérateur *weak_ update()* peut être implémenté au niveau 1 s'il n'est pas disponible au niveau 0.

La mise à jour parallèle de plusieurs dimensions, *parallel_ update*, a aussi été envisagée, mais non retenue lors de la première réunion. Bertrand Jeannet le demande maintenant avec les arguments suivants:

- meilleure efficacité dans les cas affines pour les polyèdres;
- éventuellement, meilleure précision pour les expressions non-linéaires en cas de sous-expressions communes;
- meilleure efficacité pour les cas non-linéaires pour les octogones;
- meilleure efficacité pour les cas linéaires non-octogonaux pour les octogones;
- ...

et comme inconvénient la taille de la bibliothèque à implanter. En conséquence, l'opérateur *parallel_ update* est pris en compte.

Bertrand Jeannet demande les opérateurs *split_ dim* et *split_ dims* et *merge_ dims*. Le CRI est interessé dans le cas des tests de dépendance qui nécessitent la prise en compte de deux états.

3.4 Opérateurs de conversion entre treillis

Les fonctions de conversion inter-treillis sont envisagées, *bien que l'utilisation de noms génériques pour les opérateurs en C ne permettent pas d'utiliser simultanement des treillis différents*: Non.

Le problème est résolu dans PIPS en utilisant les noms génériques pour des pointeurs vers des fonctions et en initialisant ces pointeurs en fonction du contexte, mais il n'est pas sûr qu'il soit possible d'empiler les structures qui contiennent ces pointeurs dans PIPS.

Pour réduire le nombre d'opérateurs de conversion, Nicolas Halbwachs propose l'utilisation d'un super-treillis comme pivot. Ce super-treillis pourrait être

la logique du premier ordre, l'arithmétique de Presburger (étendue aux flottants!), un format de l'Ω-library,... Voir Jérôme Leroux, les automates de Boigelot...

3.5 Opérateurs de conversion entre implémentations différentes d'un même treillis

Il faut aussi prendre en compte les fonctions de conversion entre formats différents à treillis équivalents, par exemple, le passage d'une représentation dense à une représentation creuse pour un polyèdre.

Nicolas Halbwachs voit aussi la minimisation, la normalisation et la canonicalisation comme des opérateurs de conversion.

3.6 Divers

Les opérateurs *new* et *free* seront supportés.

Les opérateurs de permutation de dimensions seront supportés au niveau 0, même s'ils n'ont pas de sens (?) pour les représentations creuses? Le maintien de la cohérence ou de la sémantique est assuré au niveau 1 ou par l'analyseur?.

3.7 Sérialisation et désérialisation

La sérialisation et la désérialisation sont ASCII dans PIPS, qui gère le sharing via NewGen et qui évite le sharing dans l'analyseur sémantique, et binaire dans Astrée qui profite du sharing et qui stocke globalement toutes les informations sémantiques glanées pour un programme complet.

3.8 Entrées-sorties

Les fonctions de lecture et d'impression sont nécessaires pour le debugging et pour la validation et pour le benchmarking. Le CRI souhaite que des noms de variables puissent être associés aux dimensions via une fonction passée en argument.

Le CRI souhaite aussi pouvoir lire son format externe creux pour humains.

3.9 Impact VMCAI'05 et NSAD'05

Utilisation d'un template. Papier *Scalable Analysis of Linear Systems using Mathematical Programming* de Sankaranarayan, Sipma et Manna.

Les opérations doivent-elles prendre en compte le template d'arrivée?.

Faut-il utiliser le manager pour passer le template? Y a-t-il un template par objet? Pourrait-on passer par une initialisation d'un treillis paramétré?

Widening Operators for Weakly Relational Numeric Abstractions de Bagnara, Hill, Mazzi et Zaffanella: de nouveaux algorithmes sur octogones pouvant être intégrés via le champ *algorithm* du *manger*.

Papier Microsoft, Leino (?), sur-couche générique gérant des symboles fonctionnels non-interprétés et interfacée avec plusieurs domaines abstaits sous-jacents. Opérateurs de query vers les domaines abstraits pour savoir ce qu'ils savent faire ou non. Passage d'une expression quelconque et/ou d'un operateur quelconque au niveau de l'interface générique.

3.10 Conclusion sur les opérateurs

Deux niveaux d'interface, 0 et 1, ont été mentionnés sans être définis. La réunion et donc ce document, comme le précédent, concerne les opérateurs du niveau 0, sauf mention contraire.

On peut utiliser deux bibliothèques de niveau 0 pour les combiner de manière heuristique et obtenir une nouvelle bibliothèque de niveau 0.

Peut-on mettre au niveau 0 des fonctions spécifiques au domaine abstrait, par exemple une pré-analyse?

Quid de l'égalité? De l'inclusion? De la différence ensembliste?

Quid des opérateurs de query définis par Microsoft?

4 Problèmes

4.1 Problèmes divers

1. Sérialisation: binaire, ASCII ou les deux? No progress.

2. Existe-il in domaine abstrait *congruence* avec une implémentation en C?

   Philippe Granger et François Masdupuy ont-ils laissé quelque chose d'utilisable? Les travaux de François Masdupuy n'ont jamais pu être intégré dans PIPS

   Où en est-on avec les Z-polyèdres dans la Polylib?

   Apparemment, non.

3. Comment doit-on traiter les ensembles de points entiers contenus dans un polyèdre rationnel? Est-ce que cela fait partie de l'interface? Est-ce que cela défini un autre treillis, comme pour l'intersection avec pZ? Pour le moment, on intègre cette subtilité de manière discrète dans l'interface.

   Rappel: la projection, le test à vide, la minimisation,... sont différents

   Les dimensions sont maintenant typées. Ce qui ne doit pas forcément imposer des opérateurs entiers, par exemple pour l'élimination de redondance entière qui n'est pas monotone décroissante pour les rationnels.

4. Support de la sous-approximation: au CRI d'en montrer l'intérêt lors d'une prochaine réunion. Pour le moment, on propose sur-approximation et exactitude.

   No progress.

5. Mécanisme de détection des *timeouts*: *threads concurrentes*? Exception? Discussion au sein de la Polylib pour avantages et inconvénients.

   No progress.

   Peut-on se suffire d'une unique valeur de time-outs pour tous les opérateurs?

6. Typage des dimensions: *int, float, bool, char **. Plutôt typage global? Comment?

   Typage par dimension, restreints à certains types (cf. points 20 et 1). Voir *Value* et *pkint*, point 4.

7. Utilisation de préfixes pour distinguer les différentes bibliothèques en C? Par exemple, OCT pour octagone, PPL pour Parma, HQ proposé par Duong.

   Oui, renforcé par Bertrand Jeannet. Voir 2. Préfixes envisagés: POLKA, C3, PIPS, POLYLIB.. Ne pas utiliser de macros mais une moulinette de désambiguation?!?

8. Certaines heuristiques peuvent nécessiter des paramètres. Comment les passe-t-on?

   En ajoutant des champs dans *manager* ou dans le champs *internal* de *manager* (cf. point 5.2).

9. Pour décider de la compatibilité entre deux arguments, comment peut-on se passer du mapping entre dimensions et adresses abstraites? En mettant ce test au niveau 1?

   Oui.

10. Souhaite-t-on avoir comme observateur un accès aux contraintes internes à l'objet? Compatibilité avec la généricité? Observateur renvoyant le plus petit objet contraignant une variable/dimension donnée?

    Utilisation d'un super-treillis?

11. Le CRI souhaite que tout type utilisé comme résultat d'une fonction inclut une constante *xxx_undefined*. La discussion n'a pas été terminée lors de la première réunion sur ce point, l'étude de *is_bottom* ne s'y prêtant pas.

    La valeur *undefined* n'apparait pas dans les domaines abstraits.

    Discussion à reprendre:

    - l'information ne devrait-elle pas être portée dans le *manager*? Non pour le CRI qui se place dans une perspective de mise au point.
    - *undefined* n'est-il pas *top*? Le CRI doit effectivement utiliser parfois *undefined* comme *top* pour alléger l'implémentation. La sémantique d'*undefined* est-elle bien définie? Top? Not implemented? Exception occured? Utilisation des codes d'exception?
    - *fail_with*?!?
    - Jérôme: utilisation pour des effets de bord détectés mais non traités.
    - propagation des erreurs? des exceptions? remises à zéro du *manger*?

4.2 Types annexes

Au-delà du type de base du treillis lui-même, l'interface devrait prendre en compte:

1. le type *Value*, *pkint*, *abstract_float*?

   Par exemple, $\pi = [3.14, 3.15]$

2. le type *dimension*,

3. le type *expression*,

4. le type *expression affine*,
5. le type *prédicat affine*, e.g. plus grand, égal,...
6. le type *intervalle*,
7. le type *expression affine à terme constant intervalle* (les coefficients sont des scalaires, seul le terme constant est un intervalle),

   Revoir l'utilité pour les flottants avec Antoine Miné.
8. le type *expression affine à coefficients intervalles* (tous les coefficients ainsi que le terme constant sont des intervalles), utiles pour traiter les flottants,

   Note: ce type peut être représenté syntaxiquement par deux expressions affines, une portant les minima et l'autre les maxima.
9. François Irigoin se demande si on pourrait avoir un domaine de QUAST (Paul Feautrier).
10. ...

La prise en compte des expressions générales est gênante pour l'utilisabilité puisque chaque analyseur définit différemment le type expression. Mais les expressions affines sont trop restrictives et reportent la linéarisation des expression au niveau supérieur alors qu'elle dépend du treillis utilisé.

4.3 Extracteurs

Il a semblé que les extracteurs dépendaient du treillis sous-jacent: non.

On a le droit de définir tous les extracteurs dont on a besoin en analyse et transformation de programmes. En effet les extracteurs se réfèrent au domaine concret, ou plus exactement à la concrétisation d'une valeur abstraite. C'est cela qui leur donne un sens quelque soit le domaine abstrait ou la librairie sous-jacente.

En fonction des domaines abstraits, la réponse doit toujours pouvoir être *don't know*, i.e. *undefined* pour le CRI, i.e. *top*?

Par exemple, on peut demander pour une variable ou pour un expression entière ou flottante:

1. son signe (pourrait être au niveau 1 et utiliser l'intervalle de valeurs),
2. sa parité,
3. sa valeur (pourrait être au niveau 1 et utiliser l'intervalle de valeurs),
4. une borne supérieure ou une borne inférieure,

   Il faut s'accorder sur le codage de plus et moins l'infini dans les différents types. Utilisation du flag exact?!?
5. un intervalle de valeurs,
6. l'intersection d'un intervalle de valeurs et d'une congruence,
7. une expression affine égale,

8. une expression quelconque égale,
9. la validité d'un prédicat,

   François Irigoin voudrait pouvoir demander la validité de la négation d'un prédicat, mais les autres participants voient ceci comme du sucre syntaxique.
10. etc...

On peut aussi demander la valeur d'un prédicat, au moins d'un prédicat affine.

Le typage a-t-il un impact sur ceci?

Nicolas Halbwachs suspecte que nous travaillons trop avec quelque chose qui est pratiquement une chaîne de treillis. Il serait bon, pour éviter les surprises, d'envisager les congruences et les BDDs pour les booléens.

4.4 Généricité des constructeurs et de l'interface

Les constructeurs proposés sont-ils trop orientés polyèdres? Les expressions affines sont-elles trop prises en compte par rapport aux expressions générales?

L'attention portée aux opérateurs de complexités exponentielles est-elle trop grande? Les conséquences en terme de lourdeur sont-elles acceptables pour des treillis simples comme les intervalles entiers?

5 L'opérateur *is_ bottom*

L'opérateur *is_ bottom*, qui renvoie, par exemple, vrai si l'ensemble des valeurs abstraites est vide, a été choisi du fait de sa simplicité. Simplicité apparente... Sa complexité est exponentielle et sa sémantique est différente en entier et en rationnel.

Il a permis de progresser sur la notion de *manager*. La réunion ne devait pas aboutir à une définition, mais à un ensemble de définitions possibles

5.1 Valeur retournée

1. Bertrand Jeannet propose de retourner un booléen ternaire, avec les valeurs {true, false, dontknow}. François Irigoin propose d'étendre cette valeur pour prendre en compte les débordements de capacité numérique et autres problèmes opérationnels.
2. au CRI, seulement des booléens sont retournés. La valeur *true* a pour sémantique *vrai*, la valeur *false* a pour sémantique je ne sais pas. Du coup, il faut définir une autre fonction, *is_ not_ bottom*.
3. Bertrand Jeannet et Nicolas Halbwachs trouvent que cela rend l'interface difficile à comprendre puisque *not(is_ bottom)* est différent de *is_ not_ bottom*. Nicolas suggère d'appeler la fonction *is_ known_ bottom* pour lever l'ambiguité et d'ajouter *is_ known_ not_ bottom*.
4. Comme il faut aussi être *threadsafe*, passer des arguments opérationnels et récupérer des informations opérationnelles, il est proposé d'utiliser une structure de données opaque, appelée *manager*.

5. Le CRI aurait préféré pour des raisons de lisibilité que les prédicats aient des noms se terminant par *_p* comme *is_known_bottom_p* parce que *is_xxx* est utilisé en interne pour gérer les unions.

   Le suffixe *_t* est utilisé dans Polka pour les noms de types.

5.2 Les informations contenues dans le *manager* pour *is_bottom*

Le contenu exact de *manager* doit être partiellement ignoré par l'utilisateur. Il dépend de la bibliothèque de *threads* utilisée et des besoins particuliers des implantations, comme celle de Bertrand Jeannet.

Antoine Miné demande que le manager soit restructuré pour bien mettre en évidence ce qui est IN et ce qui est OUT et ce qui est INOUT, fréquemment ou rarement, et ce qui est CONST. Ceci dit, comme il s'agit d'une structure de données opaque...

```
struct manager {
    struct internal * internal; // Bertrand Jeannet, RFFU: Reserved for
                                // Future Use?
    struct pool * pool;  // thread library: support for memory allocation

    int timeout;         // unity? millisecond? One for each operator?
    int max_object_size; // in abstract object size unit

    int algorithm;       // Algorithm selection: 0 is default algorithm
                         //                      MAX_INT is most accurate
                         //                        available
                         //                      MIN_INT is fastest
                         //                        available
                         //                      otherwise, algorithm n,
                         //                        with no speed or
                         //                        accuracy meaning for n
                         // One such field for each operator? int alg_op?

    bool flag_exact_wanted; // Return information about exactitude if
                            // possible
    bool flag_int_wanted;// Abstract set is a set of integer points, if
                         // not rational points. Meaning for intervals on
                         // floats?


    bool flag_time_out;  // timeout detected
    bool flag_out_of_space; // the size of the key (?) object exceeds
                         //    max_object_size
    bool flag_overflow;  // Magnitude overflow
    bool exception;      // If true, some operational exception was raised

    bool flag_exact;     // Result is mathematically exact or not:
                         // projection, satisfiability, union
}
```

Les opérateurs sur le *manager* n'ont pas été définis.

Duong Nguyen se demande si on ne devrait pas aussi signaler la non-implémentation d'une fonction par ce biais.

Bertrand Jeannet propose qu'une fonction appelée avec une exception levée dans l'argument manager fasse un retour immédiat. Son idée est d'éviter d'avoir à tester systématiquement le code de retour. François Irigoin préfèrerait qu'une erreur interne soit levée dans un tel cas, afin de faciliter la mise au point.

5.3 Autres opérateurs

Il reste à en discuter à la lumière des premières idées émises pour *is_bottom*. Comme un objet sera usuellement retourné, il sera difficile de généraliser l'idée du booléen ternaire *tbool*, d'où son abandon.

6 Conclusion

Différents niveaux d'interface sont prévus. Cette deuxième réunion, comme la première, a été essentiellement consacrée au niveau le plus bas, le niveau 0.

Aucun programme de travail commun n'a été défini, si ce n'est de fournir un compte-rendu de réunion et de réfléchir à son contenu. Le CRI doit nettoyer les bibliothèques et interfaces de PIPS, et faire un effort de clarification à propos des besoins de Sebastian Pop. Les problèmes de compatibilité avec l'existant, problèmes qui ont déjà été étudiés par Duong Nguyen, devront être pris en compte à nouveau dans une prochaine réunion.

Le problème posé par la prise en compte des *threads* doit être exploré.

La prochaine réunion aura lieu le 15 mars, à Paris ou à Grenoble.

Annexe: Liste des opérateurs proposée par Bertrand Jeannet (Version de décembre 2004, à peine altérée)

Cette liste de déclarations ML n'a pas été (complètement) mise à jour en fonction des besoins exprimés par les uns et les autres. Les opérateurs n'ont pas non plus été discuté en détail, faute de temps, mise à part *is_ bottom*.

L'opérateur *minimize* réduit la taille de l'objet, tandis que l'opérateur *canonicalize* renvoie une représentation *canonique* qui peut être plus grande. L'existence d'une forme canonique n'est pas évidente pour tous les treillis.

L'opérateur *minimize*, appliqué avec l'option *int*, peut renvoyer une polyèdre rationnel contenant plus de points rationnels que son argument (expérience CRI, François Irigoin).

L'utilisation des tableaux comme arguments n'enthousiasme pas le groupe.

François Irigoin aurait voulu qu'on puisse demander la négation d'une condition linéaire pour ne pas avoir à traiter la diséquation. Bertrand Jeannet fait remarquer qu'il existe déjà suffisament d'opérateur pour l'exprimer et propose que ce soit remonté à l'interface de niveau 2. François Irigoin n'est pas convaincu que ce soit indépendant du treillis sous-jacent.

Il ne semble pas y avoir de support pour l'accélération, pour le *forget*, pour l'*undefined*.

L'exportation du type *generator* n'a pas semblée très générique. De plus *Point* pourrait être appelé *Vertex*.

Certains des commentaires faits durant l'exposé de Bertrand ont été perdus, mais beaucoup sont listés dans les sections précédentes (*weak_ update, parallel_ update*, forme affine avec coefficients de type intervalle,...).

FI: JE COMPTE SUR LES AUTRES PARTICIPANTS POUR BOUCHER LES TROUS. MERCI!

```
type num

module Linear = struct
  type linexpr = num array
  type affexpr = {
    lin : linexpr;
    cst : num;
  }
  type typcond = Sup | SupEg
  type cond = typcond * affexpr

  type generator
    | Point of affexpr
    | Ray of linexpr
    | Line of linexpr
end

type bound = num option
type interval = {
  inf : bound
```

```
  sup : bound
}


module type A = sig
  type t
    (** The type of abstract values *)

  (** *) // Which functions are "idempotent"? f o f = f?
  val minimize : t -> unit // Minimal size for storage
  val canonicalize : t -> unit // Unique representation:
                               //  e.g. non-regression testing
                               // Or required for some operators
  val normalize : t -> unit // Added. E.g. sharpen integer constraints,
                            // x-2epsilon -> x-epsilon

  (** Constructors and basic operations *)
  val bottom : int -> t
  val top : int -> t
  val meet : t -> t -> t
  val join : t -> t -> t
  val meet_linearcond : t -> Linear.cond -> t
  val meet_nonlinearcond : t -> NonLinear.cond -> t

  (** Tests *)
  val is_bottom : t -> bool
  val is_top : t -> bool
  val is_leq : t -> t -> bool
  val is_eq : t -> t -> bool
  val is_leq_linearcond : t -> Linear.cond -> bool

  (** Accessors *) // Il y en a (beaucoup) plus. 20 domaines abstraits
                  // utilisés par ASTRÉE
  val dim : t -> int
  val to_constraints : t -> Linear.cond list
  val to_generators : t -> Linear.generator list
  val to_bounds : t -> interval array
  val bound_linear_expr : t -> Linear.affexpr -> interval

  (** Widening *)
  val widening : t -> t -> t
    (** Standard widening *)
  val limited_widening : t -> t -> Linear.cond list -> t
    (** Widening with thresholds *)

  (** Projections, Embeddings, ... *)
  (* At the end *)
  val add_dims_and_embed : t -> int -> t
  val add_dims_and_project : t -> int -> t
  val del_dims : t -> int -> t
```

```
  (* Anywhere *)
  val add_permute_dims_and_embed : t -> int -> int array -> t
  val add_permute_dims_and_project : t -> int -> int array -> t
  val permute_del_dims : t -> int -> int array -> t

  (** Transformations *)
  val assign_var_linear : t -> int -> Linear.affexpr -> t
  val substitute_var_linear : t -> int -> Linear.affexpr -> t
  val assign_vars_linear : t -> (int * Linear.affexpr) array -> t
  val substitute_vars_linear : t -> (int * Linear.affexpr) array -> t

  // weak_update
  // weak_updates
  // parallel_updates: substitute_vars (Antoine Miné)

end
```

Glossaire?