Analyse de dépendances ML pour les évaluateurs de logiciels critiques.

Vincent Benayoun

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS | le cnam

École Doctorale Informatique, Télécommunication et Électronique

CEDRIC - Centre d'Étude et De Recherche en Informatique et Communications

THÈSE DE DOCTORAT

présentée par : **Vincent BENAYOUN**

soutenue le : **16 MAI 2014**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique / Preuve formelle**

Analyse de dépendances ML pour les évaluateurs de logiciels critiques

THÈSE DIRIGÉE PAR

Mme. DUBOIS Catherine — *Professeur, ENSIIE - CEDRIC*
M. PESSAUX François — *Maître de conférences, ENSTA ParisTech*

RAPPORTEURS

M. DI COSMO Roberto — *Professeur, Université Paris Diderot*
Mme. BLAZY Sandrine — *Professeur, Université de Rennes 1*

EXAMINATEURS

M. RAJCHENBACH-TELLER David — *Ingénieur R&D, Mozilla*
M. POTET Marie-Laure (Présidente) — *Professeur, Ensimag - VERIMAG*

Remerciements

Je tiens tout d'abord à remercier vivement Catherine Dubois, qui m'a guidé tout au long de mon parcours depuis mon passage à l'école d'ingénieur jusqu'à aujourd'hui. Elle m'a fait découvrir OCaml sous un nouveau jour et m'a ainsi permis d'apprécier à leur juste valeur les bienfaits du typage fort. Elle m'a ensuite donné le goût des méthodes formelles et m'a introduit dans l'équipe CPR du CNAM où j'ai découvert le monde de la recherche en réalisant ma première véritable modélisation formelle sous la direction de Maria-Virginia Aponte et de Marianne Simonot, que je remercie également pour cette expérience très intéressante. Catherine a ensuite pris la direction de ma thèse lors mon entrée à MLstate jusqu'à la rédaction de ce manuscrit. Elle a su gérer les nombreuses péripéties de mon parcours de thèse et m'accompagner jusqu'au bout en me prodiguant de nombreux conseils précieux.

Je remercie Roberto Di Cosmo et Sandrine Blazy d'avoir accepté le rôle de rapporteurs de cette thèse et d'avoir fourni les efforts nécessaires à compréhension fine de mon travail. Merci pour leurs remarques judicieuses et leurs questions pertinentes. Je remercie également Marie-Laure Potet pour son implication dans mon jury de thèse en tant que présidente.

Mon parcours de thèse à commencé à MLstate. Je remercie donc Henri Binsztok de m'avoir proposé de rejoindre MLstate en thèse CIFRE. J'ai eu l'occasion d'y travailler avec une équipe géniale. Je souhaite tout particulièrement remercier mes encadrants Catherine, David Teller, Pierre Courtieu et Adam Koprowski qui m'ont beaucoup apporté. David m'a guidé au quotidien dans toutes mes activités à MLstate, il s'est battu sans cesse pour la réussite de ma thèse, comme il l'a fait pour tout le projet OPA en général. C'est un homme qui a les épaules pour être partout à la fois et s'occuper de chaque chose avec intelligence et efficacité. Pierre et Adam m'ont appris à maîtriser Coq, ce qui m'a beaucoup aidé pour réaliser les preuves que je présente dans ce manuscrit. Je remercie également tous les membres de l'équipe avec qui j'ai travaillé à MLstate, en particulier Mikołaj Konarski avec qui j'ai eu grand plaisir à travailler et pour qui j'ai un grand respect.

Catherine m'a donné toute sa confiance et ses encouragements en me permettant de continuer mon parcours de thèse après mon départ de MLstate. Elle m'a proposé avec le concours de Thérèse Hardin un nouveau sujet de thèse que je présente aujourd'hui dans ce manuscrit. Je tiens à remercier sincèrement Thérèse qui a suivi mon parcours d'un œil bienveillant et m'a donné des conseils judicieux aux moments opportuns. Sur son conseil, je me suis dirigé vers MLstate et sur son conseil, j'ai pris la suite de la thèse de Philippe Ayrault. Je remercie d'ailleurs Philippe pour le temps qu'il m'a consacré afin de m'expliquer les détails de son analyse de dépendances ainsi que les besoins spécifiques, en tant qu'évaluateur de logiciels critiques, qui l'ont amené à faire certains choix. Je remercie François Pessaux d'avoir assuré l'encadrement de ma thèse aux côtés de Catherine. Il a su me transmettre son expérience et sa rigueur intellectuelle. En particulier il m'a permis de mieux comprendre les besoins des évaluateurs de logiciels critiques en partageant son expérience dans le domaine.

Ces remerciements ne pourraient être complets sans remercier mes parents et mon frère qui m'ont toujours apporté leur soutien et leurs encouragements ainsi que ma chère femme qui me soutient par son amour et ses délicates attentions et nos deux petites princesses, qui par leurs jolies voix m'ont aidé à ne pas somnoler devant l'ordinateur pendant les nombreuses nuits passées en compagnie de Coq et de Latex.

Résumé

Les logiciels critiques nécessitent l'obtention d'une évaluation de conformité aux normes en vigueur avant leur mise en service. Cette évaluation est obtenue après un long travail d'analyse effectué par les évaluateurs de logiciels critiques. Ces derniers peuvent être aidés par des outils utilisés de manière interactive pour construire des modèles, en faisant appel à des analyses de flots d'information. Des outils comme SPARK-Ada existent pour des sous-ensembles du langage Ada utilisés pour le développement de logiciels critiques. Cependant, des langages émergents comme ceux de la famille ML ne disposent pas de tels outils adaptés. La construction d'outils similaires pour les langages ML demande une attention particulière sur certaines spécificités comme les fonctions d'ordre supérieur ou le filtrage par motifs. Ce travail présente une analyse de flot d'information pour de tels langages, spécialement conçue pour répondre aux besoins des évaluateurs. Cette analyse statique prend la forme d'une interprétation abstraite de la sémantique opérationnelle préalablement enrichie par des informations de dépendances. Elle est prouvée correcte vis-à-vis d'une définition formelle de la notion de dépendance, à l'aide de l'assistant à la preuve Coq. Ce travail constitue une base théorique solide utilisable pour construire un outil efficace pour l'analyse de tolérance aux pannes.

Mots clés : analyse de dépendances, logiciels critiques, langages fonctionnels, Coq, preuve de correction, analyse statique, interprétation abstraite

Abstract

Critical software needs to obtain an assessment before commissioning in order to ensure compliance with standards. This assessment is given after a long task of software analysis performed by assessors. They may be helped by tools, used interactively, to build models using information-flow analyses. Tools like SPARK-Ada exist for Ada subsets used for critical software. But some emergent languages such as those of the ML family lack such adapted tools. Providing similar tools for ML languages requires special attention on specific features such as higher-order functions and pattern-matching. This work presents an information-flow analysis for such a language specifically designed according to the needs of assessors. This analysis is built as an abstract interpretation of the operational semantics enriched with dependency information. It is proved correct according to a formal definition of the notion of dependency using the Coq proof assistant. This work gives a strong theoretical basis for building an efficient tool for fault tolerance analysis.

Keywords : dependency analysis, critical software, functional languages, Coq, proof of correctness, static analysis, abstract interpretation

Table des matières

Table des figures

Introduction

Depuis son apparition jusqu'à aujourd'hui, l'informatique s'est développée à une vitesse phénoménale. Les systèmes informatiques sont devenus de plus en plus complexes et « l'intelligence » de ces systèmes a progressivement migré du matériel vers le logiciel. Les logiciels sont maintenant présents partout et s'occupent de tâches de plus en plus importantes et complexes. On les retrouve aussi bien dans nos téléphones portables que dans nos voitures. Ils nous permettent de nous divertir, de communiquer mais aussi de piloter des avions ou de gérer des systèmes bancaires.

Compte tenu des conséquences dramatiques que peut engendrer une défaillance dans certains logiciels, une rigueur particulière doit être exigée pour la conception et le développement de ces logiciels dits « critiques ».

Les logiciels critiques sont présents dans divers domaines dont le nucléaire, le transport (aéronautique, ferroviaire et automobile), le médical, le bancaire et l'armement. Certains dysfonctionnements peuvent causer des pertes financières de l'ordre de plusieurs centaines de millions d'euros, comme le montrent les exemples récents de dysfonctionnements [Kni12] dans des logiciels automatiques de spéculation boursière. D'autres peuvent mettre en jeu des vies humaines, comme on a pu en être témoin lors des accidents de voiture provoqués par des dysfonctionnements du régulateur de vitesse du véhicule. Les conséquences d'un dysfonctionnement dans le logiciel de pilotage d'un train, d'un avion ou d'une fusée peuvent se révéler dramatiques. La destruction de la fusée Arianne V lors de son vol inaugural en 1996 est un exemple bien connu de catastrophe provoquée par un dysfonctionnement logiciel. Le domaine médical n'est pas non plus épargné. La machine de radiothérapie Therac-25 a provoqué le décès de plusieurs patients à cause d'un dysfonctionnement logiciel passé inaperçu pendant plusieurs années.

Évaluation des logiciels critiques

La prise de conscience de l'importance d'un haut niveau de sûreté de fonctionnement des logiciels critiques a conduit les autorités à établir différentes normes selon les différents domaines d'application (EN-50128 [Sta99] pour le ferroviaire, DO-178B/C [RTC92] pour l'avionique, ...). Ces normes spécifient les exigences que doivent respecter les logiciels critiques de la conception à la mise en fonctionnement voire au démantèlement, en fonction de leur niveau de criticité. La vérification de la conformité d'un logiciel aux normes en vigueur nécessite de faire appel à un évaluateur indépendant engageant personnellement sa responsabilité pénale. Celui-ci va alors produire une démonstration du respect des normes par le système critique évalué. Cette démonstration est constituée d'un ensemble de documents ayant pour but de justifier un certain niveau de confiance. La justification est basée sur l'utilisation de méthodes de développement rigoureuses, de tests unitaires et d'intégration (avec un taux de couverture raisonnable) voire de preuves mathématiques dans les cas les plus critiques. La démonstration produite sera alors soumise à l'approbation d'une autorité de tutelle qui délivre alors l'autorisation de mise en exploitation.

Outils automatiques, interactifs et corrects

La tâche d'évaluation d'un logiciel critique est particulièrement coûteuse en temps et en main d'œuvre. On estime [Ayr11] le coût du développement d'un logiciel critique entre 6 et 10 hommes-années pour mille lignes de code dont environ 60% sont attribuables aux phases de vérification, validation et évaluation. Des outils automatiques permettant de simplifier une partie de ce travail de vérification peuvent donc être fortement appréciés, moyennant le respect de certaines contraintes dues à leur utilisation.

Ces outils automatiques doivent être utilisés uniquement comme une aide pour l'évaluateur. L'interactivité est primordiale dans ce type d'outil pour que l'évaluateur puisse utiliser son expertise pour guider l'outil et obtenir ainsi les résultats dont il a besoin.

La question de la correction des outils utilisés par l'évaluateur de logiciels critiques est une question essentielle. Pour que l'évaluateur puisse faire confiance à l'outil et utiliser les résultats fournis par celui-ci pour prendre des décisions, il faut que cet outil bénéficie d'un

haut niveau de fiabilité. Par exemple une preuve formelle de la correction de l'outil peut être un gage de fiabilité acceptable par l'évaluateur.

Pourquoi une analyse de dépendances pour ML ?

Divers outils peuvent permettre d'alléger le travail d'évaluation d'un logiciel critique. Parmi ces outils, l'analyse des dépendances dans le code source du logiciel a une importance particulière. En effet, ce type d'analyse peut permettre à l'évaluateur d'identifier les parties du programme qui n'ont aucune influence sur la partie critique du système évalué. L'évaluateur peut alors se reposer sur la fiabilité de l'outil d'analyse pour s'épargner un long travail de vérification sur les parties du programme pour lesquelles il a une garantie qu'elles ne peuvent en aucune manière influencer les fonctionnalités critiques du système : elles ne font alors pas partie de la « cible d'analyse » (*Target of Analysis*).

D'autre part, l'analyse des dépendances du code source peut permettre à l'évaluateur de visualiser les interactions entre les différentes parties du logiciel. Il acquiert ainsi une meilleure compréhension du logiciel et de sa structure. En particulier, ces informations se révèlent être précieuses pour construire la « modélisation fonctionnelle du système » qui constitue une des étapes de l'AMDEC [X6086, MSA80], méthode d'analyse utilisée fréquemment par les évaluateurs de systèmes critiques.

Les analyses de dépendances, comme les analyses de flots d'information, les analyses de non-interférence, ou encore les analyses d'impact, sont des analyses opérant sur le code source des logiciels. Elles sont dépendantes du langage de programmation utilisé pour développer le logiciel à évaluer. Certains langages de programmation sont mieux outillés que d'autres. Des outils d'analyse de dépendances existent pour le langage C, particulièrement répandu. On peut citer par exemple le plug-in d'analyse d'impact de l'environnement d'analyse de code source Frama-C [CKK+12]. Même si cet outil n'est pas conçu spécifiquement pour répondre aux besoins des évaluateurs de logiciels critiques, il peut être utilisé dans une certaine mesure pour aider à la construction d'une modélisation fonctionnelle. Le langage SPARK [Bar03], sous-ensemble du langage ADA, est lui aussi équipé d'un outil permettant de spécifier les dépendances d'un programme et de les vérifier.

D'autres langages, quant à eux, ne bénéficient pas d'un tel outillage. C'est le cas notamment des langages fonctionnels de la famille ML qui voient leur utilisation commencer à se répandre dans le cadre du développement de logiciels critiques. Certains développements industriels utilisent d'ores et déjà des langages de la famille ML. La société Jane Street Capital développe des logiciels critiques d'investissements financiers en OCaml ([MW08]) manipulant des centaines de millions de dollars chaque jour. Le générateur de code embarqué certifié SCADE [PAC+08] est lui aussi écrit en OCaml. De nombreux autres logiciels critiques sont développés en utilisant les langages de la famille ML comme l'analyse statique Goanna [FHJ+06] permettant de rechercher des erreurs dans du code source critique écrit en C/C++, ou encore la plate-forme logicielle Apropos [JES00] développée par LexiFi pour la tarification et la gestion de produits financiers.

Contribution

Dans cette thèse, nous nous intéressons à l'analyse des dépendances d'un logiciel. Le langage sur lequel opère notre analyse est un langage comprenant les principales fonctionnalités présentes dans les langages de la famille ML, c'est-à-dire des fonctions d'ordre supérieur, des constructeurs de données algébriques et du filtrage par motif. Nous participons ainsi à combler le manque d'outils associés aux langages de la famille ML.

L'analyse que nous présentons a été spécifiquement conçue dans le but de répondre aux besoins particuliers des évaluateurs de logiciels critiques. Ces besoins, présentés ci-dessous, ont été identifiés suite au retour d'expérience de Philippe Ayrault, lui-même évaluateur de logiciels critiques dans le domaine ferroviaire [ABDP12, Ayr11].

Interactivité et souplesse Notre analyse permet à l'évaluateur d'analyser n'importe quelle partie du code source du logiciel critique à évaluer, en fournissant un environnement d'analyse qui est une spécification plus ou moins précise de l'environnement d'évaluation. Cet environnement d'analyse offre à l'évaluateur la possibilité de paramétrer l'analyse en donnant pour chaque identificateur présent dans l'environnement une valeur plus ou moins abstraite. Il est ainsi possible pour l'évaluateur d'analyser les programmes à différents niveaux d'abstraction, de manière à avoir tantôt une vision globale des différents composants

du système, tantôt une vision plus fine d'un composant en particulier. C'est l'expertise de l'évaluateur qui donne la ligne directrice du processus de vérification en utilisant l'analyse à différentes étapes du processus, à chaque fois en utilisant le niveau d'abstraction qui convient.

Haut niveau de confiance Pour garantir la fiabilité de notre analyse, nous sommes remontés jusqu'aux fondements de la notion de dépendance. Nous donnons une définition formelle de la notion de dépendance en nous appuyant directement sur la sémantique opérationnelle du langage. Nous avons ensuite construit, étape par étape, une preuve formelle permettant de fournir à l'évaluateur une garantie de la propriété de correction dont il a besoin en établissant formellement le lien entre notre analyse et la notion de dépendance. Une grande partie de la preuve a été réalisée et vérifiée à l'aide de l'assistant à la preuve COQ. Nous fournissons dans le présent manuscrit une preuve papier de la partie restantes. L'intégralité du développement Coq réalisé est disponible librement à l'adresse suivante : `https://github.com/vincent-benayoun/PhD-thesis`.

Notre analyse possède une spécificité supplémentaire : la prise en compte de l'impact d'un dysfonctionnement sur la terminaison du programme. En consultant le résultat de l'analyse d'un programme, on peut savoir non seulement quelles sont les parties du programme ayant une influence sur les valeurs calculées mais également celles ayant une influence sur la terminaison. Cette dernière information est intéressante pour l'évaluateur de logiciels critiques dans la mesure où la non-terminaison d'un programme peut constituer un réel risque de dysfonctionnement.

Plan de la thèse

La structure du manuscrit est la suivante :

Chapitre 1 : État de l'art Nous présentons le contexte scientifique de cette thèse en mettant en relation notre analyse de dépendances avec d'autres travaux. En premier lieu, nous expliquons en quoi notre travail s'inscrit dans la continuité de travaux existants dans le domaine. Puis nous explicitons les liens entre notre analyse de dépendances et

quelques travaux similaires portant sur la non-interférence et l'analyse de teintes. Enfin montrons en quoi notre analyse s'inscrit dans le cadre général de l'interprétation abstraite de programmes.

Chapitre 2 : Notion de dépendance Après avoir présenté le langage sur lequel opère notre analyse ainsi que sa sémantique opérationnelle, nous présentons une définition formelle des notions de base utilisée pour exprimer ce que nous appelons ***dépendances d'un programme***. Nous commençons par présenter l'injection de valeur qui nous permet de représenter formellement les points de programme dont on veut connaître l'impact. Nous définissons alors formellement la notion d'impact en elle-même, qui est double : l'impact sur la terminaison et l'impact sur la valeur du programme. Nous pouvons ensuite introduire la notion de dépendance qui se divise également en deux : les dépendances de terminaison et les dépendances de valeur.

Chapitre 3 : Analyse dynamique Une fois la notion de dépendance formalisée, nous définissons une analyse dynamique permettant de calculer les dépendances d'un programme. Ces dépendances seront fournies par l'analyse dynamique sous forme d'annotations sur chacun des sous-termes de la valeur du programme. Nous appelons cette analyse dynamique ***sémantique instrumentée*** et nous l'introduisons à l'aide de deux sémantiques intermédiaires qui nous permettent d'établir la correction de cette sémantique instrumentée.

Chapitre 4 : Analyse statique Ce chapitre touche enfin au but de cette thèse : présenter notre analyse statique des dépendances d'un programme. Afin d'atteindre ce résultat, nous partons de l'analyse dynamique présentée au chapitre précédent. Nous introduisons alors, à l'aide de deux nouvelles sémantiques intermédiaires, une interprétation abstraite de l'analyse dynamique que nous appelons ***sémantique abstraite***. C'est cette dernière sémantique qui constitue notre analyse statique de dépendances.

Chapitre 5 : Implémentation et preuve Nous présentons dans ce dernier chapitre les développements effectués autour de l'analyse de dépendances. D'une part, nous exposerons le développement au sein de l'atelier de preuve Coq, qui a permis de formaliser les différentes

sémantiques présentées dans cette thèse ainsi que de construire et vérifier leurs preuves de correction. D'autre part, nous discuterons des différents prototypes réalisés.

Conclusion et perspectives Enfin, nous présentons un bilan du travail réalisé en ouvrant la voie vers de possibles développements futurs. En particulier, nous parlerons des possibilités d'extension du langage analysé et des pistes d'optimisation de l'analyse (performance et précision).

Chapitre 1

État de l'art

1.1 Analyse de dépendances pour les logiciels critiques

1.1.1 Les langages ML

Cette thèse s'inscrit dans la suite de la thèse de P. Ayrault [Ayr11, ABDP12] soutenue en 2011. Elle reprend l'analyse statique de dépendances qui y était présentée, la rend plus générale en y apportant un certain nombre de modifications et en propose une preuve formelle vérifiée mécaniquement à l'aide de l'assistant à la preuve Coq.

La première modification réside en la manière d'annoter les programmes pour permettre à l'évaluateur de logiciel critique de désigner les points de programme dont il souhaite connaître l'impact. Dans l'analyse de P. Ayrault, on ne pouvait annoter que les sous-expressions du programme explicitement nommées par une instruction de liaison (let x = e). Dans l'analyse que nous présentons, toute sous-expression du programme peut être marquée pour analyser son impact sur l'évaluation du programme.

La notion de marquage concret/abstrait présentée par P. Ayrault effectuait simultanément deux opérations distinctes. Premièrement, elle permettait de désigner une sous-expression pour connaître son impact. Deuxièmement, elle cachait la valeur de la sous-expression désignée pour la considérer comme une boîte noire. Autrement dit, elle lui appliquait une sorte d'abstraction. Notre approche est plus générale et permet une souplesse accrue quant à l'utilisation de ces deux opérations. Il est maintenant possible de réaliser ces deux opérations séparément. Plus précisément, il est toujours possible de consi-

dérer comme une boîte noire toute sous-expression marquée pour l'analyse d'impact, mais il est également possible de suivre l'impact d'une sous-expression tout en utilisant sa valeur pour affiner l'analyse d'impact. Il est aussi possible de considérer une partie du programme comme une boîte noire en cachant sa valeur sans pour autant nous intéresser à son impact. Cette amélioration apporte une solution au problème de perte de la structure des valeurs. Ce problème avait été soulevé dans [ABDP12] au sujet de l'analyse de dépendances qui y était présentée.

La seconde modification est la distinction entre deux types de dépendance correspondant aux deux types d'impact que peut avoir la valeur d'une sous-expression sur l'évaluation du programme : impact sur la valeur du programme ou impact sur la terminaison de son évaluation. L'analyse présentée par P. Ayrault n'effectuait pas de distinction entre ces deux notions d'impact. Ainsi, si un identificateur se trouvait uniquement dans un certain sous-terme du résultat de l'analyse, on ne savait pas si cet identificateur avait uniquement un impact sur la valeur de ce sous-terme ou bien si celui-ci avait un impact global sur la terminaison de l'évaluation du programme. L'analyse statique que nous présentons dans cette thèse apporte une réponse plus précise quant au type d'impact. Si un identificateur se trouve uniquement dans un certain sous-terme du résultat de l'analyse et qu'il n'apparaît pas dans les dépendances de terminaison du programme, alors on peut être sûr qu'il n'a d'impact que sur la valeur de ce sous-terme précis et qu'il ne peut aucunement affecter les autres sous-termes de la valeur, ni la terminaison du calcul.

Considérons un exemple simple pour illustrer les différences entre les deux approches :

```
let a = 32;;
let b = 18;;
let c = b + 8;;

let f x = (a + 1, x);;

let d = f c;;
```

Supposons que nous souhaitons analyser l'impact de a, de b et de c. Dans l'analyse de P. Ayrault, nous marquons les trois premières définitions *top-level* à l'aide d'un tag « abstrait ». Dans notre analyse, nous annotons les trois définitions correspondantes à

l'aide de trois labels l_a, l_b et l_c. On obtient alors les résultats suivants :

Analyse de P. Ayrault	Notre analyse statique
a : a	a : $[\ \emptyset\ \mid\ [\ l_a\ \mid\ \top\ ]\ ]$
b : b	b : $[\ \emptyset\ \mid\ [\ l_b\ \mid\ \top\ ]\ ]$
c : c	c : $[\ \emptyset\ \mid\ [\ l_b, l_c\ \mid\ \top\ ]\ ]$
f : $< \lambda x.(a+1, x), (a, a); \ldots >$	f : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.(a+1, x), (a, \ldots); \ldots >]\ ]$
d : (a, c)	d : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ l_a\ \mid\ \top\ ],\ [\ l_b, l_c\ \mid\ \top\ ])\ ]\ ]$

L'analyse de P. Ayrault nous apprend que l'identificateur a ne dépend que de lui-même. Elle nous apprend la même chose pour l'identificateur b et pour c. On remarque que l'identificateur b n'apparaît pas dans le résultat de c. Ceci est dû à la notion de marquage « abstrait » qui coupe en quelque sorte les dépendances d'un identificateur marqué. C'est pour cette raison que le théorème exprimé dans la thèse de P. Ayrault n'est en général pas valable s'il y a plus d'un identificateur marqué. Le résultat obtenu pour f est simplement une fermeture. Enfin, le résultat pour d nous apprend qu'il s'agit d'un couple dont la première composante dépend de a et la seconde composante dépend de c.

Notre analyse fournit des résultats un peu différents. Pour a (resp. b), on apprend qu'aucune injection ne peut empêcher l'évaluation de cette partie de programme de terminer. Par contre, une injection sur a (resp. b) peut provoquer une modification du résultat de cette évaluation. Le symbole $\top$ indique que la valeur a été abstraite. Pour l'identificateur c, l'analyse fournit à la fois la dépendance l_b (car la valeur de c dépend de celle de b) et la dépendance l_c (car on a annoté la définition de c avec ce label). Le résultat pour f est une fermeture. Celui pour d nous indique qu'aucune injection ne peut empêcher la terminaison du programme et que la valeur de d est un couple dont la première composante dépend de l_a et la seconde dépend de l_b et l_c.

Un autre apport important par rapport à la thèse de P. Ayrault concerne la preuve de correction de l'analyse. Dans le chapitre 3 de sa thèse, P. Ayrault présente une preuve formelle de son analyse de dépendances. Cette preuve, complexe à vérifier, a été réalisée sur papier, bien que l'assistant à la preuve Coq ait été utilisé pour vérifier formellement certaines propriétés (formalisation du langage et de l'analyse, preuve du déterminisme de l'analyse). De plus, la preuve de correction proposée n'est valable que dans le cas particulier où le programme analysé n'est marqué qu'en un unique point. Il n'est donc pas possible de

se reposer sur ce théorème lors de l'analyse d'un programme marqué en plusieurs points (il est facile de trouver des contre-exemples). Nous apportons dans cette thèse une preuve formelle réalisée en Coq. Cette preuve a été réalisée par étapes successives en « instrumentant » (c'est-à-dire en enrichissant) la sémantique opérationnelle du langage pour qu'elle transporte toutes les informations nécessaires au calcul des dépendances, puis en prouvant formellement des abstractions successives de cette sémantique jusqu'à obtenir l'analyse statique désirée. Nous obtenons ainsi une plus grande clarté en mettant en exergue des résultats intermédiaires importants et compréhensibles permettant de composer la preuve finale de façon modulaire. Un avantage certain de cette approche est de pouvoir prouver une version modifiée de l'analyse statique en réutilisant une grande partie des preuves déjà effectuées. On pourra ainsi proposer par la suite de modifier l'analyse statique pour obtenir des résultats plus précis (à l'aide d'une représentation plus fine pour les types sommes, les entiers ou encore les fonctions) ou pour améliorer les performances de calcul (en ajoutant des approximations). La preuve de correction d'une telle analyse sera alors obtenue à moindre frais puisqu'il faudra uniquement prouver la toute dernière abstraction, toutes les autres preuves étant réutilisables telles quelles.

1.1.2 SPARK/Ada

Des analyses statiques de dépendances adaptées aux évaluateurs de logiciels critiques existent déjà. C'est notamment le cas pour SPARK [Bar03], un sous-ensemble du langage Ada, spécialement conçu pour le développement de logiciels critiques. Il embarque toute une série d'analyses statiques prenant en compte les annotations de l'utilisateur afin de garantir que le programme fonctionne conformément à ses spécifications. En particulier, SPARK est doté d'un langage d'annotations [CH04] permettant à l'utilisateur de spécifier diverses contraintes sur les flots d'information et de contrôle de son programme.

Il est tout d'abord possible de préciser pour chaque paramètre d'une fonction s'il s'agit d'une entrée, d'une sortie ou d'une entrée/sortie, ce qui permet de donner une direction aux flots d'information possibles et de vérifier certaines erreurs statiquement. C'est également une information précieuse pour l'évaluateur de logiciels critiques pour lui permettre de comprendre de façon simple et sûre le comportement du programme. Nous ne nous sommes

pas intéressés à ce type d'annotation puisque la distinction entre les entrées et les sorties ainsi que la nécessité d'initialiser une variable avant son utilisation découle naturellement des propriétés intrinsèques des langages purement fonctionnels.

Une autre fonctionnalité importante concernant la spécification du flot d'information est la possibilité de spécifier pour chaque sortie, quelles sont les entrées utilisées pour le calcul de sa valeur.

Voici un exemple de code annoté en SPARK spécifiant que la seule variable globale utilisée est Count et qu'après l'appel de la fonction, la nouvelle valeur de Count dépend de son ancienne valeur et de celle de X et la nouvelle valeur de X dépend uniquement de sa valeur précédente.

```
procedure Increment (X : in out Counter_Type);
--# global Count;
--# derives
--#    Count from Count, X &
--#    X from X;
```

Le développeur peut ainsi spécifier avec précision le flot d'information pour détecter d'éventuelles erreurs au plus tôt dans le cycle de développement. Cette spécification est alors vérifiée statiquement par une analyse du code source. Ce type d'annotation est un élément important dans la phase de vérification d'un logiciel critique. L'évaluateur peut alors se reposer sur les annotations du développeur puisqu'il sait qu'une analyse du code source a été faite par un outil logiciel fiable afin d'assurer la correction de ces annotations. Cette garantie de fiabilité des annotations permet à l'évaluateur de logiciels critiques de s'épargner une vérification manuelle fastidieuse et coûteuse. Cet outil de spécification et de vérification de propriétés sur le flot d'information du programme disponible pour SPARK (*SPARK Examiner*) est un outil puissant et fiable que l'évaluateur de logiciels critiques voudrait retrouver pour d'autres langages de programmation. Notre travail s'inscrit dans la satisfaction de ce besoin concernant les langages de la famille ML qui commencent à être utilisés pour des développements critiques, soit pour la construction de logiciels nécessitant un haut niveau de fiabilité [MW08], soit pour la construction d'outils logiciels nécessitant d'être qualifiés pour leur utilisation dans le développement de logiciels critiques régis par des normes (EN-50128, DO-178B/C, ...) [PAC+08, FHJ+06, JES00].

Nous présentons dans cette thèse une analyse statique des dépendances d'un programme ML qui pourra être utilisée comme une base théorique solide afin de développer un outil de spécification et de vérification de propriétés sur le flot d'information adapté aux spécificités des langages ML, en s'inspirant de l'outil analogue disponible pour SPARK.

1.2 Analyse de flot pour les langages fonctionnels

L'analyse de flot pour les langages fonctionnels d'ordre supérieur est un sujet qui a fait l'objet de nombreux travaux dont nous mentionnons les principaux dans la suite.

En 1996, M. Abadi, B.W. Lampson et J.J. Lévy [ALL96] proposent une analyse de flot d'information pour un lambda calcul. Le but de leur analyse est alors l'optimisation du temps d'exécution en stockant des valeurs déjà calculées. Lors de l'évaluation d'une expression, si celle-ci a déjà été évalué et que depuis cette première évaluation les seules valeurs mofidiées depuis sont indépendantes, alors on utilise la valeur déjà calculée au lieu de réévaluer l'expression. Dans le langage qu'ils considèrent, n'importe quelle sous-expression du programme analysé peut être annotée avec un label. Leur analyse est construite comme une extension de la sémantique opérationnelle usuelle du lambda calcul en ajoutant une règle manipulant les labels. Ainsi, il s'agit d'une analyse **dynamique** qui évalue le programme pour obtenir une valeur contenant certains des labels présents en tant qu'annotation dans le programme. Si un label n'est pas présent dans la valeur, cela signifie que la sous-expression correspondante n'a pas d'influence sur le calcul de la valeur du programme.

Pour illustrer le fonctionnement de cette analyse, reprenons l'exemple suivant que nous avons déjà présenté plus haut :

```
let a = 32;;
let b = 18;;
let c = b + 8;;

let f x = (a + 1, x);;

let d = f c;;
let d1 = fst d;;
let d2 = snd d;;
```

Analyse dynamique de M. Abadi et al.	Notre analyse statique
a : $l_a : 32$	a : $[\ \emptyset\ \mid\ [\ l_a\ \mid\ \top\ ]\ ]$
b : $l_b : 18$	b : $[\ \emptyset\ \mid\ [\ l_b\ \mid\ \top\ ]\ ]$
c : $l_b, l_c : 26$	c : $[\ \emptyset\ \mid\ [\ l_b, l_c\ \mid\ \top\ ]\ ]$
f : $\lambda x.(32+1, x)$	f : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.(a+1, x), (a, \ldots); \ldots >]\ ]$
d : $(l_a : 32+1, l_b, l_c : 26)$	d : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ l_a\ \mid\ \top\ ],\ [\ l_b, l_c\ \mid\ \top\ ])\ ]\ ]$
d1 : $l_a : 33$	d1 : $[\ \emptyset\ \mid\ [\ l_a\ \mid\ \top\ ]\ ]$
d2 : $l_b, l_c : 26$	d2 : $[\ \emptyset\ \mid\ [\ l_b, l_c\ \mid\ \top\ ]\ ]$

L'analyse dynamique de M. Abadi et al. est définie par une sémantique *small-step* et les couples sont encodés sous forme de fonctions, ce qui explique pourquoi la valeur de d n'est pas complètement réduite. Outre les dépendances de terminaison qui ne sont pas présentes dans l'analyse de M. Abadi et al. et les valeurs numériques qui sont abstraites dans notre analyse, les dépendances calculées sont identiques dans cet exemple.

M. Abadi et al. [ABHR99] montrent ensuite que plusieurs formes d'analyses de flot d'information peuvent être construites en utilisant le même calcul de dépendances. Ils proposent alors un cadre générique pour ces diverses formes d'analyses.

F. Pottier et S. Conchon [PC00] ont alors transformé l'analyse dynamique de [ALL96] en une analyse **statique** en utilisant une « simple » traduction et un système de types standard. Ils affirment qu'en combinant leur travail avec celui de [ABHR99], leur analyse statique peut aussi être utilisée pour toutes les formes d'analyse de dépendances présentées dans [ABHR99]. Plus tard, François Pottier et Vincent Simonet [PS02] ont proposé une technique pour l'analyse de flot sur un lambda calcul étendu par la gestion des références et des exceptions. C'est une nouvelle approche, basée sur un système de types spécifique, qui a été proposée pour combler les manques de l'approche précédente [PC00].

Cette dernière approche, utilisée comme base pour la construction de l'outil FlowCaml [SR03, Flo03], utilise un treillis représentant des niveaux de sécurité dans le but d'assurer des propriétés de non-interférence. Comme il a été montré précédemment dans [ABHR99], ce type d'analyse à base de treillis peut être utilisé pour effectuer diverses analyses de dépendances en utilisant un treillis adapté.

La notion de non-interférence a été introduite en 1982 par J.A. Goguen et J. Meseguer [GM82]. Il s'agit d'une propriété sur un programme manipulant des données et ayant des

interaction avec son environnement. En langage simple, cette propriété s'exprime de la manière suivante : « Un comportement du programme ayant un niveau de sécurité faible n'est pas affecté par une donnée ayant un niveau de sécurité élevé ». Cette propriété permet en particulier de ne pas révéler d'information confidentielle à des personnes non-autorisées.

On retrouve par exemple cette notion sous la forme de propriétés d'isolation sur les programmes embarqués dans les cartes à puce [ACL03].

L'analyse de non-interférence de FlowCaml est une analyse de flot d'information présentant des similarités avec notre analyse de dépendances. On peut attribuer à certaines valeurs du programme des niveaux de sécurité et définir un treillis spécifiant une relation d'ordre entre les différents niveaux de sécurité. Un système de types permet alors d'inférer les niveaux de sécurité des valeurs calculées par le programme et de vérifier la cohérence avec les annotations de l'utilisateur. On peut ainsi vérifier qu'aucune valeur confidentielle n'a d'influence sur une valeur moins confidentielle.

Dans l'analyse de dépendances dont nous avons besoin, la préoccupation sous-jacente est de savoir si une défaillance quelconque à un point donné du programme peut ou non avoir des répercussions sur une valeur considérée comme critique. Cette analyse est particulièrement utile pour garantir qu'un point de programme (une sous-expression donnée) n'a aucune influence sur une certaine valeur critique. On peut alors certifier que cette valeur restera valide en cas de défaillance au point de programme. On constate que, bien que le vocabulaire soit différent, notre besoin est assez proche de ce que propose l'analyse de non-interférence de FlowCaml.

D'après nos expérimentations, il est possible d'utiliser FlowCaml pour calculer les dépendances dont nous avons besoin (du point de vue de l'évaluateur de logiciels critiques) en obtenant des résultats plutôt bons, jusqu'à un certain point. Pour cela, on utilise un treillis plat (relation d'ordre vide) dont les éléments sont des identifiants désignant les points de programmes dont nous souhaitons connaître l'impact. L'inférence de type fournit alors l'analyse de dépendances désirée. Cependant, certains aspects de l'analyse ne sont pas adaptés. Premièrement, le système d'annotation manque de souplesse. Par exemple, on ne peut annoter que des valeurs mutuellement indépendantes, sans quoi il serait nécessaire d'expliciter dans le treillis les relations entre les valeurs annotées, ce qui nous ferait

perdre la traçabilité des différentes dépendances. Il n'est pas non plus possible d'annoter une valeur structurée (par exemple un couple) par un label unique. On notera également qu'il n'est pas possible d'annoter un point de programme quelconque, mais uniquement les valeurs nommées. Deuxièmement, l'analyse de FlowCaml ne permet pas d'abstraire un morceau de programme en le considérant comme une boîte noire. C'est un besoin important pour l'évaluateur de logiciels critiques pour lui permettre de se concentrer sur l'analyse d'un composant particulier du programme, en considérant les autres composants comme opaques. Cette fonctionnalité peut aussi lui permettre d'analyser facilement un composant isolé du programme sans connaître l'implémentation des autres composants utilisés. Par exemple, un évaluateur peut vouloir effectuer l'évaluation d'un programme faisant appel à des librairies dont il n'a pas accès au code source.

De plus, l'analyse de dépendances que nous présentons permet à la fois l'analyse d'impact sur la terminaison du programme et l'analyse d'impact sur les valeurs calculées lors de l'évaluation du programme. Cette distinction entre dépendances de terminaison et dépendances de valeurs n'est pas présente dans FlowCaml.

Toujours sur le même exemple, voici une comparaison entre l'analyse de FlowCaml et notre analyse statique :

```
flow !b < !c;;

let a : !a int = 32;;
let b : !b int = 18;;
let c : !c int = b + 8;;

let f x = (a + 1, x);;

let d = f c;;
```

Analyse statique FlowCaml	Notre analyse statique
a : `!a int`	a : $[\ \emptyset\ \mid\ [\ l_a\ \mid\ \top\ ]\ ]$
b : `!b int`	b : $[\ \emptyset\ \mid\ [\ l_b\ \mid\ \top\ ]\ ]$
c : `[>!b,!c] int`	c : $[\ \emptyset\ \mid\ [\ l_b, l_c\ \mid\ \top\ ]\ ]$
f : `'a ->!a int * 'a`	f : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.(a+1, x), (a, \ldots); \ldots >]\ ]$
d : `!a int * [>!b,!c] int`	d : $[\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ l_a\ \mid\ \top\ ],\ [\ l_b, l_c\ \mid\ \top\ ])\ ]\ ]$

FlowCaml calcule les dépendances pendant le typage et fournit les résultats de types annotés par des informations de dépendances. Les résultats nous indiquent que la valeur de

l'identificateur a (resp. b) est un entier qui ne dépend que de lui-même. Pour la définition de c, FlowCaml rejette cette définition si nous n'indiquons pas explicitement la relation entre b et c à l'aide de la directive flow !b < !c ;; qui signifie que l'on autorise un flot d'information allant d'une valeur de niveau !b vers une valeur de niveau !c. Le résultat de c correspond au type d'une fonction prenant un paramètre et retournant un couple dont la première composante est un entier dépendant de a et dont la seconde composante a le même type et les mêmes dépendances que l'argument. C'est un résultat bien plus concis et explicite que le résultat de notre analyse qui embarque le corps de la fonction ainsi que son environnement. Le résultat de d nous fournit les mêmes informations de dépendances que notre analyse.

Dans notre analyse, nous représentons tous les nombres entiers par la valeur $\top$ qui représente n'importe quelle valeur. Nous pourrions ajouter à notre algèbre de valeurs abstraites une valeur permettant de représenter uniquement les nombres entiers, mais il n'est pas évident que ceci ait un réel intérêt. Par contre, au sujet de la représentation des fonctions au sein de notre algèbre de valeurs abstraites, il serait sans doute intéressant de trouver une représentation plus compacte, à l'image du type retourné par FlowCaml.

Nous nous sommes inspirés de la méthode utilisée par M. Abadi et al. [ALL96] pour annoter à l'aide de labels les sous-expressions du programme dont nous souhaitons suivre l'impact. Notre analyse dynamique suit le même principe que leur analyse en calculant la valeur du programme tout en y accumulant les labels utilisés pour son élaboration. Il faut cependant remarquer deux différences fondamentales. Premièrement, nous utilisons une sémantique *big-step* contrairement à M. Abadi et al. qui utilisent une sémantique *small-step*. L'avantage d'une sémantique *big-step* est de pouvoir lier directement une expression avec le résultat de son évaluation, ce qui nous est d'une grande aide lorsque l'on veut parler de plusieurs évaluations simultanées d'une même expression. Le choix de la sémantique *big-step* nous a permis d'exprimer les définitions de nos sémantiques de façon plus naturelle, en particulier pour la sémantique collectrice. Ce choix nous a également permis d'obtenir des preuves plus simples, en particulier pour les sémantiques collectrice et abstraite. Deuxièmement, nous produisons des valeurs structurées dont chaque sous-terme est annoté par ses propres dépendances, contrairement à leur analyse qui regroupe en tête de la valeur

produite une liste globale de tous les labels utilisés pour produire la valeur. Notre analyse permet ainsi d'obtenir un résultat plus précis.

Nous avons ensuite transformé notre analyse dynamique en une analyse statique. Nous avons alors marqué une rupture avec les autres travaux présentés dans cette section, puisque contrairement à F. Pottier et S. Conchon, nous n'avons pas formalisé notre analyse statique sous forme d'un système de types. Nous avons fait le choix d'utiliser le formalisme de l'interprétation abstraite qui nous a permis plus de souplesse dans la définition d'une analyse adaptée à nos besoins. En particulier, ce cadre nous a permis de définir avec aisance l'abstraction d'une partie de programme sous forme de boîte noire.

1.3 Analyse de teintes

L'analyse de teintes telle que présentée dans [XBS06] est un cas particulier d'analyse de flot d'information [Kri07]. Ce type d'analyse permet d'identifier les entrées (contrôlables par l'utilisateur) ayant un impact sur certaines opérations dites « critiques » ou « dangereuses » du point de vue de la sécurité. Par exemple, si une valeur contrôlable par l'utilisateur est utilisée sans précaution particulière en tant qu'indice lors de la modification d'une case de tableau, il peut y avoir une vulnérabilité de type *buffer overflow*. Dans ce cas, l'utilisateur peut modifier des valeurs en mémoire auxquelles il ne devrait pas avoir accès.

Pour détecter ce type de vulnérabilité, l'analyse de teinte permet d'identifier tout au long du programme les valeurs dites teintées (ie. contrôlables par l'utilisateur) afin de donner une alerte si une telle valeur est utilisée pour effectuer une opération dangereuse (accès à un tableau, instruction `jump`, requête SQL, ...).

Notre analyse a pour but, quant à elle, d'identifier tout au long du programme les valeurs possiblement impactées par certains points de programmes (parties non vérifiées d'un programme ou bien valeurs possiblement issues d'un dysfonctionnement). Cette information peut alors nous permettre d'alerter l'utilisateur si une valeur critique du programme peut être impactée par un dysfonctionnement éventuel.

On peut ainsi apercevoir la similarité entre les deux approches en considérant un point de programme annoté pour notre analyse comme une valeur teintée. L'analyse de teinte

permettrait alors de faire la distinction entre les valeurs impactées par ce point de programme et celles qui ne le sont pas. Cependant, elles diffèrent par plusieurs aspects. Dans l'analyse de teintes, une valeur est soit « teintée » soit « non teintée ». Il n'y a aucune distinction entre les différents points d'entrée de l'utilisateur, Bien qu'il soit possible de relier une valeur teintée à un point d'entrée précis en utilisant des informations supplémentaires au sujet des chemins d'exécution, l'analyse de teintes en elle-même ne le permet pas.

Notre analyse permet d'indiquer pour chaque valeur quels sont les points d'entrée dont elle dépend. C'est une information importante pour aider l'évaluateur à comprendre la structure du programme et les dépendances entres ses différentes parties. De plus, comme nous l'avons montré plus haut dans la comparaison avec les analyses de flot d'information, nous avons besoin de pouvoir considérer certaines parties du programme comme opaques, ce qui n'est pas le cas des analyses de teintes en général. Enfin, notre analyse s'effectue sur un langage fonctionnel dans lequel les préoccupations de sécurité sont différentes de celles de la plupart des analyses de teintes effectuées sur des langages impératifs. En effet, les accès à des tableaux ou les instructions `jump` ne sont pas courants dans les langages fonctionnels purs.

Une piste intéressante détaillée dans [CMP10] pour une analyse de teintes est la possibilité de garder trace des chemins d'exécution permettant de relier un point d'entrée à l'instruction impactée. Ce type d'information pourrait être particulièrement utile pour notre analyse de dépendances. Ce sujet a d'ailleurs été abordé dans la thèse de P. Ayraut [Ayr11] (Partie III, chapitre 4).

1.4 Interprétation abstraite

L'interprétation abstraite est un cadre mathématique permettant de formaliser des analyses de programmes et de prouver leur correction. L'idée principale est d'exécuter des programmes sur un domaine abstrait dont les éléments représentent des propriétés sur le domaine d'évaluation standard. Pour cela, on définit explicitement des fonctions d'abstraction et de concrétisation pour établir le lien entre le domaine d'évaluation standard et le domaine abstrait. Une méthode générique permet alors de prouver la correction de ce

genre d'analyses de programmes en prouvant certaines propriétés sur la composition des fonctions d'abstraction, de concrétisation et d'évaluation abstraite.

À l'origine, la théorie de l'interprétation abstraite a été conçue par P. et R. Cousot [CC77, CC79] pour des langages impératifs. Les analyses de programmes décrites par cette méthode sont exprimées sous forme d'une sémantique opérationnelle.

Par la suite, le cadre de l'interprétation abstraite a été adapté aux langages fonctionnels du premier ordre par A. Mycroft [Myc81]. La théorie sous-jacente est alors différente. Contrairement au travail de P. et R. Cousot basé sur une description sous forme de sémantique opérationnelle, A. Mycroft fait le choix d'utiliser une sémantique dénotationnelle pour décrire les analyses de programmes.

G. Burn, C. Hankin et S. Abramsky [BHA86] ont alors étendu l'analyse de programmes de A. Mycroft en l'adaptant aux langages fonctionnels d'ordre supérieur. En plus d'étendre cette analyse qui est un cas particulier d'interprétation abstraite, ils ont également étendu le cadre général de l'interprétation abstraite aux langages fonctionnels d'ordre supérieur. Dans la lignée de A. Mycroft, leur travail fait usage d'une sémantique dénotationnelle pour définir leur analyse.

D'autres travaux, comme ceux de S. Hunt [Hun91] ont appliqué l'interprétation abstraite à des langages de programmation fonctionnels en utilisant des sémantiques dénotationnelles. D'autres approches sont présentées par N.D. Jones et F. Nielson dans [Jon94].

Nous présentons ici une approche originale de l'interprétation abstraite pour un langage fonctionnel d'ordre supérieur. Notre analyse de programme est exprimée sous forme d'une sémantique opérationnelle, comme pour les langages impératifs. De plus, notre analyse opère sur un langage de programmation comprenant des constructeurs de données récursives ainsi que du filtrage par motifs. Pour ce qui est des fonctions récursives, nous nous affranchissons du problème de la terminaison et de la recherche d'un point fixe à l'aide d'une approximation permettant de casser la récursion. Cependant, le problème pourrait resurgir en recherchant une meilleure représentation des fonctions récursives dans l'algèbre des valeurs abstraites. En effet, il serait souhaitable d'améliorer notre analyse en trouvant une meilleure représentation des fonctions, ce qui pourrait apporter de nombreux avantages

autant du point de vue de la précision de l'analyse que du point de vue de la performance.

Chapitre 2

Notion de dépendance

2.1 Introduction

La notion de dépendance est souvent utilisée de manière intuitive. De nombreuses analyses de dépendances se contentent d'une définition informelle. Et quand une définition formelle est donnée, elle est parfois incomplète et ne reflète que partiellement la notion utilisée dans l'analyse correspondante. Nous nous appliquons dans ce chapitre à donner une définition formelle à cette notion de dépendance.

Une particularité de la notion de dépendance que nous proposons est la distinction entre 2 types de dépendances :

- les dépendances concernant la valeur d'une expression,
- les dépendances concernant la terminaison d'une évaluation.

Les dépendances de terminaison sont rarement prises en compte dans les analyses de dépendances. Pourtant, la modification d'une partie d'un programme peut provoquer une boucle infinie ou une erreur lors de son évaluation. Dès lors, la connaissance des dépendances de terminaison constitue une information importante, en particulier pour les évaluateurs de logiciels critiques.

Après avoir présenté la sémantique opérationnelle de notre langage (qui est la sémantique usuelle d'un programme ML purement fonctionnel), nous expliquerons comment annoter un programme dont on veut connaître les dépendances. Nous verrons ensuite comment annoter les valeurs du langage avec des informations de dépendance. Nous pourrons alors définir la ***sémantique opérationnelle avec injection*** qui nous permettra d'introduire

les notions d'injection de valeur et d'***impact d'une injection***. Nous aurons alors à notre disposition toutes les notions sous-jacentes nécessaires à la définition formelle de la notion de dépendance. Nous présenterons alors la définition de la notion de dépendance que nous proposons.

2.2 Le langage : algèbre des expressions

Le langage sur lequel nous définissons notre analyse est un noyau purement fonctionnel de la famille ML. Il contient les principales caractéristiques des langages purement fonctionnels : les fonctions d'ordre supérieur, les constructeurs de données (types algébriques), le filtrage par motif (pattern-matching), la liaison (let-binding).

En plus des constructions habituelles, on ajoute la possibilité d'annoter une sous-expression à l'aide d'un label. Cette annotation permet de définir un point d'injection (cf. section 2.4).

$e :=$	$n \mid C \mid D(e) \mid (e_1, e_2)$	Constructeurs de données
	x	Identificateur
	$\lambda x.e \mid \texttt{rec}f.x.e$	Fonctions (récursive ou non)
	$e_1\ e_2$	Application
	$let\ x\ =\ e_1\ in\ e_2$	Liaison
	$if\ e\ then\ e_1\ else\ e_2$	Expression conditionnelle
	$match\ e\ with\ p \rightarrow e_1 \mid x \rightarrow e_2$	Filtrage par motif
	$l\ :\ e$	Annotation d'un point d'injection
$p :=$	$C \mid D(x) \mid (x_1, x_2)$	Filtres des données structurées

FIGURE 2.1 – Algèbre des expressions du langage

Etant donné un ensemble de constantes numériques noté Z, un ensemble de constructeurs de donnée sans paramètre est noté $Constr^0$ un ensemble de constructeurs de données avec paramètre est noté $Constr^1$.

2.3 Sémantique opérationnelle

Il s'agit de la sémantique opérationnelle usuelle. Nous ajoutons une seule règle supplémentaire permettant d'ignorer les annotations dans les programmes.

2.3.1 Valeurs

L'algèbre des valeurs comprend les valeurs constantes (entières et booléennes), des constructeurs de données structurées (paramétrés ou non), un constructeur de couple et des fermetures (récursives ou non).

$$
\begin{array}{rll}
v := & n \mid b \mid C \mid D(v) \mid (v_1, v_2) & \text{Constructeurs de données} \\
& < \lambda x.e, \Gamma > & \text{Fermeture} \\
& < \texttt{rec} f.x.e, \Gamma > & \text{Fermeture récursive}
\end{array}
$$

2.3.2 Environnements

Un environnement d'évaluation est une liste associative identificateurs $\rightarrow$ valeur.

$$\Gamma := \quad (x_1, v_1) \oplus \ldots \oplus (x_n, v_n) \quad \text{Environnement}$$

2.3.3 Règles d'inférence

Le jugement d'évaluation de la sémantique opérationnelle prend la forme suivante :

$$\Gamma \vdash e \mapsto v$$

Hormis la règle de l'évaluation d'une expression annotée par un label que nous avons ajoutée, les règles d'inférence définissant la sémantique opérationnelle de notre langage sont usuelles. Ces règles sont récapitulées dans les figures 2.2 et 2.3.

Nous donnons ci-dessous quelques explications concernant l'évaluation des fonctions et de leur application, du filtrage par motif et des expressions annotées.

Évaluation et application des fonctions Une fonction non-récursive est évaluée en une fermeture. Celle-ci contient la définition de la fonction ainsi que l'environnement dans lequel elle a été définie (ce qui permet de retrouver la valeur de chaque identificateur libre de la définition de la fonction). Une fonction récursive est évaluée en une fermeture récursive, similaire à une fermeture non-récursive. Les deux types de fermeture permettent de déterminer quelle règle d'inférence appliquer lors de l'application d'une fonction.

Dans notre sémantique, l'application d'une fonction (récursive ou non) se fait en appel par valeur. C'est-à-dire que la valeur de l'argument est calculée systématiquement avant l'évaluation du corps de la fonction (même si l'argument n'est pas utilisé dans le corps de la fonction).

Filtrage à motif unique Une particularité du filtrage par motif dans notre langage est qu'il ne filtre que sur un seul motif. Si la valeur filtrée correspond au motif, alors l'évaluation passe par la première branche, sinon, l'évaluation passe par la seconde branche. Bien que cette restriction contraigne la forme des programmes, elle n'affecte en rien l'expressivité du langage. En effet, il est toujours possible d'écrire des filtrages imbriqués, ce qui donne la possibilité de réécrire tout programme contenant des filtrages à plusieurs motifs en un programme équivalent contenant uniquement des filtrages à motif unique en cascade. Notre langage peut donc être vu comme un langage noyau n'étant pas directement manipulé par l'utilisateur. Une phase de réécriture simple sera nécessaire entre le langage utilisé en pratique par l'utilisateur et le langage sur lequel l'analyse sera effectuée.

Expressions annotées Les annotations sont utilisées pour définir des points d'injection (cf. section 2.4). Ces points d'injection serviront à calculer les dépendances du programme. En ce qui concerne la sémantique opérationnelle, ils n'ont aucune signification et sont donc ignorés. Techniquement, évaluer une expression annotée revient à évaluer sa sous-expression sans tenir compte de l'annotation.

2.3.4 Sémantique des programmes mal typés et filtrage par motif

Le filtrage par motif que nous présentons est défini par les règles OP-MATCH et OP-MATCH-VAR ainsi que par le jugement $v, p \vdash_p \bullet$. On considère que le typage garantit que toute valeur filtrée est soit un constructeur de donnée, soit un couple. Dans ces cas-là, les règles d'inférence définissent la sémantique du filtrage. Par contre, nous ne donnons pas de sémantique à l'évaluation des programmes mal typés pour lesquels l'expression filtrée ne s'évalue pas en une valeur filtrable (constructeur de donnée ou couple). Par exemple, si l'expression filtrée s'évalue en une fermeture, le filtrage n'a pas de sens et l'expression n'a donc pas de sémantique d'évaluation.

Ce choix est cohérent avec la sémantique d'une expression conditionnelle. En effet, nous n'explicitons pas la sémantique d'une expression conditionnelle dans le cas où la condition s'évalue en autre chose qu'un booléen.

Dans un compilateur réel, la sémantique des programmes mal typés peut dépendre du

$$\textsc{op-num}\ \frac{}{\Gamma \vdash n \mapsto n} \qquad \textsc{op-ident}\ \frac{v = \Gamma[x]}{\Gamma \vdash x \mapsto v} \qquad \textsc{op-abstr}\ \frac{}{\Gamma \vdash \lambda x.e \mapsto < \lambda x.e, \Gamma >} \qquad \textsc{op-abstr-rec}\ \frac{}{\Gamma \vdash \texttt{rec} f.x.e \mapsto < \texttt{rec} f.x.e, \Gamma >}$$

$$\textsc{op-apply}\ \frac{\Gamma \vdash e_1 \mapsto < \lambda x.e, \Gamma_1 > \quad \Gamma \vdash e_2 \mapsto v_2 \quad (x, v_2) \oplus \Gamma_1 \vdash e \mapsto v}{\Gamma \vdash e_1\ e_2 \mapsto v} \qquad \textsc{op-apply-rec}\ \frac{\Gamma \vdash e_1 \mapsto v_1 \quad v_1 = < \texttt{rec} f.x.e, \Gamma_1 > \quad \Gamma \vdash e_2 \mapsto v_2 \quad (f, v_1) \oplus (x, v_2) \oplus \Gamma_1 \vdash e \mapsto v}{\Gamma \vdash e_1\ e_2 \mapsto v}$$

$$\textsc{op-if-true}\ \frac{\Gamma \vdash e \mapsto \mathit{true} \quad \Gamma \vdash e_1 \mapsto v_1}{\Gamma \vdash \mathit{if}\ e\ \mathit{then}\ e_1\ \mathit{else}\ e_2 \mapsto v_1} \qquad \textsc{op-if-false}\ \frac{\Gamma \vdash e \mapsto \mathit{false} \quad \Gamma \vdash e_2 \mapsto v_2}{\Gamma \vdash \mathit{if}\ e\ \mathit{then}\ e_1\ \mathit{else}\ e_2 \mapsto v_2}$$

$$\textsc{op-constr-0}\ \frac{}{\Gamma \vdash C \mapsto C} \qquad \textsc{op-constr-1}\ \frac{\Gamma \vdash e \mapsto v}{\Gamma \vdash D(e) \mapsto D(v)} \qquad \textsc{op-couple}\ \frac{\Gamma \vdash e_1 \mapsto v_1 \quad \Gamma \vdash e_2 \mapsto v_2}{\Gamma \vdash (e_1, e_2) \mapsto (v_1, v_2)}$$

$$\textsc{op-match}\ \frac{\Gamma \vdash e \mapsto v \quad v, p \vdash_p \Gamma_p \quad \Gamma_p \oplus \Gamma \vdash e_1 \mapsto v_1}{\Gamma \vdash \mathit{match}\ e\ \mathit{with}\ p \to e_1 \mid x \to e_2 \mapsto v_1} \qquad \textsc{op-match-var}\ \frac{\Gamma \vdash e \mapsto v \quad v, p \vdash_p \bot \quad (x, v) \oplus \Gamma \vdash e_2 \mapsto v_2}{\Gamma \vdash \mathit{match}\ e\ \mathit{with}\ p \to e_1 \mid x \to e_2 \mapsto v_2}$$

$$\textsc{op-letin}\ \frac{\Gamma \vdash e_1 \mapsto v_1 \quad (x, v_1) \oplus \Gamma \vdash e_2 \mapsto v_2}{\Gamma \vdash \mathit{let}\ x\ =\ e_1\ \mathit{in}\ e_2 \mapsto v_2} \qquad \textsc{op-annot}\ \frac{\Gamma \vdash e \mapsto v}{\Gamma \vdash l\ :\ e \mapsto v}$$

FIGURE 2.2 – Règles d'inférence de la sémantique opérationnelle

$$\textsc{opm-constr-0}\ \frac{}{C, C \vdash_p \{\}} \qquad \textsc{opm-constr-1}\ \frac{}{D(v), D(x) \vdash_p \{(x, v)\}} \qquad \textsc{opm-couple}\ \frac{}{(v_1, v_2), (x_1, x_2) \vdash_p \{(x_1, v_1); (x_2, v_2)\}}$$

$$\textsc{opm-constr-0-not}\ \frac{p \neq C}{C, p \vdash_p \bot} \qquad \textsc{opm-constr-1-not}\ \frac{p \neq D'(\_)}{D(v), p \vdash_p \bot} \qquad \textsc{opm-couple-not}\ \frac{p \neq (\_, \_)}{(v_1, v_2), p \vdash_p \bot}$$

FIGURE 2.3 – Règles d'inférence du filtrage de la sémantique opérationnelle

choix de l'implémentation du compilateur. Il est en effet possible que pour des raisons de performance, le compilateur se repose sur le typage pour limiter le nombre d'opérations effectuées. Dans le cas de l'évaluation d'une expression conditionnelle par exemple, on pourrait ne tester que l'égalité entre la valeur de la condition est la valeur $true$ pour passer dans la première branche et que dans tous les autres cas, on passe dans la seconde branche, même si la valeur de la condition n'est pas $false$ mais un couple, une fermeture ou toute autre valeur.

De même, dans le cas de l'évaluation d'une expression de filtrage, on pourrait se reposer sur le typage pour passer dans la seconde branche dans tous les cas où la valeur filtrée ne correspond pas au motif, même dans les cas mal typés où la valeur de l'expression filtrée n'est ni un constructeur de donnée, ni un couple.

Ces choix d'optimisation du compilateur se reposant sur le typage n'ont évidemment pas d'influence sur le résultat de l'évaluation des programmes bien typés. Cependant, notre approche de l'analyse de dépendances considère à la fois l'impact des injections bien typées et l'impact des injections mal typées. Une injection mal typée peut survenir à diverses occasions, par exemple lors du dysfonctionnement d'un capteur ou bien lors d'une perturbation électromagnétique qui modifierait la mémoire de façon imprévisible.

2.4 Points d'injection

2.4.1 Dans le programme

On souhaite analyser un programme pour calculer ses dépendances vis-à-vis de certains points du programme. Ces points de programmes sont appelés « points d'injection » et sont représentés formellement par des annotations sur les sous-expressions du programme.

Par exemple, si l'utilisateur veut analyser l'impact d'une sous-expression sur l'évaluation complète d'un programme, il suffira alors d'annoter cette sous-expression à l'aide d'un label unique et de lancer l'analyse.

L'utilisateur de l'analyse (évaluateur de logiciels critiques) choisira d'après son expertise quels sont les points de programmes qu'il souhaite annoter. Ces points d'injection peuvent en particulier correspondre à des parties du programmes non-critiques dans le but de s'assurer qu'un dysfonctionnement dans une partie non-critique du programme ne peut pas se propager à une partie critique de celui-ci.

2.4.2 Dans l'environnement

La valeur d'une fonction est représentée par une fermeture contenant le corps de la fonction. Celui-ci est une expression contenant possiblement des points d'injection. Au cours de l'évaluation d'un programme, les points d'injection vont donc naturellement se retrouver dans l'environnement d'évaluation.

2.5 Sémantique opérationnelle avec injection

La sémantique opérationnelle avec injection est une extension de la sémantique opérationnelle usuelle. Son but est de donner une valeur à un programme dans un environnement donné et ceci pour n'importe quelle injection (à un point du programme ou bien dans son environnement d'évaluation).

Cette sémantique donne toujours le même résultat que la sémantique opérationnelle sur les programmes ne comportant pas de point d'injection (programmes non-annotés). La seule différence entre la sémantique opérationnelle usuelle et celle avec injection se trouve lors de l'évaluation d'une sous-expression annotée par un label. Là où la sémantique opérationnelle ignore tout simplement l'annotation, la sémantique opérationnelle avec injection va effectuer ce que l'on appelle l'injection. C'est le comportement de la sémantique opérationnelle avec injection sur les sous-expressions annotées qui constitue précisément notre définition de la notion d'injection.

La notion d'injection définie ici est à la base de notre notion de dépendance.

2.5.1 Règles d'inférence

Formellement, la sémantique opérationnelle avec injection prend la forme d'un jugement d'évaluation défini par des règles d'inférence. Ce jugement d'évaluation se note de la manière suivante :

$$\Gamma \vdash_{l:v_l} e \mapsto v$$

, où e est l'expression évaluée, Γ l'environnement d'évaluation et v la valeur résultat. Ce jugement d'évaluation est paramétré par une injection notée en indice ($\vdash_{l:v_l}$), où l représente le point du programme sur lequel on fait l'injection et v_l la valeur injectée. On dit que v est la valeur de l'expression e dans l'environnement Γ après injection de la valeur v_l au point de programme l.

Les règles d'inférence sont toutes identiques aux règles de la sémantique opérationnelle hormis la règle d'évaluation d'une sous-expression annotée. Cette dernière règle est divisée en 2 cas distincts : si l'expression est annotée par le label sur lequel on fait l'injection, alors sa valeur réelle est ignorée et remplacée par la valeur injectée v_l, sinon, l'annotation est

$$\text{OPINJ-NUM}\ \frac{}{\Gamma \vdash_{l:v_l} n \mapsto n} \qquad \text{OPINJ-IDENT}\ \frac{v = \Gamma[x]}{\Gamma \vdash_{l:v_l} x \mapsto v} \qquad \text{OPINJ-ABSTR}\ \frac{}{\Gamma \vdash_{l:v_l} \lambda x.e \mapsto <\lambda x.e, \Gamma>} \qquad \text{OPINJ-ABSTR-REC}\ \frac{}{\Gamma \vdash_{l:v_l} \texttt{rec} f.x.e \mapsto <\texttt{rec} f.x.e, \Gamma>}$$

$$\text{OPINJ-APPLY}\ \frac{\Gamma \vdash_{l:v_l} e_1 \mapsto <\lambda x.e, \Gamma_1> \quad \Gamma \vdash_{l:v_l} e_2 \mapsto v_2 \quad (x, v_2) \oplus \Gamma_1 \vdash_{l:v_l} e \mapsto v}{\Gamma \vdash_{l:v_l} e_1\ e_2 \mapsto v} \qquad \text{OPINJ-APPLY-REC}\ \frac{\Gamma \vdash_{l:v_l} e_1 \mapsto v_1 \quad v_1 = <\texttt{rec} f.x.e, \Gamma_1> \quad \Gamma \vdash_{l:v_l} e_2 \mapsto v_2 \quad (f, v_1) \oplus (x, v_2) \oplus \Gamma_1 \vdash_{l:v_l} e \mapsto v}{\Gamma \vdash_{l:v_l} e_1\ e_2 \mapsto v}$$

$$\text{OPINJ-IF-TRUE}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto true \quad \Gamma \vdash_{l:v_l} e_1 \mapsto v_1}{\Gamma \vdash_{l:v_l} if\ e\ then\ e_1\ else\ e_2 \mapsto v_1} \qquad \text{OPINJ-IF-FALSE}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto false \quad \Gamma \vdash_{l:v_l} e_2 \mapsto v_2}{\Gamma \vdash_{l:v_l} if\ e\ then\ e_1\ else\ e_2 \mapsto v_2}$$

$$\text{OPINJ-MATCH}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto v \quad v, p \vdash_p \Gamma_p \quad \Gamma_p \oplus \Gamma \vdash_{l:v_l} e_1 \mapsto v_1}{\Gamma \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_1} \qquad \text{OPINJ-MATCH-VAR}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto v \quad v, p \vdash_p \bot \quad (x, v) \oplus \Gamma \vdash_{l:v_l} e_2 \mapsto v_2}{\Gamma \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_2}$$

$$\text{OPINJ-CONSTR-0}\ \frac{}{\Gamma \vdash_{l:v_l} C \mapsto C} \qquad \text{OPINJ-CONSTR-1}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto v}{\Gamma \vdash_{l:v_l} D(e) \mapsto D(v)} \qquad \text{OPINJ-COUPLE}\ \frac{\Gamma \vdash_{l:v_l} e_1 \mapsto v_1 \quad \Gamma \vdash_{l:v_l} e_2 \mapsto v_2}{\Gamma \vdash_{l:v_l} (e_1, e_2) \mapsto (v_1, v_2)}$$

$$\text{OPINJ-LETIN}\ \frac{\Gamma \vdash_{l:v_l} e_1 \mapsto v_1 \quad (x, v_1) \oplus \Gamma \vdash_{l:v_l} e_2 \mapsto v_2}{\Gamma \vdash_{l:v_l} let\ x\ =\ e_1\ in\ e_2 \mapsto v_2}$$

FIGURE 2.4 – Sémantique opérationnelle avec injection : règles d'inférence identiques à la sémantique usuelle

$$\text{OPINJ-ANNOT-SAME}\ \frac{}{\Gamma \vdash_{l:v_l} l\ :\ e \mapsto v_l} \qquad \text{OPINJ-ANNOT-OTHER}\ \frac{\Gamma \vdash_{l:v_l} e \mapsto v \quad l \neq l'}{\Gamma \vdash_{l:v_l} l'\ :\ e \mapsto v}$$

FIGURE 2.5 – Sémantique opérationnelle avec injection : règles d'inférence spécifiques

ignorée.

On notera qu'une injection (évaluation d'une sous-expression annotée par le label sur lequel on fait l'injection) termine toujours. En effet, la sous-expression n'est pas évaluée et sa valeur (qu'elle existe ou non) est remplacée par la valeur injectée.

Les règles identiques à la sémantique opérationnelle usuelle sont réunies sur la figure 2.4. Les deux règles spécifiques à la sémantique opérationnelle avec injection se trouvent sur la figure 2.5.

2.5.2 Correction

2.5.2.1 Énoncé informel du théorème

Comme nous l'avons dit plus haut, la sémantique opérationnelle avec injection vient introduire la notion d'injection en remplaçant une règle de la sémantique opérationnelle usuelle par deux nouvelles règles d'inférence.

Il convient de vérifier qu'en ce qui concerne les programmes non-annotés, la sémantique opérationnelle avec injection fournit exactement les mêmes résultats que la sémantique opérationnelle.

En ce qui concerne la sémantique de la notion d'injection (les deux nouvelles règles d'inférence), il n'y a pas de propriété de correction à vérifier puisque ces deux règles d'inférences constituent notre définition de la notion d'injection.

Cependant, il y a une propriété intéressante concernant la notion d'injection : si un label n'apparaît pas dans le programme évalué, alors aucune injection sur ce label ne peut modifier la valeur du programme. Cette propriété engloble la propriété d'extension énoncée ci-dessus. En effet, l'évaluation avec injection d'un programme non-annoté est un cas particulier d'évaluation avec injection sur un label n'apparaissant pas dans le programme. Notre théorème de correction exprimera donc la propriété la plus forte.

2.5.2.2 Illustration par l'exemple

Afin d'illustrer la notion d'injection et le fonctionnement de la sémantique opérationnelle avec injection, voici quelques exemples d'évaluation de programmes.

Exemple 1 : programme non-annoté

Sur les programmes non-annotés les résultats de la sémantique opérationnelle avec injection et de la sémantique opérationnelle sont les mêmes.

Notons e le programme suivant :

```
let t = 18 in
let f x = (x-t, x+t) in
```

```
f 32
```

Nous avons les deux jugements suivants :

$\vdash e \mapsto (14, 50)$

$\vdash_{l:v_l} e \mapsto (14, 50)$ (pour toute injection (l, v_l))

Exemple 2 : programme annoté

Maintenant voyons le cas du même programme en lui ajoutant une annotation (label l) sur une de ses sous-expressions.

Notons e' le programme suivant :

```
let t = 18 in
let f x = (l:x-t, x+t) in
f 32
```

Nous avons les deux jugements suivants :

$\vdash e' \mapsto (14, 50)$

$\vdash_{l:v_l} e' \mapsto (v_l, 50)$ (pour toute valeur injectée v_l)

$\vdash_{l:3} e' \mapsto (3, 50)$ (exemple en injectant la valeur 3)

Lorsque la sous-expression f 32 est évaluée, le corps de la fonction est déplié. On évalue alors l'expression (l:x−t, x+t) dans l'environnement $\{(x, 32); (t, 18)\}$. La valeur v_l est alors injectée lors de l'évaluation de la première composante du couple puisque celle-ci est annotée par le label l. La seconde composante du couple est évaluée de manière usuelle.

Exemple 3 : programme annoté en plusieurs points

Modifions de nouveau notre exemple pour annoter une seconde sous-expression.

Notons e'' le programme suivant :

```
let t = l':18 in
let f x = (l:x-t, x+t) in
```

f 32

Voyons le résultat pour une injection sur l, celui pour une injection sur l' :
celui pour une injection sur un l'' n'apparaissant pas dans le programme :
$\vdash e'' \mapsto (14, 50)$
$\vdash_{l:v_l} e'' \mapsto (v_l, 50)$ (pour toute valeur injectée v_l)
$\vdash_{l':v_{l'}} e'' \mapsto (14, 32 + v_{l'})$ (pour toute valeur injectée $v_{l'}$)
$\vdash_{l':5} e'' \mapsto (14, 37)$ (exemple en injectant la valeur 5)
$\vdash l'' \mapsto v_{l''} e''(14, 50)$

Exemple 4 : programme annoté dans l'environnement

Un environnement d'évaluation peut contenir des points d'injection. En effet, un identificateur peut être lié à une fermeture (récursive ou non) dont le corps de la fonction contient des points d'injection.

Prenons, dans l'exemple ci-dessus, l'évaluation de la sous-expression $f\ 32$. Cette évaluation se fait dans un environnement contenant une valeur pour t et une pour f. Ces valeurs dépendent de l'injection considérée. Dans le cas des fermetures, l'injection est retardée pour avoir lieu lors de l'application de la fonction.

Par exemple, pour l'injection $(l', 7)$, l'environnement d'évaluation de l'expression $f\ 32$ est le suivant :
$\Gamma := (f, < \lambda x.(l : x - t, x + t), \{(t, 7)\} >); (t, 7)$
On a donc le jugement d'évaluation suivant :
$\vdash_{l':7} f\ 32 \mapsto (25, 39)$

Dans le cas de l'injection $(l, 3)$, l'environnement d'évaluation de l'expression $f\ 32$ est le suivant :
$\Gamma := (f, < \lambda x.(l : x - t, x + t), \{(t, 18)\} >); (t, 18)$
On évalue alors l'expression $(l : x - t, x + t)$ dans l'environnement $\{(t, 18)\}$ pour obtenir la valeur $(3, 50)$. On a donc le jugement d'évaluation suivant :

$\vdash_{l:3} f\ 32 \mapsto (3, 50)$

2.5.2.3 Énoncé formel du théorème

Théorème 2.5.1. $\forall(\Gamma, e, v)$,
$\Gamma \vdash e \mapsto v$
$\Rightarrow \forall l, ldna\_in\_eval(l, \Gamma, e)$
$\Rightarrow \forall v_l.\ \Gamma \vdash_{l:v_l} e \mapsto v$

Le prédicat $ldna\_in\_eval(l, \Gamma, e)$ signifie que le label l n'apparaît pas lors de l'évaluation de e dans Γ. C'est à dire qu'il n'apparaît ni dans l'expression e, ni dans les valeurs dans Γ des identificateurs apparaissant dans e.

Le prédicat $ldna\_in\_val(l, v)$ signifie que le label l n'apparaît pas dans la valeur v.

Les figures 2.6 et 2.7 fournissent la définition formelle de ces prédicats sous forme de règles d'inférence mutuellement récursives :

2.5.2.4 Preuve de correction

La preuve de correction se fait par induction sur le jugement d'évaluation de la sémantique opérationnelle.

Pour effectuer la preuve par induction, nous avons besoin de prouver une propriété un peu plus forte que celle énoncée ci-dessus. Nous modifions donc l'énoncé de la propriété à prouver, notre théorème de correction en sera une conséquence triviale. La propriété à prouver se formule alors :
$\forall(\Gamma, e, v)$,
$\Gamma \vdash e \mapsto v$
$\Rightarrow \forall l, ldna\_in\_eval(l, \Gamma, e)$
$\Rightarrow \forall v_l.\ \Gamma \vdash_{l:v_l} e \mapsto v$
$\wedge\ ldna\_in\_val(l, v)$

Dans la majorité des cas, la preuve est triviale. Les cas intéressants sont :

– le cas Annot

$$\text{LDNA-E-NUM}\ \frac{}{ldna\_in\_eval(l,\Gamma,n)} \qquad \text{LDNA-E-CONSTR-0}\ \frac{}{ldna\_in\_eval(l,\Gamma,C)} \qquad \text{LDNA-E-CONSTR-1}\ \frac{ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_eval(l,\Gamma,D(e))}$$

$$\text{LDNA-E-IDENT}\ \frac{ldna\_in\_val(l,\Gamma[x])}{ldna\_in\_eval(l,\Gamma,x)} \qquad \text{LDNA-E-IDENT-UNBOUND}\ \frac{\nexists v, \Gamma[x] = v}{ldna\_in\_eval(l,\Gamma,x)}$$

$$\text{LDNA-E-ABSTR}\ \frac{ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_eval(l,\Gamma,\lambda x.e)} \qquad \text{LDNA-E-ABSTR-REC}\ \frac{ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_eval(l,\Gamma,\texttt{rec}f.x.e)}$$

$$\text{LDNA-E-APPLY}\ \frac{ldna\_in\_eval(l,\Gamma,e_1)\quad ldna\_in\_eval(l,\Gamma,e_2)}{ldna\_in\_eval(l,\Gamma,e_1\ e_2)} \qquad \text{LDNA-E-IF}\ \frac{ldna\_in\_eval(l,\Gamma,e)\quad ldna\_in\_eval(l,\Gamma,e_1)\quad ldna\_in\_eval(l,\Gamma,e_2)}{ldna\_in\_eval(l,\Gamma,if\ e\ then\ e_1\ else\ e_2)}$$

$$\text{LDNA-E-MATCH}\ \frac{ldna\_in\_eval(l,\Gamma,e)\quad ldna\_in\_eval(l,\Gamma,e_1)\quad ldna\_in\_eval(l,\Gamma,e_2)}{ldna\_in\_eval(l,\Gamma,match\ e\ with\ p \to e_1 \mid x \to e_2)} \qquad \text{LDNA-E-ANNOT}\ \frac{l \neq l' \quad ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_eval(l,\Gamma,l\ :\ e)}$$

$$\text{LDNA-E-COUPLE}\ \frac{ldna\_in\_eval(l,\Gamma,e_1)\quad ldna\_in\_eval(l,\Gamma,e_2)}{ldna\_in\_eval(l,\Gamma,(e_1,e_2))} \qquad \text{LDNA-E-LETIN}\ \frac{ldna\_in\_eval(l,\Gamma,e_1)\quad ldna\_in\_eval(l,\Gamma,e_2)}{ldna\_in\_eval(l,\Gamma,let\ x\ =\ e_1\ in\ e_2)}$$

FIGURE 2.6 – Prédicat de non-apparition d'un label lors d'une évaluation

$$\text{LDNA-V-NUM}\ \frac{}{ldna\_in\_val(l,n)} \qquad \text{LDNA-V-BOOL}\ \frac{}{ldna\_in\_val(l,b)} \qquad \text{LDNA-V-CONSTR-0}\ \frac{}{ldna\_in\_val(l,C)} \qquad \text{LDNA-V-CONSTR-1}\ \frac{ldna\_in\_val(l,v)}{ldna\_in\_val(l,D(v))}$$

$$\text{LDNA-V-COUPLE}\ \frac{ldna\_in\_val(l,v_1)\quad ldna\_in\_val(l,v_2)}{ldna\_in\_val(l,(v_1,v_2))} \qquad \text{LDNA-V-CLOSURE}\ \frac{ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_val(l,<\lambda x.e,\Gamma>)} \qquad \text{LDNA-V-CLOSURE-REC}\ \frac{ldna\_in\_eval(l,\Gamma,e)}{ldna\_in\_val(l,<\texttt{rec}f.x.e,\Gamma>)}$$

FIGURE 2.7 – Prédicat de non-apparition d'un label dans une valeur

- les cas effectuant une liaison dans l'environnement (Letin, Match et Apply)
- le cas Ident

cas ANNOT Pour le cas Annot, nous utilisons l'hypothèse que le label l n'apparaît pas dans l'expression évaluée. Dans ce cas, la règle de la sémantique opérationnelle avec injection est la même que celle de la sémantique opérationnelle (elle ignore l'annotation).

cas des liaison Pour les cas effectuant une liaison dans l'environnement, nous avons besoin de savoir que le label n'apparaît pas dans la valeur liée pour appliquer notre hypothèse d'induction. Nous avons cette information grâce au renforcement de l'énoncé de notre théorème car toute valeur liée est issue d'une évaluation, donc puisque le label n'apparaît pas dans l'expression évaluée alors il n'apparaît pas dans la valeur résultat. Pour illustrer nos propos, prenons le cas du LET-BINDING. Nous avons évalué la sous-expression liée e_1 dans le même environnement que l'expression globale, donc nous savons par hypothèse d'induction que le label n'apparaît pas dans sa valeur. Cette information nous permet de prouver que le label n'apparaît pas dans l'environnement d'évaluation de la sous-expression e_2, puisque la valeur ajoutée à l'environnement est précisément le résultat de l'évaluation de e_1. Nous pouvons donc appliquer également l'hypothèse d'induction sur la seconde prémisse de la règle du LET-BINDING et conclure.

cas IDENT La règle Ident de la sémantique opérationnelle avec injection s'applique de façon identique à celle de la sémantique opérationnelle. Le jugement de la sémantique opérationnelle avec injection est donc immédiat à trouver. On déduit que le label n'apparaît pas dans la valeur trouvée dans l'environnement à partir de l'hypothèse que le label n'apparaît pas lors de l'évaluation.

2.6 Impact d'une injection

Maintenant que la notion d'injection a été clairement définie, nous pouvons aborder la notion d'***impact***. Cette dernière nous sera utile pour définir formellement la notion de dépendance.

Une fois un programme annoté par des points d'injection, on se demande si ces points d'injection ont un « impact » sur l'évaluation du programme. Autrement dit, on souhaite déterminer si l'injection d'une valeur à un certain point d'injection va modifier le comportement du programme. Puisque nous nous limitons à l'analyse de programmes qui terminent, un changement de comportement peut prendre deux formes possibles. Suite à l'injection, l'évaluation peut terminer sur une valeur différente ou bien ne pas terminer.

2.6.1 Impact sur la terminaison

Si le programme considéré a une valeur par la sémantique opérationnelle mais qu'il n'a pas de valeur par la sémantique avec injection pour une injection particulière (l, v_l), alors on dit que cette injection a un impact sur la terminaison de l'évaluation du programme.

On dira également que ce point d'injection (représenté par le label l) a un impact sur la terminaison du programme puisqu'il existe une injection sur ce label ayant un impact sur la terminaison du programme.

2.6.2 Impact sur la valeur

Si le programme considéré a une valeur par la sémantique opérationnelle et qu'il a une valeur différente par la sémantique avec injection pour une injection particulière (l, v_l), alors on dit que cette injection a un impact sur la valeur du programme.

On dira également que ce point d'injection (représenté par le label l) a un impact sur la valeur du programme puisqu'il existe une injection sur ce label ayant un impact sur la valeur du programme.

2.7 Dépendances d'une expression

On distingue deux types de dépendances distincts, issus des deux types d'impact présentés plus haut : les dépendances de terminaison et les dépendances de valeur.

2.7.1 Dépendances de terminaison

La terminaison de l'évaluation d'un programme dépend d'un point d'injection (représenté par un label) s'il existe une injection sur ce label ayant un impact sur la terminaison de l'évaluation. On dit alors que ce label fait partie des t-dépendances de ce programme (dans un environnement donné).

Lors de l'analyse d'un programme, le résultat de l'analyse contiendra un ensemble de labels représentant l'ensemble des t-dépendances du programme.

2.7.2 Dépendances de valeur

La valeur résultant de l'évaluation d'un programme dépend d'un point d'injection (représenté par un label) si il existe une injection sur ce label ayant un impact sur la valeur de l'évaluation. On dit alors que ce label fait partie des v-dépendances de ce programme (dans un environnement donné). Certains labels peuvent avoir un impact uniquement sur une partie de la valeur. Dans ce cas, on voudrait savoir le plus précisément possible quelle partie de la valeur dépend de quel label.

Lors de l'analyse d'un programme, le résultat de l'analyse devra contenir une valeur structurée dont chaque sous-terme peut être annoté par l'ensemble de ses v-dépendances.

2.7.3 Illustration par l'exemple

Considérons le programme suivant (noté e) :

```
let c = l : true in
let rec f x = match x with
              | (a,b) -> if a then b else f x
in
f (c, l' : 17)
```

Ce programme termine lors de son évaluation par la sémantique opérationnelle sur la valeur 17. On a le jugement suivant :

$\vdash e \mapsto 17$

Un label n'apparaissant pas dans le programme Un label l'' n'apparaissant pas dans le programme e ne peut pas faire partie de ses dépendances (que ce soit ses t-dépendances ou ses v-dépendances) puisqu'aucune injection sur l'' n'affecte le comportement du programme. On a le jugement suivant :
$\vdash_{l'':v_{l''}} e \mapsto 17$ (pour toute valeur injectée $v_{l''}$)

Le label l Le point d'injection représenté par le label l fait partie des t-dépendances du programme. En effet, il existe une injection $(l, false)$ ayant un impact sur la terminaison du programme. La sémantique avec injection ne fournit aucune valeur pour l'évaluation de e avec l'injection $(l, false)$.
$\nexists v. \vdash_{l:false} e \mapsto v$

Le label l' Le point d'injection représenté par le label l' fait partie des v-dépendances du programme. En effet, il existe une injection $(l', 18)$ ayant un impact sur la valeur du programme. La sémantique avec injection fournit une valeur (i.e. 18) pour l'évaluation de e avec l'injection $(l', 18)$ et cette valeur est différente de la valeur opérationnelle du programme (i.e. 17).
$\vdash_{l':18} e \mapsto 18$

Par contre, le label l' ne fait pas partie des t-dépendances du programme puisque pour toute injection sur l', le programme termine :
$\vdash_{l':v_{l'}} e \mapsto v_{l'}$ (pour toute valeur injectée $v_{l'}$).

Chapitre 3

Analyse dynamique

3.1 Introduction

Ce chapitre a pour but de présenter une analyse dynamique des dépendances d'un programme et sa preuve de correction, étape préliminaire à la définition et à la preuve d'une analyse statique présentée dans le chapitre suivant.

Notre analyse dynamique est appelée ***sémantique instrumentée*** et se trouve présentée à la fin de ce chapitre (cf. section 3.5).

Pour présenter cette analyse et en prouver sa correction, nous devons tout d'abord introduire deux sémantiques intermédiaires : la sémantique avec injection dans un t-environnement sur-instrumenté (cf. section 3.3) et la sémantique sur-instrumentée (cf. section 3.4). Ces deux sémantiques intermédiaires permettent de faire le lien entre la notion de dépendance et la sémantique dynamique étape par étape. Chaque étape a son importance dans la simplification de la preuve de correction. La figure 3.1 présente une vue globale des différentes sémantiques intermédaires entre la sémantique opérationnelle avec injection et la sémantique instrumentée.

L'analyse dynamique (sémantique instrumentée) a pour but d'analyser le programme au cours de son évaluation. Le résultat fourni est la valeur du programme (au sens de la sémantique opérationnelle) sur laquelle on ajoute des annotations de dépendance. Chaque sous-terme de la valeur contient ses propres annotations pour savoir avec précision quelles parties de la valeur dépendent de tel label et quelles parties n'en dépendent pas. Si on

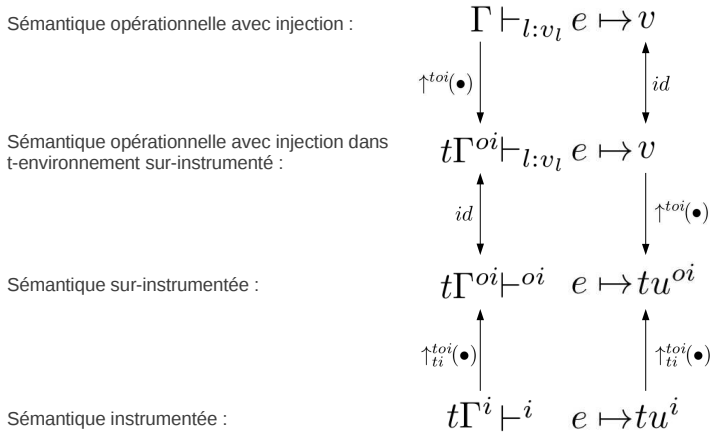

FIGURE 3.1 – Sémantiques intermédiaires pour l'analyse dynamique

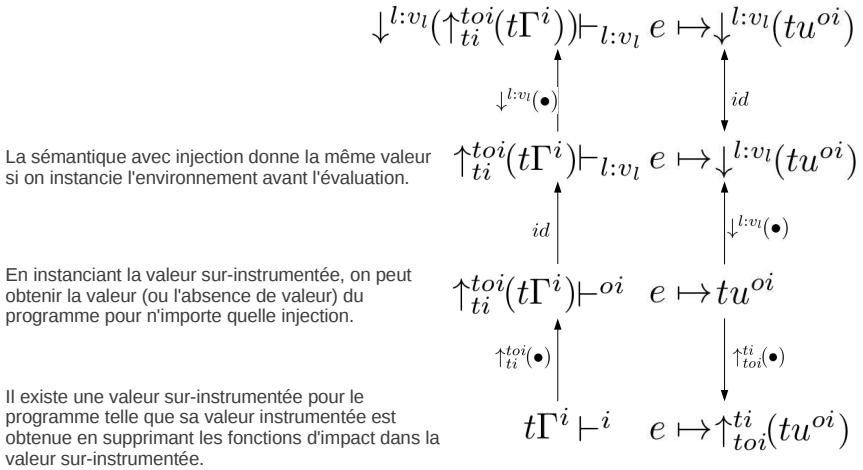

FIGURE 3.2 – Enchaînement des preuves des sémantiques intermédiaires pour l'analyse dynamique

supprime toutes les annotations de dépendance, on obtient exactement la même valeur qu'après l'évaluation du programme par la sémantique opérationnelle.

Les annotations ajoutées par l'analyse dynamique correspondent aux dépendances du programme (c'est-à-dire l'ensemble des labels ayant un impact sur l'évaluation de ce programme, comme défini plus haut (cf. section 2.7)). Nous prouvons formellement cette affirmation par le théorème de correction de l'analyse dynamique (cf. section 3.6.4).

Pour arriver à cette preuve, nous prouvons successivement la correction de la sémantique avec injection dans un t-environnement sur-instrumenté (cf. section 3.3.2), puis la correction de la sémantique sur-instrumentée (cf. section 3.4.2) et enfin la correction de la sémantique instrumentée (cf. section 3.5.3). Les propriétés de correction de ces trois sémantiques s'enchaînent pour obtenir le théorème de correction de l'analyse dynamique. La figure 3.2 illustre l'enchaînement de ces propriétés de correction.

Dans ce chapitre, nous présenterons tout d'abord la sur-instrumentation des valeurs (cf. section 3.2) qui nous permettra d'introduire la première sémantique intermédiaire pour l'analyse dynamique : la sémantique opérationnelle avec injection dans un t-environnement sur-instrumenté (cf. section 3.3). Nous présenterons ensuite la seconde sémantique intermédiaire : la sémantique sur-instrumentée (cf. section 3.4). Enfin nous pourrons présenter l'analyse dynamique elle-même : la sémantique instrumentée (cf. section 3.5). La présentation de chaque sémantique sera accompagnée de l'énoncé de son théorème de correction par rapport à la sémantique précédente ainsi que d'une explication sur sa preuve. Nous terminerons ce chapitre avec l'énoncé du théorème global de correction de l'analyse dynamique et une explication sur sa preuve utilisant la propriété de correction de chaque sémantique par rapport à la sémantique précédente.

3.2 Sur-instrumentation des valeurs

Jusqu'à maintenant, nous avons vu comment évaluer un programme à l'aide de la sémantique opérationnelle (cf. section 2.3) pour obtenir sa valeur opérationnelle (également appelée valeur de référence du programme). Nous avons également vu comment évaluer un programme à l'aide de la sémantique avec injection (cf. section 2.5) pour obtenir sa valeur

après injection, pour n'importe quelle injection possible.

Un même programme possède donc plusieurs valeurs : sa valeur de référence (obtenue par la sémantique opérationnelle) et une valeur après injection (ou une absence de valeur après injection) pour chaque injection possible.

Nous souhaitons maintenant intégrer toutes ces informations sur un programme en un seul et unique terme appelé « valeur sur-instrumentée ». La valeur sur-instrumentée d'un programme permettra alors de retrouver, à l'aide d'un mécanisme d'instanciation, la valeur de référence de ce programme ainsi que la valeur (ou l'absence de valeur) de ce programme après n'importe quelle injection.

Pour cela, nous allons annoter la valeur de référence du programme avec des informations de dépendance sur chaque sous-terme de celle-ci.

3.2.1 Valeurs

Comme nous l'avons vu plus haut (cf. section 2.7), un programme possède des t-dépendances (dépendances de terminaison) ainsi que des v-dépendances (dépendances de valeur). Lors de la sur-instrumentation des valeurs, nous associons à toute valeur (résultat de l'évaluation d'un programme) l'ensemble des t-dépendances du programme évalué. Ensuite, nous associons à chaque sous-terme de la valeur un ensemble de v-dépendances correspondant aux points d'injection ayant un impact sur ce sous-terme.

Une *t-valeur sur-instrumentée* (notée tu^{oi}) est composée d'un ensemble de t-dépendances td^{oi} et d'une v-valeur sur-instrumentée u^{oi}.

$$tu^{oi} := \quad [\ td^{oi}\ \mid\ u^{oi}\ ] \quad \text{Valeur avec annotation de } t\text{-dépendances}$$

Une *v-valeur sur-instrumentée* (notée u^{oi}) est composée d'un v-ensemble de dépendances d^{oi} et d'une valeur simple sur-instrumentée v^{oi}.

$$u^{oi} := \quad [\ d^{oi}\ \mid\ v^{oi}\ ] \quad \text{Valeur avec annotation de } v\text{-dépendances}$$

Enfin, une *valeur simple sur-instrumentée* (notée v^{oi}) est une valeur simple dont les

sous-termes sont des v-valeurs sur-instrumentées u^{oi}.

$$\begin{array}{rll} v^{oi} := & n \mid b \mid C \mid D(u^{oi}) \mid (u_1^{oi}, u_2^{oi}) & \text{Constructeurs de données} \\ & < \lambda x.e, \Gamma^{oi} > & \text{Fermeture} \\ & < \texttt{rec}f.x.e, \Gamma^{oi} > & \text{Fermeture récursive} \end{array}$$

3.2.2 Ensembles de dépendances

On distingue 2 types de dépendances :

- les *t-dépendances*, concernant la terminaison de l'évaluation
- les *v-dépendances*, concernant le résultat de l'évaluation

Un *ensemble de t-dépendances* td^{oi} est un ensemble de labels dans lequel on associe à chaque label une t-fonction d'impact. Cette *t-fonction d'impact* est une fonction booléenne sur les valeurs nous indiquant pour chaque injection possible, si l'évaluation terminerait ou non (ce qui permet de modéliser un impact aboutissant soit à une boucle infinie, soit à une erreur d'exécution). Si une t-fonction d'impact retourne *true*, cela signifie qu'il y a non-terminaison, sinon, cela signifie que l'évaluation correspondante termine.

$$\begin{array}{rll} td^{oi} := & (l_1, tf_1); \ldots (l_n, tf_n) & t\text{-dépendances} \\ tf : & v \rightarrow \mathit{bool} & t\text{-fonction d'impact} \end{array}$$

Un *ensemble de v-dépendances* d^{oi} est un ensemble de labels dans lequel on associe à chaque label une v-fonction d'impact. Cette *v-fonction d'impact* est une fonction des valeurs vers les valeurs nous indiquant pour chaque injection possible, quelle serait la valeur de l'évaluation avec injection.

$$\begin{array}{rll} d^{oi} := & (l_1, f_1); \ldots (l_n, f_n) & v\text{-dépendances} \\ f : & v \rightarrow v & v\text{-fonction d'impact} \end{array}$$

3.2.3 Environnements

On distingue 2 types d'environnements sur-instrumentés :

- les environnements dans lesquels on évalue les expressions
- les environnements qui se trouvent encapsulés dans les valeurs (fermetures et les fermetures récursives).

Dans le cas de l'évaluation d'une expression, on a besoin de connaître les t-dépendances de chaque identificateur. On utilisera donc la notion de *t-environnement sur-instrumenté*.

En revanche, pour les environnements encapsulés dans une valeur (via une fermeture), il suffit de connaître les t-dépendances globales de cette valeur et il n'est pas nécessaire d'avoir les t-dépendances de chaque identificateur lié. Dans ce cas, on utilisera un *v-environnement sur-instrumenté* sans t-dépendances. En effet, pour qu'un tel environnement ait pu être créé et emprisonné dans une fermeture, il faut nécessairement que tout ce qui a précédé la création de cette fermeture ait terminé. Les valeurs présentes dans l'environnement proviennent de l'évaluation d'une expression de liaison (liaison *let in*, filtrage par motif ou application de fonction). Dans tous les cas possibles, lorsqu'on effectue la liaison d'une valeur dans l'environnement, les t-dépendances de cette valeur sont toujours ajoutés à la valeur de l'expression évaluée (cf. figure 3.5, règles OI-LETIN, OI-MATCH, OI-MATCH-VAR, OI-APPLY et OI-REC-APPLY).

Un *t-environnement sur-instrumenté* permet de lier chaque identificateur à une t-valeur sur-instrumentée.

$$t\Gamma^{oi} := \quad (x_1, tu_1^{oi}); \ldots; (x_n, tu_n^{oi}) \quad t\text{-environnement sur-instrumenté}$$

Un *v-environnement sur-instrumenté* permet de lier chaque identificateur à une v-valeur sur-instrumentée.

$$\Gamma^{oi} := \quad (x_1, u_1^{oi}); \ldots; (x_n, u_n^{oi}) \quad v\text{-environnement sur-instrumenté}$$

3.2.4 Valeur de référence d'une valeur sur-instrumentée

Pour obtenir la valeur de référence d'une valeur sur-instrumentée, il suffit de retirer toutes les annotations de dépendance. La valeur de référence d'une valeur sur-instrumentée est notée $\uparrow_{toi}(tu^{oi})$.

Formellement, l'extraction de la valeur de référence d'une t-valeur sur-instrumentée se définit comme suit :

Valeur de référence d'une t-valeur sur-instrumentée :

$$\uparrow_{toi}(\ [\ td^{oi}\ \mid\ u^{oi}\ ]) :=\ \uparrow_{oi}(u^{oi})$$

Valeur de référence d'une v-valeur sur-instrumentée :

$$\uparrow_{oi}(\ [\ d^{oi}\ \mid\ v^{oi}\ ]) :=\ \uparrow_{oi}(v^{oi})$$

Valeur de référence d'une valeur simple sur-instrumentée :

$$\begin{array}{rl}
\uparrow_{oi}(n) := & n \\
\uparrow_{oi}(b) := & b \\
\uparrow_{oi}(C) := & C \\
\uparrow_{oi}(D(u^{oi})) := & D(\uparrow_{oi}(u^{oi})) \\
\uparrow_{oi}((u_1^{oi}, u_2^{oi})) := & (\uparrow_{oi}(u_1^{oi}), \uparrow_{oi}(u_2^{oi})) \\
\uparrow_{oi}(<\lambda x.e, \Gamma^{oi}>) := & <\lambda x.e, \uparrow_{oi}(\Gamma^{oi})> \\
\uparrow_{oi}(<\texttt{rec}f.x.e, \Gamma^{oi}>) := & <\texttt{rec}f.x.e, \uparrow_{oi}(\Gamma^{oi})>
\end{array}$$

Environnement de référence d'un t-environnement sur-instrumenté :

$$\begin{array}{rl}
\uparrow_{toi}(\emptyset) := & \emptyset \\
\uparrow_{toi}((x, tu^{oi}) \oplus t\Gamma^{oi}) := & (x, \uparrow_{toi}(tu^{oi})) \oplus \uparrow_{toi}(t\Gamma^{oi})
\end{array}$$

Environnement de référence d'un v-environnement sur-instrumenté :

$$\begin{array}{rl}
\uparrow_{oi}(\emptyset) := & \emptyset \\
\uparrow_{oi}((x, u^{oi}) \oplus \Gamma^{oi}) := & (x, \uparrow_{oi}(u^{oi})) \oplus \uparrow_{oi}(\Gamma^{oi})
\end{array}$$

3.2.5 Instanciation d'une valeur sur-instrumentée

La valeur sur-instrumentée d'un programme contient toute l'information nécessaire pour savoir quelle serait le comportement du programme lors de son évaluation par la sémantique avec injection et ce pour n'importe quelle injection.

Pour extraire la valeur après une injection (l, v_l) depuis une valeur sur-instrumentée tu^{oi}, on utilise une fonction d'instanciation notée $\downarrow^{l:v_l}(tu^{oi})$.

Tout d'abord, il faut regarder si le label l appartient à l'ensemble des t-dépendances. Si c'est le cas, la t-fonction d'impact associée au label nous indique si le programme aurait ou non terminé sur une valeur lors de son évaluation par la sémantique avec injection. Si le label l n'appartient pas aux t-dépendances, alors le programme a forcément une valeur si on l'évalue à l'aide de la sémantique avec injection.

Ensuite, pour chaque sous-terme de la valeur, on regarde si le label l appartient aux v-dépendances correspondant au sous-terme considéré. Si c'est le cas, alors la v-fonction d'impact nous indique la valeur du sous-terme pour l'injection considérée. Si le label l n'appartient pas aux v-dépendances du sous-terme, alors la valeur du sous-terme après injection correspond à sa valeur de référence dans laquelle on a instancié chaque sous-terme.

On remarquera que puisque les valeurs simples sur-instrumentées (v^{oi}) ne contiennent que des valeurs sur-instrumentées (u^{oi}), il n'est pas question de terminaison lors du parcours de ces sous-termes. Ceci est parfaitement justifié puisque la terminaison est une propriété globale pour une t-valeur sur-instrumentée.

La fonction d'instanciation d'une t-valeur sur-instrumentée se définit formellement comme suit :

Instanciation des t-valeurs sur-instrumentées :

$$\downarrow^{l:v_l}(\ [\ td^{oi}\ \mid\ u^{oi}\ ]) := \downarrow^{l:v_l}(u^{oi}) \quad \text{si } atif_{l:v_l}(td^{oi}) = false$$

La fonction d'instanciation des t-valeurs sur-instrumentées est une fonction partielle. Elle n'est définie que lorsque $atif_{l:v_l}(td^{oi}) = false$. Dans le cas où $atif_{l:v_l}(td^{oi}) = true$, on dit que la t-valeur sur-instrumentée n'est pas instanciable et on écrit $\nexists v.\ \downarrow^{l:v_l}(\ [\ td^{oi}\ \mid\ u^{oi}\ ]) = v$.

Instanciation des valeurs sur-instrumentées :

$$\begin{array}{rcll} \downarrow^{l:v_l}(\ [\ d^{oi}\ \mid\ v^{oi}\ ]) & := & aif_{l:v_l}(d^{oi}) & \text{si } l \in d^{oi} \\ \downarrow^{l:v_l}(\ [\ d^{oi}\ \mid\ v^{oi}\ ]) & := & \downarrow^{l:v_l}(v^{oi}) & \text{si } l \notin d^{oi} \end{array}$$

Instanciation des valeurs simples sur-instrumentées :

$$\begin{array}{rcl} \downarrow^{l:v_l}(n) & := & n \\ \downarrow^{l:v_l}(b) & := & b \\ \downarrow^{l:v_l}(C) & := & C \\ \downarrow^{l:v_l}(D(u^{oi})) & := & D(\downarrow^{l:v_l}(u^{oi})) \\ \downarrow^{l:v_l}((u_1^{oi}, u_2^{oi})) & := & (\downarrow^{l:v_l}(u_1^{oi}), \downarrow^{l:v_l}(u_2^{oi})) \\ \downarrow^{l:v_l}(< \lambda x.e, \Gamma^{oi} >) & := & < \lambda x.e, \downarrow^{l:v_l}(\Gamma^{oi}) > \\ \downarrow^{l:v_l}(< \texttt{rec} f.x.e, \Gamma^{oi} >) & := & < \texttt{rec} f.x.e, \downarrow^{l:v_l}(\Gamma^{oi}) > \end{array}$$

Instanciation des environnements sur-instrumentés :

$$\begin{array}{rcl} \downarrow^{l:v_l}(\{\}) & := & \{\} \\ \downarrow^{l:v_l}((x, u^{oi}) \oplus \Gamma^{oi}) & := & (x, \downarrow^{l:v_l}(u^{oi})) \oplus \downarrow^{l:v_l}(\Gamma^{oi}) \end{array}$$

Application des t-fonctions d'impact :

$$\begin{array}{rcll} atif_{l:v_l}((l, tf^{oi}); td^{oi}) & := & tf^{oi}(v_l) \vee atif_{l:v_l}(td^{oi}) & \\ atif_{l:v_l}((l', tf^{oi}); td^{oi}) & := & atif_{l:v_l}(td^{oi}) & \text{si } l' \neq l \\ atif_{l:v_l}(\emptyset) & := & false & \end{array}$$

Application des fonctions d'impact :

$$\begin{array}{rcll} aif_{l:v_l}((l, f^{oi}); d^{oi}) & := & f^{oi}(v_l) & \\ aif_{l:v_l}((l', f^{oi}); d^{oi}) & := & aif_{l:v_l}(d^{oi}) & \text{si } l' \neq l \end{array}$$

3.2.6 Conversion

Compte tenu du nombre de notions similaires à propos des environnements et des valeurs, et pour éviter toute confusion, les conversions seront toujours explicites.

Par exemple, la notation Γ^{oi} désigne un environnement sur-instrumenté qui n'est pas forcément en relation avec le t-environnement sur-instrumenté $t\Gamma^{oi}$. Lorsqu'il y a une relation entre deux environnements, celle-ci est exprimée explicitement à l'aide d'une fonction de conversion. Par exemple $\Gamma^{oi} = \uparrow^{oi}_{toi}(t\Gamma^{oi})$ exprime que l'environnement Γ^{oi} est le résultat de l'application de la fonction de conversion $\uparrow^{oi}_{toi}(\bullet)$ à l'environnement $t\Gamma^{oi}$. Cette fonction convertit un t-environnement sur-instrumenté (resp. une t-valeur sur-instrumentée) en un v-environnement sur-instrumenté (resp. une v-valeur sur-instrumentée).

3.3 Sémantique avec injection dans un t-environnement sur-instrumenté

La sémantique avec injection que nous avons présentée plus haut (cf. section 2.5) nous a permis de définir formellement la notion de dépendance vis-à-vis de laquelle nous pourrons prouver la correction de notre analyse.

Nous avons également défini (cf. section 3.2) une manière d'annoter la valeur d'un programme avec des informations de dépendance de façon à intégrer en un seul et même terme le comportement du programme pour toute injection. Plus loin, nous définirons (cf. section 3.4) une sémantique permettant d'inférer une valeur sur-instrumentée pour tout programme dont l'évaluation de référence termine. Cette sémantique sur-instrumentée devra manipuler au sein de l'environnement d'évaluation des valeurs elles-mêmes sur-instrumentées (obtenues par l'évaluation d'un programme précédent ou spécifiées par l'utilisateur) et non des valeurs simples comme dans la sémantique opérationnelle.

Afin de faciliter la preuve de correction de la sémantique sur-instrumentée, nous introduisons maintenant une extension de la sémantique avec injection. Cette extension donnera un sens à l'évaluation avec injection d'un programme dans un environnement sur-instrumenté.

L'expression est évaluée dans un t-environnement sur-instrumenté, ce qui permet de spécifier pour chaque identificateur présent dans l'environnement, quelle aurait été sa valeur si on l'avait obtenue après telle ou telle injection. Le résultat de l'évaluation d'une expression par la sémantique avec injection dans un t-environnement sur-instrumenté est une valeur sans annotation, tout comme pour la sémantique opérationnelle avec injection ou pour la sémantique opérationnelle usuelle. Lorsque les annotations du t-environnement sur-instrumenté sont vides, alors la sémantique avec injection dans le t-environnement sur-instrumenté coïncide avec la sémantique avec injection dans son environnement de référence.

3.3.1 Règles d'inférence

Formellement, la sémantique avec injection dans un t-environnement sur-instrumenté prend la forme d'un jugement d'évaluation similaire à celui de la sémantique opérationnelle avec injection. On note le jugement d'évaluation de la manière suivante :

$$t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v$$

La seule différence par rapport à la sémantique opérationnelle avec injection est l'environnement d'évaluation. Dans la sémantique que nous définissons ici, l'environnement d'évaluation est un t-environnement sur-instrumenté. Il contient plus d'information que l'environnement d'évaluation de la sémantique opérationnelle avec injection. En effet dans l'environnement opérationnel, chaque identificateur est lié à une unique valeur. Celle-ci représente la valeur de l'identificateur quelque soit l'injection considérée. Par contre, dans un t-environnement sur-instrumenté, la valeur d'un identificateur dans l'environnement peut changer en fonction de l'injection considérée.

Cette nouvelle sémantique est plus générale que la sémantique opérationnelle avec injection que nous avions définie pour introduire notre notion de dépendance. En utilisant la fonction injective $\uparrow^{toi}(\bullet)$ qui décore tout environnement opérationnel avec des annotations de dépendance vides, on obtient un plongement de l'ensemble des jugements de la sémantique opérationnelle avec injection vers l'ensemble des jugements de la sémantique avec injection dans un t-environnement sur-instrumenté.

Ainsi lorsque nous prouverons la correction de la sémantique avec injection dans un t-environnement sur-instrumenté, nous en déduirons trivialement la correction de la sémantique opérationnelle avec injection par une simple spécialisation du théorème de correction.

De même lorsque nous prouverons la correction de la sémantique sur-instrumentée par rapport à la sémantique avec injection dans un t-environnement sur-instrumenté, nous en déduirons trivialement la correction de la sémantique sur-instrumentée par rapport à la sémantique opérationnelle avec injection.

Remarque : de la même manière que la sémantique opérationnelle avec injection, la sémantique avec injection dans un t-environnement sur-instrumenté est paramétrée par le

label l sur lequel on effectue l'injection et par la valeur injectée v_l. Ces paramètres sont notés en indice du jugement d'évaluation ($\vdash_{l:v_l}$).

Bien que l'environnement d'évaluation contienne des annotations de dépendance, la sémantique avec injection ne fait qu'utiliser ces annotations. Cette sémantique n'a pas pour but de calculer des dépendances. Lors de l'évaluation d'un programme, elle ne produit aucune annotation. En particulier, lorsqu'une valeur est ajoutée à l'environnement, elle contient toujours des annotations vides, la valeur ajoutée n'étant utilisée que pour une injection particulière : l'injection se trouvant en paramètre du jugement d'évaluation.

Les règles identiques à la sémantique opérationnelle usuelle sont réunies sur la figure 3.3. Les deux règles spécifiques à la sémantique opérationnelle avec injection se trouvent sur la figure 3.4.

La majorité des règles sont identiques à la sémantique opérationnelle (modulo l'ajout d'annotations vides via la fonction $\uparrow^{toi}(\bullet)$ lors de la construction d'un environnement). En plus des deux règles d'évaluation d'une expression annotée, trois autres règles sont spécifiques à la sémantique avec injection dans un t-environnement sur-instrumenté. Lorsqu'on va chercher la valeur d'un identificateur dans l'environnement, on instancie la t-valeur sur-instrumentée trouvée pour l'injection considérée. Lorsqu'on construit une fermeture (ou une fermeture récursive), on instancie le t-environnement sur-instrumenté pour l'injection considérée avant de l'encapsuler dans la fermeture.

3.3.2 Correction

3.3.2.1 Énoncé informel du théorème

La sémantique avec injection dans un t-environnement sur-instrumenté permet de calculer la valeur d'un programme dans un t-environnement sur-instrumenté. Pour chaque identificateur, cet environnement contient une t-valeur sur-instrumentée. Cette valeur nous permet de connaître la valeur opérationnelle de l'identificateur pour n'importe quelle injection. En particulier, lorsque l'on évalue un programme pour une injection particulière (l, v_l), sa valeur opérationnelle est l'instanciation de sa t-valeur sur-instrumentée pour l'in-

$$\text{INJ-NUM}\ \frac{}{t\Gamma^{oi} \vdash_{l:v_l} n \mapsto n} \qquad \text{INJ-LETIN}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \qquad (x, \uparrow^{toi}(v_1)) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2}{t\Gamma^{oi} \vdash_{l:v_l} let\ x\ =\ e_1\ in\ e_2 \mapsto v_2}$$

$$\text{INJ-APPLY}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto <\lambda x.e, \Gamma_1> \quad t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \quad (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e \mapsto v}{t\Gamma^{oi} \vdash_{l:v_l} e_1\ e_2 \mapsto v}$$

$$\text{INJ-APPLY-REC}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \qquad v_1 = <\texttt{rec}f.x.e, \Gamma_1> \quad t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \quad (f, \uparrow^{toi}(v_1)) \oplus (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e \mapsto v}{t\Gamma^{oi} \vdash_{l:v_l} e_1\ e_2 \mapsto v}$$

$$\text{INJ-IF-TRUE}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto true \qquad t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1}{t\Gamma^{oi} \vdash_{l:v_l} if\ e\ then\ e_1\ else\ e_2 \mapsto v_1} \qquad \text{INJ-IF-FALSE}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto false \qquad t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2}{t\Gamma^{oi} \vdash_{l:v_l} if\ e\ then\ e_1\ else\ e_2 \mapsto v_2}$$

$$\text{INJ-MATCH}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \qquad v, p \vdash_p \Gamma_p \quad \uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1}{t\Gamma^{oi} \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_1} \qquad \text{INJ-MATCH-VAR}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \qquad v, p \vdash_p \bot \quad (x, \uparrow^{toi}(v)) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2}{t\Gamma^{oi} \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_2}$$

$$\text{INJ-CONSTR-0}\ \frac{}{t\Gamma^{oi} \vdash_{l:v_l} C \mapsto C} \qquad \text{INJ-CONSTR-1}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v}{t\Gamma^{oi} \vdash_{l:v_l} D(e) \mapsto D(v)} \qquad \text{INJ-COUPLE}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \qquad t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2}{t\Gamma^{oi} \vdash_{l:v_l} (e_1, e_2) \mapsto (v_1, v_2)}$$

FIGURE 3.3 – Sémantique avec injection dans un t-environnement sur-instrumenté : règles d'inférence identiques à la sémantique usuelle

$$\text{INJ-IDENT}\ \frac{v = \downarrow^{l:v_l}(t\Gamma^{oi}[x])}{t\Gamma^{oi} \vdash_{l:v_l} x \mapsto v} \qquad \text{INJ-ANNOT-SAME}\ \frac{}{t\Gamma^{oi} \vdash_{l:v_l} l\ :\ e \mapsto v_l} \qquad \text{INJ-ANNOT-OTHER}\ \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \qquad l \neq l'}{t\Gamma^{oi} \vdash_{l:v_l} l'\ :\ e \mapsto v}$$

$$\text{INJ-ABSTR}\ \frac{}{t\Gamma^{oi} \vdash_{l:v_l} \lambda x.e \mapsto <\lambda x.e, \downarrow^{l:v_l}(t\Gamma^{oi})>} \qquad \text{INJ-ABSTR-REC}\ \frac{}{t\Gamma^{oi} \vdash_{l:v_l} \texttt{rec}f.x.e \mapsto <\texttt{rec}f.x.e, \downarrow^{l:v_l}(t\Gamma^{oi})>}$$

FIGURE 3.4 – Sémantique avec injection dans un t-environnement sur-instrumenté : règles d'inférence spécifiques

jection (l, v_l). L'évaluation d'un programme dans un t-environnement sur-instrumenté pour une injection (l, v_l) doit donc retourner la même valeur que son évaluation opérationnelle pour l'injection (l, v_l) dans l'environnement opérationnel obtenu par instanciation du t-environnement sur-instrumenté pour l'injection (l, v_l).

Dit d'une manière plus concise, on peut exprimer informellement le théorème de correction de notre sémantique avec injection dans un t-environnement sur-instrumenté ainsi : la sémantique avec injection doit être la même si on instancie ou non l'environnement avant l'évaluation.

3.3.2.2 Illustration par l'exemple

Pour mieux appréhender la signification de ce théorème de correction, voici un exemple très simple d'évaluation de programme par la sémantique avec injection dans un t-environnement sur-instrumenté et dans son instanciation.

Notons $t\Gamma^{oi}$ l'environnement suivant :

$t\Gamma^{oi} \equiv (y,\ [\ td_y^{oi}\ \mid\ [\ d_y^{oi}\ \mid\ 7\ ]\ ]) \oplus (z,\ [\ td_z^{oi}\ \mid\ [\ d_z^{oi}\ \mid\ 4\ ]\ ])$

avec :

$td_y^{oi} \equiv (l, tf_y^l)$ où $tf_y^l(v)$ vaut $false$ si v est un entier positif, $true$ sinon

$d_y^{oi} \equiv (l, f_y^l)$ où $f_y^l(v)$ vaut le double de v si v est un entier positif, 0 sinon

$td_z^{oi} \equiv \emptyset$

$d_z^{oi} \equiv (l', f_z^{l'})$ où $f_z^{l'}(v)$ vaut 3 si v est le constructeur de donnée C, 5 sinon

$l \neq l'$

Notons e le programme suivant :

```
let f x = (x, x+z) in
f (y + l'':2)
```

avec $l \neq l''$ et $l' \neq l''$

Exemple 1 : impact sur la valeur d'un identificateur

Considérons l'injection de la valeur 8 sur le label l. La sémantique avec injection dans le t-environnement sur-instrumenté nous donne le jugement suivant :

$t\Gamma^{oi} \vdash_{l:8} e \mapsto (18, 22)$

En instanciant le t-environnement sur-instrumenté, nous obtenons :
$\downarrow^{l:8}(t\Gamma^{oi}) = (y, 16) \oplus (z, 4)$

La sémantique opérationnelle avec injection dans cet environnement nous donne alors le même résultat que la sémantique avec injection dans le t-environnement sur-instrumenté :
$(y, 16) \oplus (z, 4) \vdash_{l:8} e \mapsto (18, 22)$

Exemple 2 : impact sur la valeur d'un identificateur

Considérons l'injection de la valeur C sur le label l'. La sémantique avec injection dans le t-environnement sur-instrumenté nous donne le jugement suivant :
$t\Gamma^{oi} \vdash_{l':C} e \mapsto (9, 12)$

En instanciant le t-environnement sur-instrumenté, nous obtenons :
$\downarrow^{l':C}(t\Gamma^{oi}) = (y, 7) \oplus (z, 3)$

La sémantique opérationnelle avec injection dans cet environnement nous donne alors le même résultat que la sémantique avec injection dans le t-environnement sur-instrumenté :
$(y, 7) \oplus (z, 3) \vdash_{l':C} e \mapsto (9, 12)$

Exemple 3 : impact sur la terminaison d'un identificateur

Considérons l'injection de la valeur -4 sur le label l. La t-valeur sur-instrumentée de l'identificateur y n'est pas instanciable. La sémantique avec injection dans le t-environnement sur-instrumenté ne fournit donc aucun jugement, puisque la sous-expression y ne trouve aucune règle d'inférence permettant de déduire un jugement d'évaluation.

Le t-environnement sur-instrumenté n'est donc pas instanciable lui non plus. Nous sommes dans une situation où l'évaluation de la partie précédente du programme (celle qui a permi de construire l'environnement) ne termine pas pour l'injection considérée $(l, -4)$.

Il n'est alors pas possible de donner une valeur au programme par la sémantique opérationnelle avec l'injection $(l, -4)$ puisque l'environnement d'évaluation correspondant

n'existe pas.

Exemple 4 : impact sur un point d'injection sur programme

Considérons l'injection de la valeur 17 sur le label l''. La sémantique avec injection dans le t-environnement sur-instrumenté nous donne le jugement suivant :
$t\Gamma^{oi} \vdash_{l'':17} e \mapsto (24, 28)$

En instanciant le t-environnement sur-instrumenté, nous obtenons :
$\downarrow^{l'':17}(t\Gamma^{oi}) = (y, 7) \oplus (z, 4)$

La sémantique opérationnelle avec injection dans cet environnement nous donne alors le même résultat que la sémantique avec injection dans le t-environnement sur-instrumenté :
$(y, 7) \oplus (z, 4) \vdash_{l'':17} e \mapsto (24, 28)$

3.3.2.3 Énoncé formel du théorème

Théorème 3.3.1 (Correction de la sémantique avec injection dans un t-environnement sur-instrumenté)**.**
$\forall(t\Gamma^{oi}, \Gamma, e, v, l, v_l),\ t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \Rightarrow \Gamma = \downarrow^{l:v_l}(t\Gamma^{oi}) \Rightarrow \Gamma \vdash_{l:v_l} e \mapsto v$

3.3.2.4 Preuve de correction

La preuve se fait trivialement par induction sur le jugement de la sémantique avec injection dans le t-environnement sur-instrumenté.

Voici quelques explications sommaires sur quelques cas de la preuve :

INJ-ABSTR Dans le cas de l'évaluation d'une abstraction (récursive ou non), l'environnement encapsulé dans la fermeture est l'instanciation du t-environnement sur-instrumenté. La sémantique opérationnelle avec injection coïncide donc avec notre nouvelle sémantique.

INJ-IDENT Dans le cas de l'évaluation d'un identificateur, il suffit de remarquer que l'instanciation de la t-valeur sur-instrumentée d'un identificateur est la même que la valeur opérationnelle de cet identificateur dans l'instanciation du t-environnement sur-instrumenté.

INJ-MATCH Le cas du filtrage par motif est lui-aussi très simple.

Supposons que nous avons un jugement de la forme suivante :

$$\text{INJ-MATCH}\quad \frac{t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \qquad v, p \vdash_p \Gamma_p \qquad \uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1}{t\Gamma^{oi} \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_1}$$

Les hypothèses d'induction nous permettent de déduire ces jugements :

$\downarrow^{l:v_l}(t\Gamma^{oi}) \vdash_{l:v_l} e \mapsto v$

et $\downarrow^{l:v_l}(\uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi}) \vdash_{l:v_l} e_1 \mapsto v_1$

Enfin, puisque $\uparrow^{toi}(\bullet)$ n'ajoute que des dépendances vides, nous avons :

$$\downarrow^{l:v_l}(\uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi}) = \Gamma_p \oplus \downarrow^{l:v_l}(t\Gamma^{oi})$$

Nous pouvons alors appliquer la règle de la sémantique opérationnelle avec injection permettant de conclure :

$$\text{OPINJ-MATCH}\quad \frac{\downarrow^{l:v_l}(t\Gamma^{oi}) \vdash_{l:v_l} e \mapsto v \qquad v, p \vdash_p \Gamma_p \qquad \Gamma_p \oplus \downarrow^{l:v_l}(t\Gamma^{oi}) \vdash_{l:v_l} e_1 \mapsto v_1}{\downarrow^{l:v_l}(t\Gamma^{oi}) \vdash_{l:v_l} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto v_1}$$

3.4 Sémantique sur-instrumentée

Ce chapitre a pour but de présenter une seconde sémantique intermédiaire utilisée pour la preuve de notre analyse dynamique : la sémantique sur-instrumentée. Cette sémantique n'est pas nécessaire pour définir l'analyse dynamique mais a pour but de simplifier et de faciliter la preuve de correction de cette dernière.

Nous avons commencé par définir la sémantique opérationnelle avec injection qui donne un sens à la notion d'injection lors de l'évaluation d'un programme. Cette sémantique évalue un programme dans un environnement opérationnel pour une injection donnée. L'environnement d'évaluation doit alors contenir la valeur opérationnelle de chaque identificateur pour l'injection considérée. Si nous souhaitons considérer plusieurs injections, il nous faut évaluer le programme dans autant d'environnements, puisqu'à chaque injection correspond un certain environnement.

Nous avons ensuite présenté la définition de la sémantique avec injection dans un t-environnement sur-instrumenté. Cette sémantique évalue un programme dans un t-environnement sur-instrumenté pour une injection donnée. Ce t-environnement sur-instrumenté regroupe tous les environnements correspondant aux différentes injections. Si on souhaite considérer plusieurs injections, il faut toujours effectuer plusieurs évaluations du programme, mais toutes ces évaluations se font alors dans le même environnement. En effet, cette sémantique, bien qu'elle manipule des t-valeurs sur-instrumentées dans l'environnement d'évaluation, produit uniquement des valeurs simples au sens de la sémantique opérationnelle, tout comme la sémantique opérationnelle avec injection. La définition de cette sémantique est un premier pas permettant de faire le lien entre la notion d'injection et l'analyse dynamique.

Dans cette section, nous allons définir la sémantique sur-instrumentée qui constitue un pas supplémentaire entre la notion d'injection et l'analyse dynamique. La sémantique sur-instrumentée évalue un programme dans un t-environnement sur-instrumenté pour fournir une t-valeur sur-instrumentée. Cette fois-ci, nous avons la possibilité d'analyser un programme pour toutes les injections simultanément. En une seule évaluation, nous obtenons une t-valeur sur-instrumentée contenant toutes les informations nécessaires pour déduire

le comportement du programme pour toute injection. On se rapproche ainsi de notre analyse dynamique. Cependant, la sémantique sur-instrumentée n'est pas calculable. C'est la contrepartie nécessaire pour avoir une sémantique donnant le comportement exact du programme pour toute injection. Lorsque nous définirons l'analyse dynamique, nous obtiendrons une sémantique calculable fournissant une approximation du comportement du programme pour toute injection.

3.4.1 Règles d'inférence

La sémantique sur-instrumentée est représentée formellement par un jugement d'évaluation de la forme suivante :

$$t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi}$$

où e est l'expression évaluée, $t\Gamma^{oi}$ le t-environnement sur-instrumenté dans lequel on effectue l'évaluation et tu^{oi} la t-valeur sur-instrumentée résultat de l'évaluation.

Les règles d'inférence (figure 3.5) sont en grande partie similaires aux règles habituelles de la sémantique opérationnelle. Cependant, trois d'entre elles sont notablement différentes : l'application de fonction (récursive ou non), la structure conditionnelle et le filtrage par motif.

Nous allons détailler ci-dessous chacune des règles pour en donner les explications indispensables à leur compréhension.

OI-NUM Le résultat de l'évaluation d'une constante entière est une t-valeur sur-instrumentée comprenant un ensemble de t-dépendances vide, un ensemble de v-dépendances vide et une valeur simple sur-instrumentée étant la constante elle-même. L'ensemble des t-dépendances est vide car aucune injection ne peut avoir d'impact sur la terminaison de l'évaluation de cette expression. L'ensemble des v-dépendances est vide car aucune injection ne peut avoir d'impact sur la valeur de cette expression.

OI-IDENT L'évaluation d'un identificateur se fait de manière habituelle, en allant chercher la valeur correspondante dans l'environnement. Les dépendances de la t-valeur sur-instrumentée retournée sont celles qui ont été enregistrées dans l'environnement pour cet

$$\textsc{oi-num}\ \frac{}{t\Gamma^{oi} \vdash^{oi} n \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ n\ ]\ ]} \qquad \textsc{oi-abstr}\ \frac{}{t\Gamma^{oi} \vdash^{oi} \lambda x.e \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.e, \uparrow^{oi}_{toi}(t\Gamma^{oi}) >\ ]\ ]}$$

$$\textsc{oi-ident}\ \frac{tu^{oi} = t\Gamma^{oi}[x]}{t\Gamma^{oi} \vdash^{oi} x \mapsto tu^{oi}} \qquad \textsc{oi-abstr-rec}\ \frac{}{t\Gamma^{oi} \vdash^{oi} \texttt{rec}f.x.e \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \texttt{rec}f.x.e, \uparrow^{oi}_{toi}(t\Gamma^{oi}) >\ ]\ ]}$$

$$\textsc{oi-apply}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \qquad tu_2^{oi} = [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (x, tu_2^{oi}) \oplus \uparrow^{toi}_{oi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \qquad deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]}$$

$$\textsc{oi-rec-apply}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto tu_1^{oi} \qquad tu_1^{oi} = [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \texttt{rec}f.x.e, \Gamma_1^{oi} >\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \qquad tu_2^{oi} = [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (f, tu_1^{oi}) \oplus (x, tu_2^{oi}) \oplus \uparrow^{toi}_{oi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \qquad deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]}$$

$$\textsc{oi-letin}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto tu_1^{oi} \qquad tu_1^{oi} = [\ td_1^{oi}\ \mid\ u_1^{oi}\ ] \\ (x, tu_1^{oi}) \oplus t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \end{array}}{t\Gamma^{oi} \vdash^{oi} let\ x\ =\ e_1\ in\ e_2 \mapsto [\ td_1^{oi} \cup td_2^{oi}\ \mid\ u_2^{oi}\ ]}$$

$$\textsc{oi-if-true}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ true\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ] \qquad deps\_spec\_if(t\Gamma^{oi}, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} if\ e\ then\ e_1\ else\ e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_1^{oi}\ \mid\ [\ d'^{oi} \cup d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ]}$$

$$\textsc{oi-if-false}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ false\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ] \qquad deps\_spec\_if(t\Gamma^{oi}, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} if\ e\ then\ e_1\ else\ e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_2^{oi}\ \mid\ [\ d'^{oi} \cup d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ]}$$

$$\textsc{oi-match}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \qquad tu^{oi} = [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ tu^{oi}, p \vdash^{oi}_{p} t\Gamma^{oi}_{p} \\ t\Gamma^{oi}_{p} \oplus t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ] \qquad deps\_spec\_match(t\Gamma^{oi}, p, x, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_1^{oi}\ \mid\ [\ d'^{oi} \cup d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ]}$$

$$\textsc{oi-match-var}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \qquad tu^{oi} = [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ tu^{oi}, p \vdash^{oi}_{p} \bot \\ (x, tu^{oi}) \oplus t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ] \qquad deps\_spec\_match(t\Gamma^{oi}, p, x, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_2^{oi}\ \mid\ [\ d'^{oi} \cup d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ]}$$

$$\textsc{oi-constr-0}\ \frac{}{t\Gamma^{oi} \vdash^{oi} C \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ C\ ]\ ]} \qquad \textsc{oi-constr-1}\ \frac{t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ u^{oi}\ ]}{t\Gamma^{oi} \vdash^{oi} D(e) \mapsto [\ td^{oi}\ \mid\ [\ \emptyset\ \mid\ D(u^{oi})\ ]\ ]}$$

$$\textsc{oi-couple}\ \frac{t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ u_1^{oi}\ ] \qquad t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]}{t\Gamma^{oi} \vdash^{oi} (e_1, e_2) \mapsto [\ td_1^{oi} \cup td_2^{oi}\ \mid\ [\ \emptyset\ \mid\ (u_1^{oi}, u_2^{oi})\ ]\ ]}$$

$$\textsc{oi-annot}\ \frac{t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ]}{t\Gamma^{oi} \vdash^{oi} l\ :\ e \mapsto [\ td^{oi}\ \mid\ [\ (l, fun\ x \Rightarrow x); d^{oi}\ \mid\ v^{oi}\ ]\ ]}$$

FIGURE 3.5 – Sémantique sur-instrumentée

$$\textsc{oim-constr-0}\ \frac{}{[\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ C\ ]\ ], C \vdash_p^{oi} \{\}} \qquad \textsc{oim-constr-1}\ \frac{}{[\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ D(u^{oi})\ ]\ ], D(x) \vdash_p^{oi} \{(x,\ [\ \emptyset\ \mid\ u^{oi}\ ])\}}$$

$$\textsc{oim-couple}\ \frac{}{(\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ u_1^{oi}\ ]\ ],\ [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ u_2^{oi}\ ]\ ]), (x_1, x_2) \vdash_p^{oi} \{(x_1,\ [\ \emptyset\ \mid\ u_1^{oi}\ ]); (x_2,\ [\ \emptyset\ \mid\ u_2^{oi}\ ])\}}$$

$$\textsc{oim-constr-0-not}\ \frac{p \neq C}{[\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ C\ ]\ ], p \vdash_p^{oi} \bot} \qquad \textsc{oim-constr-1-not}\ \frac{p \neq D'(\_)}{[\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ D(u^{oi})\ ]\ ], p \vdash_p^{oi} \bot}$$

$$\textsc{oim-couple-not}\ \frac{p \neq (\_, \_)}{(\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ u_1^{oi}\ ]\ ],\ [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ u_2^{oi}\ ]\ ]), p \vdash_p^{oi} \bot}$$

FIGURE 3.6 – Sémantique sur-instrumentée : règles de filtrage

$$\uparrow_{toi}^{oi}(\ [\ td^{oi}\ \mid\ u^{oi}\ ]) = u^{oi}$$

$$\uparrow_{toi}^{oi}(\{\}) = \{\} \qquad \uparrow_{toi}^{oi}((x, tu^{oi}) \oplus t\Gamma^{oi}) = (x, \uparrow_{toi}^{oi}(tu^{oi})) \oplus \uparrow_{toi}^{oi}(t\Gamma^{oi})$$

FIGURE 3.7 – Valeurs sur-instrumentées : suppression des t-dépendances

identificateur.

OI-ABSTR Dans la t-valeur sur-instrumentée d'une abstraction, l'ensemble des t-dépendances est vide puisque l'évaluation d'une abstraction termine toujours (on ne considère que des injections pour lesquelles l'environnement est instanciable). Pour la même raison, les ensembles de t-dépendances présentes dans l'environnement sont ignorées. On les supprime à l'aide de la fonction $\uparrow_{toi}^{oi}(\bullet)$ définie en figure 3.7. L'ensemble des v-dépendances est vide lui-aussi car l'évaluation d'une abstraction retourne toujours une fermeture contenant le même corps de fonction, seul l'environnement encapsulé peut différer d'une injection à l'autre. L'impact d'une injection sur l'environnement encapsulé dans la fermeture est déjà encodé à l'aide des v-dépendances des v-valeurs sur-instrumentées de l'environnement.

Attention, le fait qu'aucune dépendance n'est introduite par la règle d'évaluation de l'abstraction ne signifie pas qu'une fermeture ne contient jamais de dépendances. D'une part, des dépendances peuvent être ajoutées à la fermeture lorsqu'elle est le résultat de l'évaluation d'une expression conditionnelle par exemple (cf. règle OI-IF-TRUE). D'autre

$$\uparrow^{toi}_{oi}(u^{oi}) = [\ \emptyset\ \mid\ u^{oi}\ ]$$

$$\uparrow^{toi}_{oi}(\{\}) = \{\} \qquad \uparrow^{toi}_{oi}((x, u^{oi}) \oplus \Gamma^{oi}) = (x, \uparrow^{toi}_{oi}(u^{oi})) \oplus \uparrow^{toi}_{oi}(\Gamma^{oi})$$

FIGURE 3.8 – Valeurs sur-instrumentées : ajout de t-dépendances

part, il y a des v-dépendances présentes dans l'environnement encapsulé dans la fermeture. Celles-ci seront prises en compte lors de l'application de la fonction. Il est également intéressant de noter que le corps de la fonction peut contenir des points d'injection qui généreront probablement des dépendances lors de l'application de la fonction.

OI-ABSTR-REC L'évaluation d'une abstraction récursive suit exactement le même modèle que l'évaluation d'une abstraction non-récursive.

OI-APPLY La règle de l'application suit le schéma habituel de l'évaluation d'une application dans la sémantique opérationnelle. Elle évalue e_1 et impose que sa valeur soit une fermeture, puis elle évalue e_2 avant d'évaluer le corps de la fermeture dans son environnement en ajoutant à l'environnement une liaison pour la valeur de l'argument. Pour évaluer le corps de la fermeture, on doit transformer le v-environnement encapsulé en un t-environnement. On utilise alors la fonction $\uparrow^{toi}_{oi}(\bullet)$ (définie en figure 3.8) qui ajoute des t-dépendances vides. Cette approche est valide puisque les t-dépendances des identificateurs utilisés ont déjà été pris en compte.

A ce schéma habituel vient s'ajouter le calcul des dépendances qui nécessite une spécification complexe. Compte tenu de la complexité de cette spécification, des explications détaillées sont données dans la section 3.4.1.1.

OI-REC-APPLY La règle de l'application d'une fonction récursive est similaire à la règle de l'application d'une fonction non-récursive. Elle suit le schéma d'évaluation habituel en y greffant une spécification complexe des dépendances. Une explication détaillée de cette règle d'inférence est donnée en section 3.4.1.1.

OI-LETIN L'évaluation d'une liaison (`let in`) se fait de la même manière que pour la sémantique opérationnelle. On commence par évaluer la sous-expression e_1 pour obtenir sa valeur, puis on évalue la sous-expression e_2 en ajoutant à l'environnement une liaison pour la valeur de e_1. La t-valeur sur-instrumentée de l'expression évaluée est alors constituée d'un ensemble de t-dépendances et de la v-valeur sur-instrumentée de e_2. L'ensemble des t-dépendances est la concaténation des t-dépendances des deux sous-expressions car si une des deux n'a pas de valeur, alors l'expression globale n'a pas de valeur non plus.

OI-IF-TRUE et OI-IF-FALSE Pour l'évaluation d'une expression conditionnelle, on évalue tout d'abord la condition puis l'une des deux branches en fonction de la valeur (de référence) de la condition. On retrouve alors dans le résultat les t-dépendances de chacune des deux sous-expressions évaluées. On ajoute au résultat un ensemble de t-dépendance td'^{oi} et un ensemble de v-dépendance d'^{oi} qui permettent de spécifier le comportement du programme pour les injections ayant un t-impact ou un v-impact sur l'évaluation de la condition. En effet, si un label a un impact sur l'évaluation de la condition, alors l'évaluation de l'expression conditionnelle ne passe pas forcément par la même branche. Pour de plus amples détails sur cette règle d'inférence, se référer à la section 3.4.1.1.

OI-MATCH et OI-MATCH-VAR Dans la règle d'évaluation d'un filtrage par motif, on retrouve le même principe que dans la règle d'évaluation d'une expression conditionnelle. Une prémisse supplémentaire permet de déterminer la branche évaluée à l'aide d'un prédicat de filtrage prenant en paramètre la valeur (de référence) de l'expression filtrée et le motif de filtrage. On retrouve également les ensembles de dépendance td'^{oi} et d'^{oi} permettant de spécifier le comportement du programme pour les injections ayant un impact sur l'évaluation de l'expression filtrée. De plus amples explications sur cette règle d'inférence sont données dans la section 3.4.1.1.

OI-CONSTR-0 L'évaluation d'un constructeur sans paramètre suit la même logique que l'évalution d'une constante numérique. Aucune t-dépendance, ni aucune v-dépendance n'est introduite.

OI-CONSTR-1 Pour l'évaluation d'un constructeur paramétré, on évalue la sous-expression en paramètre puis on applique le constructeur de donnée à sa v-valeur sur-instrumentée. L'ensemble des t-dépendances est exactement le même que pour la sous-expression, puisque l'évaluation d'une telle expression termine sur une valeur si et seulement si l'évaluation de la sous-expression termine sur une valeur. En ce qui concerne l'ensemble des v-dépendances, il est vide puisqu'aucune injection ne peut modifier le constructeur de tête de la valeur de l'expression. L'impact des injections sur les sous-termes de la valeur est déjà encodé dans la v-valeur sur-instrumentée de la sous-expression.

OI-COUPLE Pour l'évaluation d'un couple, on évalue les sous-expressions pour obtenir les deux sous-termes de la valeur sur-instrumentée finale que l'on réunit à l'aide du constructeur de couple. On concatène les t-dépendances des deux sous-expressions pour obtenir les t-dépendances de la t-valeur sur-instrumentée finale, puisque l'évaluation du couple termine sur une valeur si et seulement si l'évaluation des deux sous-expressions termine. L'ensemble des v-dépendances est vide pour la même raison que pour l'évaluation d'un constructeur paramétré : les v-dépendances correspondant à un impact sur un sous-terme de la valeur sont déjà présents dans les sous-termes de la valeur simple sur-instrumentée.

OI-ANNOT Une expression annotée a les mêmes dépendances que sa sous-expression auxquelles on ajoute le label de l'annotation que l'on associe à la fonction identité. Cette fonction indique que si on injecte une valeur sur un label lors de l'évaluation d'une expression annotée par ce même label, alors la valeur de l'expression annotée est directement la valeur injectée.

C'est la seule règle qui permet d'introduire de nouvelles dépendances. Au moment de son introduction, une dépendance est toujours associée à la fonction identité. On peut remarquer également que l'on introduit uniquement des v-dépendances et qu'aucune règle ne permet d'introduire des t-dépendances sans que celles-ci ne soient déjà présentes dans les t-dépendances ou dans les v-dépendances d'une des prémisses. Ce sont les règles manipulant des dépendances indirectes (cf. section 3.4.1.1) qui permettent d'associer aux labels des fonctions autres que l'identité et ce sont elles également qui introduisent des t-dépendances

à partir de v-dépendances des prémisses.

Règles de filtrage Les règles de filtrage sont divisées en deux groupes de trois règles. Le premier regroupe les règles positives (une règle pour chaque forme de motif). Elles définissent un jugement de la forme $tu^{oi}, p \vdash_p^{oi} \Gamma^{oi}$. Ce jugement indique que la t-valeur sur-instrumentée tu^{oi} correspond bien au motif de filtrage p et retourne un v-environnement sur-instrumentée Γ^{oi} contenant les liaisons des identificateurs du motif avec les sous-termes de tu^{oi} correspondants. Le second groupe réunit les règles négatives (une règle pour chaque forme de motif). Ces règles définissent un jugement de la forme $tu^{oi}, p \vdash_p^{oi} \bot$. Ce jugement signifie que la t-valeur sur-instrumentée tu^{oi} ne correspond pas au motif de filtrage p.

3.4.1.1 Explication des règles avec dépendances indirectes

Dans la majorité des règles d'inférence, le calcul des t-dépendances (resp. des v-dépendances) se fait par simple concaténation des t-dépendances (resp. des v-dépendances) présentes dans les prémisses. Les règles de l'évaluation de l'application, de l'expression conditionnelle et du filtrage par motif nécessitent, quant à elles, une spécification plus complexe des dépendances. Il n'est pas possible dans ces cas-là d'obtenir les dépendances du résultat par simple concaténation des dépendances des prémisses, ni par aucun autre procédé calculatoire. On utilise dans alors des fonctions de spécification des dépendances indirectes. Nous allons tout d'abord présenter en détail les règles d'inférence concernées, en expliquant de quelle manière les fonctions de spécification des dépendances indirectes sont utilisées. Ensuite nous reviendrons sur ces fonctions de spécification (cf. section 3.4.1.2) en leur donnant une définition formelle.

OI-APPLY

Rappellons tout d'abord la règle d'inférence que nous allons expliquer :

$$\text{OI-APPLY}\quad \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \\ tu_2^{oi} = [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]}$$

La première prémisse correspond à l'évaluation de la sous-expression e_1. Celle-ci doit obligatoirement avoir pour résultat une fermeture (notée $< \lambda x.e, \Gamma_1^{oi} >$) pour que la règle de l'application non récursive s'applique. À cette fermeture sont associés un ensemble de t-dépendances (noté td_1^{oi}) et un ensemble de v-dépendances (noté d_1^{oi}).

La deuxième prémisse est l'évaluation de la sous-expression e_2 qui retourne une t-valeur sur-instrumentée (notée tu_2^{oi}) composée d'un ensemble de t-dépendances td_2^{oi} et d'une v-valeur sur-instrumentée u_2^{oi}.

La troisième prémisse évalue le corps de la fermeture dans l'environnement de la fermeture, en y ajoutant la liaison de la valeur du paramètre. Les composantes de la t-valeur sur-instrumentée résultant de cette évaluation sont notées td^{oi}, d^{oi} et v^{oi}. On notera que pour évaluer le corps de la fermeture, on ajoute des t-dépendances vides à l'environnement (fonction $\uparrow_{oi}^{toi}()$). Cette pratique est justifiée par le fait que les t-dépendances sont toujours regroupées en tête de la t-valeur sur-instrumentée. Les t-dépendances de l'évaluation de e_1 sont donc toutes présentes dans l'ensemble td_1^{oi} qui fait partie des t-dépendances du résultat final.

Jusque là, nous avons suivi la manière habituelle d'évaluer une application de fonction dans un langage fonctionnel. En ce qui concerne la dernière prémisse, elle est propre à la sémantique sur-instrumentée car elle ne participe pas à l'élaboration de la valeur simple sur-instrumentée du résultat mais uniquement à ses dépendances. Plus précisément, elle spécifie les dépendances indirectes td'^{oi} et d'^{oi} qui vont apparaître dans les dépendances du résultat final. Pour plus de détails concernant la spécification des dépendances indirectes, se reporter à la section 3.4.1.2.

Intéressons-nous maintenant au résultat final : la t-valeur sur-instrumentée de l'appli-

cation de e_1 à e_2. Celle-ci est composée d'un ensemble de t-dépendances, d'un ensemble de v-dépendances et d'une valeur simple sur-instrumentée.

L'ensemble de ses t-dépendances est la concaténation des quatres ensembles de t-dépendances suivants, obtenus dans les prémisses :

- td_1^{oi} et td_2^{oi} car si l'évaluation de e_1 ou celle de e_2 ne termine pas, alors l'évaluation de l'application ne termine pas non plus,
- td^{oi} car si l'évaluation du corps de la fonction ne termine pas, alors l'évaluation de l'application ne termine pas non plus,
- et td'^{oi} qui spécifie les cas de non-terminaison de l'application pour les labels dont dépend la valeur de l'expression e_1.

Notons que le fait d'ajouter l'ensemble td^{oi} en entier constitue une sur-approximation car il est inutile de considérer la non-terminaison de l'évaluation du corps de la fermeture de la valeur de référence de e_1 lorsque le label de l'injection considérée a un impact sur la valeur de e_1. En effet, dans un tel cas, la fonction appliquée n'est pas celle de la valeur de référence mais celle spécifiée par d_1^{oi}. Il est donc possible que la sémantique sur-instrumentée indique qu'une certaine injection provoque une non-terminaison de l'évaluation de l'application en ce basant sur le fait que cette injection provoque une non-terminaison de l'évaluation du corps de la fonction de référence alors que le label en question apparaît dans d_1^{oi} et que l'évaluation de la fonction correspondante termine. Pour que la sémantique sur-instrumentée soit exacte, il ne faudrait ajouter que les labels de td^{oi} n'apparaissant pas dans d_1^{oi}. C'est une amélioration qu'il serait souhaitable d'envisager puisque la sémantique sur-instrumentée a pour objectif de spécifier de façon exacte le comportement du programme pour toute injection sans faire d'approximation. La même remarque pourrait être faite concernant les v-dépendances. On pourrait n'ajouter que les labels de d^{oi} n'apparaissant pas dans d_1^{oi}. Cependant, la sémantique sur-instrumentée ne serait en rien affectée puisque dans les v-dépendances sur-instrumentées, seule la première occurence d'un label est prise en compte (cf. définition de la fonction d'instanciation $aif_{l:v_l}(\bullet)$). Ces améliorations n'auraient cependant aucune influence concernant la précision de l'analyse dynamique ou de l'analyse statique car les ensembles de labels resteraient identiques.

En ce qui concerne l'ensemble des v-dépendances du résultat final, il est constitué de la concaténation des deux ensembles de v-dépendances suivants :

- d'^{oi} qui spécifie la valeur de l'application pour une injection sur un label dont dépend la valeur de l'expression e_1,
- et d^{oi} car si un label n'est pas dans d'^{oi} (i.e. n'est pas dans d_1^{oi}) alors la valeur de l'application pour une injection sur ce label est celle de l'évaluation du corps de la fermeture $< \lambda x.e, \Gamma_1^{oi} >$.

La valeur simple sur-instrumentée du résultat final est la valeur simple sur-instrumentée v^{oi} issue de l'évaluation du corps de la fermeture. En effet, c'est elle qui détermine la valeur de l'application dans le cas d'une injection sur un label n'ayant pas d'impact sur la valeur de e_1 et n'apparaissant pas dans d^{oi}.

OI-REC-APPLY

$$\frac{\begin{array}{c}t\Gamma^{oi} \vdash^{oi} e_1 \mapsto tu_1^{oi} \qquad tu_1^{oi} = [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \texttt{rec} f.x.e, \Gamma_1^{oi} >]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \qquad tu_2^{oi} = [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (f, tu_1^{oi}) \oplus (x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi})\end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]}\ \text{OI-REC-APPLY}$$

La règle de l'application récursive suit le même schéma que la règle de l'application. Les deux premières prémisses correspondent à l'évaluation de e_1. Elles donnent des noms aux différentes composantes de sa t-valeur sur-instrumentée et imposent que sa valeur de référence soit une fermeture récursive.

Les deux prémisses suivantes décrivent l'évaluation de e_2 et donnent des noms aux différentes composantes de sa t-valeur sur-instrumentée.

La prémisse suivante est un jugement d'évaluation pour le corps de la fermeture de référence. Lors de cette évaluation, on ajoute dans l'environnement une liaison pour la fonction récursive et une seconde liaison pour l'argument de la fonction.

On retrouve dans la dernière prémisse la fonction de spécification des dépendances indirectes *deps_spec_apply* utilisée de la même manière que dans la règle de l'application. Elle permet de spécifier les t-dépendances ainsi que les v-dépendances de l'application dans

le cas où la fonction appliquée n'est pas la valeur de référence de e_1, c'est-à-dire dans le cas où le label concerné a un impact sur la valeur de e_1. On peut remarquer que le c'est la même fonction de spécification qui est utilisée dans la règle de l'application et dans celle de l'application récursive. Ceci s'explique par le fait que la valeur de l'application dans le cas d'une injection sur un label ayant un impact sur la valeur de e_1 ne dépend pas de la valeur de référence de e_1. De plus, une injection peut conduire à une fermeture récursive pour e_1 alors que sa valeur de référence était une fermeture non-récursive ou inversement.

Enfin, la t-valeur sur-instrumentée de l'application de e_1 à e_2 se construit de la même manière que dans le cas de l'application non-récursive. Les t-dépendances sont la concaténation des t-dépendances de e_1, des t-dépendances de e_2, de celles de l'évaluation du corps de la fermeture de référence et des t-dépendances indirectes. De même pour la v-valeur sur-instrumentée, il s'agit de la v-valeur sur-instrumentée issue de l'évaluation du corps de la fermeture de référence à laquelle on a ajouté les v-dépendances indirectes spécifiées par la dernière prémisse.

OI-IF-TRUE

OI-IF-TRUE

$$\frac{t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ true\ ]\ ] \qquad t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ] \qquad deps\_spec\_if(t\Gamma^{oi}, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi})}{t\Gamma^{oi} \vdash^{oi} if\ e\ then\ e_1\ else\ e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_1^{oi}\ \mid\ [\ d'^{oi} \cup d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ]}$$

Expliquons maintenant la règle d'inférence de l'évaluation sur-instrumentée d'une expression conditionnelle. Nous expliquons tout d'abord la règle OI-IF-TRUE qui correspond au cas où la valeur de référence de la condition est $true$. Nous verrons ensuite la règle OI-IF-FALSE qui donne la t-valeur sur-instrumentée de l'expression conditionnelle dans le cas où la valeur de référence est $false$. On peut remarquer qu'on ne donne pas de t-valeur sur-instrumentée aux expresssions conditionnelles dont la valeur de référence de la condition n'est ni $true$ ni $false$. En effet, notre analyse dynamique ne permet d'analyser que des programmes qui terminent sur une valeur lorsqu'on les évalue sans injection.

La première prémisse est un jugement d'évaluation pour la condition e. Elle impose que sa valeur de référence soit $true$ pour que cette règle puisse s'appliquer. Les t-dépendances de la condition sont notées td^{oi} et ses v-dépendances sont notées d^{oi}

Puisque la valeur de référence de la condition est $true$, on doit évaluer la première branche e_1 par la sémantique sur-instrumentée pour obtenir sa valeur de référence et ses dépendances. En effet, la valeur de référence de e_1 est également la valeur de référence du résultat. Les v-dépendances de e_1 permettent de spécifier la valeur de l'expression conditionnelle pour toute injection sur un label qui n'a pas d'impact sur la valeur de la condition, puisque dans tous ces cas, on passe dans la première branche.

Dans le cas d'une injection sur un label ayant un impact sur la valeur de la condition (ie. un label présent dans d^{oi}), c'est la dernière prémisse qui fournit les dépendances correspondantes à l'aide de la fonction de spécification des dépendances indirectes $deps\_spec\_if$. L'idée est que l'injection a pu modifier la valeur de la condition et donc qu'on ne passe plus forcément par la première branche.

Le résultat de l'évaluation de l'expression conditionnelle est une t-valeur sur-instrumentée. Celle-ci est constituée de t-dépendances obtenues par l'union des t-dépendances indirectes et des t-dépendances directes (les t-dépendances des deux sous-expressions évaluées e et e_1 et d'une v-valeur sur-instrumentée construite en ajoutant les v-dépendances indirectes à la v-valeur sur-instrumentée de e_1.

L'ordre des t-dépendances n'a pas d'importance puisque si l'une des sous-expressions évaluées n'a pas de valeur, alors l'expression complète n'a pas non plus de valeur. Par exemple, un label peut apparaître à la fois dans td^{oi} et dans td_1^{oi}, associé respectivement à des t-fonctions d'impact tf^{oi} et tf_1^{oi}. Pour une certaine injection v_l, on peut avoir $tf^{oi}(v_l) = true$ et $tf_1^{oi}(v_l) = false$ et pour une autre injection v'_l, on peut avoir $tf^{oi}(v_l) = false$ et $tf_1^{oi}(v_l) = true$. Dans les deux cas, $atif_{l:v_l}(td^{oi} \cup td_1^{oi}) = true$. Par contre, l'ordre des v-dépendances est important car la fonction $aif_{l:v_l}(\bullet)$ ne prend en compte que la première occurence du label. Ainsi, dans notre cas, les v-dépendances indirectes d'^{oi} sont placées avant les v-dépendances directes d_1^{oi} puisque ces dernières n'ont de signification que lorsqu'on est sûr que l'on passe par la première branche, c'est-à-dire dans le cas d'une injection sur un label n'apparaîssant pas dans d^{oi} (donc dans d'^{oi}).

OI-IF-FALSE

OI-IF-FALSE

$$\frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ false\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ] \qquad deps\_spec\_if(t\Gamma^{oi}, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} if\ e\ then\ e_1\ else\ e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_2^{oi}\ \mid\ [\ d'^{oi} \cup d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ]}$$

Cette règle est pratiquement identique à la précédente. La seule différence est que la valeur de référence de la condition e est $false$. C'est donc la sous-expression correspondant à la seconde branche qui est évaluée pour obtenir la valeur de référence de l'expression conditionnelle ainsi que les dépendances directes. Les dépendances indirectes sont quant à elles obtenues de la même manière que dans la règle précédente. On utilise pour cela la fonction de spécification $deps\_spec\_if$ qui est appelée avec les mêmes arguments puisque les dépendances indirectes ne dépendent pas de la valeur de référence.

OI-MATCH

OI-MATCH

$$\frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \qquad tu^{oi} = [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ tu^{oi}, p \vdash_p^{oi} t\Gamma_p^{oi} \\ t\Gamma_p^{oi} \oplus t\Gamma^{oi} \vdash^{oi} e_1 \mapsto [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ] \\ deps\_spec\_match(t\Gamma^{oi}, p, x, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_1^{oi}\ \mid\ [\ d'^{oi} \cup d_1^{oi}\ \mid\ v_1^{oi}\ ]\ ]}$$

L'évaluation des expressions de filtrage se définit également à l'aide de deux règles d'inférence. Nous allons expliquer la règle correspondant à une évaluation de référence passant dans la première branche. Nous verrons ensuite la règle correspondant à une évaluation de référence passant dans la seconde branche.

La première prémisse est un jugement d'évaluation sur-instrumentée pour la sous-expression filtrée e. La t-valeur sur-instrumentée de la sous-expression e est notée tu^{oi}.

La deuxième prémisse permet de nommer les différentes composantes de tu^{oi}. Les t-dépendances sont notées td^{oi}, les v-dépendances d^{oi} et la valeur simple sur-instrumentée est notée v^{oi}.

La troisième prémisse indique que la t-valeur sur-instrumentée de la sous-expression filtrée correspond au motif de filtrage p (cf. figure 3.6). Cette correspondance produit un environnement de liaison $t\Gamma_p^{oi}$ qui associe à chaque identificateur présent dans le motif p

une t-valeur sur-instrumentée issue du sous-terme correspondant dans tu^{oi}. Ce jugement garantit que la valeur de référence extraite de tu^{oi} correspond elle aussi au motif de filtrage p.

La quatrième prémisse correspond à l'évaluation sur-instrumentée de la première branche du filtrage. On évalue cette branche-ci puisque l'évaluation de référence passe dans cette branche. Les composantes de la t-valeur sur-instrumentée de e_1 sont notées td_1^{oi}, d_1^{oi} et v_1^{oi}.

La dernière prémisse est la spécification des dépendances indirectes du filtrage. Elle utilise la fonction de spécification $deps\_spec\_match$ pour spécifier les t-dépendances indirectes td'^{oi} et les v-dépendances indirectes d'^{oi}. Ces dépendances indirectes décrivent le comportement de l'évaluation de l'expression lorsque l'injection considérée est faite sur un label ayant un impact sur la valeur de la sous-expression filtrée. En effet, une modification de la valeur de la sous-expression filtrée provoquée par une injection peut entraîner un changement de branche.

Enfin, le résultat de l'évaluation sur-instrumentée de l'expression de filtrage est la t-valeur sur-instrumentée correspondant à l'évaluation de la branche de référence à laquelle on a ajouté des dépendances. Les t-dépendances ajoutées sont les t-dépendances indirectes et les t-dépendances de la sous-expression filtrée. Les v-dépendances ajoutées sont uniquement les v-dépendances indirectes.

OI-MATCH-VAR

OI-MATCH-VAR

$$\frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \qquad tu^{oi} = [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ tu^{oi}, p \vdash_p^{oi} \bot \\ (x, tu^{oi}) \oplus t\Gamma^{oi} \vdash^{oi} e_2 \mapsto [\ td_2^{oi}\ \mid\ [\ d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ] \\ deps\_spec\_match(t\Gamma^{oi}, p, x, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ td'^{oi} \cup td^{oi} \cup td_2^{oi}\ \mid\ [\ d'^{oi} \cup d_2^{oi}\ \mid\ v_2^{oi}\ ]\ ]}$$

Cette règle ressemble fortement à la règle correspondant à un passage dans la première branche pour l'évaluation de référence. Nous allons souligner uniquement les différences entre ces deux règles d'inférence.

La troisième prémisse nous indique que cette règle n'est appliquée que lorsque la t-valeur sur-instrumentée de la sous-expression filtrée ne correspond pas au motif de filtrage.

La correspondance (ou non-correspondance) d'une t-valeur sur-instrumentée à un motif est équivalente à la correspondance (resp. non-correspondance) de la valeur de référence à ce motif. C'est pourquoi cette règle correspond à l'évaluation sur-instrumentée d'un programme dont l'évaluation de référence passe par la seconde branche.

La quatrième prémisse évalue la seconde branche au lieu de la première puisque c'est la seconde branche qui est évaluée lors de l'évaluation de référence.

La spécification des dépendances indirectes est identique à la règle précédente.

Enfin, le résultat est construit de la même manière, en ajoutant les t-dépendances indirectes, les t-dépendances de la sous-expression filtrée et les v-dépendances indirectes à la t-valeur sur-instrumentée résultat de l'évaluation de la branche empruntée par l'évaluation de référence.

3.4.1.2 Spécification des dépendances indirectes

Dans les règles d'inférence les plus simples, les t-dépendances (resp. les v-dépendances) du résultat sont des dépendances directes, c'est-à-dire qu'elles sont la concaténation des t-dépendances (des v-dépendances) apparaissant dans les prémisses. C'est le cas des règles suivantes : OI-NUM, OI-IDENT, OI-ABSTR, OI-ABSTR-REC, OI-LETIN, OI-CONSTR-0, OI-CONSTR-1, et OI-COUPLE.

La règle OI-ANNOT, quant à elle, introduit une v-dépendance supplémentaire. La v-fonction d'impact associée à cette dépendance est toujours l'identité. C'est cette introduction de dépendance qui correspond à la sémantique de l'injection, telle que définie dans la sémantique opérationnelle avec injection et dans la sémantique avec injection dans un t-environnement sur-instrumenté. Le théorème de correction de la sémantique sur-instrumentée nous fournira une justification formelle de cette correspondance.

En ce qui concerne les autres règles (OI-APPLY, OI-REC-APPLY, OI-IF-TRUE, OI-IF-FALSE, OI-MATCH, OI-MATCH-VAR), on introduit une fonction de spécification des dépendances indirectes. Ces fonctions permettent de spécifier une partie des dépendances du résultat en fonction des dépendances des prémisses.

Dépendances indirectes dans les règles de l'application

$$deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi})$$

La fonction *deps_spec_apply* est utilisée dans les règles de l'application et de l'application récursive. Lorsqu'un label apparaît dans les v-dépendances de e_1, toute injection sur ce label est susceptible de modifier la valeur de la fonction appliquée. La valeur de retour de l'application est donc également susceptible d'être modifiée, c'est pourquoi le label en question doit apparaître dans les v-dépendances du résultat. En ce qui concerne la v-fonction d'impact associée à un tel label, elle va dépendre de la fonction effectivement appliquée (valeur de e_1 pour cette injection) ainsi que du paramètre effectivement passé à la fonction (valeur de e_2 pour cette injection). Il se peut également que la fonction effectivement appliquée ne termine pas, le label en question doit donc apparaître dans les t-dépendances du résultat et la t-fonction d'impact associée doit permettre de déterminer pour chaque injection sur ce label si l'application de la fonction va effectivement terminer ou non. C'est le rôle de la fonction *deps_spec_apply* de spécifier ces dépendances indirectes.

Elle prend en argument la t-valeur sur-instrumentée de e_2 (notée tu_2^{oi}) ainsi que l'ensemble des v-dépendances de e_1 (noté d_1^{oi}). Elle retourne les t-dépendances ainsi que les v-dépendances qu'il faut ajouter au résultat pour prendre en compte ces dépendances indirectes.

Il est nécessaire de préciser que cette fonction n'est pas calculable, ce qui rend la sémantique sur-instrumentée elle-même non-calculable.

Pour spécifier td'^{oi} et d'^{oi}, on commence par définir leur liste de labels comme étant exactement la même que celle de d_1^{oi} (les v-dépendances de e_1). Ensuite, il faut associer à chaque label de cette liste une t-fonction d'impact ou une v-fonction d'impact. La t-fonction d'impact (notée tf') dans td'^{oi} et la v-fonction d'impact (notée f') dans d'^{oi} associées à un certain label sont spécifiées à l'aide de la fonction $deps\_spec\_apply\_fun$ en fonction de la v-fonction d'impact correspondante dans d_1^{oi} et d'autres paramètres explicités ci-dessous.

$$deps\_spec\_apply\_fun(tu_2^{oi}, l, f_1) = (tf', f')$$

On donne en paramètre de cette fonction la t-valeur sur-instrumentée de e_2 (notée

tu_2^{oi}), un label présent dans d_1^{oi} (noté l) et la v-fonction d'impact associée à ce label dans d_1^{oi} (notée f_1). Pour toute valeur v_l, on va spécifier les valeurs de retour de tf' et de f' en fonction de tu_2^{oi} (la t-valeur sur-instrumentée de e_2) et de la valeur $f_1(v_l)$.

On distingue trois cas possibles :

- $f_1(v_l)$ est une fermeture $< \lambda x.e, \Gamma >$

  Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

  $$\Big(\forall (v_2, v'). \downarrow^{l:v_l}(tu_2^{oi}) = v_2 \wedge (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma) \vdash_{l:v_l} e \mapsto v' \\ \Rightarrow tf'(v_l) = false \wedge f'(v_l) = v'\Big) \\ \wedge \\ \Big(\big(\ \nexists (v_2, v'). \downarrow^{l:v_l}(tu_2^{oi}) = v_2 \wedge (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma) \vdash_{l:v_l} e \mapsto v'\big) \\ \Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

  Cette formule signifie que si e_2 a une valeur v_2 pour cette injection (l'instanciation de tu_2^{oi} pour l'injection considérée retourne une valeur v_2) et que l'application de la fermeture à cet argument termine sur une valeur v', alors l'évaluation de l'application de e_1 à e_2 termine ($tf'(v_l) = false$) sur la valeur v' ($f'(v_l) = v'$).
  Dans le cas contraire (si tu_2^{oi} n'est pas instanciable ou si l'application de la fermeture ne termine pas) alors $tf'(v_l) = true$, ce qui signifie que l'évaluation de l'expression $e_1\ e_2$ ne termine pas pour l'injection (l, v_l), et $f'(v_l) = v_{dummy}$, ce qui signifie que la valeur de $f'(v_l)$ n'a aucune importance puisque $tf'(v_l) = true$.
  Remarque : une autre solution serait de définir les v-fonctions d'impact comme des fonctions partielles et de spécifier ici que f' n'a pas de valeur pour l'argument v_l, cependant il n'est pas évident que cette solution soit préférable. C'est une piste qui reste à explorer.

- $f_1(v_l)$ est une fermeture récursive $< \texttt{rec}f.x.e, \Gamma >$

Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

$$\Big(\forall(v_2, v'). \downarrow^{l:v_l}(tu_2^{oi}) = v_2$$
$$\wedge\ (f, \uparrow^{toi}(< \texttt{rec} f.x.e, \Gamma >)) \oplus (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma) \vdash_{l:v_l} e \mapsto v'$$
$$\Rightarrow tf'(v_l) = false \wedge f'(v_l) = v'\Big)$$
$$\wedge$$
$$\Big(\big(\ \nexists(v_2, v'). \downarrow^{l:v_l}(tu_2^{oi}) = v_2$$
$$\wedge\ (f, \uparrow^{toi}(< \texttt{rec} f.x.e, \Gamma >)) \oplus (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma) \vdash_{l:v_l} e \mapsto v'\big)$$
$$\Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

Le principe est le même que dans le cas précédent sauf qu'on applique une fermeture récursive dans le cas présent. On peut remarquer que, comme d'habitude, dans le jugement d'application de la fermeture, on ajoute deux liaisons à l'environnement : une pour la fonction récursive et une pour l'argument.

- $f_1(v_l)$ est autre chose

  On a affaire à une erreur de type donc dans ce cas, la spécification est la suivante : $tf'(v_l) = true \wedge f'(v_l) = v_{dummy}$.

Dépendances indirectes dans les règles de l'expression conditionnelle

$$deps\_spec\_if(t\Gamma^{oi}, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi})$$

La fonction $deps\_spec\_if$ est utilisée dans les deux règles d'évaluation de l'expression conditionnelle. Lorsqu'un label apparaît dans les v-dépendances de la condition e, toute injection sur ce label est susceptible de modifier la valeur de la condition et donc la valeur du résultat de l'évaluation de l'expression conditionnelle. Le label en question doit donc apparaître dans les v-dépendances du résultat. La v-fonction d'impact associée à ce label est alors spécifiée par $deps\_spec\_if$. Une injection sur un tel label peut également provoquer

la non-terminaison de l'évaluation de la condition e ou bien la terminaison sur une valeur provoquant une erreur de type. C'est pourquoi le label doit également apparaître dans les t-dépendances de la t-valeur sur-instrumentée de l'expression conditionnelle. Ces dépendances indirectes (un impact sur l'évaluation de la condition e qui provoque soit un impact sur la valeur de l'expression conditionnelle soit un impact sur la terminaison de l'évaluation de l'expression conditionnelle) sont spécifiées à l'aide de la fonction $deps\_spec\_if$.

Cette fonction de spécification prend quatre arguments : le t-environnement sur-instrumenté $t\Gamma^{oi}$ dans lequel on évalue l'expression conditionnelle, la sous-expression e_1 de la première branche, à évaluer dans le cas où la condition est vraie, la sous-expression e_2 de la seconde branche, à évaluer dans le cas où la condition est fausse et l'ensemble de v-dépendances d^{oi} de l'expression testée. Le résultat de cette fonction est une spécification caractérisant de façon unique les t-dépendances ainsi que les v-dépendances qu'il est nécessaire d'ajouter à la t-valeur sur-instrumentée de l'expression conditionnelle pour prendre en compte les dépendances indirectes.

La fonction de spécification $deps\_spec\_if$ ainsi que ses deux homologues $deps\_spec\_apply$ et $deps\_spec\_match$ donnent uniquement une caractérisation des ensembles de dépendance td'^{oi} et d'^{oi} sous forme de formule logique mais ne fournissent pas un procédé calculatoire permettant de construire les fonctions d'impact inclues dans td'^{oi} et d'^{oi}.

Pour spécifier td'^{oi} et d'^{oi}, on commence par définir leur liste de labels comme étant exactement la même que celle de d^{oi} (les v-dépendances de l'expression testée). Ensuite, il faut associer à chaque label de cette liste une t-fonction d'impact dans td'^{oi} et une v-fonction d'impact dans d'^{oi}. La t-fonction d'impact (notée tf') et la v-fonction d'impact (notée f') associées à un certain label sont spécifiées en fonction de la v-fonction d'impact (notée f) correspondant à ce label dans d^{oi} et d'autres paramètres explicités ci-dessous. Nous notons alors $deps\_spec\_if\_fun$ la fonction qui permet de spécifier les t-fonctions d'impact de td'^{oi} et les v-fonctions d'impact de d'^{oi}.

$$deps\_spec\_if\_fun(t\Gamma^{oi}, e_1, e_2, l, f) = (tf', f')$$

On donne en paramètre de cette fonction le t-environnement sur-instrumenté (noté

$t\Gamma^{oi}$) dans lequel on évalue l'expression, les deux sous-expressions e_1 et e_2 correspondant aux deux branches de l'expression conditionnelle, un label présent dans d^{oi} (noté l) et la v-fonction d'impact associée à ce label dans d^{oi} (notée f). Pour toute valeur v_l, on va spécifier les valeurs de retour de tf' et de f'.

On distingue trois cas possibles :

- $f(v_l)$ vaut $true$

  Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

  $$\Big(\forall v_1.t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \\ \Rightarrow tf'(v_l) = false \wedge f'(v_l) = v_1\Big) \\ \wedge \\ \Big(\big(\nexists v_1.t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1\big) \\ \Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

  Cette spécification fait la distinction entre deux cas : soit il existe une valeur v_1 telle que l'évaluation de e_1 pour l'injection considérée termine sur v_1, et dans ce cas l'évaluation de l'expression conditionnelle termine également pour cette même injection ($tf'(v_l) = false$) et sa valeur est v_1 ($f'(v_l) = v_1$), soit il n'existe pas de jugement d'évaluation de e_1 pour l'injection considérée, et dans ce cas l'évaluation de l'expression conditionnelle ne termine pas non plus sur une valeur ($tf'(v_l) = true$ et $f'(v_l) = v_{dummy}$).

- $f(v_l)$ vaut $false$

  Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

$$\Big(\forall v_2.t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2$$
$$\Rightarrow tf'(v_l) = false \wedge f'(v_l) = v_2\Big)$$
$$\wedge$$
$$\Big(\big(\ \nexists v_2.t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2\big)$$
$$\Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

L'injection considérée entraîne le passage dans la seconde branche de l'expression conditionnelle en donnant $false$ pour valeur à la condition e. La spécification est donc la même que la précédente, sauf que l'on évalue e_2 au lieu de e_1.

- $f(v_l)$ ne vaut ni $true$ ni$false$

  On a affaire à une erreur de type donc dans ce cas, la spécification est la suivante : $tf'(v_l) = true \wedge f'(v_l) = v_{dummy}$.

Dépendances indirectes dans les règles du filtrage par motif

$$deps\_spec\_match(t\Gamma^{oi}, p, x, e_1, e_2, d^{oi}) = (td'^{oi}, d'^{oi})$$

La fonction $deps\_spec\_match$ est utilisée dans les deux règles d'évaluation du filtrage par motif. Lorsqu'un label apparaît dans les v-dépendances de l'expression filtrée e, toute injection sur ce label est susceptible de modifier la branche choisie lors du filtrage et donc la valeur du résultat. Ce label doit donc apparaître dans les v-dépendances de la t-valeur sur-instrumentée résultant de l'évaluation du filtrage. Il doit également apparaître dans ses t-dépendances puisqu'une injection sur ce label peut entraîner la non-terminaison de l'évaluation de l'expression filtrée e ou bien provoquer une erreur de type. La fonction $deps\_spec\_match$ donne une spécification permettant de caractériser de façon unique ces dépendances indirectes à ajouter aux t-dépendances et aux v-dépendances de la t-valeur sur-instrumentée du filtrage.

Il est intéressant de remarquer que si l'injection considérée n'a d'impact que sur un sous-terme de la valeur de l'expression filtrée, alors ces dépendances ne sont pas des dépendances indirectes. Ces dépendances sont prises en compte dans le calcul de la t-valeur

sur-instrumentée de la branche de référence. En effet, une injection sur un tel label ne peut pas avoir d'influence sur le choix de la branche et on évaluera donc la branche de référence. Ceci est dû à notre sémantique du filtrage qui ne discrimine que sur le constructeur de tête de la valeur filtrée. C'est une manière courante de définir le filtrage puisque tout filtrage à une profondeur arbitraire peut être réécrit en une imbrication de filtrages successifs.

La fonction de spécification $deps\_spec\_match$ prend quatre arguments : le t-environnement sur-instrumenté $t\Gamma^{oi}$ dans lequel on évalue l'expression de filtrage, le motif de filtrage de la première branche ainsi que l'identificateur de liaison de la seconde branche, la sous-expression e_1 de la première branche, la sous-expression e_2 de la seconde branche et l'ensemble de v-dépendances d^{oi} de l'expression filtrée. Le résultat de cette fonction est une spécification permettant de caractériser sans ambiguïté les t-dépendances ainsi que les v-dépendances correspondant aux dépendances indirectes de l'expression de filtrage.

Formellement, on ne peut pas exprimer la fonction $deps\_spec\_match$ (ni ses homologues $deps\_spec\_apply$ et $deps\_spec\_if$) sous forme d'une fonction calculable. On définit cependant le graphe correspondant à cette fonction à l'aide d'un prédicat.

Comme pour les deux autres fonctions de spécification, on définit les listes de labels de td'^{oi} et d'^{oi} comme étant exactement les mêmes que celle de d^{oi}. Pour spécifier la t-fonction d'impact (notée tf') dans td'^{oi} correspondant à un de ces labels, ou la v-fonction d'impact (notée f') dans d'^{oi}, on définit la fonction $deps\_spec\_match\_fun$ qui en donne une spécification dépendant entre autre de la v-fonction d'impact (notée f) correspondant à ce label dans d^{oi}.

$$deps\_spec\_match\_fun(t\Gamma^{oi}, p, x, e_1, e_2, l, f) = (tf', f')$$

Les paramètres de cette fonction sont : le t-environnement sur-instrumenté $t\Gamma^{oi}$ dans lequel on évalue l'expression, le motif p correspondant à la première branche, l'identificateur x permettant de lier la valeur filtrée dans la seconde branche, des deux sous-expressions e_1 et e_2 correspondant aux corps des branches, d'un label l présent dans d^{oi} et de la v-fonction d'impact f correspondant à ce label dans d^{oi}. Pour toute valeur v_l, on va alors spécifier les valeurs de retour de tf' et de f'.

On distingue trois cas possibles :

- $f(v_l)$ correspond au motif de filtrage p (on a $f(v_l), p \vdash_p \Gamma_p$)

  Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

  $$\Big(\forall v_1.\uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \\ \Rightarrow tf'(v_l) = false \wedge f'(v_l) = v_1\Big) \\ \wedge \\ \Big(\big(\nexists v_1.\uparrow^{toi}(\Gamma_p) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1\big) \\ \Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

  On tente ici d'évaluer e_1 dans l'environnement $t\Gamma^{oi}$ augmenté de l'environnement de liaison issu du filtrage Γ_p. On distingue alors deux cas : soit l'évaluation retourne une valeur, soit elle n'en retourne pas. Dans le premier cas, l'évaluation du filtrage termine sur une valeur, ce qui est traduit par la formule $tf'(v_l) = false$ et sa valeur est v_1, ce qui est traduit par la formule $f'(v_l) = v_1$.
  On constate ici qu'il est impossible de définir la fonction de spécification sous forme de fonction calculable car elle nécessiterait une fonction calculable permettant de déterminer si l'évaluation de e_1 ***termine ou non. Cette remarque est valable pour les trois fonctions de spécification.***

- $f(v_l)$ ne correspond pas au motif de filtrage p (on a $f(v_l), p \vdash_p \perp$)

  Dans ce cas, les valeurs de retour de la t-fonction d'impact tf' et de la v-fonction d'impact f' sont spécifiées par la formule suivante :

  $$\Big(\forall v_2.(x, \uparrow^{toi}(f(v_l))) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \\ \Rightarrow tf'(v_l) = false \wedge f'(v_l) = v_2\Big) \\ \wedge$$

$$\Big(\big(\ \nexists v_2.(x, \uparrow^{toi}(f(v_l))) \oplus t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2\big)$$
$$\Rightarrow tf'(v_l) = true \wedge f'(v_l) = v_{dummy}\Big)$$

Si la valeur de l'expression filtrée ne correspond pas au motif, on évalue alors la seconde branche du filtrage. Si l'évaluation de cette seconde branche termine sur une valeur v_2, alors on a la spécification suivante : $tf'(v_l) = false \wedge f'(v_l) = v_2$. Si l'évaluation de e_2 ne termine pas sur une valeur, alors la spécification qui s'applique est : $tf'(v_l) = true \wedge f'(v_l) = v_{dummy}$.

- $f(v_l)$ n'est pas une valeur filtrable

  Ce cas correspond à une erreur de type.
  La spécification est donc la suivante : $tf'(v_l) = true \wedge f'(v_l) = v_{dummy}$.

3.4.2 Correction

3.4.2.1 Énoncé informel du théorème

La sémantique sur-instrumentée permet de simuler en une seule évaluation toutes les évaluations avec injection possibles. En effet, le résultat de l'évaluation d'un programme par cette sémantique est une t-valeur sur-instrumentée contenant la valeur de référence du programme ainsi que, en puissance, la valeur du programme pour toute évaluation avec injection.

Pour extraire la valeur du programme pour une injection donnée, il suffit d'instancier la t-valeur sur-instrumentée. Si l'instanciation retourne une valeur, alors nous pouvons être sûrs que l'évaluation de ce programme par la sémantique avec injection termine pour l'injection considérée et que la valeur retournée est la même que l'instanciation de la t-valeur sur-instrumentée.

Le théorème auquel nous nous intéressons exprime le lien qu'il doit y avoir entre la sémantique sur-instrumentée et la sémantique avec injection dans un t-environnement sur-instrumenté. En combinant ce résultat avec le théorème de correction de la sémantique avec injection dans un t-environnement sur-instrumenté et le théorème de correction de

la sémantique opérationnelle avec injection, il sera possible de déduire que si un label n'apparaît pas dans la t-valeur sur-instrumentée d'un programme alors ce label n'a pas d'impact sur l'évaluation du programme. Cependant, cette propriété ne nous intéresse pas en elle-même car il n'est pas possible en pratique d'analyser un programme avec la sémantique sur-instrumentée. On prouvera cette propriété sur la sémantique instrumentée qui constitue notre analyse dynamique.

3.4.2.2 Illustration par l'exemple

Pour se rendre compte concrètement de la signification du théorème de correction de la sémantique sur-instrumentée, nous illustrons ici par quelques exemples le lien entre la sémantique sur-instrumentée et la sémantique avec injection dans un t-environnement sur-instrumenté.

Exemple 1 : évaluation d'un couple

Notons e_1 le programme suivant : (3, 1:7)

Considérons maintenant le jugement d'évaluation de ce programme par la sémantique sur-instrumentée dans le t-environnement sur-instrumenté vide (son arbre de dérivation est donné en figure 3.9) :

$$\emptyset \vdash^{oi} e_1 \mapsto tu^{oi} \text{ avec } tu^{oi} = [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ \emptyset\ \mid\ 3\ ],\ [\ (l, id)\ \mid\ 7\ ])\ ]\ ]$$

où id représente la fonction identité.

Donnons des noms aux différentes composantes de tu^{oi} :

$$tu^{oi} = [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ (\ [\ d_1^{oi}\ \mid\ 3\ ],\ [\ (l, f_2)\ \mid\ 7\ ])\ ]\ ]$$

L'instanciation de tu^{oi} retourne toujours une valeur puisque son ensemble de t-dépendances td^{oi} est vide. L'ensemble de v-dépendances d^{oi} est vide lui aussi, ce qui indique que pour toute injection, l'instanciation de tu^{oi} aura toujours une structure de couple. L'ensemble des v-dépendances de la première composante du couple d_1^{oi} est également vide, donc pour toute injection, la première composante du couple est la valeur 3. En ce

qui concerne l'ensemble des v-dépendances de la seconde composante du couple, il contient un unique label l et sa v-fonction d'impact associée f_2. Si on instancie tu^{oi} pour une injection sur un label autre que l, on obtiendra un couple dont la seconde composante est spécifiée par f_2. Si on instancie tu^{oi} pour une injection sur le label l, alors le couple résultat aura pour seconde composante la valeur 7 qui est la valeur de référence de ce sous-terme. La fonction f_2 est l'identité donc on a :

$$\forall v_l.\downarrow^{l:v_l}(tu^{oi}) = (3, v_l) \text{ et } \forall v_l.\downarrow^{l':v_{l'}}(tu^{oi}) = (3, 7) \text{ si } l \neq l'$$

Maintenant, intéressons nous à l'évaluation de e_1 par la sémantique avec injection dans le même t-environnement sur-instrumenté (l'environnement vide). La sémantique avec injection évalue le programme pour une injection donnée. Considérons une injection que nous noterons $(l', v_{l'})$.

La figure 3.10 présente l'arbre de dérivation du jugement d'évaluation de e_1 par la sémantique avec injection dans le cas où $l = l'$. En réalité, il existe un tel arbre de dérivation pour chaque valeur de $v_{l'}$, cependant tous ces arbres de dérivation ont la même structure. Nous présentons donc un arbre d'évaluation paramétré par la valeur $v_{l'}$.

Dans le cas ou $l \neq l'$, tous les arbres d'évaluation ont eux aussi la même structure. La figure 3.11 présente donc un arbre d'évaluation paramétré par l'injection $(l', v_{l'})$ dans le cas ou $l \neq l'$.

Nous obtenons les deux jugements paramétrés suivants, correspondant aux deux situations possibles :

$$\forall v_{l'}.t\Gamma^{oi} \vdash_{l:v_{l'}} (3, l \ : \ 7) \mapsto (3, v_{l'})$$
$$\text{et } \forall (l', v_{l'}).t\Gamma^{oi} \vdash_{l':v_{l'}} (3, l \ : \ 7) \mapsto (3, 7) \text{ si } l \neq l'$$

Nous pouvons alors constater que pour toute injection possible, l'instanciation de la t-valeur sur-instrumentée de e_1 retourne bien la valeur retournée par l'évaluation de e_1 via la sémantique avec injection. En outre, la sémantique avec injection termine toujours sur une valeur (car la t-valeur sur-instrumentée de e_1 est instanciable pour toute injection).

$$\text{OI-COUPLE}\ \frac{\text{OI-NUM}\ \dfrac{}{t\Gamma^{oi} \vdash^{oi} 3 \mapsto\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ 3\ ]\ ]} \qquad \text{OI-ANNOT}\ \dfrac{\text{OI-NUM}\ \dfrac{}{t\Gamma^{oi} \vdash^{oi} 7 \mapsto\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ 7\ ]\ ]}}{t\Gamma^{oi} \vdash^{oi} l\ :\ 7 \mapsto\ [\ \emptyset\ \mid\ [\ (l, fun\ x \Rightarrow x)\ \mid\ 7\ ]\ ]}}{t\Gamma^{oi} \vdash^{oi} (3, l\ :\ 7) \mapsto\ [\ td_1^{oi} \cup td_2^{oi}\ \mid\ [\ \emptyset\ \mid\ (\ [\ \emptyset\ \mid\ 3\ ],\ [\ (l, fun\ x \Rightarrow x)\ \mid\ 7\ ])\ ]\ ]}$$

FIGURE 3.9 – Exemple 1 : arbre de dérivation du jugement d'évaluation sur-instrumentée

$$\text{INJ-COUPLE}\ \frac{\text{INJ-NUM}\ \dfrac{}{t\Gamma^{oi} \vdash_{l:v_{l'}} 3 \mapsto 3} \qquad \text{INJ-ANNOT-SAME}\ \dfrac{}{t\Gamma^{oi} \vdash_{l:v_{l'}} l\ :\ 7 \mapsto v_{l'}}}{t\Gamma^{oi} \vdash_{l:v_{l'}} (3, l\ :\ 7) \mapsto (3, v_{l'})}$$

FIGURE 3.10 – Exemple 1 : arbre de dérivation du jugement d'évaluation avec injection si $l = l'$

$$\text{INJ-COUPLE}\ \frac{\text{INJ-NUM}\ \dfrac{}{t\Gamma^{oi} \vdash_{l':v_{l'}} 3 \mapsto 3} \qquad \text{INJ-ANNOT-OTHER}\ \dfrac{\text{INJ-NUM}\ \dfrac{}{t\Gamma^{oi} \vdash_{l':v_{l'}} 7 \mapsto 7} \qquad l' \neq l}{t\Gamma^{oi} \vdash_{l':v_{l'}} l\ :\ 7 \mapsto 7}}{t\Gamma^{oi} \vdash_{l':v_{l'}} (3, l\ :\ 7) \mapsto (3, 7)}$$

FIGURE 3.11 – Exemple 1 : arbre de dérivation du jugement d'évaluation avec injection si $l \neq l'$

Exemple 2 : liaison d'une fonction

Notons e_2 le programme suivant :

```
let f o =
  match o with
  | Some(x) → x + l1 : 8
  | none → d
in
(f (Some 18), l2 : 17)
```

Remarque : Some est écrit avec une majuscule puisqu'il s'agit d'une constructeur de donnée alors que none est écrit sans majuscule puisqu'il s'agit d'une variable (cf. algèbre des valeurs sur-instrumentées).

Nous souhaitons évaluer ce programme dans l'environnement suivant :

$$t\Gamma_2^{oi} = (d,\ [\ td_d^{oi}\ \mid\ [\ d_d^{oi}\ \mid\ 26\ ]\ ])$$
$$\text{avec } td_d^{oi} = (l_3, tf_d) \text{ et } td_d^{oi} = (l_4, f_d)$$

Le jugement d'évaluation de e_2 par la sémantique sur-instrumentée est :

$$t\Gamma_2^{oi} \vdash^{oi} e_2 \mapsto\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ (l_1, f_1)\ \mid\ 26\ ],\ [\ (l_2, id)\ \mid\ 17\ ])\ ]\ ]$$
$$\text{avec } f_1 = fun\ x \Rightarrow 18 + x$$

Pour s'en convaincre, l'arbre d'évaluation de ce programme par la sémantique sur-instrumentée est présenté dans les figures 3.12 et 3.13. Pour faciliter la lecture de l'arbre d'évaluation, on écrit à gauche de chaque règle le nom de la règle d'inférence appliquée et à droite le niveau de profondeur de la règle au sein de l'arbre d'évaluation du programme complet e_2. Le niveau 0 correspond à la racine de l'arbre, le niveau 1 correspond à ses prémisses et ainsi de suite. Le sous-arbre d'évaluation du filtrage par motif est présenté dans une figure séparée pour plus de lisibilité.

Notre exemple contient le symbole d'addition. Pour donner une sémantique à ce symbole, il serait nécessaire d'ajouter une règle d'inférence. Nous ne détaillons pas l'ajout de cette règle ici et cachons le sous-arbre d'évaluation correspondant. Il est possible de manipuler les nombres entiers dans notre langage sans avoir recours à l'ajout de règle supplémentaire. Pour cela, il suffit d'encoder les entiers à l'aide des constructeurs de données

et de définir les fonctions usuelles (addition, soustraction, division, multiplication, ...) directemement dans le langage. Cependant, nous avons choisi de ne pas utiliser cet encodage dans l'exemple afin de simplifier la présentation.

Nous pouvons constater que les labels l_3 et l_4 n'apparaissent pas dans la t-valeur sur-instrumentée de e_2. En effet, puisque l'évaluation du filtrage passe toujours par la première branche la valeur de l'identificateur d dans l'environnement n'a pas d'importance. L'analyse donne une t-valeur sur-instrumentée à la sous-expression filtrée. L'ensemble des v-dépendances de cette t-valeur sur-instrumentée est vide donc il n'y a aucune dépendance indirecte lors de l'évaluation du filtrage (aucune injection susceptible de modifier le choix de la branche lors de l'évaluation). L'analyse « déduit » donc de cette manière que l'évaluation du filtrage passe toujours pas la première branche.

Notons tu_2^{oi} la t-valeur sur-instrumentée de e_2 dans $t\Gamma_2^{oi}$. L'instanciation de tu_2^{oi} ne dépend que des labels l_1 et l_2. Si on instancie pour une injection sur l_1, seule la première composante du couple est modifiée. Si on instancie pour une injection sur l_2, seule la seconde composante du couple est modifiée. Si on instancie pour une injection sur un autre label, on obtient la valeur de référence. Ces trois cas sont récapitulés ci-dessous :

$$\begin{array}{rcll} \downarrow^{l_1:v_{l_1}}(tu_2^{oi}) & = & (18+v_{l_1},17) & \\ \downarrow^{l_2:v_{l_2}}(tu_2^{oi}) & = & (26,v_{l_2}) & \\ \downarrow^{l:v_l}(tu_2^{oi}) & = & (26,17) & \text{si } l \neq l_1 \text{ et } l \neq l_2 \end{array}$$

Intéressons-nous maintenant à l'évaluation de l'expression e_2 à l'aide de la sémantique avec injection. En fonction du label sur lequel on effectue l'injection, l'arbre de dérivation du jugement de la sémantique avec injection change de forme. On distingue 3 cas, suivant qu'il s'agisse d'une injection sur le label l_1, sur le label l_2 ou sur un autre label.

Pour une injection sur le label l_1, l'arbre de dérivation est donné en deux parties, dans les figures 3.14 et 3.15.

Pour une injection sur le label l_2, l'arbre de dérivation est donné dans les figures 3.16 et 3.17.

Pour une injection sur un label autre que l_1 et l_2, l'arbre de dérivation est donné en deux parties, dans les figures 3.18 et 3.19.

On obtient les trois jugements suivants qui correspondent aux troix cas de l'instancia-

tion de la t-valeur sur-instrumentée du programme :

$$
\begin{array}{ll}
t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} e_2 \mapsto (18 + v_{l_1}, 17) & \\
t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} e_2 \mapsto (26, v_{l_2}) & \\
t\Gamma_2^{oi} \vdash_{l\ :v_l} e_2 \mapsto (26, 17) & \text{si } l \neq l_1 \text{ et } l \neq l_2
\end{array}
$$

On constate ainsi que pour tout injection, la sémantique avec injection termine sur une valeur qui est la même que l'instanciation de la t-valeur sur-instrumentée.

Pour chaque arbre de dérivation, le niveau 0 correspond à l'évaluation de l'expression de liaison. On a ensuite deux sous-arbres au niveau 1 : un pour l'évaluation de la sous-expression liée et un autre pour l'évaluation du couple. La forme du premier sous-arbre est commune à tous les cas (l_1, l_2 ou autre). Par contre, la forme du second sous-arbre dépend du label sur lequel on effectue l'injection. Il s'agit de l'évaluation de la sous-expression couple. Cette évaluation comprend elle-même deux sous-arbres (de niveau 2), un pour chaque composante du couple.

On peut constater que le sous-arbre de niveau 2 correspondant à l'évaluation de la première composante du couple est le même pour toute injection sur un label différent de l_1 (c'est-à-dire pour tout label n'apparaissant pas dans l'ensemble des v-dépendances sur-instrumentées de la première composante du couple). De même, le sous-arbre correspondant à la seconde composante du couple est le même pour toute injection sur un label différent de l_2 (c'est-à-dire pour tout label n'apparaissant pas dans l'ensemble des v-dépendances sur-instrumentées de la seconde composante du couple).

$$\textsc{oi-letin}\ 0\ \dfrac{\textsc{oi-couple}\ 1\ \dfrac{\textsc{oi-annot}\ 2\ \dfrac{\textsc{oi-num}\ 3\ \dfrac{}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} 17 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ 17\ ]\ ]}}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} l_2\ :\ 17 \mapsto [\ \emptyset\ \mid\ [\ (l_2, fun\ x \Rightarrow x)\ \mid\ 17\ ]\ ]} \quad \textsc{oi-apply}\ 2\ \dfrac{\textsc{oi-match}\ 3\ \dfrac{\begin{array}{c}\textsc{oi-constr-1}\ 3\ \dfrac{\textsc{oi-num}\ 4\ \dfrac{}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} 18 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ 18\ ]\ ]}}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} Some\ 18 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ Some(\ [\ \emptyset\ \mid\ 18\ ])\ ]\ ]} \\ \vdots\ \text{(figure 3.13)}\ \vdots \\ deps\_spec\_apply(\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ Some(\ [\ \emptyset\ \mid\ 18\ ])\ ]\ ], \emptyset) = (\emptyset, \emptyset)\end{array}}{(o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})) \vdash^{oi} match\ o\ with\ Some\ x \to x + l_1\ :\ 8 \mid none \to d \mapsto [\ \emptyset\ \mid\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ]\ ]}}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} f(Some\ 18) \mapsto [\ \emptyset\ \mid\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ]\ ]}}{(f,...)\oplus t\Gamma_2^{oi} \vdash^{oi} (f(Some\ 18), l_2\ :\ 17) \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ],\ [\ (l_2, fun\ x \Rightarrow x)\ \mid\ 17\ ])\ ]\ ]}}{t\Gamma_2^{oi} \vdash^{oi} e_2 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ],\ [\ (l_2, fun\ x \Rightarrow x)\ \mid\ 17\ ])\ ]\ ]}$$

FIGURE 3.12 – Exemple 2 : arbre de dérivation du jugement d'évaluation sur-instrumentée

$$\textsc{oi-match}\ 3\ \dfrac{\textsc{oi-plus}\ 4\ \dfrac{\begin{array}{c}\textsc{oim-constr-1}\ 4\ \dfrac{\textsc{oi-ident}\ 4\ \dfrac{[\ \emptyset\ \mid\ [\ \emptyset\ \mid\ Some(\ [\ \emptyset\ \mid\ 18\ ])\ ]\ ] = ((o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})))[o]}{(o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})) \vdash^{oi} o \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ Some(\ [\ \emptyset\ \mid\ 18\ ])\ ]\ ]}}{[\ \emptyset\ \mid\ [\ \emptyset\ \mid\ Some(\ [\ \emptyset\ \mid\ 18\ ])\ ]\ ], Some\ x \vdash_p^{oi} (x,\ [\ \emptyset\ \mid\ 18\ ])} \\ \vdots\end{array}}{(x,\ [\ \emptyset\ \mid\ 18\ ]) \oplus (o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})) \vdash^{oi} x + l_1\ :\ 8 \mapsto [\ \emptyset\ \mid\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ]\ ]} \\ deps\_spec\_match((o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})), Some\ x, none, x + l_1\ :\ 8, d, \emptyset) = (\emptyset, \emptyset)}{(o,...)\oplus \uparrow_{oi}^{toi}(\uparrow_{toi}^{oi}(t\Gamma_2^{oi})) \vdash^{oi} match\ o\ with\ Some\ x \to x + l_1\ :\ 8 \mid none \to d \mapsto [\ \emptyset\ \mid\ [\ (l_1, fun\ x \Rightarrow 18 + x)\ \mid\ 26\ ]\ ]}$$

FIGURE 3.13 – Exemple 2 : sous-arbre de dérivation sur-instrumentée du filtrage

$$\textsc{inj-letin}\ 0\ \dfrac{\textsc{inj-couple}\ 1\ \dfrac{\textsc{inj-apply}\ 2\ \dfrac{\textsc{inj-match}\ 3\ \dfrac{\textsc{inj-ident}\ 3\ \dfrac{\textsc{inj-abstr}\ 1\ \dfrac{}{t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} \lambda o.match\ o\ with... \mapsto < \lambda o.match\ o\ with..., (d,26) >}}{< \lambda o.match\ o\ with..., (d,26) > =\downarrow^{l_1:v_{l_1}}(((f,...) \oplus t\Gamma_2^{oi})[f])}}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} f \mapsto < \lambda o.match\ o\ with..., (d,26) >}}{\vdots\ \text{(figure 3.15)}\ \vdots}}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d,26) \vdash_{l_1:v_{l_1}} match\ o\ with... \mapsto 18 + v_{l_1}}}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} f(Some\ 18) \mapsto 18 + v_{l_1}}}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} (f(Some\ 18), l_2\ :\ 17) \mapsto (18 + v_{l_1}, 17)}}{t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} e_2 \mapsto (18 + v_{l_1}, 17)}$$

$$\textsc{inj-annot-other}\ 2\ \dfrac{\textsc{inj-num}\ 3\ \dfrac{}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} 17 \mapsto 17}}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} l_2\ :\ 17 \mapsto 17}\ l_1 \neq l_2$$

$$\textsc{inj-constr-1}\ 3\ \dfrac{\textsc{inj-num}\ 4\ \dfrac{}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} 18 \mapsto 18}}{(f,...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} Some\ 18 \mapsto Some\ 18}$$

FIGURE 3.14 – Exemple 2 : arbre de dérivation du jugement d'évaluation avec injection sur l_1

$$\textsc{inj-match}\ 3\ \dfrac{\textsc{inj-ident}\ 4\ \dfrac{Some\ 18 =\downarrow^{l_1:v_{l_1}}(\uparrow^{toi}(Some\ 18))}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d,26) \vdash_{l_1:v_{l_1}} o \mapsto Some\ 18} \qquad \textsc{opm-constr-1}\ 4\ \dfrac{}{Some\ 18, Some\ x \vdash_p (x, 18)}}{\vdots\ \textsc{inj-plus}\ 4\ \dfrac{}{(x, \uparrow^{toi}(18)) \oplus (o, \uparrow^{toi}(v_2)) \oplus (d,26) \vdash_{l_1:v_{l_1}} x + l_1\ :\ 8 \mapsto 18 + v_{l_1}}}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d,26) \vdash_{l_1:v_{l_1}} match\ o\ with\ Some\ x \to x + l_1\ :\ 8 \mid none \to d \mapsto 18 + v_{l_1}}$$

FIGURE 3.15 – Exemple 2 : sous-arbre de dérivation du filtrage avec injection sur l_1

$$\textsc{inj-letin}\ \dfrac{\textsc{inj-couple}\ \dfrac{\textsc{inj-apply}\ \dfrac{\textsc{inj-match}\ \dfrac{\textsc{inj-constr-1}\ \dfrac{\textsc{inj-ident}\ \dfrac{\textsc{inj-abstr}\ \dfrac{}{t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} \lambda o.match\ o\ with... \mapsto < \lambda o.match\ o\ with..., (d, 26) >}\ 1}{< \lambda o.match\ o\ with..., (d, 26) > = \downarrow^{l_2:v_{l_2}}(((f, ...) \oplus t\Gamma_2^{oi})[f])}}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} f \mapsto < \lambda o.match\ o\ with..., (d, 26) >}\ 3 \quad \textsc{inj-num}\ \dfrac{}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} 18 \mapsto 18}\ 4}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} Some\ 18 \mapsto Some\ 18}\ 3 \quad \vdots \text{ (figure 3.17)}}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l_2:v_{l_2}} match\ o\ with... \mapsto 26}\ 3}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} f(Some\ 18) \mapsto 26}\ 2 \quad \textsc{inj-annot-same}\ \dfrac{}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} l_2\ :\ 17 \mapsto v_{l_2}}\ 2}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} (f(Some\ 18), l_2\ :\ 17) \mapsto (26, v_{l_2})}\ 1}{t\Gamma_2^{oi} \vdash_{l_2:v_{l_2}} e_2 \mapsto (26, v_{l_2})}\ 0$$

FIGURE 3.16 – Exemple 2 : arbre de dérivation du jugement d'évaluation avec injection sur l_2

$$\textsc{inj-match}\ \dfrac{\textsc{inj-ident}\ \dfrac{Some\ 18 = \downarrow^{l_2:v_{l_2}}(\uparrow^{toi}(Some\ 18))}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l_2:v_{l_2}} o \mapsto Some\ 18}\ 4 \quad \textsc{opm-constr-1}\ \dfrac{}{Some\ 18, Some\ x \vdash_p (x, 18)}\ 4 \quad \vdots \quad \textsc{inj-plus}\ \dfrac{}{(x, \uparrow^{toi}(18)) \oplus (o, \uparrow^{toi}(v_2)) \oplus (d, 26) \vdash_{l_2:v_{l_2}} x + l_2\ :\ 8 \mapsto 26}\ 4}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l_2:v_{l_2}} match\ o\ with\ Some\ x \to x + l_2\ :\ 8 \mid none \to d \mapsto 26}\ 3$$

FIGURE 3.17 – Exemple 2 : sous-arbre de dérivation du filtrage avec injection sur l_2

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{t\Gamma_2^{oi} \vdash_{l:v_l} \lambda o.match\ o\ with... \mapsto < \lambda o.match\ o\ with..., (d, 26) >}\ 1}{< \lambda o.match\ o\ with..., (d, 26) > = \downarrow^{l:v_l}(((f, ...) \oplus t\Gamma_2^{oi})[f])}\ 3}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} f \mapsto < \lambda o.match\ o\ with..., (d, 26) >}\ 4}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} 18 \mapsto 18}\ 3}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} Some\ 18 \mapsto Some\ 18}\ 3}{\vdots\ \text{(figure 3.19)}\ \vdots}\ 3}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l:v_l} match\ o\ with... \mapsto 26}\ 3}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} f(Some\ 18) \mapsto 26}\ 2 \qquad \dfrac{\dfrac{}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l_1:v_{l_1}} 17 \mapsto 17}\ 3}{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} l\ :\ 17 \mapsto 17}\ l_1 \neq l_2$$

$$\dfrac{(f, ...) \oplus t\Gamma_2^{oi} \vdash_{l:v_l} (f(Some\ 18), l\ :\ 17) \mapsto (26, 17)}{t\Gamma_2^{oi} \vdash_{l:v_l} e_2 \mapsto (26, 17)}\ 0$$

(INJ-ABSTR, INJ-IDENT, INJ-NUM, INJ-CONSTR-1, INJ-MATCH, INJ-APPLY, INJ-ANNOT-OTHER, INJ-NUM, INJ-COUPLE, INJ-LETIN, 5, 4, 3, 2, 1)

FIGURE 3.18 – Exemple 2 : arbre de dérivation du jugement d'évaluation avec injection sur $l \notin \{l_1, l_2\}$

$$\dfrac{\dfrac{\dfrac{Some\ 18 = \downarrow^{l:v_l}(\uparrow^{toi}(Some\ 18))}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l:v_l} o \mapsto Some\ 18}\ 4 \qquad \dfrac{}{Some\ 18, Some\ x \vdash_p (x, 18)}\ 4}{\vdots}}{(o, \uparrow^{toi}(Some\ 18)) \oplus (d, 26) \vdash_{l:v_l} match\ o\ with\ Some\ x \to x + l\ :\ 8 \mid none \to d \mapsto 26}\ 3$$

(INJ-IDENT, OPM-CONSTR-1, INJ-PLUS, INJ-MATCH, 4, 3)

FIGURE 3.19 – Exemple 2 : sous-arbre de dérivation du filtrage avec injection sur $l \notin \{l_1, l_2\}$

3.4.2.3 Énoncé formel du théorème

Théorème 3.4.1 (Correction de la sémantique sur-instrumentée)**.**
$\forall(t\Gamma^{oi}, e, tu^{oi}).t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi}$
$\Rightarrow \forall(l, v_l).\exists\Gamma.\Gamma = \downarrow^{l:v_l}(t\Gamma^{oi})$
$\Rightarrow \forall v.v = \downarrow^{l:v_l}(tu^{oi}) \Rightarrow t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v$

Ce théorème énonce la correction de la sémantique sur-instrumentée. Si l'évaluation sur-instrumentée d'un programme e dans un environnement $t\Gamma^{oi}$ termine sur une valeur sur-instrumentée tu^{oi}, alors cette valeur doit être correcte. Pour toute injection, si le t-environnement sur-instrumenté $t\Gamma^{oi}$ et la t-valeur sur-instrumentée tu^{oi} sont instanciables alors la sémantique avec injection termine sur une valeur et cette valeur est précisément l'instanciation de la t-valeur sur-instrumentée tu^{oi}.

L'hypothèse d'instanciabilité du t-environnement sur-instrumenté assure que l'injection considérée peut conduire à l'évaluation de l'expression e. Autrement dit, si le t-environnement sur-instrumenté n'est pas instanciable, cela signifie que l'injection considérée provoquera une erreur avant même l'évaluation de l'expression e, par exemple lors de l'évaluation d'une autre partie du programme précédant celle de e.

3.4.2.4 Preuve de correction

La preuve de correction se fait par induction sur le jugement d'évaluation de la sémantique sur-instrumentée. Cette induction implique la preuve de quinze cas : un pour chaque règle d'inférence. Nous allons présenter ici les grandes lignes de la preuve en détaillant quelques uns de ces cas. La preuve complète est disponible sous forme de code Coq.

cas OI-NUM Ce cas est trivial. Dans ce cas, l'expression e n'est autre qu'une constante numérique n, la t-valeur sur-instrumentée tu^{oi} ne contient que des dépendances vides et la valeur simple sur-instrumentée n. L'instanciation de tu^{oi} est donc la valeur n et il suffit d'appliquer la règle d'inférence INJ-NUM de la sémantique avec injection pour conclure.

cas OI-ABSTR À partir d'un jugement de la sémantique sur-instrumentée obtenu par la règle OI-ABSTR, d'une injection (l, v_l) donnée, et de la valeur v obtenue par instanciation de la valeur sur-instrumentée pour l'injection (l, v_l), on veut déduire le jugement correspondant pour la sémantique avec injection.

Voici les trois hypothèses à notre disposition :

$$\frac{}{t\Gamma^{oi} \vdash^{oi} \lambda x.e \mapsto \ [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.e, \uparrow^{oi}_{toi}(t\Gamma^{oi}) >]\ ]}\ \text{OI-ABSTR}$$

$$\exists \Gamma.\Gamma = \downarrow^{l:v_l}(t\Gamma^{oi})$$

$$v = \downarrow^{l:v_l}(\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.e, \uparrow^{oi}_{toi}(t\Gamma^{oi}) >]\ ])$$

Puisque $t\Gamma^{oi}$ est instanciable pour l'injection (l, v_l), alors on a :

$$\downarrow^{l:v_l}(\uparrow^{oi}_{toi}(t\Gamma^{oi})) = \downarrow^{l:v_l}(t\Gamma^{oi})$$

En simplifiant la dernière hypothèse, on obtient :

$$v = < \lambda x.e, \downarrow^{l:v_l}(\uparrow^{oi}_{toi}(t\Gamma^{oi})) > = < \lambda x.e, \downarrow^{l:v_l}(t\Gamma^{oi}) >$$

On peut alors conclure en appliquant la règle d'inférence suivante :

$$\frac{}{t\Gamma^{oi} \vdash_{l:v_l} \lambda x.e \mapsto < \lambda x.e, \downarrow^{l:v_l}(t\Gamma^{oi}) >}\ \text{INJ-ABSTR}$$

cas OI-APPLY À partir d'un jugement de la sémantique sur-instrumentée obtenu par la règle OI-APPLY, d'une injection (l, v_l) donnée, de la valeur v obtenue par instanciation de la valeur sur-instrumentée pour l'injection (l, v_l) et de trois hypothèses d'induction correspondant aux trois jugements de la sémantique sur-instrumentée présents dans les prémisses de la règle OI-APPLY, on veut déduire le jugement correspondant pour la sémantique avec injection.

Voici les six hypothèses à notre disposition :

$$\frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto \ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \qquad tu_2^{oi} = \ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (x, tu_2^{oi}) \oplus \uparrow^{toi}_{oi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto \ [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ] \\ deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto \ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]}\ \text{OI-APPLY}$$

$$\exists \Gamma.\Gamma =\downarrow^{l:v_l}(t\Gamma^{oi}) \qquad (H_{inst}^{t\Gamma^{oi}})$$

$$v =\downarrow^{l:v_l}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]) \qquad (H_{inst}^{tu^{oi}})$$

$$\begin{aligned} &\exists \Gamma.\Gamma =\downarrow^{l:v_l}(t\Gamma^{oi}) \\ &\Rightarrow \forall v.v =\downarrow^{l:v_l}(\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ]) \qquad (IH_1) \\ &\Rightarrow t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v \end{aligned}$$

$$\exists \Gamma.\Gamma =\downarrow^{l:v_l}(t\Gamma^{oi}) \Rightarrow \forall v.v =\downarrow^{l:v_l}(tu_2^{oi}) \Rightarrow t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v \qquad (IH_2)$$

$$\begin{aligned} &\exists \Gamma.\Gamma =\downarrow^{l:v_l}((x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi})) \\ &\Rightarrow \forall v.v =\downarrow^{l:v_l}(\ [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ]) \qquad (IH) \\ &\Rightarrow (x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash_{l:v_l} e \mapsto v \end{aligned}$$

Nous commençons par utiliser les hypothèses d'induction pour obtenir des jugements d'évaluation avec injection pour e_1, pour e_2 et pour le corps de la fonction appliquée. Nous pourrons ensuite appliquer la règle INJ-APPLY pour conclure.

Le corps de la fonction appliquée n'est pas le même selon que le label l sur lequel on fait l'injection appartient à d_1^{oi} ou non. En effet, si $l \notin d_1^{oi}$ alors la valeur de e_1 est bien une fermeture dont le corps est l'expression e. Dans le cas contraire, si $l \in d_1^{oi}$ alors la valeur de e_1 est spécifiée par la v-fonction d'impact associée à l dans d_1^{oi}. Nous allons donc séparer la preuve en deux parties pour prouver ces deux cas séparément.

Traitons tout d'abord le cas où $l \notin d_1^{oi}$.

Nous savons d'une part que l'environnement $t\Gamma^{oi}$ est instanciable, c'est exactement l'hypothèse $H_{inst}^{t\Gamma^{oi}}$. D'autre part, l'hypothèse $H_{inst}^{tu^{oi}}$ nous apprend que la t-valeur sur-instrumentée $(\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ])$ est instanciable. En effet, $H_{inst}^{tu^{oi}}$ implique $atif_{l:v_l}(td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}) = false$ ce qui implique $atif_{l:v_l}(td_1^{oi}) = false$. Nous pouvons donc appliquer l'hypothèse d'induction IH_1 :

$$\downarrow^{l:v_l}(\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ]) =< \lambda x.e, \downarrow^{l:v_l}(\Gamma_1^{oi}) >$$

$$t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto < \lambda x.e, \downarrow^{l:v_l}(\Gamma_1^{oi}) >$$

Pour obtenir un jugement d'évaluation avec injection pour e_2, nous procédons de la même manière que pour e_1. L'instanciabilité de $t\Gamma^{oi}$ est fournie par $H_{inst}^{t\Gamma^{oi}}$. En ce qui concerne l'instanciation de tu_2^{oi}, on déduit $atif_{l:v_l}(td_2^{oi}) = false$ à partir de la propriété $atif_{l:v_l}(td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}) = false$ qui a déjà été montrée. Puisque l'instanciabilité d'une t-valeur sur-instrumentée dépend uniquement de ses t-dépendances sur-instrumentées et de l'injection considérée (une v-valeur sur-instrumentée est toujours instanciable), tu_2^{oi} est instanciable en une certaine valeur v_2.

$$\downarrow^{l:v_l}(\ [\ td_2^{oi} \ \mid \ u_2^{oi} \ ]) = \downarrow^{l:v_l}(u_2^{oi}) = v_2$$

$$t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2$$

Pour obtenir un jugement d'évaluation avec injection pour e, nous procédons une troisième fois de la même manière. Il faut prouver l'instanciabilité de $(x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi})$. La t-valeur sur-instrumentée tu_2^{oi} est instanciable, comme on l'a vu plus haut. L'environnement $\uparrow_{oi}^{toi}(\Gamma_1^{oi})$ est instanciable car Γ_1^{oi} est instanciable et la fonction $\uparrow_{oi}^{toi}(\bullet)$ n'ajoute que des annotations vides. On a donc :

$$\downarrow^{l:v_l}((x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi})) = (x, v_2) \oplus \downarrow^{l:v_l}(\Gamma_1^{oi})$$

Nous prouvons que $[\ td^{oi} \ \mid \ [\ d^{oi} \ \mid \ v^{oi} \ ] \ ]$ s'instancie en v, en réduisant l'hypothèse $H_{inst}^{tu^{oi}}$ à l'aide du fait que $atif_{l:v_l}(td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}) = false$ et de l'hypothèse $l \notin d_1^{oi}$:

$$\begin{aligned} v &= \downarrow^{l:v_l}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi} \ \mid \ [\ d'^{oi} \cup d^{oi} \ \mid \ v^{oi} \ ] \ ]) \\ &= \downarrow^{l:v_l}(\ [\ d'^{oi} \cup d^{oi} \ \mid \ v^{oi} \ ]) = \downarrow^{l:v_l}(\ [\ d^{oi} \ \mid \ v^{oi} \ ]) \\ &= \downarrow^{l:v_l}(\ [\ td^{oi} \ \mid \ [\ d^{oi} \ \mid \ v^{oi} \ ] \ ]) \end{aligned}$$

Cette dernière égalité étant justifiée car nous déduisons $atif_{l:v_l}(td^{oi}) = false$ de la propriété prouvée précédemment $atif_{l:v_l}(td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}) = false$. On obtient ainsi le jugement de la sémantique avec injection en appliquant l'hypothèse d'induction IH :

$$(x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash_{l:v_l} e \mapsto v$$

Avant de pouvoir appliquer la règle d'inférence de l'application, il nous faut appliquer la propriété suivante notée P^{inj}_{inst} (dont la preuve est fournie en Coq). Cette propriété énonce que si, pour un programme e, la sémantique avec injection dans un t-environnement sur-instrumenté $t\Gamma^{oi}$ termine sur une valeur v alors elle termine sur la même valeur v dans l'environnement $\uparrow^{toi}(\downarrow^{l:v_l}(\Gamma^{oi}))$. Cette propriété est très proche du théorème de correction de la sémantique avec injection dans un t-environnement sur-instrumenté.

$$\begin{aligned}\forall(t\Gamma^{oi}, l, v_l, e, v, \Gamma),\ & t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \\ & \Rightarrow \Gamma = \downarrow^{l:v_l}(t\Gamma^{oi}) \\ & \Rightarrow \uparrow^{toi}(\Gamma) \vdash_{l:v_l} e \mapsto v\end{aligned} \qquad (P^{inj}_{inst})$$

On obtient alors :

$$\uparrow^{toi}(\downarrow^{l:v_l}((x, tu_2^{oi}) \oplus \uparrow^{toi}_{oi}(\Gamma_1^{oi}))) \vdash_{l:v_l} e \mapsto v$$

$$(x, \uparrow^{toi}(\downarrow^{l:v_l}(tu_2^{oi}))) \oplus \uparrow^{toi}(\downarrow^{l:v_l}(\uparrow^{toi}_{oi}(\Gamma_1^{oi}))) \vdash_{l:v_l} e \mapsto v$$

$$(x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\downarrow^{l:v_l}(\Gamma_1^{oi})) \vdash_{l:v_l} e \mapsto v$$

Les trois jugements d'évaluation avec injection ainsi obtenus pour e_1, e_2 et e nous servent alors de prémisses pour appliquer la règle de la sémantique avec injection INJ-APPLY qui nous permet de conclure :

$$\text{INJ-APPLY}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto < \lambda x.e, \downarrow^{l:v_l}(\Gamma_1^{oi}) > \\ t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \\ (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\downarrow^{l:v_l}(\Gamma_1^{oi})) \vdash_{l:v_l} e \mapsto v \end{array}}{t\Gamma^{oi} \vdash_{l:v_l} e_1\ e_2 \mapsto v}$$

Supposons maintenant que $l \in d_1^{oi}$.

On prouve ce cas-là par induction sur la structure de la liste d_1^{oi}. On élimine le premier cas, où d_1^{oi} est vide, puisque d_1^{oi} doit contenir au moins l. On suppose donc que d_1^{oi} est de la forme $(l', f_1); d^{oi}_{1,tl}$ pour une certaine v-fonction d'impact f_1 et un certain v-ensemble de dépendances $d^{oi}_{1,tl}$. Par la définition de *deps_spec_apply*, on déduit qu'il existe une t-fonction d'impact tf' et un t-ensemble de dépendances td'^{oi}_{tl} tels que $td'^{oi} = (l', tf'); td'^{oi}_{tl}$

et qu'il existe une v-fonction d'impact f' et un v-ensemble de dépendances d'^{oi}_{tl} tels que $d'^{oi} = (l', f'); d'^{oi}_{tl}$.

On a alors les propriétés suivantes ainsi que l'hypothèse d'induction IH_{d_1} :

$$\begin{gathered} deps\_spec\_apply(tu_2^{oi}, d_{1,tl}^{oi}) = (td'^{oi}_{tl}, d'^{oi}_{tl}) \\ deps\_spec\_apply\_fun(tu_2^{oi}, l', f_1) = (tf', f') \end{gathered}$$

$$\begin{aligned} \forall v.v = &\downarrow^{l:v_l}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}_{tl} \ \mid \ [\ d'^{oi}_{tl} \cup d^{oi} \ \mid \ v^{oi} \ ] \ ]) \\ &\Rightarrow t\Gamma^{oi} \vdash_{l:v_l} e_1 \ e_2 \mapsto v \end{aligned} \qquad (IH_{d_1})$$

On distingue alors deux cas, $l = l'$ ou $l \neq l'$.

Commençons par supposer $l = l'$.

On a forcément la propriété $tf'(v_l) = false$. Si ce n'était pas le cas, la t-valeur sur-instrumentée $[\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi} \ \mid \ [\ d'^{oi} \cup d^{oi} \ \mid \ v^{oi} \ ] \ ]$ ne serait pas instanciable, ce qui contredirait l'hypothèse $H^{tu^{oi}}_{inst}$. Puisque $tf'(v_l) = false$, il y a uniquement deux cas possibles pour la valeur de $f_1(v_l)$ (cf. définition de $deps\_spec\_apply\_fun$, section 3.4.1.2).

- $f_1(v_l)$ est une fermeture $< \lambda x'.e', \Gamma_1 >$

  Dans ce cas, il existe deux valeurs v_2 et v' avec les propriétés suivantes :

  $$\downarrow^{l:v_l}(tu_2^{oi}) = v_2$$
  $$(x', \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e' \mapsto v'$$
  $$f'(v_l) = v'$$

  De plus, on prouve que $v' = v$ en réduisant l'égalité de l'hypothèse $H^{tu^{oi}}_{inst}$.

  En utilisant les hypothèses d'induction IH_1 et IH_2, on obtient les jugements d'évaluation avec injection suivants pour e_1 et e_2 :

  $$t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto < \lambda x'.e', \Gamma_1 >$$
  $$t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2$$

  Il suffit donc d'appliquer la règle d'inférence INJ-APPLY pour conclure :

  $$\frac{\begin{array}{c} t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto < \lambda x'.e', \Gamma_1 > \\ t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \\ (x', \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e' \mapsto v \end{array}}{t\Gamma^{oi} \vdash_{l:v_l} e_1 \ e_2 \mapsto v} \text{INJ-APPLY}$$

- $f_1(v_l)$ est une fermeture récursive $< \texttt{rec} f'.x'.e', \Gamma_1 >$

  Dans ce cas, il existe deux valeurs v_2 et v' avec les propriétés suivantes :

$$\downarrow^{l:v_l}(tu_2^{oi}) = v_2$$
$$(f', \uparrow^{toi}(< \texttt{rec} f'.x'.e', \Gamma_1 >)) \oplus (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e' \mapsto v'$$
$$f'(v_l) = v'$$

  Comme ci-dessus, on utilise l'hypothèse $H_{inst}^{tu^{oi}}$ pour prouver que $v' = v$ et on déduit les jugements d'évaluation avec injection pour e_1 et e_2 à l'aide des hypothèses d'induction IH_1 et IH_2. On applique ensuite la règle INJ-APPLY-REC pour conclure :

$$\frac{\begin{array}{c} t\Gamma^{oi} \vdash_{l:v_l} e_1 \mapsto v_1 \qquad v_1 =< \texttt{rec} f'.x'.e', \Gamma_1 > \\ t\Gamma^{oi} \vdash_{l:v_l} e_2 \mapsto v_2 \\ (f, \uparrow^{toi}(v_1)) \oplus (x, \uparrow^{toi}(v_2)) \oplus \uparrow^{toi}(\Gamma_1) \vdash_{l:v_l} e' \mapsto v \end{array}}{t\Gamma^{oi} \vdash_{l:v_l} e_1\ e_2 \mapsto v} \text{INJ-APPLY-REC}$$

Maintenant supposons $l \neq l'$.

Puisque $l \neq l'$, on a l'égalité suivante :

$$\begin{aligned} v &= \downarrow^{l:v_l}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]) \\ &= \downarrow^{l:v_l}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td^{oi} \cup td'^{oi}_{tl}\ \mid\ [\ d'^{oi}_{tl} \cup d^{oi}\ \mid\ v^{oi}\ ]\ ]) \end{aligned} \qquad (H_{inst}^{tu^{oi}})$$

On peut alors appliquer l'hypothèse d'induction IH_{d_1} qui nous permet de conclure ce dernier cas :

$$t\Gamma^{oi} \vdash_{l:v_l} e_1\ e_2 \mapsto v$$

3.5 Sémantique instrumentée

Nous arrivons maintenant à la présentation de la sémantique instrumentée, qui constitue ce que nous appelons notre analyse dynamique. Le but de cette analyse est de pouvoir évaluer un programme de la même manière qu'avec la sémantique opérationnelle usuelle mais en gardant trace pour chaque sous-terme de la valeur produite des dépendances de ce sous-terme. On veut alors pouvoir garantir que si un label n'apparaît pas parmi les dépendances d'un programme, alors ce label n'a pas d'impact sur l'évaluation de ce programme.

Pour prouver la correction de l'analyse dynamique par rapport à la notion d'impact présentée en section 2.6, nous prouverons tout d'abord une propriété de correction de la sémantique instrumentée vis-à-vis de la sémantique sur-instrumentée puis nous nous reposerons sur la correction des deux sémantiques intermédiaires présentées plus haut : la sémantique avec injection dans un t-environnement sur-instrumenté (cf. section 3.3) et la sémantique sur-instrumentée (cf. section 3.4).

La sémantique instrumentée évalue une expression dans un t-environnement instrumenté pour retourner une t-valeur instrumentée. C'est une simplification de la sémantique sur-instrumentée. En effet, on peut remarquer dans la sémantique sur-instrumentée que le calcul de la présence d'un label dans un ensemble de v-dépendances (resp. de t-dépendances) ne dépend pas des v-fonctions d'impact (resp. des t-fonctions d'impact) qui sont associées. On peut donc calculer les ensembles de labels sans leur associer de fonction d'impact. C'est justement ce que fait la sémantique instrumentée.

On obtient ainsi une **sémantique décidable sur les programmes qui terminent** puisque le caractère indécidable de la sémantique sur-instrumentée venait de la spécification des fonctions d'impact.

La sémantique instrumentée constitue en elle-même l'analyse dynamique de dépendances que nous souhaitions définir, puisque l'analyse qui nous intéresse est justement le calcul des ensembles de labels. Sa définition ne nécessite aucunement une définition préalable de la sémantique sur-instrumentée ou bien de la sémantique avec injection. L'intérêt d'avoir défini la sémantique sur-instrumentée est uniquement de simplifier et de clarifier la preuve de correction de la sémantique instrumentée.

3.5.1 Algèbre des valeurs instrumentées

La sémantique instrumentée manipule des valeurs instrumentées. Ces valeurs peuvent être vues comme des valeurs abstraites par rapport aux valeurs sur-instrumentées. La fonction d'abstraction est tout simplement la fonction qui supprime toutes les fonctions d'impact. Ainsi, chaque valeur sur-instrumentée correspond à une valeur instrumentée unique et plusieurs valeurs sur-instrumentées peuvent correspondre à la même valeur instrumentée.

Cette section présente la composition des valeurs instrumentées.

3.5.1.1 Valeurs

Une t-valeur instrumentée (notée tu^i) est composée d'un ensemble de t-dépendances td^i et d'une v-valeur instrumentée u^i.

$$tu^i := \quad [\ td^i \ \mid \ u^i\ ] \quad \text{Valeur avec annotation de } t\text{-dépendance}$$

Une v-valeur instrumentée (notée u^i) est composée d'un ensemble de v-dépendances d^i et d'une valeur simple instrumentée v^i.

$$u^i := \quad [\ d^i \ \mid \ v^i\ ] \quad \text{Valeur avec annotation de } v\text{-dépendance}$$

Enfin, une valeur simple instrumentée (notée v^i) est une valeur simple dont les sous-termes sont des v-valeurs instrumentées u^i. On note $\mathcal{V}^i$ l'ensemble des valeurs simples instrumentées.

$$\begin{array}{lll} v^i := & n \mid b \mid C \mid D(u^i) \mid (u_1^i, u_2^i) & \text{Constructeurs de données} \\ & < \lambda x.e, \Gamma^i > & \text{Fermeture} \\ & < \texttt{rec}f.x.e, \Gamma^i > & \text{Fermeture récursive} \end{array}$$

3.5.1.2 Ensembles de dépendances

Dans les t-valeurs instrumentées (resp. les v-valeurs instrumentées), les ensembles de t-dépendances (resp. de v-dépendances) sont simplement des ensembles de labels.

$$\begin{array}{rll} td^i := & \{l_1; \ldots; l_n\} & t\text{-dépendances} \\ d^i := & \{l_1; \ldots; l_m\} & v\text{-dépendances} \end{array}$$

3.5.1.3 Environnements

De même que pour la sémantique sur-instrumentée, on distingue 2 types d'environnements.

Un t-environnement instrumenté permet de lier chaque identificateur à une t-valeur instrumentée.

$$t\Gamma^i := \quad (x_1, tu_1^i); \ldots; (x_n, tu_n^i) \quad t\text{-environnement instrumenté}$$

Un v-environnement instrumenté permet de lier chaque identificateur à une v-valeur instrumentée.

$$\Gamma^i := \quad (x_1, u_1^i); \ldots; (x_n, u_n^i) \quad v\text{-environnement instrumenté}$$

3.5.2 Règles d'inférence

Le jugement d'évaluation de la sémantique instrumentée prend la forme suivante :

$$t\Gamma^i \vdash^i e \mapsto tu^i$$

où e est l'expression évaluée, $t\Gamma^i$ le t-environnement instrumenté dans lequel on effectue l'évaluation et tu^i la t-valeur instrumentée résultat de l'évaluation.

Ce jugement est défini par les règles d'inférence présentée en figure 3.20.

Les règles simples de la sémantique sur-instrumentée restent les mêmes dans la sémantique instrumentée, seule la signification de la concaténation de dépendances est différente puisqu'on concatène des listes de labels au lieu de concaténer des listes d'associations. En ce qui concerne les règles contenant des dépendances indirectes, la sémantique instrumentée n'effectue plus le calcul des fonctions d'impact mais se concente de calculer les ensembles de labels.

Des explications détaillées de ces règles sont données ci-dessous.

I-NUM La règle d'évaluation d'une constante entière est identique à celle de la sémantique sur-instrumentée. L'ensemble des t-dépendances instrumentées est vide, de même que l'ensemble des v-dépendances instrumentées et la valeur simple instrumentée est la constante entière.

$$\textsc{i-num}\ \frac{}{t\Gamma^i \vdash^i n \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]} \qquad \textsc{i-ident}\ \frac{tu^i = t\Gamma^i[x]}{t\Gamma^i \vdash^i x \mapsto tu^i} \qquad \textsc{i-letin}\ \frac{t\Gamma^i \vdash^i e_1 \mapsto tu_1^i \qquad tu_1^i = [\ td_1^i\ |\ u_1^i\ ] \qquad (x, tu_1^i) \oplus t\Gamma^i \vdash^i e_2 \mapsto [\ td_2^i\ |\ u_2^i\ ]}{t\Gamma^i \vdash^i let\ x\ =\ e_1\ in\ e_2 \mapsto [\ td_1^i \cup td_2^i\ |\ u_2^i\ ]}$$

$$\textsc{i-abstr}\ \frac{}{t\Gamma^i \vdash^i \lambda x.e \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ <\lambda x.e, \uparrow_{ti}^i(t\Gamma^i)>\ ]\ ]} \qquad \textsc{i-abstr-rec}\ \frac{}{t\Gamma^i \vdash^i \texttt{rec} f.x.e \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ <\texttt{rec} f.x.e, \uparrow_{ti}^i(t\Gamma^i)>\ ]\ ]}$$

$$\textsc{i-apply}\ \frac{t\Gamma^i \vdash^i e_1 \mapsto [\ td_1^i\ |\ [\ d_1^i\ |<\lambda x.e, \Gamma_1^i>\ ]\ ] \qquad t\Gamma^i \vdash^i e_2 \mapsto tu_2^i \qquad tu_2^i = [\ td_2^i\ |\ u_2^i\ ] \qquad (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ]}{t\Gamma^i \vdash^i e_1\ e_2 \mapsto [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ |\ [\ d_1^i \cup d^i\ |\ v^i\ ]\ ]}$$

$$\textsc{i-apply-rec}\ \frac{t\Gamma^i \vdash^i e_1 \mapsto tu_1^i \qquad tu_1^i = [\ td_1^i\ |\ [\ d_1^i\ |<\texttt{rec} f.x.e, \Gamma_1^i>\ ]\ ] \qquad t\Gamma^i \vdash^i e_2 \mapsto tu_2^i \qquad tu_2^i = [\ td_2^i\ |\ u_2^i\ ] \qquad (f, tu_1^i) \oplus (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ]}{t\Gamma^i \vdash^i e_1\ e_2 \mapsto [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ |\ [\ d_1^i \cup d^i\ |\ v^i\ ]\ ]}$$

$$\textsc{i-if-true}\ \frac{t\Gamma^i \vdash^i e \mapsto [\ td^i\ |\ [\ d^i\ |\ true\ ]\ ] \qquad t\Gamma^i \vdash^i e_1 \mapsto [\ td_1^i\ |\ [\ d_1^i\ |\ v_1^i\ ]\ ]}{t\Gamma^i \vdash^i if\ e\ then\ e_1\ else\ e_2 \mapsto [\ d^i \cup td^i \cup td_1^i\ |\ [\ d^i \cup d_1^i\ |\ v_1^i\ ]\ ]}$$

$$\textsc{i-if-false}\ \frac{t\Gamma^i \vdash^i e \mapsto [\ td^i\ |\ [\ d^i\ |\ false\ ]\ ] \qquad t\Gamma^i \vdash^i e_2 \mapsto [\ td_2^i\ |\ [\ d_2^i\ |\ v_2^i\ ]\ ]}{t\Gamma^i \vdash^i if\ e\ then\ e_1\ else\ e_2 \mapsto [\ d^i \cup td^i \cup td_2^i\ |\ [\ d^i \cup d_2^i\ |\ v_2^i\ ]\ ]}$$

$$\textsc{i-match}\ \frac{t\Gamma^i \vdash^i e \mapsto tu^i \qquad tu^i = [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ] \qquad tu^i, p \vdash_p^i t\Gamma_p^i \qquad t\Gamma_p^i \oplus t\Gamma^i \vdash^i e_1 \mapsto [\ td_1^i\ |\ [\ d_1^i\ |\ v_1^i\ ]\ ]}{t\Gamma^i \vdash^i match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^i \cup td^i \cup td_1^i\ |\ [\ d^i \cup d_1^i\ |\ v_1^i\ ]\ ]}$$

$$\textsc{i-match-var}\ \frac{t\Gamma^i \vdash^i e \mapsto tu^i \qquad tu^i = [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ] \qquad tu^i, p \vdash_p^i \bot \qquad (x, tu^i) \oplus t\Gamma^i \vdash^i e_2 \mapsto [\ td_2^i\ |\ [\ d_2^i\ |\ v_2^i\ ]\ ]}{t\Gamma^i \vdash^i match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^i \cup td^i \cup td_2^i\ |\ [\ d^i \cup d_2^i\ |\ v_2^i\ ]\ ]}$$

$$\textsc{i-constr-0}\ \frac{}{t\Gamma^i \vdash^i C \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ C\ ]\ ]} \qquad \textsc{i-constr-1}\ \frac{t\Gamma^i \vdash^i e \mapsto [\ td^i\ |\ u^i\ ]}{t\Gamma^i \vdash^i D(e) \mapsto [\ td^i\ |\ [\ \emptyset\ |\ D(u^i)\ ]\ ]}$$

$$\textsc{i-couple}\ \frac{t\Gamma^i \vdash^i e_1 \mapsto [\ td_1^i\ |\ u_1^i\ ] \qquad t\Gamma^i \vdash^i e_2 \mapsto [\ td_2^i\ |\ u_2^i\ ]}{t\Gamma^i \vdash^i (e_1, e_2) \mapsto [\ td_1^i \cup td_2^i\ |\ [\ \emptyset\ |\ (u_1^i, u_2^i)\ ]\ ]} \qquad \textsc{i-annot}\ \frac{t\Gamma^i \vdash^i e \mapsto [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ]}{t\Gamma^i \vdash^i l\ :\ e \mapsto [\ td^i\ |\ [\ l; d^i\ |\ v^i\ ]\ ]}$$

FIGURE 3.20 – Sémantique instrumentée

$$\textsc{im-constr-0}\ \frac{}{[\ td^i\ |\ [\ d^i\ |\ C\ ]\ ], C \vdash_p^i \{\}} \qquad \textsc{im-constr-1}\ \frac{}{[\ td^i\ |\ [\ d^i\ |\ D(u^i)\ ]\ ], D(x) \vdash_p^i \{(x,\ [\ \emptyset\ |\ u^i\ ])\}}$$

$$\textsc{im-couple}\ \frac{}{[\ td^i\ |\ [\ d^i\ |\ (u_1^i, u_2^i)\ ]\ ], (x_1, x_2) \vdash_p^i \{(x_1,\ [\ \emptyset\ |\ u_1^i\ ]); (x_2,\ [\ \emptyset\ |\ u_2^i\ ])\}}$$

$$\textsc{im-constr-0-not}\ \frac{p \neq C}{[\ td^i\ |\ [\ d^i\ |\ C\ ]\ ], p \vdash_p^i \bot} \qquad \textsc{im-constr-1-not}\ \frac{p \neq D(\_)}{[\ td^i\ |\ [\ d^i\ |\ D(u^i)\ ]\ ], p \vdash_p^i \bot}$$

$$\textsc{im-couple-not}\ \frac{p \neq (\_, \_)}{[\ td^i\ |\ [\ d^i\ |\ (u_1^i, u_2^i)\ ]\ ], p \vdash_p^i \bot}$$

FIGURE 3.21 – Sémantique instrumentée : règles de filtrage

$$\uparrow^i_{ti}([\ td^i\ \mid\ u^i\ ]) = u^i$$

$$\uparrow^i_{ti}(\{\}) = \{\} \qquad \uparrow^i_{ti}((x, tu^i) \oplus t\Gamma^i) = (x, \uparrow^i_{ti}(tu^i)) \oplus \uparrow^i_{ti}(t\Gamma^i)$$

FIGURE 3.22 – Valeurs instrumentées : suppression des t-dépendances

I-IDENT Cette règle est identique à celle de la sémantique sur-instrumentée. L'évaluation d'un identificateur se fait de manière habituelle, en allant chercher la valeur correspondante dans l'environnement. Les dépendances de la t-valeur instrumentée retournée sont celles qui ont été enregistrées dans l'environnement pour cet identificateur.

I-ABSTR De même que pour la sémantique sur-instrumentée, les t-dépendances et les v-dépendances sont vides. La valeur simple instrumentée est une fermeture contenant l'environnement d'évaluation préalablement nettoyé de ses t-dépendances à l'aide de la fonction $\uparrow^i_{ti}(\bullet)$ définie en figure 3.22.

I-ABSTR-REC L'évaluation d'une abstraction récursive suit exactemement le même modèle que l'évaluation d'une abstraction non-récursive.

I-APPLY La règle de l'application de la sémantique sur-instrumentée fait partie des règles avec dépendances indirectes. La règle correspondante de la sémantique instrumentée présentée ici lui ressemble à une différence près : les dépendances indirectes ne nécessitent pas de spécification complexe.

En effet, dans la sémantique sur-instrumentée, la fonction de spécification des dépendances indirectes *deps_spec_apply* permettait de spécifier les t-dépendances indirectes td'^{oi} ainsi que les v-dépendances indirectes d'^{oi}. Cette spécification précisait que la liste des labels présents dans td'^{oi} (de même pour d'^{oi}) était la même que la liste des labels présents dans d_1^{oi} (l'ensemble v-dépendances sur-instrumentée de e_1) et donnait une spécification pour chaque fonction d'impact associée à une dépendance indirecte. Dans la sémantique instrumentée, nous ne considérons plus que les listes de labels et ignorons les fonctions d'impact. Ainsi, nous pouvons remplacer td'^{oi} et d'^{oi} par la liste de labels d_1^i. Le caractère

$$\uparrow_i^{ti}(u^i) = [\ \emptyset\ \mid\ u^i\ ]$$

$$\uparrow_i^{ti}(\{\}) = \{\} \qquad \uparrow_i^{ti}((x, u^i) \oplus \Gamma^i) = (x, \uparrow_i^{ti}(u^i)) \oplus \uparrow_i^{ti}(\Gamma^i)$$

FIGURE 3.23 – Valeurs instrumentées : ajout de t-dépendances

indécidable de cette règle disparaît ainsi, avec la disparition de la fonction de spécification des dépendances indirectes.

Comme dans la sémantique sur-instrumentée, nous utilisons une fonction ajoutant des t-dépendances vides à l'environnement encapsulé pour évaluer le corps de la fermeture. Cette fonction, notée $\uparrow_i^{ti}(\bullet)$ est définie en figure 3.23.

I-REC-APPLY De la même manière que pour la règle de l'application d'une fonction non récursive, cette règle reprend la règle de la sémantique sur-instrumentée en remplaçant les dépendances indirectes par la liste des v-dépendances de la sous-expression e_1.

I-LETIN La règle d'évaluation d'une liaison est identique à la règle correspondante dans la sémantique sur-instrumentée. Elle évalue e_1 puis ajoute une liaison à l'environnement pour évaluer e_2. Le résultat est une t-valeur instrumentée constituée de la concaténation des t-dépendances des sous-expressions évaluées et de la v-valeur instrumentée de e_2.

I-IF-TRUE et I-IF-FALSE La règle d'évaluation d'une expression conditionnelle contient des dépendances indirectes. Dans la sémantique sur-instrumentée, cette règle faisait appel à la fonction de spécification des dépendances indirectes $deps\_spec\_if$ pour spécifier la liste des labels des dépendances indirectes ainsi que leurs fonctions d'impact. Cette fonction n'est plus nécessaire ici puisque la sémantique instrumentée ignore les fonctions d'impact. Nous reprennons donc la règle de la sémantique sur-instrumentée en remplaçant les t-dépendances indirectes et les v-dépendances indirectes par d^i, la liste des v-dépendances de la sous-expression testée.

I-MATCH et I-MATCH-VAR Comme pour les règles d'évaluation de l'application et de l'expression conditionnelle, la fonction de spécification des dépendances indirectes n'est plus nécessaire. Les dépendances indirectes, qui ne sont plus que des listes de labels, sont données immédiatement par la liste des v-dépendances de la sous-expression filtrée.

I-CONSTR-0, I-CONSTR-1 et I-COUPLE Ces trois règles reprennent à l'identique les règles de la sémantique sur-instrumentée. Le fait que les dépendances ne contiennent plus de fonction d'impact n'a aucune influence sur la logique de ces règles. Ceci est dû au fait qu'aucune dépendance n'est introduite, on ne fait que regrouper les t-dépendances des éventuelles sous-expressions en les concaténant et on construit la v-valeur instrumentée à l'aide des v-valeurs instrumentées des sous-expressions. La concaténation des dépendances a une signification légèrement différente puisqu'elle concatène des listes de labels au lieu de concaténer des listes associatives.

I-ANNOT La règle concernant une expression annotée est elle aussi similaire à son homologue dans la sémantique sur-instrumentée. Elle ajoute simplement le label l aux dépendances de la valeur de la sous-expression. La seule différence avec la sémantique sur-instrumentée est qu'elle n'ajoute pas de fonction d'impact.

Règles de filtrage De même que pour la sémantique sur-instrumentée, les règles de filtrage de la sémantique instrumentée se divisent en deux groupes de trois règles. Le premier groupe définit un jugement de la forme $tu^i, p \vdash^i_p \Gamma^i$ qui indique qu'une valeur instrumentée tu^i correspond au motif p et retourne le v-environnement instrumenté Γ^i. Le second groupe de règles définit un second jugement qui prend la forme suivante : $tu^i, p \vdash^i_p \bot$. Ce jugement signifie que la t-valeur instrumentée tu^i ne correspond pas au motif de filtrage p. Toutes ces règles d'inférence sont identiques à celle de la sémantique sur-instrumentée, ce qui se comprend aisément puisque la filtrage ne dépend pas des fonctions d'impact.

3.5.3 Correction

3.5.3.1 Énoncé informel du théorème

La sémantique instrumentée est une simplification de la sémantique sur-instrumentée. Elle permet de définir une analyse dynamique calculable pour tout programme **dont l'évaluation de référence termine**. Pour obtenir une sémantique calculable, nous avons retiré les fonctions d'impact de toutes les valeurs. En effet, dans les règles d'inférence avec dépendances indirectes, les fonctions d'impact contenues dans les dépendances indirectes sont spécifiées par rapport aux fonctions d'impact des prémisses. Ces spécifications ne permettent pas de construire les fonctions d'impact des dépendances indirectes et il n'est pas possible de définir un procédé calculatoire permettant de les construire puisque le problème de l'arrêt est indécidable (cf. section 3.4.1.2).

On remarque qu'il est possible de supprimer les fonctions d'impact de toutes les valeurs car dans les règles d'inférence de la sémantique sur-instrumentée, les fonctions d'impact des prémisses ne servent qu'à définir d'autres fonctions d'impact et ne sont pas du tout utilisées pour le calcul des autres parties de la valeur (la valeur de référence et les listes de labels).

Pour assurer la correction de la sémantique instrumentée il nous faut donc montrer que la seule différence entre la valeur d'un programme par la sémantique sur-instrumentée et sa valeur par la sémantique instrumentée est l'absence de fonction d'impact dans cette dernière.

3.5.3.2 Illustration par l'exemple

Pour illustrer le fonctionnement de la sémantique instrumentée ainsi que la nécessité du théorème de correction que nous venons d'énoncer informellement, reprenons les deux exemples présentés précédemment.

Exemple 1 : évaluation d'un couple

Notons e_1 le programme suivant : (3, 1:7)

$$\text{I-COUPLE}\ \dfrac{\text{I-NUM}\ \dfrac{}{t\Gamma^i \vdash^i 3 \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ 3\ ]\ ]} \qquad \text{I-ANNOT}\ \dfrac{\text{I-NUM}\ \dfrac{}{t\Gamma^i \vdash^{oi} 7 \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ 7\ ]\ ]}}{t\Gamma^i \vdash^i l\ :\ 7 \mapsto [\ \emptyset\ |\ [\ l\ |\ 7\ ]\ ]}}{t\Gamma^i \vdash^i (3, l\ :\ 7) \mapsto [\ td_1^i \cup td_2^i\ |\ [\ \emptyset\ |\ (\ [\ \emptyset\ |\ 3\ ],\ [\ l\ |\ 7\ ])\ ]\ ]}$$

FIGURE 3.24 – Exemple 1 : arbre de dérivation du jugement d'évaluation instrumentée

Nous avions présenté son jugement d'évaluation sur-instrumentée dans le t-environnement sur-instrumenté vide :

$\emptyset \vdash^{oi} e_1 \mapsto tu^{oi}$ avec $tu^{oi} = [\ \emptyset\ |\ [\ \emptyset\ |\ (\ [\ \emptyset\ |\ 3\ ],\ [\ (l, id)\ |\ 7\ ])\ ]\ ]$

Intéressons-nous maintenant au jugement d'évaluation instrumenté dans le t-environnent instrumenté vide. L'arbre d'évaluation correspondant est donné en figure 3.24 et voici le jugement obtenu :

$\emptyset \vdash^{i} e_1 \mapsto tu^{i}$ avec $tu^{i} = [\ \emptyset\ |\ [\ \emptyset\ |\ (\ [\ \emptyset\ |\ 3\ ],\ [\ l\ |\ 7\ ])\ ]\ ]$

On remarque la disparition de la fonction d'impact associée au label l dans l'ensemble des v-dépendances de la seconde composante du couple. Outre ce changement, toutes les autres parties de la valeur instrumentée sont identiques à celles de la valeur sur-instrumentée car les autres ensembles de dépendances sont vides.

Pour l'évaluation de e_1, on obtient donc la même t-valeur instrumentée, que l'on évalue le programme directement par la sémantique instrumentée ou bien en évaluant le programme par la sémantique sur-instrumentée puis en retirant toutes les fonctions d'impact.

Exemple 2 : liaison d'une fonction

Notons e_2 le programme suivant :

```
let f o =
  match o with
  | Some(x) → x + l1 : 8
  | none → d
in
```

(f (Some 18), l_2 : 17)

Nous allons évaluer ce programme par la sémantique sur-instrumentée dans l'environnement $t\Gamma_2^{oi}$ et par la sémantique instrumentée dans l'environnement $t\Gamma_2^{i} = \uparrow_{toi}^{ti}(t\Gamma_2^{oi})$. On pose :

$$t\Gamma_2^{oi} = (d,\ [\ td_d^{oi}\ \mid\ [\ d_d^{oi}\ \mid\ 26\ ]\ ])$$
$$\text{avec } td_d^{oi} = (l_3, tf_d) \text{ et } d_d^{oi} = (l_4, f_d)$$

L'arbre de dérivation du jugement d'évaluation sur-instrumentée de e_2 a été donné dans les figures 3.12 et 3.13. On rappelle ici le jugement obtenu :

$$t\Gamma_2^{oi} \vdash^{oi} e_2 \mapsto\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ (l_1, f_1)\ \mid\ 26\ ],\ [\ (l_2, id)\ \mid\ 17\ ])\ ]\ ]$$
$$\text{avec } f_1 = fun\ x \Rightarrow 18 + x$$

En ce qui concerne l'évaluation de e_2 par la sémantique instrumentée, nous donnons l'arbre de dérivation dans les figures 3.25 et 3.26. Voici le jugement d'évaluation de e_2 par la sémantique instrumentée :

$$\uparrow_{toi}^{ti}(t\Gamma_2^{oi}) \vdash^{i} e_2 \mapsto\ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ l_1\ \mid\ 26\ ],\ [\ l_2\ \mid\ 17\ ])\ ]\ ]$$

Comme dans l'exemple précédent, on constate que la propriété de correction de la sémantique instrumentée est également vraie pour l'évaluation de e_2. En effet, on obtient exactement la même valeur en utilisant la sémantique instrumentée ou la sémantique sur-instrumentée. En évaluant e_2 par la sémantique sur-instrumentée puis en supprimant toutes les fonctions d'impact dans la t-valeur sur-instrumentée, on obtient la t-valeur instrumentée [∅ | [∅ | ([l_1 | 26], [l_2 | 17])]]. Aussi, en supprimant dans un premier temps toutes les fonctions d'impact dans l'environnement d'évaluation $t\Gamma_2^{oi}$ puis en utilisant la sémantique instrumentée pour évaluer e_2 on obtient la même t-valeur instrumentée [∅ | [∅ | ([l_1 | 26], [l_2 | 17])]].

$$\textsc{i-letin}\ 0\ \dfrac{\textsc{i-couple}\ 1\ \dfrac{\textsc{i-annot}\ 2\ \dfrac{\textsc{i-num}\ 3}{(f,...)\oplus t\Gamma_2^i \vdash^i 17 \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ 17\ ]\ ]}}{(f,...)\oplus t\Gamma_2^i \vdash^i l_2\ :\ 17 \mapsto [\ \emptyset\ |\ [\ l_2\ |\ 17\ ]\ ]}}{(f,...)\oplus t\Gamma_2^i \vdash^i (f(\mathit{Some}\ 18), l_2\ :\ 17) \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ (\ [\ l_1\ |\ 26\ ],\ [\ l_2\ |\ 17\ ])\ ]\ ]}$$

$$t\Gamma_2^i \vdash^i e_2 \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ (\ [\ l_1\ |\ 26\ ],\ [\ l_2\ |\ 17\ ])\ ]\ ]$$

$$\textsc{i-apply}\ 2\ \dfrac{\textsc{i-match}\ 3\ \dfrac{\vdots\ \text{(figure 3.26)}\ \vdots}{(o,...)\oplus \uparrow_i^{ti}(\uparrow_{ti}^i(t\Gamma_2^i)) \vdash^i \mathit{match}\ o\ \mathit{with}\ \mathit{Some}\ x \to x + l_1\ :\ 8 \mid \mathit{none} \to d \mapsto [\ \emptyset\ |\ [\ l_1\ |\ 26\ ]\ ]}}{(f,...)\oplus t\Gamma_2^i \vdash^i f(\mathit{Some}\ 18) \mapsto [\ \emptyset\ |\ [\ l_1\ |\ 26\ ]\ ]}$$

$$\textsc{i-constr-1}\ 3\ \dfrac{\textsc{i-num}\ 4}{(f,...)\oplus t\Gamma_2^i \vdash^i \mathit{Some}\ 18 \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ \mathit{Some}(\ [\ \emptyset\ |\ 18\ ])\ ]\ ]}$$

$$\textsc{i-ident}\ 3\ \dfrac{[\ \emptyset\ |\ [\ \emptyset\ |\ <\lambda o.\mathit{match}\ o\ \mathit{with}..., \uparrow_{ti}^i(t\Gamma_2^i) >\ ]\ ] = ((f,...)\oplus t\Gamma_2^i)[f]}{(f,...)\oplus t\Gamma_2^i \vdash^i f \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ <\lambda o.\mathit{match}\ o\ \mathit{with}..., \uparrow_{ti}^i(t\Gamma_2^i) >\ ]\ ]}$$

$$\textsc{i-abstr}\ 1\ \dfrac{}{t\Gamma_2^i \vdash^i \lambda o.\mathit{match}\ o\ \mathit{with}... \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ <\lambda o.\mathit{match}\ o\ \mathit{with}..., \uparrow_{ti}^i(t\Gamma_2^i) >\ ]\ ]}$$

FIGURE 3.25 – Exemple 2 : arbre de dérivation du jugement d'évaluation instrumentée

$$\textsc{i-match}\ 3\ \dfrac{\textsc{i-plus}\ 4\ \dfrac{}{(x,\ [\ \emptyset\ |\ 18\ ])\oplus(o,...)\oplus\uparrow_i^{ti}(\uparrow_{ti}^i(t\Gamma_2^i)) \vdash^i x + l_1\ :\ 8 \mapsto [\ \emptyset\ |\ [\ l_1\ |\ 26\ ]\ ]}}{(o,...)\oplus\uparrow_i^{ti}(\uparrow_{ti}^i(t\Gamma_2^i)) \vdash^i \mathit{match}\ o\ \mathit{with}\ \mathit{Some}\ x \to x + l_1\ :\ 8 \mid \mathit{none} \to d \mapsto [\ \emptyset\ |\ [\ l_1\ |\ 26\ ]\ ]}$$

$$\vdots$$

$$\textsc{im-constr-1}\ 4\ \dfrac{\textsc{i-ident}\ 4\ \dfrac{[\ \emptyset\ |\ [\ \emptyset\ |\ \mathit{Some}(\ [\ \emptyset\ |\ 18\ ])\ ]\ ] = ((o,...)\oplus\uparrow_i^{ti}(\uparrow_{ti}^i(t\Gamma_2^i)))[o]}{(o,...)\oplus\uparrow_i^{ti}(\uparrow_{ti}^i(t\Gamma_2^i)) \vdash^i o \mapsto [\ \emptyset\ |\ [\ \emptyset\ |\ \mathit{Some}(\ [\ \emptyset\ |\ 18\ ])\ ]\ ]}}{[\ \emptyset\ |\ [\ \emptyset\ |\ \mathit{Some}(\ [\ \emptyset\ |\ 18\ ])\ ]\ ], \mathit{Some}\ x \vdash_p^i (x,\ [\ \emptyset\ |\ 18\ ])}$$

FIGURE 3.26 – Exemple 2 : sous-arbre de dérivation instrumentée du filtrage

3.5.3.3 Énoncé formel du théorème

On veut prouver que la sémantique instrumentée donne les mêmes résultats que la sémantique sur-instrumentée hormis les fonctions d'impact. Si un programme a une valeur instrumentée, alors on peut être sûr qu'il a aussi une valeur sur-instrumentée et qu'en supprimant les fonctions d'impact de cette dernière, on obtient la même valeur instrumentée.

On commence par définir formellement la fonction qui supprime toutes les fonctions d'impact d'une valeur sur-instrumentée pour obtenir la valeur instrumentée correspondante. Cette fonction est notée $\uparrow^{ti}_{toi}(\bullet)$. On dit que c'est une abstraction des valeurs sur-instrumentées en valeurs instrumentées.

Abstraction d'une t-valeur sur-instrumentée :

$$\uparrow^{ti}_{toi}(\ [\ td^{oi}\ \mid\ u^{oi}\ ]) := \quad [\ \uparrow^{ti}_{toi}(td^{oi})\ \mid \uparrow^{i}_{oi}(u^{oi})\ ]$$

Abstraction d'une v-valeur sur-instrumentée :

$$\uparrow^{i}_{oi}(\ [\ d^{oi}\ \mid\ v^{oi}\ ]) := \quad [\ \uparrow^{i}_{oi}(d^{oi})\ \mid \uparrow^{i}_{oi}(v^{oi})\ ]$$

Abstraction d'un ensemble de t-dépendances sur-instrumenté :

$$\begin{array}{rl} \uparrow^{ti}_{toi}(\emptyset) := & \emptyset \\ \uparrow^{ti}_{toi}((l, tf^{oi}); td^{oi}) := & l; \uparrow^{ti}_{toi}(td^{oi}) \end{array}$$

Abstraction d'un ensemble de v-dépendances sur-instrumenté :

$$\begin{array}{rl} \uparrow^{i}_{oi}(\emptyset) := & \emptyset \\ \uparrow^{i}_{oi}((l, f^{oi}); d^{oi}) := & l; \uparrow^{i}_{oi}(d^{oi}) \end{array}$$

Abstraction d'une valeur simple sur-instrumentée :

$$\begin{array}{rl} \uparrow^{i}_{oi}(n) := & n \\ \uparrow^{i}_{oi}(b) := & b \\ \uparrow^{i}_{oi}(C) := & C \\ \uparrow^{i}_{oi}(D(u^{oi})) := & D(\uparrow^{i}_{oi}(u^{oi})) \\ \uparrow^{i}_{oi}((u^{oi}_1, u^{oi}_2)) := & (\uparrow^{i}_{oi}(u^{oi}_1), \uparrow^{i}_{oi}(u^{oi}_2)) \\ \uparrow^{i}_{oi}(< \lambda x.e, \Gamma^{oi} >) := & < \lambda x.e, \uparrow^{i}_{oi}(\Gamma^{oi}) > \\ \uparrow^{i}_{oi}(< \texttt{rec} f.x.e, \Gamma^{oi} >) := & < \texttt{rec} f.x.e, \uparrow^{i}_{oi}(\Gamma^{oi}) > \end{array}$$

Abstraction d'un t-environnement sur-instrumenté :

$$\begin{array}{rl} \uparrow^{ti}_{toi}(\emptyset) := & \emptyset \\ \uparrow^{ti}_{toi}((x, tu^{oi}) \oplus t\Gamma^{oi}) := & (x, \uparrow^{ti}_{toi}(tu^{oi})) \oplus \uparrow^{ti}_{toi}(t\Gamma^{oi}) \end{array}$$

Abstraction d'un v-environnement sur-instrumenté :

$$\begin{array}{rcl} \uparrow^{i}_{oi}(\emptyset) & := & \emptyset \\ \uparrow^{i}_{oi}((x, u^{oi}) \oplus \Gamma^{oi}) & := & (x, \uparrow^{i}_{oi}(u^{oi})) \oplus \uparrow^{i}_{oi}(\Gamma^{oi}) \end{array}$$

Il nous faut également définir formellement la fonction de concrétisation. Cette fonction transforme toute valeur instrumentée en une valeur sur-instrumentée dont elle est l'image par la fonction d'abstraction. Pour cela, la fonction de concrétisation doit associer une fonction d'impact à tout label présent dans un ensemble de dépendances de la valeur instrumentée. Le choix de ces fonctions d'impact est complètement arbitraire. Cette fonction de concrétisation est notée $\uparrow^{toi}_{ti}(\bullet)$.

Notons tf^{oi}_{dummy} et f^{oi}_{dummy} une t-fonction d'impact et une v-fonction d'impact quelconques qui seront introduites lors de la concrétisation des ensembles de dépendances instrumentés.

Concrétisation d'une t-valeur instrumentée :

$$\uparrow^{toi}_{ti}(\ [\ td^{i}\ \mid\ u^{i}\ ]) := \quad [\ \uparrow^{toi}_{ti}(td^{i})\ \mid \uparrow^{oi}_{i}(u^{i})\ ]$$

Concrétisation d'une v-valeur instrumentée :

$$\uparrow^{oi}_{i}(\ [\ d^{i}\ \mid\ v^{i}\ ]) := \quad [\ \uparrow^{oi}_{i}(d^{i})\ \mid \uparrow^{oi}_{i}(v^{i})\ ]$$

Concrétisation d'un ensemble de t-dépendances instrumenté :

$$\begin{array}{rcl} \uparrow^{toi}_{ti}(\emptyset) & := & \emptyset \\ \uparrow^{toi}_{ti}(l; td^{i}) & := & (l, tf^{oi}_{dummy}); \uparrow^{toi}_{ti}(td^{i}) \end{array}$$

Concrétisation d'un ensemble de v-dépendances instrumenté :

$$\begin{array}{rcl} \uparrow^{oi}_{i}(\emptyset) & := & \emptyset \\ \uparrow^{oi}_{i}(l; d^{i}) & := & (l, f^{oi}_{dummy}); \uparrow^{oi}_{i}(d^{i}) \end{array}$$

Concrétisation d'une valeur simple instrumentée :

$$\begin{array}{rcl} \uparrow^{oi}_{i}(n) & := & n \\ \uparrow^{oi}_{i}(b) & := & b \\ \uparrow^{oi}_{i}(C) & := & C \\ \uparrow^{oi}_{i}(D(u^{i})) & := & D(\uparrow^{oi}_{i}(u^{i})) \\ \uparrow^{oi}_{i}((u^{i}_{1}, u^{i}_{2})) & := & (\uparrow^{oi}_{i}(u^{i}_{1}), \uparrow^{oi}_{i}(u^{i}_{2})) \\ \uparrow^{oi}_{i}(< \lambda x.e, \Gamma^{i} >) & := & < \lambda x.e, \uparrow^{oi}_{i}(\Gamma^{i}) > \\ \uparrow^{oi}_{i}(< \texttt{rec} f.x.e, \Gamma^{i} >) & := & < \texttt{rec} f.x.e, \uparrow^{oi}_{i}(\Gamma^{i}) > \end{array}$$

Concrétisation d'un t-environnement instrumenté :

$$\begin{aligned} \uparrow_{ti}^{toi}(\emptyset) &:= \emptyset \\ \uparrow_{ti}^{toi}((x, tu^i) \oplus t\Gamma^i) &:= (x, \uparrow_{ti}^{toi}(tu^i)) \oplus \uparrow_{ti}^{toi}(t\Gamma^i) \end{aligned}$$

Concrétisation d'un v-environnement instrumenté :

$$\begin{aligned} \uparrow_{i}^{oi}(\emptyset) &:= \emptyset \\ \uparrow_{i}^{oi}((x, u^i) \oplus \Gamma^i) &:= (x, \uparrow_{i}^{oi}(u^i)) \oplus \uparrow_{i}^{oi}(\Gamma^i) \end{aligned}$$

Nous pouvons alors énoncer le thèorème de correction :

Théorème 3.5.1 (Correction de la sémantique instrumentée)**.**

$\forall (t\Gamma^i, e, tu^i).t\Gamma^i \vdash^i e \mapsto tu^i$

$\Rightarrow \exists (tu^{oi}).\ \uparrow_{ti}^{toi}(t\Gamma^i) \vdash^{oi} e \mapsto tu^{oi} \wedge tu^i = \uparrow_{toi}^{ti}(tu^{oi})$

Le théorème de correction énonce que s'il existe un jugement pour la sémantique instrumentée évaluant un programme e dans un t-environnement instrumenté $t\Gamma^i$ et retournant une t-valeur instrumentée tu^i, alors il existe un jugement évaluant e dans le t-environnement sur-instrumenté obtenu en ajoutant des fonctions d'impact quelconques dans $t\Gamma^i$ et en supprimant les fonctions d'impact dans la t-valeur sur-instrumentée obtenue, on retrouve la t-valeur instrumentée tu^i.

La propriété suivante est plus forte que le théorème de correction. Elle permet de s'affranchir de la fonction de concrétisation qui n'a que peu de signification puisqu'elle ne fait qu'introduire des fonctions d'impact quelconques dans les ensembles de dépendances. Il aurait été sans doute plus judicieux de prouver cette propriété à la place du théorème de correction exprimé ci-dessus. Elle exprime que pour tout t-environnement sur-instrumenté $t\Gamma^{oi}$, s'il existe un jugement instrumenté pour e dans l'environnement obtenu en supprimant toutes les fonctions d'impact de $t\Gamma^{oi}$ et que ce jugement retourne une t-valeur instrumentée tu^i, alors il existe un jugement sur-instrumenté dans $t\Gamma^{oi}$ et en supprimant les fonctions d'impact dans la t-valeur sur-instrumentée obtenue, on retrouve la t-valeur instrumentée tu^i. Si on spécialise cette propriété en prenant $t\Gamma^{oi} = \uparrow_{ti}^{toi}(t\Gamma^i)$, on retrouve le théorème de correction exprimé ci-dessus qui est donc un cas particulier. La preuve de cette propriété pourra faire l'objet d'un travail futur.

$$\begin{aligned} &\forall (t\Gamma^{oi}, e, tu^i).\ \uparrow_{toi}^{ti}(t\Gamma^{oi}) \vdash^i e \mapsto tu^i \\ &\Rightarrow \exists (tu^{oi}).t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \wedge tu^i = \uparrow_{toi}^{ti}(tu^{oi}) \end{aligned}$$

3.5.3.4 Preuve de correction

La preuve est faite par induction sur le jugement de la sémantique instrumentée. Nous expliquons ici quelques cas pour permettre au lecteur de comprendre facilement la manière dont la preuve a été faite. La preuve complète est disponible sous forme de code source Coq.

cas I-NUM Ce cas est trivial. Dans ce cas, l'expression e n'est autre qu'une constante numérique n, la t-valeur instrumentée tu^i ne contient que des dépendances vides et la valeur simple sur-instrumentée n. Il suffit d'appliquer la règle d'inférence OI-NUM de la sémantique sur-instrumentée pour conclure.

cas I-IDENT Dans ce cas, l'expression e est un identificateur que nous noterons x. Cet identificateur est présent dans l'environnement $t\Gamma^i$ associé à la valeur tu^i. L'identificateur x est donc également présent dans l'environnement $\uparrow_{ti}^{toi}(t\Gamma^i)$ associé à la valeur $\uparrow_{ti}^{toi}(tu^i)$. Il suffit donc d'appliquer la règle d'inférence OI-IDENT de la sémantique sur-instrumentée pour conclure :

$$\uparrow_{ti}^{toi}(t\Gamma^i) \vdash^{oi} x \mapsto \uparrow_{ti}^{toi}(tu^i) \wedge tu^i = \uparrow_{toi}^{ti}(\uparrow_{ti}^{toi}(tu^i))$$

cas I-ABSTR Nous avons le jugement de la sémantique instrumentée suivant :

$$t\Gamma^i \vdash^i \lambda x.e \mapsto \ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{ti}^{i}(t\Gamma^i) >]\ ]$$

Nous allons montrer que la t-valeur sur-instrumentée suivante convient :

$$\uparrow_{ti}^{toi}(\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{ti}^{i}(t\Gamma^i) >]\ ])$$
$$\text{qui est égale à } \ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^i)) >]\ ]$$

Il nous faut alors montrer la propriété suivante :

$$\uparrow_{ti}^{toi}(t\Gamma^i) \vdash^{oi} e \mapsto \ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^i)) >]\ ]$$
$$\wedge\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{ti}^{i}(t\Gamma^i) >]\ ] = \uparrow_{toi}^{ti}(\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^i)) >]\ ])$$

La première partie de la propriété se montre en appliquant la règle d'inférence OI-ABSTR et en remarquant que :

$$\uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^{i})) = \uparrow_{toi}^{oi}(\uparrow_{ti}^{toi}(t\Gamma^{i}))$$

La seconde partie de la propriété se montre en remarquant que :

$$\uparrow_{toi}^{ti}(\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^{i})) >\ ]\ ]) =\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid < \lambda x.e, \uparrow_{oi}^{i}(\uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^{i}))) >\ ]\ ]$$
$$\text{et } \uparrow_{ti}^{i}(t\Gamma^{i}) = \uparrow_{oi}^{i}(\uparrow_{i}^{oi}(\uparrow_{ti}^{i}(t\Gamma^{i})))$$

cas I-APPLY Nous avons les jugements de la sémantique instrumentée issus de la règle d'inférence I-APPLY ainsi que les trois hypothèses d'induction suivantes :

$$t\Gamma^{i} \vdash^{i} e_1 \mapsto\ [\ td_1^{i}\ \mid\ \ [\ d_1^{i}\ \mid < \lambda x.e, \Gamma_1^{i} >\ ]\ ]$$
$$t\Gamma^{i} \vdash^{i} e_2 \mapsto tu_2^{i}$$
$$tu_2^{i} =\ [\ td_2^{i}\ \mid\ u_2^{i}\ ]$$
$$(x, tu_2^{i}) \oplus \uparrow_{i}^{ti}(\Gamma_1^{i}) \vdash^{i} e \mapsto\ [\ td^{i}\ \mid\ \ [\ d^{i}\ \mid\ v^{i}\ ]\ ]$$
$$t\Gamma^{i} \vdash^{i} e_1\ e_2 \mapsto\ [\ td_1^{i} \cup td_2^{i} \cup td^{i} \cup d_1^{i}\ \mid\ \ [\ d_1^{i} \cup d^{i}\ \mid\ v^{i}\ ]\ ]$$

$$\uparrow_{ti}^{toi}(t\Gamma^{i}) \vdash^{oi} e_1 \mapsto\ [\ td_1^{oi}\ \mid\ \ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ]$$
$$\wedge\ [\ td_1^{i}\ \mid\ \ [\ d_1^{i}\ \mid < \lambda x.e, \Gamma_1^{i} >\ ]\ ] = \uparrow_{toi}^{ti}(\ [\ td_1^{oi}\ \mid\ \ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ])$$

$$\uparrow_{ti}^{toi}(t\Gamma^{i}) \vdash^{oi} e_2 \mapsto\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]$$
$$\wedge\ [\ td_2^{i}\ \mid\ u_2^{i}\ ] = \uparrow_{toi}^{ti}(\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ])$$

$$\uparrow_{ti}^{toi}((x, tu_2^{i}) \oplus \uparrow_{i}^{ti}(\Gamma_1^{i})) \vdash^{oi} e \mapsto\ [\ td^{oi}\ \mid\ \ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ]$$
$$\wedge\ [\ td^{i}\ \mid\ \ [\ d^{i}\ \mid\ v^{i}\ ]\ ] = \uparrow_{toi}^{ti}(\ [\ td^{oi}\ \mid\ \ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ])$$

Nous souhaitons dériver un jugement d'évaluation sur-instrumentée pour l'expression $e_1\ e_2$ en utilisant la règle d'inférence OI-APPLY ainsi que les jugements d'évaluation de e_1 et de e_2 issus des hypothèses d'induction ci-dessus. Il nous manque un jugement d'évaluation adéquat pour e et une hypothèse concernant la spécification des dépendances indirectes.

Tout d'abord, nous remarquons que la fonction de spécification des dépendances indirectes *deps_spec_apply* est une fonction totale (la preuve de cette propriété est assez

simple mais nécessite cependant de faire appel à l'axiome du choix ainsi qu'à l'axiome du tiers exclu). Il existe donc des ensembles de dépendances td'^{oi} et d'^{oi} tels que :

$$deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi})$$

En ce qui concerne le jugement d'évaluation sur-instrumentée de e, on reprend celui qui provient des hypothèses d'induction en l'adaptant. Remarquons tout d'abord qu'il est possible de réécrire l'environnement d'évaluation ainsi :

$$\begin{array}{rcl}
\uparrow_{ti}^{toi}((x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)) & = & \uparrow_{ti}^{toi}((x, \uparrow_{toi}^{ti}(\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ])) \oplus \uparrow_i^{ti}(\uparrow_{oi}^{i}(\Gamma_1^{oi}))) \\
& = & (x, \uparrow_{ti}^{toi}(\uparrow_{toi}^{ti}(\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]))) \oplus \uparrow_{ti}^{toi}(\uparrow_i^{ti}(\uparrow_{oi}^{i}(\Gamma_1^{oi}))) \\
& = & (x, \uparrow_{ti}^{toi}(\uparrow_{toi}^{ti}(\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]))) \oplus \uparrow_{ti}^{toi}(\uparrow_{toi}^{ti}(\uparrow_{oi}^{toi}(\Gamma_1^{oi}))) \\
& = & \uparrow_{ti}^{toi}(\uparrow_{toi}^{ti}((x,\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi})))
\end{array}$$

On a donc le jugement suivant :

$$\uparrow_{ti}^{toi}(\uparrow_{toi}^{ti}((x,\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}))) \vdash^{oi} e \mapsto\ [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ]$$

On en déduit qu'il existe un jugement d'évaluation dans le t-environnement sur-instrumenté $(x,\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi})$ qui évalue e en une certaine t-valeur sur-instrumentée $[\ td''^{oi}\ \mid\ [\ d''^{oi}\ \mid\ v''^{oi}\ ]\ ]$ qui vérifie les propriétés suivantes :

$$\begin{array}{c}
(x,\ [\ td_2^{oi}\ \mid\ u_2^{oi}\ ]) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto\ [\ td''^{oi}\ \mid\ [\ d''^{oi}\ \mid\ v''^{oi}\ ]\ ] \\
\uparrow_{toi}^{ti}(\ [\ td''^{oi}\ \mid\ [\ d''^{oi}\ \mid\ v''^{oi}\ ]\ ]) = \uparrow_{toi}^{ti}(\ [\ td^{oi}\ \mid\ [\ d^{oi}\ \mid\ v^{oi}\ ]\ ])
\end{array}$$

On peut alors appliquer la règle d'inférence OI-APPLY pour déduire un jugement d'évaluation sur-instrumentée pour l'expression $e_1\ e_2$:

$$\text{OI-APPLY}\ \frac{\begin{array}{c} t\Gamma^{oi} \vdash^{oi} e_1 \mapsto\ [\ td_1^{oi}\ \mid\ [\ d_1^{oi}\ \mid < \lambda x.e, \Gamma_1^{oi} >\ ]\ ] \\ t\Gamma^{oi} \vdash^{oi} e_2 \mapsto tu_2^{oi} \qquad tu_2^{oi} = [\ td_2^{oi}\ \mid\ u_2^{oi}\ ] \\ (x, tu_2^{oi}) \oplus \uparrow_{oi}^{toi}(\Gamma_1^{oi}) \vdash^{oi} e \mapsto\ [\ td''^{oi}\ \mid\ [\ d''^{oi}\ \mid\ v''^{oi}\ ]\ ] \\ deps\_spec\_apply(tu_2^{oi}, d_1^{oi}) = (td'^{oi}, d'^{oi}) \end{array}}{t\Gamma^{oi} \vdash^{oi} e_1\ e_2 \mapsto\ [\ td_1^{oi} \cup td_2^{oi} \cup td''^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d''^{oi}\ \mid\ v''^{oi}\ ]\ ]}$$

Pour conclure, il ne nous reste plus qu'à prouver qu'en supprimant les fonctions d'impact de la t-valeur sur-instrumentée de $e_1\ e_2$, on retrouve sa t-valeur instrumentée :

$$
\begin{array}{l}
\uparrow^{ti}_{toi}(\ [\ td_1^{oi} \cup td_2^{oi} \cup td''^{oi} \cup td'^{oi}\ \mid\ [\ d'^{oi} \cup d''^{oi}\ \mid\ v''^{oi}\ ]\ ]) \\
=\ [\ \uparrow^{ti}_{toi}(td_1^{oi}) \cup \uparrow^{ti}_{toi}(td_2^{oi}) \cup \uparrow^{ti}_{toi}(td''^{oi}) \cup \uparrow^{ti}_{toi}(td'^{oi})\ \mid\ [\ \uparrow^{i}_{oi}(d'^{oi}) \cup \uparrow^{i}_{oi}(d''^{oi})\ \mid \uparrow^{i}_{oi}(v''^{oi})\ ]\ ] \\
=\ [\ \uparrow^{ti}_{toi}(td_1^{oi}) \cup \uparrow^{ti}_{toi}(td_2^{oi}) \cup \uparrow^{ti}_{toi}(td^{oi}) \cup \uparrow^{ti}_{toi}(td'^{oi})\ \mid\ [\ \uparrow^{i}_{oi}(d'^{oi}) \cup \uparrow^{i}_{oi}(d^{oi})\ \mid \uparrow^{i}_{oi}(v^{oi})\ ]\ ] \\
=\ [\ td_1^{i} \cup td_2^{i} \cup td^{i} \cup \uparrow^{ti}_{toi}(td'^{oi})\ \mid\ [\ \uparrow^{i}_{oi}(d'^{oi}) \cup d^{i}\ \mid\ v^{i}\ ]\ ] \\
=\ [\ td_1^{i} \cup td_2^{i} \cup td^{i} \cup d_1^{i}\ \mid\ [\ d_1^{i} \cup d^{i}\ \mid\ v^{i}\ ]\ ]
\end{array}
$$

3.6 Correction de l'analyse dynamique

Nous souhaitons, dans cette section, établir une preuve de correction de notre analyse dynamique vis-à-vis de la notion de dépendance établie formellement en section 2.7. Pour cela, nous avons présenté plus haut plusieurs sémantiques intermédiaires en établissant à chaque étape la correction de la sémantique présentée par rapport à la précédente. Nous allons maintenant pouvoir enchaîner les preuves intermédiaires pour établir un théorème de correction de l'analyse dynamique (la sémantique instrumentée) par rapport à la notion de dépendances introduite formellement à l'aide de la sémantique avec injection (cf. section 3.3).

3.6.1 Énoncé informel du théorème

Le but de notre analyse est de pouvoir affirmer que si un label n'apparaît pas dans le résultat de l'analyse d'un programme alors ce label n'a pas d'impact sur l'évaluation de ce programme.

Autrement dit, après avoir obtenu un jugement d'évaluation instrumentée du programme à analyser et après avoir vérifié que le label en question n'apparaît pas dans les dépendances calculées alors on peut affirmer que peu importe l'injection considérée sur ce label, l'évaluation avec injection de ce programme retournera toujours la valeur de référence du programme (la valeur du programme lors d'une évaluation sans injection).

3.6.2 Illustration par l'exemple

Exemple 1 : évaluation d'un couple

Notons e_1 le programme suivant : (3, 1:7)
Son jugement d'évaluation instrumenté dans le t-environnent instrumenté vide est :

$$\emptyset \vdash^i e_1 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ \emptyset\ \mid\ 3\ ],\ [\ l\ \mid\ 7\ ])\ ]\ ]$$

Seul le label l apparaît dans la t-valeur instrumentée de e_1. On peut donc affirmer que

tout autre label n'a pas d'impact sur l'évaluation de e_1. On a en effet pour toute injection $(l', v_{l'})$ sur un label l' différent de l :

$$\emptyset \vdash_{l':v_{l'}} e_1 \mapsto (3, 7)$$

Exemple 2 : liaison d'une fonction

Notons e_2 le programme suivant :

```
let f o =
  match o with
  | Some(x) → x + l1 : 8
  | none → d + l2 : 8
in
(f (Some 18), l3 : 17)
```

Voici le jugement d'évaluation instrumentée de e_2 dans un t-environnement instrumenté $t\Gamma_2^i = (d, [\, td_d^i \mid [\, d_d^i \mid 26\,]\,])$ où d est un identificateur, td_d^i un ensemble de t-dépendances et d_d^i un ensemble de v-dépendances :

$$t\Gamma_2^i \vdash^i e_2 \mapsto [\, \emptyset \mid [\, \emptyset \mid ([\, l_1 \mid 26\,], [\, l_3 \mid 17\,])\,]\,]$$

Le label l_2 n'étant pas présent dans la t-valeur instrumentée du programme e_2, nous pouvons en conclure qu'aucune injection sur l_2 n'a d'impact sur l'évaluation de e_2 :

$$\forall v_{l_2}.(d, 26) \vdash_{l_2:v_{l_2}} e_2 \mapsto (26, 17)$$

Par contre, les labels l_1 et l_3 apparaissent dans la t-valeur instrumentée du programme e_2. Il n'est donc pas impossible qu'une injection sur un de ces labels puisse modifier le comportement du programme, comme le montrent les exemples ci-dessous :

$$(d, 26) \vdash_{l_1:24} e_2 \mapsto (32, 17)$$

$$(d, 26) \vdash_{l_3:613} e_2 \mapsto (26, 613)$$

$$\textsc{ldna-e-i-num}\ \frac{}{ldna\_in\_eval^i(l, t\Gamma^i, n)} \qquad \textsc{ldna-e-i-constr-0}\ \frac{}{ldna\_in\_eval^i(l, t\Gamma^i, C)} \qquad \textsc{ldna-e-i-constr-1}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e)}{ldna\_in\_eval^i(l, t\Gamma^i, D(e))}$$

$$\textsc{ldna-e-i-ident}\ \frac{ldna\_in\_val^i(l, t\Gamma^i[x])}{ldna\_in\_eval^i(l, t\Gamma^i, x)} \qquad \textsc{ldna-e-i-ident-unbound}\ \frac{x \notin support(t\Gamma^i)}{ldna\_in\_eval^i(l, t\Gamma^i, x)}$$

$$\textsc{ldna-e-i-abstr}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e)}{ldna\_in\_eval^i(l, t\Gamma^i, \lambda x.e)} \qquad \textsc{ldna-e-i-abstr-rec}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e)}{ldna\_in\_eval^i(l, t\Gamma^i, \texttt{rec} f.x.e)}$$

$$\textsc{ldna-e-i-apply}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e_1) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_2)}{ldna\_in\_eval^i(l, t\Gamma^i, e_1\ e_2)} \qquad \textsc{ldna-e-i-if}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_1) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_2)}{ldna\_in\_eval^i(l, t\Gamma^i, if\ e\ then\ e_1\ else\ e_2)}$$

$$\textsc{ldna-e-i-match}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_1) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_2)}{ldna\_in\_eval^i(l, t\Gamma^i, match\ e\ with\ p \to e_1 \mid x \to e_2)} \qquad \textsc{ldna-e-i-annot}\ \frac{l \neq l' \quad ldna\_in\_eval^i(l, t\Gamma^i, e)}{ldna\_in\_eval^i(l, t\Gamma^i, l\ :\ e)}$$

$$\textsc{ldna-e-i-couple}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e_1) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_2)}{ldna\_in\_eval^i(l, t\Gamma^i, (e_1, e_2))} \qquad \textsc{ldna-e-i-letin}\ \frac{ldna\_in\_eval^i(l, t\Gamma^i, e_1) \quad ldna\_in\_eval^i(l, t\Gamma^i, e_2)}{ldna\_in\_eval^i(l, t\Gamma^i, let\ x\ =\ e_1\ in\ e_2)}$$

FIGURE 3.27 – Prédicat de non-apparition d'un label lors d'une évaluation instrumentée

3.6.3 Énoncé formel du théorème

Pour énoncer formellement le théorème de correction, il nous faut tout d'abord définir formellement les notions de non-apparition d'un label dans une valeur instrumentée et dans un t-environnement instrumenté. On définit pour cela les prédicats $ldna\_in\_eval^i$, $ldna\_in\_val^i$ et $ldna\_in\_env^i$. Leur définition est donnée respectivement en figure 3.27, en figure 3.28 et en figure 3.29.

Théorème 3.6.1 (Correction de l'analyse dynamique)**.**

$\forall(t\Gamma^i, e, tu^i, l).$

$t\Gamma^i \vdash^i e \mapsto tu^i$

$\Rightarrow ldna\_in\_env^{ti}(l, t\Gamma^i)$

$\Rightarrow ldna\_in\_val^{ti}(l, tu^i)$

$\Rightarrow \forall v_l.\ \uparrow_{ti}(t\Gamma^i) \vdash_{l:v_l} e \mapsto \uparrow_{ti}(tu^i)$

$$\text{LDNA-V-I-NUM}\ \frac{}{ldna\_in\_val^i(l,n)} \qquad \text{LDNA-V-I-BOOL}\ \frac{}{ldna\_in\_val^i(l,b)} \qquad \text{LDNA-V-I-CONSTR-0}\ \frac{}{ldna\_in\_val^i(l,C)} \qquad \text{LDNA-V-I-CONSTR-1}\ \frac{ldna\_in\_val^i(l,u)}{ldna\_in\_val^i(l,D(u))}$$

$$\text{LDNA-V-I-COUPLE}\ \frac{ldna\_in\_val^i(l,u_1) \quad ldna\_in\_val^i(l,u_2)}{ldna\_in\_val^i(l,(u_1,u_2))} \qquad \text{LDNA-V-I-CLOSURE}\ \frac{ldna\_in\_eval^i(l,\uparrow_i^{ti}(\Gamma^i),e)}{ldna\_in\_val^i(l,<\lambda x.e,\Gamma^i>)} \qquad \text{LDNA-V-I-CLOSURE-REC}\ \frac{ldna\_in\_eval^i(l,\uparrow_i^{ti}(\Gamma^i),e)}{ldna\_in\_val^i(l,<\texttt{rec}f.x.e,\Gamma^i>)}$$

$$\text{LDNA-V-I-V-VAL}\ \frac{l \notin d^i \qquad ldna\_in\_val^i(l,v^i)}{ldna\_in\_val^i(l,\ [\ d^i\ \mid\ v^i\ ])} \qquad \text{LDNA-V-I-T-VAL}\ \frac{l \notin td^i \qquad ldna\_in\_val^i(l,u^i)}{ldna\_in\_val^i(l,\ [\ td^i\ \mid\ u^i\ ])}$$

FIGURE 3.28 – Prédicat de non-apparition d'un label dans une valeur instrumentée

$$\text{LDNA-ENV-I-EMPTY}\ \frac{}{ldna\_in\_env^i(l,\emptyset)} \qquad \text{LDNA-ENV-I-CONS}\ \frac{ldna\_in\_env^i(l,tu^i) \qquad ldna\_in\_env^i(l,t\Gamma^i)}{ldna\_in\_env^i(l,(tu^i)\oplus t\Gamma^i)}$$

FIGURE 3.29 – Prédicat de non-apparition d'un label dans un environnement instrumenté

3.6.4 Preuve de correction

La preuve de ce théorème repose sur les théorèmes déjà prouvés précédemment concernant la correction de la sémantique instrumentée vis-à-vis de la sémantique sur-instrumentée, la correction de la sémantique sur-instrumentée vis-à-vis de la sémantique avec injection et la correction de la sémantique avec injection vis-à-vis de la sémantique opérationnelle avec injection.

L'enchaînement de ces théorèmes permet ainsi d'établir le lien entre la sémantique instrumentée et la sémantique opérationnelle avec injection.

On part des hypothèses suivantes :

$$t\Gamma^i \vdash^i e \mapsto tu^i$$

$$ldna\_in\_env^{ti}(l,t\Gamma^i)$$

$$ldna\_in\_val^{ti}(l,tu^i)$$

Rappel du théorème de correction de la sémantique instrumentée :

$$\forall(t\Gamma^i, e, tu^i).t\Gamma^i \vdash^i e \mapsto tu^i \Rightarrow \exists(tu^{oi}).\uparrow_{ti}^{toi}(t\Gamma^i) \vdash^{oi} e \mapsto tu^{oi} \wedge tu^i = \uparrow_{toi}^{ti}(tu^{oi})$$

En utilisant le jugement d'évaluation instrumenté et le théorème de correction de la sémantique instrumentée vis-à-vis de la sémantique sur-instrumentée, on déduit qu'il existe une t-valeur sur-instrumentée tu^{oi} vérifiant les deux propriétés suivantes :

$$\uparrow_{ti}^{toi}(t\Gamma^i) \vdash^{oi} e \mapsto tu^{oi}$$

$$tu^i = \uparrow_{toi}^{ti}(tu^{oi})$$

D'autre part, puisque le label l n'apparaît pas dans le t-environnement instrumenté $t\Gamma^i$, on en déduit que ce dernier est instanciable et l'environnement instancié est son environnement de référence. C'est-à-dire qu'il existe un environnement Γ tel que :

$$\Gamma = \downarrow^{l:v_l}(t\Gamma^{oi}) = \uparrow_{toi}(t\Gamma^{oi}) = \uparrow_{ti}(t\Gamma^i)$$

Puisque le label l n'apparaît pas non plus dans la t-valeur instrumentée tu^i, cette dernière est instanciable et la valeur de l'instanciation est sa valeur de référence. Il existe donc une valeur v telle que :

$$v = \downarrow^{l:v_l}(tu^{oi}) = \uparrow_{toi}(tu^{oi}) = \uparrow_{ti}(tu^i)$$

Rappel du théorème de correction de la sémantique sur-instrumentée vis-à-vis de la sémantique avec injection dans un t-environnement sur-instrumenté :

$$t\Gamma^{oi} \vdash^{oi} e \mapsto tu^{oi} \Rightarrow \forall(l, v_l).\exists\Gamma.\Gamma = \downarrow^{l:v_l}(t\Gamma^{oi}) \Rightarrow \forall v.v = \downarrow^{l:v_l}(tu^{oi}) \Rightarrow t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v$$

On utilise alors ce théorème pour obtenir :

$$t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v$$

Rappel du théorème de correction de la sémantique avec injection dans un t-environnement sur-instrumenté vis-à-vis de la sémantique opérationnelle avec injection :

$$\forall(t\Gamma^{oi}, \Gamma, e, v, l, v_l), t\Gamma^{oi} \vdash_{l:v_l} e \mapsto v \Rightarrow \Gamma = \downarrow^{l:v_l}(t\Gamma^{oi}) \Rightarrow \Gamma \vdash_{l:v_l} e \mapsto v$$

On utilise enfin ce théorème pour déduire :

$$\Gamma \vdash_{l:v_l} e \mapsto v$$

On peut alors conclure en utilisant les égalités ci-dessus concernant Γ et v :

$$\uparrow_{ti}(t\Gamma^i) \vdash_{l:v_l} e \mapsto \uparrow_{ti}(tu^i)$$

Chapitre 4

Analyse statique

4.1 Introduction

Ce chapitre a pour but de présenter une analyse statique des dépendances d'un programme et sa preuve de correction. Cette analyse permettra d'analyser tout programme écrit dans notre langage et de fournir une approximation raisonnable de ses dépendances. Elle permettra également d'analyser un programme sans en connaître précisément l'environnement d'évaluation. Il suffira de spécifier un environnement d'évaluation abstrait correspondant à un ensemble d'environnements d'évaluation possibles.

La présentation de cette analyse statique fait suite à la présentation dans le chapitre précédent d'une analyse dynamique. Là où l'analyse dynamique permettait d'analyser un programme uniquement dans un environnement d'évaluation de référence connu, l'analyse statique permet de spécifier de façon souple l'environnement d'évaluation.

Bien que la définition de l'analyse statique, que nous appelons ***sémantique abstraite***, soit indépendante de celle de la sémantique instrumentée ainsi que des autres sémantiques présentées, sa présentation fait suite à la présentation dans le chapitre précédent de l'analyse dynamique (sémantique instrumentée). En effet, dans le but de faciliter la preuve de correction de l'analyse statique, nous définissons la sémantique abstraite comme une interprétation abstraite de la sémantique instrumentée. De la même manière que dans le chapitre précédent, nous serons amené à introduire des sémantiques intermédiaires permettant une preuve par étapes, plus simple qu'une approche directe.

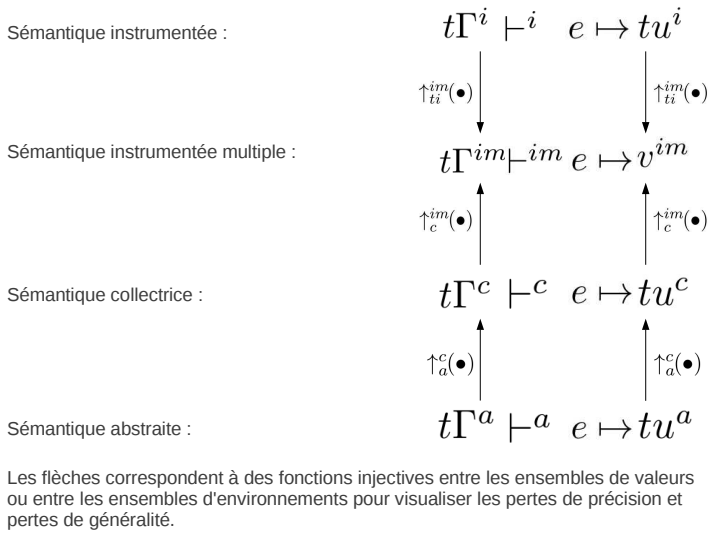

FIGURE 4.1 – Sémantiques intermédiaires pour l'analyse statique

La figure 4.1 présente une vue globale des différentes sémantiques intermédaires entre la sémantique instrumentée et la sémantique abstraite.

Ainsi, dans ce chapitre, nous utilisons la sémantique instrumentée comme une sémantique de base dont nous allons proposer une interprétation abstraite en plusieurs étapes. La preuve de correction de l'analyse statique sera alors constituée de l'enchaînement des preuves de correction des différentes sémantiques intermédiaires. La figure 4.2 propose une représentation graphique de l'enchaînement des différentes preuves de correction.

Tout d'abord, nous présenterons la sémantique instrumentée multiple qui permet d'analyser un programme simultanément pour un ensemble de t-environnements instrumentés possibles (cf. section 4.2). Ensuite, nous aborderons la présentation de la sémantique collectrice (cf. section 4.3), qui permet d'analyser un programme dans un environnement collecteur dans lequel chaque identificateur est lié à un ensemble de valeurs possibles. Enfin nous présenterons la définition et la preuve de correction de la sémantique abstraite (cf. section 4.4), dans laquelle les ensembles de valeurs présents dans la sémantique collectrice sont remplacés par des valeurs abstraites représentant certains ensembles de valeurs particuliers.

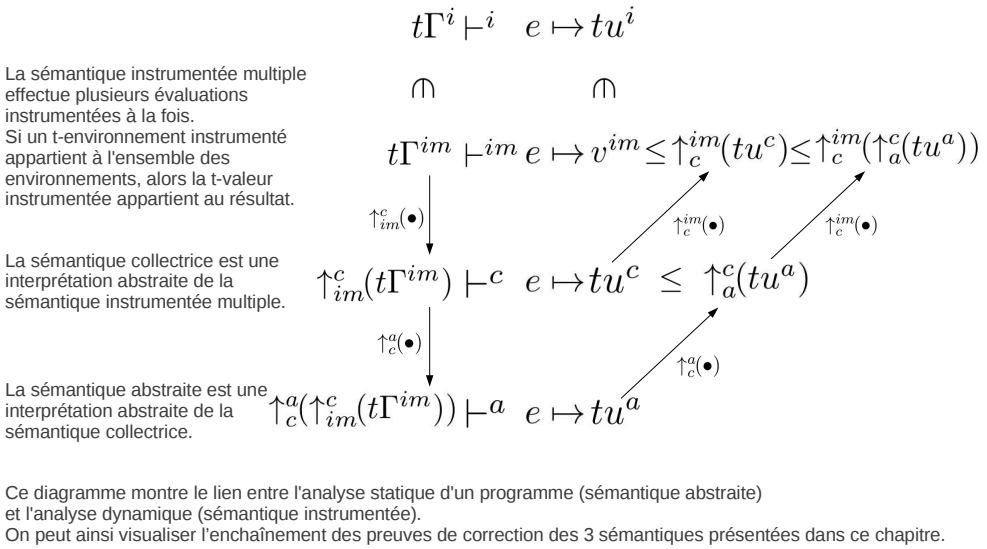

FIGURE 4.2 – Enchaînement des preuves des sémantiques intermédiaires pour l'analyse statique

4.2 Sémantique instrumentée multiple

La sémantique instrumentée multiple est la première sémantique intermédiaire permettant de faire le lien entre la sémantique instrumentée et la sémantique abstraite. Il s'agit d'un outil technique permettant de faciliter la preuve de correction de la sémantique collectrice qui sera présentée dans la section suivante.

4.2.1 Présentation informelle

Un des buts de notre analyse statique est de pouvoir obtenir en une unique analyse un résultat contenant une approximation des dépendances d'un programme pour un ensemble d'évaluations possibles. La sémantique instrumentée multiple franchit ce premier pas en regroupant dans un jugement d'évaluation unique un ensemble d'évaluations instrumentées possibles.

4.2.2 Définition formelle

La définition formelle de la sémantique instrumentée multiple ne comprend qu'une seule règle d'inférence. Celle-ci se base sur la sémantique instrumentée précédemment définie.

Le point de départ est un ensemble $t\Gamma^{im}$ d'environnements d'évaluations instrumentés possibles. Chacun de ces environnements instrumentés fournit ou non un jugement d'évaluation instrumenté pour le programme e aboutissant sur une certaine t-valeur instrumentée. Toutes les t-valeurs instrumentées tu^i pour lesquelles il existe un jugement d'évaluation instrumentée du programme e dans un t-environnement instrumenté présent dans $t\Gamma^{im}$ sont regroupées dans un ensemble de t-valeurs instrumentées v^{im} qui est le résultat du jugement d'évaluation instrumentée multiple.

Il est utile de remarquer que pour tout ensemble de t-environnements instrumentés et tout programme e, il existe un jugement d'évaluation instrumentée multiple de ce programme en un certain ensemble de t-valeurs instrumentées (éventuellement vide).

$$\frac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i e \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} e \mapsto v^{im}}\ \text{I-MULTIPLE}$$

4.3 Sémantique collectrice

La sémantique collectrice a pour but de faire apparaître le calcul des dépendances lors de l'analyse simultanée d'un ensemble d'évaluations possibles d'un programme. La sémantique instrumentée multiple permettait d'analyser simultanément un ensemble d'évaluations possibles d'un programme mais le calcul des dépendances se faisait séparément pour chacune des évaluations possibles. La sémantique collectrice évalue un programme dans un environnement collecteur et fournit comme résultat **une** valeur collectrice. Cette valeur collectrice comprend des t-dépendances ainsi que des v-dépendances globales (partagées par toutes les valeurs possibles) ainsi qu'un ensemble de valeurs possibles.

L'intérêt d'une telle sémantique est de séparer le calcul des dépendances du calcul des valeurs dans chaque environnement possible. Cette étape permettra par la suite de définir une abstraction du calcul des valeurs en gardant le calcul des dépendances défini ici.

4.3.1 Algèbre des valeurs

4.3.1.1 Valeurs

Une t-valeur collectrice (ou simplement « valeur collectrice » puisqu'ici il n'y a pas d'ambiguïté, notée tu^c) est composée d'un ensemble de t-dépendances td^c, d'un ensemble de v-dépendances d^c ainsi que d'un ensemble de valeurs simples instrumentées vs^i.

$$tu^c := \quad [\ td^c \ \mid \ d^c \ \mid \ vs^i\ ] \quad \text{Valeur collectrice}$$

Les annotations de dépendance td^c et d^c sont globales et chaque valeur simple instrumentée présente dans vs^i contient des annotations de dépendance concernant ses propres sous-termes.

Comme dans toutes les autres sémantiques présentées, les ensembles de t-dépendances ainsi que les ensembles de v-dépendances sont des ensembles nécessairement finis. Par contre, l'ensemble des valeurs simples instrumentées contenu dans une valeur collectrice est un ensemble potentiellement infini.

4.3.1.2 Ensembles de dépendances

Dans les valeurs collectrices, les ensembles de t-dépendances (resp. de v-dépendances) sont simplement des ensembles de labels, comme dans la sémantique instrumentée.

$$\begin{array}{rll} td^c := & \{l_1; \ldots; l_n\} & t\text{-dépendances} \\ d^c := & \{l_1; \ldots; l_m\} & v\text{-dépendances} \end{array}$$

4.3.1.3 Environnements

Contrairement à la sémantique instrumentée, il n'y a qu'un seul type d'environnement. La représentation des fonctions sous forme de valeur collectrice est constituée d'un ensemble de fermetures instrumentées, contenant chacune un v-environnement instrumenté. Il n'y a donc pas besoin d'un type d'environnement collecteur spécial pour la représentation des fonctions.

Un t-environnement collecteur (ou simplement « environnement collecteur » puisqu'ici il n'y a pas d'ambiguïté) permet de lier chaque identificateur à une t-valeur collectrice.

$$t\Gamma^c := \quad (x_1, tu_1^c); \ldots; (x_n, tu_n^c) \quad t\text{-environnement collecteur}$$

4.3.2 Règles d'inférence

La sémantique collectrice évalue une expression dans un environnement collecteur et fournit une valeur collectrice. Tout comme la sémantique instrumentée multiple, elle fournit toujours un résultat (pour toute expression et tout environnement collecteur, il existe une valeur collectrice). La valeur collectrice peut éventuellement contenir un ensemble vide de valeurs simples, ce qui correspond à une erreur d'évaluation. Le jugement d'évaluation prend la forme suivante :

$$t\Gamma^c \vdash^c e \mapsto tu^c$$

C-NUM La règle d'évaluation d'une constante numérique n'introduit aucune dépendance. Les t-dépendances ainsi que les v-dépendances sont donc vides. L'ensemble des valeurs simples instrumentées est réduit à un singleton contenant uniquement la constante numérique correspondant à l'expression.

$$\textsc{c-num}\ \frac{}{t\Gamma^c \vdash^c n \mapsto [\ \emptyset\ |\ \emptyset\ |\ \{n\}\ ]} \qquad \textsc{c-ident}\ \frac{tu^c = t\Gamma^c[x]}{t\Gamma^c \vdash^c x \mapsto tu^c} \qquad \textsc{c-ident-empty}\ \frac{x \notin support(t\Gamma^c)}{t\Gamma^c \vdash^c x \mapsto [\ \emptyset\ |\ \emptyset\ |\ \emptyset\ ]}$$

$$\textsc{c-abstr}\ \frac{vs^i = \{< \lambda x.e, \Gamma^i > \ |\ \Gamma^i \in \uparrow_c^{im}(t\Gamma^c)\}}{t\Gamma^c \vdash^c \lambda x.e \mapsto [\ \emptyset\ |\ \emptyset\ |\ vs^i\ ]} \qquad \textsc{c-abstr-rec}\ \frac{vs^i = \{< \texttt{rec} f.x.e, \Gamma^i > \ |\ \Gamma^i \in \uparrow_c^{im}(t\Gamma^c)\}}{t\Gamma^c \vdash^c \texttt{rec} f.x.e \mapsto [\ \emptyset\ |\ \emptyset\ |\ vs^i\ ]}$$

$$\textsc{c-apply}\ \frac{\begin{array}{c} t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ |\ d_1^c\ |\ vs_1^i\ ] \\ t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ |\ d_2^c\ |\ vs_2^i\ ] \qquad v^{im} = multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i) \\ (\forall l.\ l \in td^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in td^i \wedge [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ] \in v^{im})) \\ (\forall l.\ l \in d^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in d^i \wedge [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ] \in v^{im})) \\ vs^i = \{v^i \mid \exists (td^i, d^i).\ [\ td^i\ |\ [\ d^i\ |\ v^i\ ]\ ] \in v^{im}\} \end{array}}{t\Gamma^c \vdash^c e_1\ e_2 \mapsto [\ td^c\ |\ d^c\ |\ vs^i\ ]}$$

$$\textsc{c-if-true}\ \frac{t\Gamma^c \vdash^c e \mapsto [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad true \in vs^i \qquad false \notin vs^i \qquad t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ |\ d_1^c\ |\ vs_1^i\ ] \qquad tu'^c = [\ d^c \cup td^c \cup td_1^c\ |\ d^c \cup d_1^c\ |\ vs_1^i\ ]}{t\Gamma^c \vdash^c if\ e\ then\ e_1\ else\ e_2 \mapsto tu'^c}$$

$$\textsc{c-if-false}\ \frac{t\Gamma^c \vdash^c e \mapsto [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad false \in vs^i \qquad true \notin vs^i \qquad t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ |\ d_2^c\ |\ vs_2^i\ ] \qquad tu'^c = [\ d^c \cup td^c \cup td_2^c\ |\ d^c \cup d_2^c\ |\ vs_2^i\ ]}{t\Gamma^c \vdash^c if\ e\ then\ e_1\ else\ e_2 \mapsto tu'^c}$$

$$\textsc{c-if-unknown}\ \frac{\begin{array}{c} t\Gamma^c \vdash^c e \mapsto [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad true \in vs^i \qquad false \in vs^i \qquad t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ |\ d_1^c\ |\ vs_1^i\ ] \\ t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ |\ d_2^c\ |\ vs_2^i\ ] \qquad tu'^c = [\ d^c \cup td^c \cup td_1^c \cup td_2^c\ |\ d^c \cup d_1^c \cup d_2^c\ |\ vs_1^i \cup vs_2^i\ ] \end{array}}{t\Gamma^c \vdash^c if\ e\ then\ e_1\ else\ e_2 \mapsto tu'^c}$$

$$\textsc{c-if-empty}\ \frac{t\Gamma^c \vdash^c e \mapsto [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad true \notin vs^i \qquad false \notin vs^i}{t\Gamma^c \vdash^c if\ e\ then\ e_1\ else\ e_2 \mapsto [\ \emptyset\ |\ \emptyset\ |\ \emptyset\ ]}$$

$$\textsc{c-match}\ \frac{\begin{array}{c} t\Gamma^c \vdash^c e \mapsto tu^c \\ tu^c = [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad vs^i \cap vs^i_{matchable} \neq \emptyset \qquad tu^c, p \vdash^c_p t\Gamma^c_p \qquad t\Gamma^c_p \oplus t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ |\ d_1^c\ |\ vs_1^i\ ] \end{array}}{t\Gamma^c \vdash^c match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^c \cup td^c \cup td_1^c\ |\ d^c \cup d_1^c\ |\ vs_1^i\ ]}$$

$$\textsc{c-match-var}\ \frac{\begin{array}{c} t\Gamma^c \vdash^c e \mapsto tu^c \qquad tu^c = [\ td^c\ |\ d^c\ |\ vs^i\ ] \\ vs^i \cap vs^i_{matchable} \neq \emptyset \qquad tu^c, p \vdash^c_p \bot \qquad (x, tu^c) \oplus t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ |\ d_2^c\ |\ vs_2^i\ ] \end{array}}{t\Gamma^c \vdash^c match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^c \cup td^c \cup td_2^c\ |\ d^c \cup d_2^c\ |\ vs_2^i\ ]}$$

$$\textsc{c-match-unknown}\ \frac{\begin{array}{c} t\Gamma^c \vdash^c e \mapsto tu^c \qquad tu^c = [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad vs^i \cap vs^i_{matchable} \neq \emptyset \\ tu^c, p \vdash^c_p ?\ t\Gamma^c_p \qquad t\Gamma^c_p \oplus t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ |\ d_1^c\ |\ vs_1^i\ ] \qquad (x, tu^c) \oplus t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ |\ d_2^c\ |\ vs_2^i\ ] \end{array}}{t\Gamma^c \vdash^c match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^c \cup td^c \cup td_1^c \cup td_2^c\ |\ d^c \cup d_1^c \cup d_2^c\ |\ vs_1^i \cup vs_2^i\ ]}$$

$$\textsc{c-match-empty}\ \frac{t\Gamma^c \vdash^c e \mapsto tu^c \qquad tu^c = [\ td^c\ |\ d^c\ |\ vs^i\ ] \qquad vs^i \cap vs^i_{matchable} = \emptyset}{t\Gamma^c \vdash^c match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ \emptyset\ |\ \emptyset\ |\ \emptyset\ ]}$$

FIGURE 4.3 – Sémantique collectrice : première partie

$$\textsc{c-constr-0}\ \frac{}{t\Gamma^c \vdash^c C \mapsto [\ \emptyset\ \mid\ \emptyset\ \mid\ \{C\}\ ]} \qquad \textsc{c-constr-1}\ \frac{t\Gamma^c \vdash^c e \mapsto [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]}{t\Gamma^c \vdash^c D(e) \mapsto [\ td^c\ \mid\ \emptyset\ \mid\ \{D([\ d^c\ \mid\ v^i\ ]) \mid v^i \in vs^i\}\ ]}$$

$$\textsc{c-couple}\ \frac{t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ] \qquad t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ]}{t\Gamma^c \vdash^c (e_1, e_2) \mapsto [\ td_1^c \cup td_2^c\ \mid\ \emptyset\ \mid\ \{([\ d_1^c\ \mid\ v_1^i\ ], [\ d_2^c\ \mid\ v_2^i\ ]) \mid v_1^i \in vs_1^i \wedge v_2^i \in vs_2^i\}\ ]}$$

$$\textsc{c-letin}\ \frac{t\Gamma^c \vdash^c e_1 \mapsto tu_1^c \qquad tu_1^c = [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ] \qquad vs_1^i \neq \emptyset \qquad (x, tu_1^c) \oplus t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ]}{t\Gamma^c \vdash^c let\ x\ =\ e_1\ in\ e_2 \mapsto [\ td_1^c \cup td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ]}$$

$$\textsc{c-letin-empty}\ \frac{t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c\ \mid\ d_1^c\ \mid\ \emptyset\ ]}{t\Gamma^c \vdash^c let\ x\ =\ e_1\ in\ e_2 \mapsto [\ \emptyset\ \mid\ \emptyset\ \mid\ \emptyset\ ]} \qquad \textsc{c-annot}\ \frac{t\Gamma^c \vdash^c e \mapsto [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]}{t\Gamma^c \vdash^c l\ :\ e \mapsto [\ td^c\ \mid\ \{l\} \cup d^c\ \mid\ vs^i\ ]}$$

FIGURE 4.4 – Sémantique collectrice : seconde partie

$$vs^i_{matchable} = \{C | \forall C \in Constr^0\} \cup \{D(u^i) | \forall D \in Constr^1, \forall u^i\} \cup \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\}$$

$$\textsc{cm-constr-0}\ \frac{vs^i \cap vs^i_{matchable} \subseteq \{C | \forall C \in Constr^0\}}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], C \vdash^c_p \{\}}$$

$$\textsc{cm-constr-1}\ \frac{\begin{array}{c} vs^i \cap vs^i_{matchable} \subseteq \{D(u^i) | \forall D \in Constr^1, \forall u^i\} \\ vs'^i = \{v^i | \exists d^i.D([\ d^i\ \mid\ v^i\ ]) \in vs^i\} \qquad \forall l.l \in d'^c \Leftrightarrow \exists d^i.l \in d^i \wedge \exists v^i.D([\ d^i\ \mid\ v^i\ ]) \in vs^i \end{array}}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], D(x) \vdash^c_p \{(x, [\ \emptyset\ \mid\ d'^c\ \mid\ vs'^i\ ])\}}$$

$$\textsc{cm-couple}\ \frac{\begin{array}{c} vs^i \cap vs^i_{matchable} \subseteq \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} \qquad vs_1^i = \{v_1^i | \exists (d_1^i, d_2^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i\} \\ \forall l.l \in d_1^c \Leftrightarrow \exists d_1^i.l \in d_1^i \wedge \exists (v_1^i, d_2^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \\ vs_2^i = \{v_2^i | \exists (d_1^i, v_1^i, d_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i\} \\ \forall l.l \in d_2^c \Leftrightarrow \exists d_2^i.l \in d_2^i \wedge \exists (d_1^i, v_1^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \end{array}}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], (x_1, x_2) \vdash^c_p \{(x_1, [\ \emptyset\ \mid\ d_1^c\ \mid\ vs_1^i\ ]); (x_2, [\ \emptyset\ \mid\ d_2^c\ \mid\ vs_2^i\ ])\}}$$

$$\textsc{cm-constr-0-not}\ \frac{vs^i \cap \{C | \forall C \in Constr^0\} = \emptyset}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], C \vdash^c_p \bot} \qquad \textsc{cm-constr-1-not}\ \frac{vs^i \cap \{D(u^i) | \forall D \in Constr^1, \forall u^i\} = \emptyset}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], D(x) \vdash^c_p \bot}$$

$$\textsc{cm-couple-not}\ \frac{vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} = \emptyset}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], (x_1, x_2) \vdash^c_p \bot}$$

$$\textsc{cm-constr-0-unknown}\ \frac{vs^i \cap vs^i_{matchable} \not\subseteq \{C | \forall C \in Constr^0\} \qquad vs^i \cap \{C | \forall C \in Constr^0\} \neq \emptyset}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], C \vdash^c_p ?\ \{\}}$$

$$\textsc{cm-constr-1-unknown}\ \frac{\begin{array}{c} vs^i \cap vs^i_{matchable} \not\subseteq \{D(u^i) | \forall D \in Constr^1, \forall u^i\} \\ vs^i \cap \{D(u^i) | \forall D \in Constr^1, \forall u^i\} \neq \emptyset \qquad vs'^i = \{v^i | \exists d^i.D([\ d^i\ \mid\ v^i\ ]) \in vs^i \cap \{D(u^i) | \forall D \in Constr^1, \forall u^i\}\} \\ \forall l.l \in d'^c \Leftrightarrow \exists d^i.l \in d^i \wedge \exists v^i.D([\ d^i\ \mid\ v^i\ ]) \in vs^i \cap \{D(u^i) | \forall D \in Constr^1, \forall u^i\} \end{array}}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], D(x) \vdash^c_p ?\ \{(x, [\ \emptyset\ \mid\ d'^c\ \mid\ vs'^i\ ])\}}$$

$$\textsc{cm-couple-unknown}\ \frac{\begin{array}{c} vs^i \cap vs^i_{matchable} \not\subseteq \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} \\ vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} \neq \emptyset \qquad vs_1^i = \{v_1^i | \exists (d_1^i, d_2^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\}\} \\ \forall l.l \in d_1^c \Leftrightarrow \exists d_1^i.l \in d_1^i \wedge \exists (v_1^i, d_2^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} \\ vs_2^i = \{v_2^i | \exists (d_1^i, v_1^i, d_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\}\} \\ \forall l.l \in d_2^c \Leftrightarrow \exists d_2^i.l \in d_2^i \wedge \exists (d_1^i, v_1^i, v_2^i).([\ d_1^i\ \mid\ v_1^i\ ], [\ d_2^i\ \mid\ v_2^i\ ]) \in vs^i \cap \{(u_1^i, u_2^i) | \forall u_1^i, \forall u_2^i\} \end{array}}{[\ td^c\ \mid\ d^c\ \mid\ vs^i\ ], (x_1, x_2) \vdash^c_p ?\ \{(x_1, [\ \emptyset\ \mid\ d_1^c\ \mid\ vs_1^i\ ]); (x_2, [\ \emptyset\ \mid\ d_2^c\ \mid\ vs_2^i\ ])\}}$$

FIGURE 4.5 – Sémantique collectrice : règles de filtrage

C-IDENT La règle d'évaluation d'un identificateur est semblable à celle des autres sémantiques. Elle retourne la valeur associée à cet identificateur dans l'environnement.

C-IDENT-EMPTY Une nouvelle règle est introduite pour donner une valeur à un identificateur dans le cas où il n'apparaît pas dans l'environnement. Dans ce cas, l'ensemble des valeurs simples est vide pour signifier qu'il n'existe aucune évaluation instrumentée correspondante. Cette règle est nécessaire pour que la sémantique collectrice soit totale, c'est-à-dire qu'elle fournisse une valeur pour n'importe quelle expression dans n'importe quel environnement.

C-ABSTR L'évaluation d'une fonction ne produit aucune dépendance, comme dans les sémantiques précédentes. En ce qui concerne l'ensemble des valeurs simples instrumentées, on remarquera qu'il contient un ensemble de fermetures. Puisque l'environnement collecteur correspond à un ensemble d'environnements instrumentés, on construit une fermeture pour chaque environnement instrumenté possible. Pour cela, on utilise la fonction $\uparrow_c^{im}(\bullet)$ qui transforme un t-environnement collecteur en l'ensemble des environnements instrumentés lui correspondant. Cette fonction est définie en figure 4.8.

C-ABSTR-REC Selon le même principe que pour la règle C-ABSTR, on construit une fermeture récursive pour chaque environnement instrumenté possible.

C-APPLY La règle de l'application, est plus complexe car il faut construire l'ensemble des valeurs possibles pour l'application à partir de l'ensemble des valeurs possibles pour e_1 et e_2.

Tout d'abord, on évalue e_1 et e_2 dans l'environnement collecteur pour obtenir leur valeur collectrice. On construit ensuite l'ensemble des t-valeurs instrumentées de l'application en utilisant la fonction *multiple_instrumented_application* avec les valeurs collectrices de e_1 et e_2. À partir du résultat de cette fonction, on extrait alors les t-dépendances, les v-dépendances ainsi que l'ensemble des valeurs simples instrumentées de l'application.

La fonction *multiple_instrumented_application* a pour but de définir l'ensemble des

$$
\begin{array}{l}
multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i) = \\
\quad \{\ tu^i \mid \quad (\ \exists < \lambda x.e, \Gamma_1^i > \in vs_1^i. \\
\qquad \exists v_2^i \in vs_2^i. \\
\qquad \exists (td^i, t^i, v^i). \\
\qquad \exists tu_2^i = [\ td_2^c \mid [\ d_2^c \mid v_2^i\ ]\ ]. \\
\qquad (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i \mid [\ d^i \mid v^i\ ]\ ] \\
\qquad \wedge\ tu^i = [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c \mid [\ d_1^c \cup d^i \mid v^i\ ]\ ]\ ) \\
\quad \vee\ (\ \exists < \mathtt{rec} f.x.e, \Gamma_1^i > \in vs_1^i. \\
\qquad \exists v_2^i \in vs_2^i. \\
\qquad \exists (td^i, t^i, v^i). \\
\qquad \exists tu_f^i = [\ td_1^c \mid [\ d_1^c \mid < \mathtt{rec} f.x.e, \Gamma_1^i >\ ]\ ]. \\
\qquad \exists tu_2^i = [\ td_2^c \mid [\ d_2^c \mid v_2^i\ ]\ ]. \\
\qquad (f, tu_f^i) \oplus (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i \mid [\ d^i \mid v^i\ ]\ ] \\
\qquad \wedge\ tu^i = [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c \mid [\ d_1^c \cup d^i \mid v^i\ ]\ ]\ )\ \}
\end{array}
$$

FIGURE 4.6 – Sémantique collectrice : prédicat d'application instrumentée multiple

t-valeurs instrumentées d'une application de fonction à partir de la valeur collectrice de la fonction et de la valeur collectrice de l'argument. Cet ensemble est défini par un prédicat d'appartenance. Une t-valeur instrumentée appartient à cet ensemble si il existe une fermeture (récursive ou non) et une valeur argument telles que l'application de cette fermeture à cette valeur par la sémantique instrumentée fournit la t-valeur considérée. La définition formelle de cette fonction est donnée en figure 4.6.

Expressions conditionnelles

Pour l'évaluation collectrice d'une expresssion conditionnelle, on distingue 4 cas :

- pour tout environnement possible, seule la première branche est évaluée
- pour tout environnement possible, seule la seconde branche est évaluée
- selon l'environnement possible, l'une ou l'autre des branches est évaluée
- il n'y a aucune évaluation possible

C-IF-TRUE Le premier cas est caractérisé par l'appartenance de la valeur *true* à l'ensemble des valeurs possible de la condition (ce qui indique qu'il existe une évaluation possible passant par la première branche) et la non-appartenance de la valeur $false$ (ce qui indique qu'aucune évaluation possible ne passe par la seconde branche). Dans ce cas, le calcul des dépendances s'effectue de manière similaire à la sémantique instrumentée.

C-IF-FALSE Le second cas est caractérisé de la même manière, par l'appartenance de la valeur $false$ et la non-appartenance de la valeur $true$ à l'ensemble des valeurs possibles de la condition. Dans ce cas également, le calcul des dépendances s'effectue de manière similaire à la sémantique instrumentée.

C-IF-UNKNOWN Le cas incertain, où l'on peut passer parfois par la première branche et parfois par la seconde branche est caractérisé par l'appartenance des deux valeurs $true$ et $false$ à l'ensemble des valeurs possibles de la condition. Dans ce cas, les dépendances de la première branche ainsi que celles de la seconde branche se retrouvent toutes les deux dans les dépendances du résultat. C'est une des sources d'approximation dues à l'incertitude sur les valeurs de l'environnement.

C-IF-EMPTY Le dernier cas de l'évaluation d'une expression conditionnelle correspond au cas d'erreur. Le programme analysé est incorrect et son analyse par notre calcul de dépendances n'a pas d'intérêt.

Pattern-matching

Pour l'évaluation collectrice d'un pattern-matching, on distingue 4 cas, homologues aux 4 cas concernant une expression conditionnelle :

- pour tout environnement possible, seule la première branche est évaluée
- pour tout environnement possible, seule la seconde branche est évaluée
- selon l'environnement possible, l'une ou l'autre des branches est évaluée
- il n'y a aucune évaluation possible

Cette distinction en 4 cas suit les sémantiques opérationnelles et instrumentées présentées plus haut dans lesquelles seules les valeurs « filtrables » aboutissent à une évaluation. Par exemple, si l'expression filtrée s'évalue en une fermeture ou un nombre, alors aucune des deux branches du filtrage n'est évaluée, le résultat est donc une erreur lors de l'évaluation. Il se peut que cette sémantique « théorique » ne corresponde pas à l'implémentation d'un compilateur en pratique. Ce point est discuté plus en détail en section 2.3.4.

La discrimination entre les 4 cas se fait tout d'abord en vérifiant si la valeur collectrice contient au moins une valeur filtrable (cf. définition de $vs^i_{matchable}$ en figure 4.5). Si c'est le cas, on distingue alors trois possibilités à l'aide d'un prédicat de filtrage défini en figure 4.5. Ce prédicat est noté $tu^c, p \vdash^c_p result$ où tu^c est la valeur collectrice filtrée, p est le filtre et $result$ peut prendre trois formes différentes. Il s'agit soit d'un environnement collecteur (si la valeur est toujours filtrée), soit du symbole $\perp$ (si la valeur n'est jamais filtrée), soit du symbole ? accompagné d'un environnement collecteur (si la valeur collectrice contient à la fois des valeurs filtrées et des valeurs non filtrées).

C-MATCH Le cas où seule la première branche peut être évaluée est caractérisé par deux conditions. Premièrement, il doit exister des valeurs « filtrables » dans l'ensemble des valeurs possibles de l'expression filtrée (sinon, aucune branche n'est jamais évaluée). Deuxièmement, la valeur collectrice de l'expression filtrée est filtrée par le motif (ce qui signifie que toutes les valeurs instrumentées « filtrables » correspondantes sont filtrées par ce motif). L'environnement collecteur obtenu par le filtrage contient les liaisons des éventuels identificateurs contenus dans le motif. Chaque identificateur est lié à l'ensemble des valeurs filtrées par le motif.

C-MATCH-VAR Le cas où seule la seconde branche peut être évaluée est aussi caractérisé par la non-vacuité de l'ensemble des valeurs « filtrables » de l'expression filtrée ainsi que par une seconde condition. La seconde condition impose qu'aucune valeur « filtrable » de l'expression filtrée ne correspond au motif.

C-MATCH-UNKNOWN Comme pour l'évaluation d'une expression conditionnelle, il existe un cas incertain où on ne peut pas garantir que l'évaluation passera toujours pas la même branche. On caractérise ce cas par la condition de non-vacuité de l'ensemble des valeurs « filtrables » de l'expression filtrée et une seconde condition qui impose qu'il existe parmi les valeurs « filtrables » de l'expression filtrée certaines qui correspondent au motif et d'autres qui n'y correspondent pas. Dans ce cas, les dépendances des deux branches se retrouvent dans les dépendances du résultat.

C-MATCH-EMPTY S'il n'existe aucune valeur « filtrable » pour l'expression filtrée alors on est en présence d'un programme erroné.

C-CONSTR-0 L'évaluation collectrice d'un constructeur constant est similaire à l'évaluation d'une constante numérique. Les t-dépendances ainsi que les v-dépendances sont donc vides et l'ensemble des valeurs simples instrumentées est réduit à un singleton contenant uniquement le constructeur.

C-CONSTR-1 L'évaluation d'un constructeur paramétré n'ajoute pas de dépendances. Les t-dépendances sont exactement les t-dépendances de la sous-expression et l'ensemble des v-dépendances est vide. Pour obtenir l'ensembles des valeurs simples instrumentées, on enrobe avec le constructeur chacune des valeurs simples instrumentées de la sous-expression.

C-COUPLE L'évaluation d'un couple suit le même principe que celle d'un constructeur paramétré sauf qu'il y a deux sous-expressions.

C-LETIN La règle d'évaluation d'une liaison est habituelle. On ajoute simplement comme condition qu'il existe au moins une valeur possible pour e_1. Cette condition n'est pas absolument nécessaire mais permet à l'analyse de fournir un résultat plus précis.

C-LETIN-EMPTY Dans le cas où il n'existe aucune valeur pour e_1, il n'existe pas non plus de valeur pour l'expression globale.

C-ANNOT La règle d'évaluation d'une expression annotée est la même que dans la sémantique instrumentée. On ajoute simplement le label à l'ensemble des v-dépendances de l'expression.

4.3.3 Correction

4.3.3.1 Énoncé informel du théorème

On veut montrer que la sémantique collectrice est une interprétation abstraite de la sémantique instrumentée multiple. C'est-à-dire qu'une évaluation par la sémantique

instrumentée multiple peut être simulée par une évaluation collectrice et que la valeur ainsi obtenue sera moins précise que celle obtenue par une évaluation directe. Cette simulation nécessite une conversion de l'environnement et de la valeur retournée définies formellement à l'aide d'une fonction d'abstraction notée $\uparrow^{c}_{im}(\bullet)$ (cf. figure 4.7) et d'une fonction de concrétisation notée $\uparrow^{im}_{c}(\bullet)$ (cf. figure 4.8).

Le lien entre l'évaluation directe par la sémantique instrumentée multiple et l'évaluation passant par la sémantique collectrice et les fonctions de conversion, est illustré en figure 4.2.

4.3.3.2 Illustration par l'exemple

Pour illustrer le fonctionnement de la sémantique collectrice ainsi que la signification du théorème de correction que nous venons d'énoncer informellement, examinons un exemple simple. Nous nous limitons à un exemple très simple car les jugements d'évaluation deviennent vite volumineux. Rappelons que la sémantique instrumentée multiple ainsi que la sémantique collectrice n'ont pas pour but d'être utilisées en pratique. Leur rôle est de simplifier et de rendre modulaire la preuve de correction de l'analyse statique qui, elle, est utilisable en pratique.

Exemple 1 : évaluation d'un couple

Notons e_1 le programme suivant : (x, l_1:7)

Nous nous intéressons à l'évaluation instrumentée multiple de ce programme dans un environnement instrumenté multiple $t\Gamma^{im}_1$ contenant deux t-environnements instrumentés.

$$t\Gamma^{im}_1 = \{(x,\ [\ \emptyset\ \mid\ \ [\ \{l_2\}\ \mid\ C_1\ ]\ ]); (x,\ [\ \emptyset\ \mid\ \ [\ \{l_3\}\ \mid\ C_2\ ]\ ])\}$$

En évaluant le programme e_1 par la sémantique instrumentée multiple, dans l'environnement instrumenté multiple $t\Gamma^{im}_1$, on obtient le jugement suivant :

$$\begin{array}{l} t\Gamma^{im}_1 \vdash^{im} e_1 \mapsto \{\quad [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_2\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ]; \\ \qquad\qquad\qquad\quad [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ]\quad \} \end{array}$$

Nous allons maintenant nous intéresser à l'évaluation du même programme, dans le même environnement, mais en utilisant l'interprétation abstraite fournie par la sémantique collectrice.

Commençons par abstraire l'environnement instrumenté multiple $t\Gamma_1^{im}$ en un environnement collecteur que nous noterons $t\Gamma_1^c$:

$$\uparrow_{im}^{c}(t\Gamma_1^{im}) = (x,\ [\ \emptyset\ \mid\ \{l_2; l_3\}\ \mid\ \{C_1; C_2\}\ ]) = t\Gamma_1^c$$

Nous procédons alors à l'évaluation de e_1 dans l'environnement collecteur obtenu après abstraction :

$$\begin{array}{l} t\Gamma_1^c \vdash^c e_1 \mapsto\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \{\ \ ([\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ]);\\ \qquad\qquad\qquad\qquad\qquad\quad ([\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\quad \}\ ] \end{array}$$

Nous obtenons enfin la valeur instrumentée multiple correspondante en utilisant la fonction de concrétisation :

$$\begin{array}{rl} \uparrow_c^{im}([\ \emptyset\ \mid\ \emptyset\ \mid\ \{ & ([\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ]);\\ & ([\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ]) \qquad\qquad \}\ ])\\ = \{ & [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ];\\ & [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ] \quad \} \end{array}$$

Nous constatons alors que la valeur instrumentée multiple obtenue directement est plus précise que celle obtenue par l'interprétation abstraite. En effet, cette dernière contient des dépendances supplémentaires. Formellement, nous avons la relation suivante :

$$\begin{array}{rl} & \{\ \ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_2\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ];\\ & \quad [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ] \quad \}\\ \subseteq^{im} & \{\ \ [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ];\\ & \quad [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ ([\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ] \quad \} \end{array}$$

4.3.3.3 Énoncé formel du théorème

Avant d'énoncer formellement le théorème de correction de la sémantique collectrice, il nous faut définir formellement quelques notions qui serviront à exprimer le théorème.

Tout d'abord, nous définissons une fonction d'abstraction (cf. figure 4.7) ainsi qu'une fonction de concrétisation (cf. figure 4.8). La fonction d'abstraction (notée $\uparrow_{im}^{c}(\bullet)$) permet de transformer un ensemble de t-environnements instrumentés en un environnement

$$\frac{t\Gamma^{im} = \{\{\}\}}{\uparrow^c_{im}(t\Gamma^{im}) = \{\}}\ \text{ABSTR-IM-ENV-EMPTY}$$

$$\frac{t\Gamma^{im} = \{(x, tu^i) \oplus t\Gamma^i \mid \forall tu^i \in v^{im}.\forall t\Gamma^i \in t\Gamma^c\} \qquad \uparrow^c_{im}(v^{im}) = tu^c}{\uparrow^c_{im}(t\Gamma^{im}) = (x, tu^c) \oplus \uparrow^c_{im}(t\Gamma^c)}\ \text{ABSTR-IM-ENV-CONS}$$

$$\frac{\begin{array}{c}\forall l.l \in td^c \Leftrightarrow \exists td^i.l \in td^i \wedge \exists u^i.\ [\ td^i\ \mid\ u^i\ ] \in v^{im} \\ \forall l.l \in d^c \Leftrightarrow \exists d^i.l \in d^i \wedge \exists (td^i, v^i).\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v^{im} \\ vs^i = \{v^i \mid \exists (td^i, d^i).\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v^{im}\}\end{array}}{\uparrow^c_{im}(v^{im}) = [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]}\ \text{ABSTR-IM-VAL}$$

FIGURE 4.7 – Sémantique collectrice : Définition de la fonction d'abstraction

collecteur. La règle d'inférence ABSTR-IM-ENV-CONS se lit ainsi : pour abstraire un environnement $t\Gamma^{im}$ qui est de la forme $\{(x, tu^i) \oplus t\Gamma^i \mid \forall tu^i \in v^{im}.\forall t\Gamma^i \in t\Gamma^c\}$ pour une certaine valeur instrumentée multiple v^{im} et un certain environnement collecteur $t\Gamma^c$, on construit une valeur collectrice tu^c pour x à partir de v^{im} et on ajoute cette liaison à l'abstraction de $t\Gamma^c$. La fonction de concrétisation (notée $\uparrow^{im}_c(\bullet)$) permet quant à elle de transformer une valeur collectrice en un ensemble de t-valeurs instrumentées. Nous pouvons alors composer ces fonctions avec la sémantique collectrice pour établir une simulation de la sémantique instrumentée multiple. Pour cela, nous commençons par abstraire l'environnement d'évaluation, puis nous évaluons le programme considéré à l'aide de la sémantique collectrice et enfin nous concrétisons la valeur collectrice.

Il nous faut ensuite pouvoir comparer la valeur instrumentée multiple obtenue directement à l'aide de la sémantique instrumentée multiple et celle obtenue en passant par la sémantique collectrice. Pour cela, nous définissons une relation d'ordre (cf. figure 4.9 et 4.10) sur les valeurs instrumentées multiples. Nous exprimerons alors dans le théorème de correction que la valeur instrumentée multiple obtenue directement à l'aide de la sémantique instrumentée multiple est plus petite (ie. plus précise) que celle obtenue en passant par la sémantique collectrice.

Le théorème de correction de la sémantique collectrice s'exprime alors ainsi :

Théorème 4.3.1 (Correction de la sémantique collectrice)**.**
$\forall (t\Gamma^{im}, e, v^{im}, tu^c, t\Gamma^c).$

$$\uparrow_c^{im}(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]) := \ \{\ [\ td^c\ \mid\ [\ d^c\ \mid\ v^i\ ]\ ]\ |\ v^i \in vs^i\}$$

$$\uparrow_c^{im}(t\Gamma^c) := \ append_c^{im}(t\Gamma^c, \{\})$$

$$\frac{t\Gamma^c = \{\}}{append_c^{im}(t\Gamma^c, t\Gamma^{im}) = t\Gamma^{im}}\ \text{APPEND-C-IM-EMPTY}$$

$$\frac{t\Gamma^c = (x,\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]) \oplus t\Gamma'^c \qquad t\Gamma'^{im} = append_c^{im}(t\Gamma'^c, t\Gamma^{im})}{append_c^{im}(t\Gamma^c, t\Gamma^{im}) = \{(x,\ [\ td^c\ \mid\ [\ d^c\ \mid\ v^i\ ]\ ]) \oplus t\Gamma^i\ |\ v^i \in vs^i, t\Gamma^i \in t\Gamma'^{im}\}}\ \text{APPEND-C-IM-CONS}$$

FIGURE 4.8 – Sémantique collectrice : Définition de la fonction de concrétisation

$$\frac{\forall l \in td_1^i.\ l \in td_2^i}{td_1^i \subseteq td_2^i}\ \text{LE-TD} \qquad \frac{\forall l \in d_1^i.\ l \in d_2^i}{d_1^i \subseteq d_2^i}\ \text{LE-D}$$

$$\frac{td_1^i \subseteq td_2^i \qquad u_1^i \subseteq^i u_2^i}{[\ td_1^i\ \mid\ u_1^i\ ] \subseteq^i\ [\ td_2^i\ \mid\ u_2^i\ ]}\ \text{LE-TI-VAL} \qquad \frac{d_1^i \subseteq d_2^i \qquad v_1^i \subseteq^i v_2^i}{[\ d_1^i\ \mid\ v_1^i\ ] \subseteq^i\ [\ d_2^i\ \mid\ v_2^i\ ]}\ \text{LE-I-VAL}$$

$$\frac{}{n \subseteq^i n}\ \text{LE-I-VAL-NUM} \qquad \frac{}{b \subseteq^i b}\ \text{LE-I-VAL-BOOL} \qquad \frac{}{C \subseteq^i C}\ \text{LE-I-VAL-CONSTR0} \qquad \frac{u_1^i \subseteq^i u_2^i}{D(u_1^i) \subseteq^i D(u_2^i)}\ \text{LE-I-VAL-CONSTR1}$$

$$\frac{u_1^i \subseteq^i u_2^i \qquad u_1'^i \subseteq^i u_2'^i}{(u_1^i, u_1'^i) \subseteq^i (u_2^i, u_2'^i)}\ \text{LE-I-VAL-COUPLE} \qquad \frac{\Gamma_1^i \subseteq^i \Gamma_2^i}{< \lambda x.e, \Gamma_1^i > \subseteq^i < \lambda x.e, \Gamma_2^i >}\ \text{LE-I-VAL-CLOSURE}$$

$$\frac{\Gamma_1^i \subseteq^i \Gamma_2^i}{< \texttt{rec} f.x.e, \Gamma_1^i > \subseteq^i < \texttt{rec} f.x.e, \Gamma_2^i >)}\ \text{LE-I-VAL-REC-CLOSURE}$$

$$\frac{}{\{\} \subseteq^i \{\}}\ \text{LE-I-ENV-EMPTY} \qquad \frac{u_1^i \subseteq^i u_2^i \qquad \Gamma_1^i \subseteq^i \Gamma_2^i}{(x, u_1^i) \oplus \Gamma_1^i \subseteq^i (x, u_2^i) \oplus \Gamma_2^i}\ \text{LE-I-ENV-CONS}$$

$$\frac{}{\{\} \subseteq^i \{\}}\ \text{LE-TI-ENV-EMPTY} \qquad \frac{tu_1^i \subseteq^i u_2^i \qquad t\Gamma_1^i \subseteq^i t\Gamma_2^i}{(x, tu_1^i) \oplus t\Gamma_1^i \subseteq^i (x, tu_2^i) \oplus t\Gamma_2^i}\ \text{LE-TI-ENV-CONS}$$

FIGURE 4.9 – Relation d'ordre sur les valeurs et environnements instrumentés

$$\frac{\forall tu_1^i \in v_1^{im}.\ \exists tu_2^i \in v_2^{im}.\ tu_1^i \subseteq^i tu_2^i}{v_1^{im} \subseteq^{im} v_2^{im}}\ \text{LE-IM-VAL}$$

$$\frac{\forall t\Gamma_1^i \in t\Gamma_1^{im}.\ \exists t\Gamma_2^i \in t\Gamma_2^{im}.\ t\Gamma_1^i \subseteq^i t\Gamma_2^i}{t\Gamma_1^{im} \subseteq^{im} t\Gamma_2^{im}}\ \text{LE-IM-ENV}$$

FIGURE 4.10 – Relation d'ordre sur les valeurs et environnements instrumentés multiples

$t\Gamma^{im} \vdash^{im} e \mapsto v^{im} \Rightarrow t\Gamma^{c} = \uparrow^{c}_{im}(t\Gamma^{im}) \Rightarrow t\Gamma^{c} \vdash^{c} e \mapsto tu^{c} \Rightarrow v^{im} \subseteq^{im} \uparrow^{im}_{c}(tu^{c})$

4.3.3.4 Preuve de correction

La preuve est faite par induction sur le jugement de la sémantique collectrice. On obtient alors 20 cas à prouver, suivant les 20 règles d'inférences de la sémantique collectrice. Nous expliquons ici quelques cas pour permettre au lecteur de comprendre plus facilement la manière dont la preuve a été faite. La preuve complète est disponible sous forme de code source Coq.

cas C-NUM Ce cas est trivial, tout comme le cas C-CONSTR-0. Nous allons tout de même l'expliquer rapidement pour illustrer le principe de la preuve qui est également utilisé dans les autres cas.

Nous avons les hypothèses suivantes :

$$t\Gamma^{im} \vdash^{im} n \mapsto v^{im}$$
$$t\Gamma^{c} = \uparrow^{c}_{im}(t\Gamma^{im})$$
$$t\Gamma^{c} \vdash^{c} n \mapsto [\ \emptyset\ |\ \emptyset\ |\ \{n\}\ ]$$

On veut alors prouver la propriété ci-dessous :

$$v^{im} \subseteq^{im} \uparrow^{im}_{c}([\ \emptyset\ |\ \emptyset\ |\ \{n\}\ ])$$

La règle de la sémantique instrumentée multiple I-MULTIPLE nous fait appliquer la règle I-NUM de la sémantique instrumentée. Pour tout environnement $t\Gamma^{i} \in t\Gamma^{im}$, cette dernière nous donne la valeur $[\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]$. On en déduit que v^{im} est le singleton $\{\ [\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]\}$. D'autre part, la concrétisation de la valeur collectrice vaut : $\uparrow^{im}_{c}([\ \emptyset\ |\ \emptyset\ |\ \{n\}\ ]) = \{\ [\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]\}$. Pour conclure, il nous suffit donc de prouver la propriété ci-dessous qui est évidente puisque la relation d'ordre sur les valeurs instrumentées multiples est réflexive.

$$\{\ [\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]\} \subseteq^{im} \{\ [\ \emptyset\ |\ [\ \emptyset\ |\ n\ ]\ ]\}$$

cas C-CONSTR-1 Ce cas est légèrement plus complexe que le précédent du fait que l'expression évaluée possède une sous-expression.

Nous avons les hypothèses suivantes dont une hypothèse d'induction :

$$\frac{\text{I-MULTIPLE}\quad v^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i D(e) \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} D(e) \mapsto v^{im}}$$

$$t\Gamma^c = \uparrow^c_{im}(t\Gamma^{im})$$

$$\frac{\text{C-CONSTR-1}\quad t\Gamma^c \vdash^c e \mapsto\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]}{t\Gamma^c \vdash^c D(e) \mapsto\ [\ td^c\ \mid\ \emptyset\ \mid\ \{D(\ [\ d^c\ \mid\ v^i\ ])\mid v^i \in vs^i\}\ ]}$$

$$\frac{\text{I-MULTIPLE}\quad v'^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i e \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} e \mapsto v'^{im}}$$

$$v'^{im} \subseteq^{im} \uparrow^{im}_c(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ])$$

Nous devons alors montrer cette propriété :

$$v^{im} \subseteq^{im} \uparrow^{im}_c(\ [\ td^c\ \mid\ \emptyset\ \mid\ \{D(\ [\ d^c\ \mid\ v^i\ ])\mid v^i \in vs^i\}\ ])$$

Pour cela, nous montrons que pour un élément quelconque de $tu^i \in v^{im}$, il existe un élément plus grand dans $\uparrow^{im}_c(\ [\ td^c\ \mid\ \emptyset\ \mid\ \{D(\ [\ d^c\ \mid\ v^i\ ])\mid v^i \in vs^i\}\ ])$.

Puisque tu^i appartient à v^{im}, il est issu d'un jugement d'évaluation instrumentée déduit de la règle d'inférence ci-dessous (en notant $tu^i =\ [\ td^i\ \mid\ [\ \emptyset\ \mid\ D(u^i)\ ]\ ]$) :

$$\frac{\text{I-CONSTR-1}\quad t\Gamma^i \vdash^i e \mapsto\ [\ td^i\ \mid\ u^i\ ]}{t\Gamma^i \vdash^i D(e) \mapsto\ [\ td^i\ \mid\ [\ \emptyset\ \mid\ D(u^i)\ ]\ ]}$$

Nous avons alors $[\ td^i\ \mid\ u^i\ ] \in v'^{im}$ et nous pouvons donc appliquer notre hypothèse d'induction pour en déduire qu'il existe une valeur $[\ td^i_2\ \mid\ [\ d^i_2\ \mid\ v^i_2\ ]\ ]$ telle que :

$$[\ td^i_2\ \mid\ [\ d^i_2\ \mid\ v^i_2\ ]\ ] \in \uparrow^{im}_c(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ])$$

$$[\ td^i\ \mid\ u^i\ ] \subseteq^i\ [\ td^i_2\ \mid\ [\ d^i_2\ \mid\ v^i_2\ ]\ ]$$

De la première de ces deux propriétés, nous déduisons que $td^i_2 = td^c$, $d^i_2 = d^c$ et $v^i_2 \in vs^i$. Nous pouvons alors construire la valeur $[\ td^c\ \mid\ [\ \emptyset\ \mid\ D(\ [\ d^c\ \mid\ v^i_2\ ])\ ]\ ]$ et montrer qu'elle

respecte les conditions nous permettant de conclure :

$$[\ td^c\ \mid\ [\ \emptyset\ \mid\ D(\ [\ d^c\ \mid\ v_2^i\ ])\ ]\ ] \in \uparrow_c^{im}(\ [\ td^c\ \mid\ \emptyset\ \mid\ \{D(\ [\ d^c\ \mid\ v^i\ ])\mid v^i \in vs^i\}\ ])$$

$$[\ td^i\ \mid\ [\ \emptyset\ \mid\ D(u^i)\ ]\ ] \subseteq^i [\ td^c\ \mid\ [\ \emptyset\ \mid\ D(\ [\ d^c\ \mid\ v_2^i\ ])\ ]\ ]$$

cas C-IDENT Une des particularités de la sémantique collectrice est d'avoir des règles d'inférence donnant une sémantique à un programme erroné, par exemple un programme évalué dans un environnement où certains identificateurs libres ne sont pas définis. Nous présentons donc la preuve des cas C-IDENT et C-IDENT-EMPTY pour se rendre compte de la manière dont sont traités les cas d'erreur.

Commençons par le cas C-IDENT. Nous avons les hypothèses suivantes :

$$\frac{\text{I-MULTIPLE}}{\dfrac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i x \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} x \mapsto v^{im}}} \qquad t\Gamma^c = \uparrow_{im}^c(t\Gamma^{im}) \qquad \frac{\text{C-IDENT}}{\dfrac{tu^c = t\Gamma^c[x]}{t\Gamma^c \vdash^c x \mapsto tu^c}}$$

Nous devons alors montrer cette proriété :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(tu^c)$$

Pour cela, supposons que nous avons une t-valeur instrumentée $tu^i \in v^{im}$ et montrons qu'il existe une t-valeur instrumentée $tu_2^i \in \uparrow_c^{im}(tu^c)$ telle que $tu^i \subseteq^i tu_2^i$. Puisque $tu^i \in v^{im}$, il existe un t-environnement instrumenté $t\Gamma^i \in t\Gamma^{im}$ tel que :

$$\frac{\text{I-IDENT}}{\dfrac{tu^i = t\Gamma^i[x]}{t\Gamma^i \vdash^i x \mapsto tu^i}}$$

D'après la définition des fonctions d'abstraction et de concrétisation, nous pouvons déduire qu'il existe une valeur $tu_2^i \in \uparrow_c^{im}(t\Gamma^c[x])$ telle que $tu^i \subseteq^i tu_2^i$, ce qui nous permet de conclure.

cas C-IDENT-EMPTY Dans ce cas, nous avons les hypothèses suivantes :

$$\frac{\text{I-MULTIPLE}}{\dfrac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i x \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} x \mapsto v^{im}}} \qquad t\Gamma^c = \uparrow_{im}^c(t\Gamma^{im}) \qquad \frac{\text{C-IDENT-EMPTY}}{\dfrac{x \notin support(t\Gamma^c)}{t\Gamma^c \vdash^c x \mapsto\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \emptyset\ ]}}$$

Et nous devons montrer :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \emptyset\ ])$$

Puisque $\uparrow_c^{im}(\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \emptyset\ ]) = \emptyset$, il nous faut montrer que $v^{im} = \emptyset$. D'après la définition de la fonction d'abstraction, les identificateurs présents dans n'importe quel élément de $t\Gamma^{im}$ sont les mêmes que ceux présents dans $t\Gamma^c$. Ainsi, puisque l'identificateur x n'est pas présent dans l'environnement $t\Gamma^c$, alors il n'est présent dans aucun t-environnement instrumenté appartenant à $t\Gamma^{im}$. Il n'existe alors aucune règle d'inférence permettant d'évaluer l'expression x dans un t-environnement instrumenté appartenant à $t\Gamma^{im}$. On peut donc conclure puisque $v^{im} = \emptyset$.

cas C-APPLY Le cas de l'application est un des cas les plus complexes en raison de la complexité des règles d'inférence correspondantes dans la sémantique collectrice et dans la sémantique instrumentée. Nous avons à notre disposition les hypothèses de l'énoncé du thèorème ainsi que deux hypothèses d'induction, correspondant aux deux sous-termes de l'expression :

$$\dfrac{\overset{\text{I-MULTIPLE}}{\dfrac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i e_1\ e_2 \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} e_1\ e_2 \mapsto v^{im}}} \qquad t\Gamma^c = \uparrow_{im}^c(t\Gamma^{im})}{\overset{\text{C-APPLY}}{\dfrac{\begin{array}{c} t\Gamma^c \vdash^c e_1 \mapsto\ [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ] \qquad t\Gamma^c \vdash^c e_2 \mapsto\ [\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ] \\ v'^{im} = multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i) \\ (\forall l.\ l \in td^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in td^i \wedge\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v'^{im})) \\ (\forall l.\ l \in d^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in d^i \wedge\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v'^{im})) \\ vs^i = \{v^i \mid \exists (td^i, d^i).\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v'^{im}\} \end{array}}{t\Gamma^c \vdash^c e_1\ e_2 \mapsto\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]}}}$$

$$\dfrac{\overset{\text{I-MULTIPLE}}{\dfrac{v_1^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i e_1 \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} e_1 \mapsto v_1^{im}}} \qquad \overset{\text{I-MULTIPLE}}{\dfrac{v_2^{im} = \{tu^i \mid \exists t\Gamma^i \in t\Gamma^{im}.\ t\Gamma^i \vdash^i e_2 \mapsto tu^i\}}{t\Gamma^{im} \vdash^{im} e_1 \mapsto v_2^{im}}}}{v_1^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ]) \qquad v_2^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ])}$$

Et nous devons montrer :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ])$$

Pour cela, supposons que nous avons une t-valeur instrumentée $tu^i \in v^{im}$ et montrons qu'il existe une t-valeur instrumentée $tu'^i \in \uparrow_c^{im}(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ])$ telle que $tu^i \subseteq^i tu'^i$. Puisque $tu^i \in v^{im}$, il existe une évaluation instrumentée de l'expression $e_1\ e_2$ en la t-valeur instrumentée tu^i dans un certain t-environnement instrumenté $t\Gamma^i \in t\Gamma^{im}$. Il y a alors deux cas possibles selon que la règle d'inférence de cette évaluation est I-APPLY ou I-APPLY-REC.

– cas I-APPLY

$$\text{I-APPLY}\ \frac{\begin{array}{c} t\Gamma^i \vdash^i e_1 \mapsto\ [\ td_1^i\ \mid\ [\ d_1^i\ \mid < \lambda x.e, \Gamma_1^i >\ ]\ ] \\ t\Gamma^i \vdash^i e_2 \mapsto tu_2^i \qquad tu_2^i =\ [\ td_2^i\ \mid\ u_2^i\ ] \\ (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \end{array}}{t\Gamma^i \vdash^i e_1\ e_2 \mapsto\ [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ \mid\ [\ d_1^i \cup d^i\ \mid\ v^i\ ]\ ]}$$

$$tu^i =\ [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ \mid\ [\ d_1^i \cup d^i\ \mid\ v^i\ ]\ ]$$

D'après nos hypothèses d'induction, nous savons qu'il existe des t-valeurs instrumentées $tu_1'^i \in \uparrow_c^{im}(\ [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ])$ et $tu_2'^i \in \uparrow_c^{im}(\ [\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ])$ vérifiant les propriétés suivantes :

$$[\ td_1^i\ \mid\ [\ d_1^i\ \mid < \lambda x.e, \Gamma_1^i >\ ]\ ] \subseteq^i\ tu_1'^i \qquad\qquad tu_2^i \subseteq^i\ tu_2'^i$$

Puisque $tu_1'^i \in \uparrow_c^{im}(\ [\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ])$, nous savons qu'il existe une valeur simple instrumentée $v_1'^i$ telle que $tu_1'^i =\ [\ td_1^c\ \mid\ [\ d_1^c\ \mid\ v_1'^i\ ]\ ]$ et $v_1'^i \in vs_1^i$. De même, il existe une valeur simple instrumentée $v_2'^i$ telle que $tu_2'^i =\ [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2'^i\ ]\ ]$ et $v_2'^i \in vs_2^i$. Nous déplions alors la définition de la relation d'ordre pour obtenir les relations suivantes (en notant $tu_2^i =\ [\ td_2^i\ \mid\ [\ d_2^i\ \mid\ v_2^i\ ]\ ]$) :

$$td_1^i \subseteq^i td_1^c \qquad d_1^i \subseteq^i d_1^c \qquad < \lambda x.e, \Gamma_1^i > \subseteq^i v_1'^i$$

$$v_1'^i =< \lambda x.e, \Gamma_1'^i > \qquad \Gamma_1^i \subseteq^i \Gamma_1'^i$$

$$td_2^i \subseteq^i td_2^c \qquad d_2^i \subseteq^i d_2^c \qquad v_2^i \subseteq^i v_2'^i$$

Nous utilisons le lemme intermédiaire suivant pour déduire un jugement d'évaluation instrumentée de l'expression e dans l'environnement $(x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i)$. La preuve de ce lemme a été réalisée en Coq (`Lemma ival_of_in_le_itenv`). Voici son énoncé :

$$\forall(\Gamma^i, \Gamma'^i, e, tu^i).(\Gamma_1^i \subseteq^i \Gamma_1'^i) \Rightarrow (t\Gamma^i \vdash^i e \mapsto tu^i) \Rightarrow \exists tu'^i.(t\Gamma'^i \vdash^i e \mapsto tu'^i) \wedge (tu^i \subseteq^i tu'^i)$$

Nous obtenons alors une t-valeur instrumentée $[\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$ vérifiant le jugement d'évaluation instrumentée suivant et la relation d'ordre suivante :

$$(x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i) \vdash^i e \mapsto [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$$

$$[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$$

Pour conclure le cas I-APPLY, il nous faut maintenant montrer que la t-valeur instrumentée $[\ td^c \ \mid \ [\ d^c \ \mid \ v'^i\ ]\ ]$ convient. C'est à dire qu'elle vérifie les deux propriétés suivantes : $[\ td^c \ \mid \ [\ d^c \ \mid \ v'^i\ ]\ ] \in \uparrow_c^{im}(\ [\ td^c \ \mid \ d^c \ \mid \ vs^i\ ])$ et $tu^i \subseteq^i [\ td^c \ \mid \ [\ d^c \ \mid \ v'^i\ ]\ ]$.

Commençons par la première des deux propriétés à prouver. En dépliant la définition de la fonction de concrétisation, la propriété à prouver se réduit à $v'^i \in vs^i$. Pour cela, il nous faut montrer $\exists(td^i, d^i).\ [\ td^i \ \mid \ [\ d^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$. Nous allons donc montrer que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$. Nous déplions ensuite la valeur de v'^{im} ainsi que celle de $multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i)$. Ainsi, la propriété à montrer devient :

$[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in$

$$\begin{array}{lll}
\{\ tu^i \mid & (& \exists < \lambda x.e, \Gamma_1^i > \in vs_1^i. \\
& & \exists v_2^i \in vs_2^i. \\
& & \exists(td^i, t^i, v^i). \\
& & \exists tu_2^i = [\ td_2^c \ \mid \ [\ d_2^c \ \mid \ v_2^i\ ]\ ]. \\
& & (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \\
& & \wedge\ tu^i = [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c \ \mid \ [\ d_1^c \cup d^i \ \mid \ v^i\ ]\ ]\ ) \\
& \vee\ (& \exists < \mathtt{rec} f.x.e, \Gamma_1^i > \in vs_1^i. \\
& & \exists v_2^i \in vs_2^i. \\
& & \exists(td^i, t^i, v^i). \\
& & \exists tu_f^i = [\ td_1^c \ \mid \ [\ d_1^c \ \mid < \mathtt{rec} f.x.e, \Gamma_1^i >\ ]\ ]. \\
& & \exists tu_2^i = [\ td_2^c \ \mid \ [\ d_2^c \ \mid \ v_2^i\ ]\ ]. \\
& & (f, tu_f^i) \oplus (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \\
& & \wedge\ tu^i = [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c \ \mid \ [\ d_1^c \cup d^i \ \mid \ v^i\ ]\ ]\ )\ \}
\end{array}$$

Puisque nous sommes dans le cas de l'application d'une fonction non récursive, c'est la première partie de la disjonction que nous allons prouver. Nous prouvons cette propriété à l'aide des propriétés suivantes déjà prouvées plus haut :

$$< \lambda x.e, \Gamma_1'^i >\in vs_1^i$$

$$v_2'^i \in vs_2^i$$

$$(x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i) \vdash^i e \mapsto \ [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$$

Montrons maintenant la seconde propriété qui nous permettra de conclure. Après avoir déplié la valeur de tu^i elle exprime ainsi :

$$[\ td_1^i \cup td_2^i \cup td^i \cup d_1^i \ \mid \ [\ d_1^i \cup d^i \ \mid \ v^i\ ]\ ] \subseteq^i \ [\ td^c \ \mid \ [\ d^c \ \mid \ v'^i\ ]\ ]$$

En dépliant la définition de la relation d'ordre, nous obtenons les trois propriétés suivantes à prouver :

$$\forall l \in (td_1^i \cup td_2^i \cup td^i \cup d_1^i).l \in td^c \qquad \forall l \in (d_1^i \cup d^i).l \in d^c \qquad v^i \subseteq^i v'^i$$

Prouvons la propriété concernant les t-dépendances.
Soit $l \in (td_1^i \cup td_2^i \cup td^i \cup d_1^i)$. D'après la définition de td^c, il nous faut montrer que $\exists(td^i, d^i, v^i).\ l \in td^i \wedge [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \in v'^{im}$. Nous savons déjà que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$ Montrons donc que $l \in (td_1^c \cup td_2^c \cup td'^i \cup d_1^c)$. Cette propriété est issue des relations d'ordre déjà montrées et de la compatibilité de la relation d'ordre avec la concaténation [1]. Les relations concernées sont : $td_1^i \subseteq^i td_1^c$, $td_2^i \subseteq^i td_2^c$, $[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i \ [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$ et $d_1^i \subseteq^i d_1^c$.
Prouvons la propriété concernant les v-dépendances.
Soit $l \in (d_1^i \cup d^i)$. D'après la définition de d^c, il nous faut montrer que $\exists(td^i, d^i, v^i).\ l \in d^i \wedge [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \in v'^{im}$. Nous savons déjà que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$ Montrons donc que $l \in (d_1^c \cup d'^i)$. Cette propriété est issue des relations d'ordre déjà montrées et de la compatibilité de la relation d'ordre avec la concaténation [2]. Les relations concernées sont : $d_1^i \subseteq^i d_1^c$ et $[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$.

1. $\forall(td_1, td_2, td_1', td_2').\ (td_1 \subseteq td_1') \wedge (td_2 \subseteq td_2') \Rightarrow td_1 \cup td_2 \subseteq td_1' \cup td_2'$
2. $\forall(d_1, d_2, d_1', d_2').\ (d_1 \subseteq d_1') \wedge (d_2 \subseteq d_2') \Rightarrow d_1 \cup d_2 \subseteq d_1' \cup d_2'$

La propriété concernant les valeurs simples instrumentées vient directement de la relation $[\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \subseteq^i [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$ prouvée précédemment.
Le cas I-APPLY est donc complètement prouvé.

– cas I-APPLY-REC

$$\frac{\begin{array}{c}t\Gamma^i \vdash^i e_1 \mapsto tu_1^i \qquad tu_1^i = [\ td_1^i\ \mid\ [\ d_1^i\ \mid < \texttt{rec}f.x.e, \Gamma_1^i >\ ]\ ] \\ t\Gamma^i \vdash^i e_2 \mapsto tu_2^i \qquad tu_2^i = [\ td_2^i\ \mid\ u_2^i\ ] \\ (f, tu_1^i) \oplus (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ]\end{array}}{t\Gamma^i \vdash^i e_1\ e_2 \mapsto [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ \mid\ [\ d_1^i \cup d^i\ \mid\ v^i\ ]\ ]}$$

$$tu^i = [\ td_1^i \cup td_2^i \cup td^i \cup d_1^i\ \mid\ [\ d_1^i \cup d^i\ \mid\ v^i\ ]\ ]$$

D'après nos hypothèses d'induction, nous savons qu'il existe des t-valeurs instrumentées $tu_1'^i \in \uparrow_c^{im}([\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ])$ et $tu_2'^i \in \uparrow_c^{im}([\ td_2^c\ \mid\ d_2^c\ \mid\ vs_2^i\ ])$ vérifiant les propriétés suivantes :

$$[\ td_1^i\ \mid\ [\ d_1^i\ \mid < \texttt{rec}f.x.e, \Gamma_1^i >\ ]\ ] \subseteq^i tu_1'^i \qquad\qquad tu_2^i \subseteq^i tu_2'^i$$

Puisque $tu_1'^i \in \uparrow_c^{im}([\ td_1^c\ \mid\ d_1^c\ \mid\ vs_1^i\ ])$, nous savons qu'il existe une valeur simple instrumentée $v_1'^i$ telle que $tu_1'^i = [\ td_1^c\ \mid\ [\ d_1^c\ \mid\ v_1'^i\ ]\ ]$ et $v_1'^i \in vs_1^i$. De même, il existe une valeur simple instrumentée $v_2'^i$ telle que $tu_2'^i = [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2'^i\ ]\ ]$ et $v_2'^i \in vs_2^i$. Nous déplions alors la définition de la relation d'ordre pour obtenir les relations suivantes (en notant $tu_2^i = [\ td_2^i\ \mid\ [\ d_2^i\ \mid\ v_2^i\ ]\ ]$) :

$$td_1^i \subseteq^i td_1^c \qquad d_1^i \subseteq^i d_1^c \qquad < \texttt{rec}f.x.e, \Gamma_1^i > \subseteq^i v_1'^i$$

$$v_1'^i = < \texttt{rec}f.x.e, \Gamma_1'^i > \qquad \Gamma_1^i \subseteq^i \Gamma_1'^i$$

$$td_2^i \subseteq^i td_2^c \qquad d_2^i \subseteq^i d_2^c \qquad v_2^i \subseteq^i v_2'^i$$

Comme dans le cas précédent, nous utilisons le lemme intermédiaire suivant pour déduire un jugement d'évaluation instrumentée de l'expression e dans l'environnement $(f, tu_1'^i) \oplus (x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i)$. La preuve de ce lemme a été réalisée en Coq (`Lemma ival_of_in_le_itenv`). Voici son énoncé :

$$\forall(\Gamma^i, \Gamma'^i, e, tu^i).(\Gamma_1^i \subseteq^i \Gamma_1'^i) \Rightarrow (t\Gamma^i \vdash^i e \mapsto tu^i) \Rightarrow \exists tu'^i.(t\Gamma'^i \vdash^i e \mapsto tu'^i) \wedge (tu^i \subseteq^i tu'^i)$$

Nous obtenons alors une t-valeur instrumentée $[\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$ vérifiant le jugement d'évaluation instrumentée suivant et la relation d'ordre suivante :

$$(f, tu_1'^i) \oplus (x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i) \vdash^i e \mapsto\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$$

$$[\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \subseteq^i\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$$

Pour conclure le cas I-APPLY-REC, il nous faut maintenant montrer que la t-valeur instrumentée $[\ td^c\ \mid\ [\ d^c\ \mid\ v'^i\ ]\ ]$ convient. C'est à dire qu'elle vérifie les deux propriétés suivantes : $[\ td^c\ \mid\ [\ d^c\ \mid\ v'^i\ ]\ ] \in \uparrow_c^{im}(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ])$ et $tu^i \subseteq^i\ [\ td^c\ \mid\ [\ d^c\ \mid\ v'^i\ ]\ ]$.

Commençons par la première des deux propriétés à prouver. En dépliant la définition de la fonction de concrétisation, la propriété à prouver se réduit à $v'^i \in vs^i$. Pour cela, il nous faut montrer $\exists(td^i, d^i).\ [\ td^i\ \mid\ [\ d^i\ \mid\ v'^i\ ]\ ] \in v'^{im}$. Nous allons donc montrer que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c\ \mid\ [\ d_1^c \cup d'^i\ \mid\ v'^i\ ]\ ] \in v'^{im}$. Nous déplions ensuite la valeur de v'^{im} ainsi que celle de $multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i)$. Ainsi, la propriété à montrer devient :

$[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c\ \mid\ [\ d_1^c \cup d'^i\ \mid\ v'^i\ ]\ ] \in$

$$\begin{array}{lll}
\{\ tu^i \mid & (& \exists < \lambda x.e, \Gamma_1^i > \in vs_1^i. \\
& & \exists v_2^i \in vs_2^i. \\
& & \exists(td^i, t^i, v^i). \\
& & \exists tu_2^i =\ [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2^i\ ]\ ]. \\
& & (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \\
& & \wedge\ tu^i =\ [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c\ \mid\ [\ d_1^c \cup d^i\ \mid\ v^i\ ]\ ]\ ) \\
& \vee\ (& \exists < \mathtt{rec} f.x.e, \Gamma_1^i > \in vs_1^i. \\
& & \exists v_2^i \in vs_2^i. \\
& & \exists(td^i, t^i, v^i). \\
& & \exists tu_f^i =\ [\ td_1^c\ \mid\ [\ d_1^c\ \mid < \mathtt{rec} f.x.e, \Gamma_1^i >\ ]\ ]. \\
& & \exists tu_2^i =\ [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2^i\ ]\ ]. \\
& & (f, tu_f^i) \oplus (x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto\ [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \\
& & \wedge\ tu^i =\ [\ td_1^c \cup td_2^c \cup td^i \cup d_1^c\ \mid\ [\ d_1^c \cup d^i\ \mid\ v^i\ ]\ ]\ )\ \}
\end{array}$$

Puisque nous sommes dans le cas de l'application d'une fonction récursive, c'est la seconde partie de la disjonction que nous allons prouver. Nous prouvons cette propriété à l'aide des propriétés suivantes déjà prouvées plus haut :

$$< \mathtt{rec} f.x.e, \Gamma_1'^i > \in vs_1^i$$

$$v_2'^i \in vs_2^i$$

$$(f, tu_1'^i) \oplus (x, tu_2'^i) \oplus \uparrow_i^{ti}(\Gamma_1'^i) \vdash^i e \mapsto [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$$

Montrons maintenant la seconde propriété qui nous permettra de conclure. Après avoir déplié la valeur de tu^i elle exprime ainsi :

$$[\ td_1^i \cup td_2^i \cup td^i \cup d_1^i \ \mid \ [\ d_1^i \cup d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td^c \ \mid \ [\ d^c \ \mid \ v'^i\ ]\ ]$$

En dépliant la définition de la relation d'ordre, nous obtenons les trois propriétés suivantes à prouver :

$$\forall l \in (td_1^i \cup td_2^i \cup td^i \cup d_1^i).l \in td^c \qquad \forall l \in (d_1^i \cup d^i).l \in d^c \qquad v^i \subseteq^i v'^i$$

Nous procédons de la même manière que dans le cas I-APPLY.

Prouvons la propriété concernant les t-dépendances.

Soit $l \in (td_1^i \cup td_2^i \cup td^i \cup d_1^i)$. D'après la définition de td^c, il nous faut montrer que $\exists(td^i, d^i, v^i).\ l \in td^i \wedge [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \in v'^{im}$. Nous savons déjà que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$ Montrons donc que $l \in (td_1^c \cup td_2^c \cup td'^i \cup d_1^c)$. Cette propriété est issue des relations d'ordre déjà montrées et de la compatibilité de la relation d'ordre avec la concaténation. Les relations concernées sont : $td_1^i \subseteq^i td_1^c$, $td_2^i \subseteq^i td_2^c$, $[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$ et $d_1^i \subseteq^i d_1^c$.

Prouvons la propriété concernant les v-dépendances.

Soit $l \in (d_1^i \cup d^i)$. D'après la définition de d^c, il nous faut montrer que $\exists(td^i, d^i, v^i).\ l \in d^i \wedge [\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \in v'^{im}$. Nous savons déjà que $[\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ] \in v'^{im}$. Montrons donc que $l \in (d_1^c \cup d'^i)$. Cette propriété est issue des relations d'ordre déjà montrées et de la compatibilité de la relation d'ordre avec la concaténation. Les relations concernées sont : $d_1^i \subseteq^i d_1^c$ et $[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$.

La propriété concernant les valeurs simples instrumentées vient directement de la relation $[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] \subseteq^i [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$ prouvée précédemment.

Le cas I-APPLY-REC est donc complètement prouvé.

Ce qui conclut le cas C-APPLY.

4.4 Sémantique abstraite

Ce chapitre présente enfin la sémantique abstraite, qui constitue ce que nous appelons notre analyse statique. Le but de cette analyse est de calculer les dépendances de n'importe quel programme écrit dans notre langage et de permettre à l'utilisateur de spécifier les environnements d'évaluation possibles avec souplesse.

La sémantique abstraite est une interprétation abstraite de la sémantique collectrice présentée dans le chapitre précédent. La sémantique collectrice avait permis de faire ressortir le calcul des dépendances lors d'un ensemble d'évaluations instrumentées simultanées. Cependant, cette sémantique n'était pas calculable compte tenu des ensembles de valeurs simples instrumentées éventuellement infinis présents dans les valeurs collectrices. La sémantique abstraite vient résoudre ce problème de non-calculabilité en remplaçant ces ensembles de valeurs simples par un nombre fini de valeurs simples abstraites.

Elle présente deux intérets par rapport à la sémantique instrumentée. D'une part, elle permet d'analyser un programme sans avoir besoin de connaître avec précision son environnement d'évaluation. En effet, un environnement abstrait permet de représenter un ensemble d'environnements instrumentés. Et une seule évaluation abstraite permet de simuler un ensemble d'évaluations instrumentées possibles. D'autre part, cette sémantique termine pour n'importe quel programme à évaluer et n'importe quel environnement d'évaluation. On est donc certain d'obtenir un résultat en un temps fini. En effet, dans la sémantique instrumentée, deux cas pouvaient provoquer la non-terminaison de l'évaluation d'un programme : une erreur de type ou une boucle infinie. La sémantique abstraite permet de résoudre ces deux cas. Le cas d'une erreur de type résulte en une valeur spéciale qui indique qu'il n'existe aucune évaluation instrumentée correspondante. Le cas de la boucle infinie est résolu à l'aide d'une sur-approximation des dépendances et de la valeur de retour qui permet de stopper les appels récursifs infinis.

4.4.1 Algèbre des valeurs

La sémantique abstraite manipule des valeurs abstraites. Ces valeurs constituent une abstraction des valeurs collectrices. Les fonctions d'abstraction et de concrétisation sont

définies formellement en figures 4.11, 4.12 et 4.13.

4.4.1.1 Valeurs

Une t-valeur abstraite (notée tu^a) est composée d'un ensemble de t-dépendances td^a et d'une v-valeur abstraite u^a.

$$tu^a := \quad [\ td^a \ \mid \ u^a\ ] \quad \text{Valeur avec annotation de } t\text{-dépendance}$$

Une v-valeur abstraite (notée u^a) est composée d'un ensemble de v-dépendances d^a et d'une valeur simple abstraite v^a.

$$u^a := \quad [\ d^a \ \mid \ v^a\ ] \quad \text{Valeur avec annotation de } v\text{-dépendance}$$

Enfin, une valeur simple abstraite (notée v^a) est une valeur simple dont les sous-termes sont des v-valeurs abstraites u^a.

$v^a :=$	$C \mid D(u^a) \mid (u_1^a, u_2^a)$	Constructeurs de données
	$< \lambda x.e, \Gamma^a >$	Fermeture
	$< \texttt{rec}f.x.e, \Gamma^a >$	Fermeture récursive
	$\bot$	Aucune valeur
	$\top$	Valeur quelconque

On peut remarquer qu'il n'existe pas de valeur abstraite permettant de représenter une constante numérique ou bien un booléen. En effet, toute valeur numérique ou booléenne est abstraite en la valeur $\top$, comme on peut le constater dans la définition formelle de la fonction d'abstraction (cf. figures 4.11 et 4.12). En pratique, cette approximation permet à l'analyse de gagner en rapidité sans trop perdre en précision. En effet, les valeurs numériques et booléennes sont rarement connues statiquement, lors de l'analyse d'un programme. Il aurait été possible d'abstraire ces valeurs de façon moins brutale (par exemple à l'aide de listes d'intervalles représentant les valeurs possibles), cependant, ce n'est pas l'objet de notre analyse et il sera tout à fait possible de proposer par la suite une interprétation abstraite plus fine de la sémantique collectrice.

4.4.1.2 Ensembles de dépendances

Dans les valeurs abstraites, les ensembles de t-dépendances (resp. de v-dépendances) sont des ensembles de labels, comme dans les sémantiques instrumentée et collectrice.

Abstraction d'une t-valeur collectrice :

$$\textsc{abstr-c-val}\quad \frac{\uparrow^a_{im}(\{\ [\ d^c\ \mid\ v^i\ ]\mid v^i \in vs^i\}) = u^a}{\uparrow^a_c(\ [\ td^c\ \mid\ d^c\ \mid\ vs^i\ ]) =\ [\ td^a\ \mid\ u^a\ ]}$$

Abstraction d'un ensemble de v-valeurs instrumentées :

$$\textsc{abstr-im-a-val-empty}\quad \frac{}{\uparrow^a_{im}(\emptyset) =\ [\ \emptyset\ \mid\ \bot\ ]}$$

$$\textsc{abstr-im-a-val-contr-0}\quad \frac{us^i \subset \{\ [\ d^i\ \mid\ C\ ]\mid \forall d^i\}\qquad us^i \neq \emptyset\qquad d\_of\_us^i(us^i) = d^a}{\uparrow^a_{im}(us^i) =\ [\ d^a\ \mid\ C\ ]}$$

$$\textsc{abstr-im-a-val-contr-1}\quad \frac{\begin{array}{c} us^i \subset \{\ [\ d^i\ \mid\ D(u^i)\ ]\mid \forall(d^i,u^i)\}\qquad us^i \neq \emptyset \\ d\_of\_us^i(us^i) = d^a\qquad \uparrow^a_{im}(\{u^i \mid \exists d^i.\ [\ d^i\ \mid\ D(u^i)\ ] \in us^i\}) = u^a \end{array}}{\uparrow^a_{im}(us^i) =\ [\ d^a\ \mid\ D(u^a)\ ]}$$

$$\textsc{abstr-im-a-val-couple}\quad \frac{\begin{array}{c} us^i \subset \{\ [\ d^i\ \mid\ (u^i_1,u^i_2)\ ]\mid \forall(d^i,u^i_1,u^i_2)\}\qquad us^i \neq \emptyset\qquad d\_of\_us^i(us^i) = d^a \\ \uparrow^a_{im}(\{u^i_1 \mid \exists(d^i,u^i_2).\ [\ d^i\ \mid\ (u^i_1,u^i_2)\ ] \in us^i\}) = u^a_1 \\ \uparrow^a_{im}(\{u^i_2 \mid \exists(d^i,u^i_1).\ [\ d^i\ \mid\ (u^i_1,u^i_2)\ ] \in us^i\}) = u^a_2 \end{array}}{\uparrow^a_{im}(us^i) =\ [\ d^a\ \mid\ (u^a_1,u^a_2)\ ]}$$

$$\textsc{abstr-im-a-val-closure}\quad \frac{\begin{array}{c} us^i \subset \{\ [\ d^i\ \mid < \lambda x.e,\Gamma^i >\ ]\mid \forall(d^i,\Gamma^i)\}\qquad us^i \neq \emptyset \\ d\_of\_us^i(us^i) = d^a\qquad \uparrow^a_{im}(\{\Gamma^i \mid \exists d^i.\ [\ d^i\ \mid < \lambda x.e,\Gamma^i >\ ] \in us^i\}) = \Gamma^a \end{array}}{\uparrow^a_{im}(us^i) =\ [\ d^a\ \mid < \lambda x.e,\Gamma^a >\ ]}$$

$$\textsc{abstr-im-a-val-closure-rec}\quad \frac{\begin{array}{c} us^i \subset \{\ [\ d^i\ \mid < \texttt{rec}f.x.e,\Gamma^i >\ ]\mid \forall(d^i,\Gamma^i)\}\qquad us^i \neq \emptyset \\ d\_of\_us^i(us^i) = d^a\qquad \uparrow^a_{im}(\{\Gamma^i \mid \exists d^i.\ [\ d^i\ \mid < \texttt{rec}f.x.e,\Gamma^i >\ ] \in us^i\}) = \Gamma^a \end{array}}{\uparrow^a_{im}(us^i) =\ [\ d^a\ \mid < \texttt{rec}f.x.e,\Gamma^a >\ ]}$$

$$\textsc{abstr-im-a-val-unknown}\quad \frac{\begin{array}{c} us^i \neq \emptyset\qquad us^i \not\subset \{\ [\ d^i\ \mid\ C\ ]\mid \forall d^i\} \\ us^i \not\subset \{\ [\ d^i\ \mid\ D(u^i)\ ]\mid \forall(d^i,u^i)\}\qquad us^i \not\subset \{\ [\ d^i\ \mid\ (u^i_1,u^i_2)\ ]\mid \forall(d^i,u^i_1,u^i_2)\} \\ us^i \not\subset \{\ [\ d^i\ \mid < \lambda x.e,\Gamma^i >\ ]\mid \forall(d^i,\Gamma^i)\}\qquad us^i \not\subset \{\ [\ d^i\ \mid < \texttt{rec}f.x.e,\Gamma^i >\ ]\mid \forall(d^i,\Gamma^i)\} \end{array}}{\uparrow^a_{im}(us^i) =\ [\ \emptyset\ \mid\ \top\ ]}$$

FIGURE 4.11 – Sémantique abstraite : Définition de la fonction d'abstraction (partie 1/2)

Récolte des v-dépendances d'un ensemble de v-valeurs instrumentées :

$$\text{ABSTR-IM-DA}\quad \frac{\forall l.l \in d^a \Leftrightarrow \exists(d^i, v^i).l \in d^i \wedge\ [\ d^i\ \mid\ v^i\ ] \in us^i}{d\_of\_us^i(us^i) = d^a}$$

Abstraction d'un ensemble de v-environnements instrumentés :

$$\text{ABSTR-IM-A-ENV-EMPTY}\quad \frac{}{\uparrow^a_{im}(\{\}) = \{\}} \qquad \text{ABSTR-IM-A-ENV-CONS}\quad \frac{\Gamma^{im} = \{(x, u^i) \oplus \Gamma^i \mid \forall u^i \in us^i.\Gamma^i \in \Gamma'^{im}\} \quad \uparrow^a_{im}(us^i) = u^a \quad \uparrow^a_{im}(\Gamma'^{im}) = \Gamma^a}{\uparrow^a_{im}(\Gamma^{im}) = (x, u^a) \oplus \Gamma^a}$$

Abstraction d'un t-environnement collecteur :

$$\text{ABSTR-C-A-ENV-EMPTY}\quad \frac{}{\uparrow^a_c(\{\}) = \{\}} \qquad \text{ABSTR-C-A-ENV-CONS}\quad \frac{\uparrow^a_c(tu^c) = tu^a \quad \uparrow^a_c(t\Gamma^c) = t\Gamma^a}{\uparrow^a_c((x, tu^c) \oplus t\Gamma^c) = (x, tu^a) \oplus t\Gamma^a}$$

FIGURE 4.12 – Sémantique abstraite : Définition de la fonction d'abstraction (partie 2/2)

Concrétisation d'une t-valeur abstraite :

$$\uparrow^c_a(\ [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ]) := \quad [\ td^a\ \mid\ d^a\ \mid \uparrow^c_a(v^a)\ ]$$

Concrétisation d'une valeur simple abstraite :

$$\begin{array}{rcl}
\uparrow^c_a(C) := & \{C\} \\
\uparrow^c_a(D(\ [\ d^a\ \mid\ v^a\ ])) := & \{D(\ [\ d^a\ \mid\ v^i\ ]) \mid \forall v^i \in \uparrow^c_a(v^a)\} \\
\uparrow^c_a((\ [\ d^a_1\ \mid\ v^a_1\ ],\ [\ d^a_2\ \mid\ v^a_2\ ])) := & \{(\ [\ d^a_1\ \mid\ v^i_1\ ],\ [\ d^a_2\ \mid\ v^i_2\ ]) \mid \forall v^i_1 \in \uparrow^c_a(v^a_1).\forall v^i_2 \in \uparrow^c_a(v^a_2)\} \\
\uparrow^c_a(<\lambda x.e, \Gamma^a>) := & \{<\lambda x.e, \Gamma^i> \mid \forall \Gamma^i \in \uparrow^c_a(\Gamma^a)\} \\
\uparrow^c_a(<\texttt{rec}f.x.e, \Gamma^a>) := & \{<\texttt{rec}f.x.e, \Gamma^i> \mid \forall \Gamma^i \in \uparrow^c_a(\Gamma^a)\} \\
\uparrow^c_a(\bot) := & \emptyset \\
\uparrow^c_a(\top) := & \{v^i \mid \forall v^i \in \mathcal{V}^i\}
\end{array}$$

Concrétisation d'un v-environnement abstrait :

$$\begin{array}{rcl}
\uparrow^c_a(\{\}) := & \{\{\}\} \\
\uparrow^c_a((x,\ [\ d^a\ \mid\ v^a\ ]) \oplus \Gamma^a) := & \{(x,\ [\ d^a\ \mid\ v^i\ ]) \oplus \Gamma^i \mid \forall v^i \in \uparrow^c_a(v^a).\forall \Gamma^i \in \uparrow^c_a(\Gamma^a)\}
\end{array}$$

Concrétisation d'un t-environnement abstrait :

$$\begin{array}{rcl}
\uparrow^c_a(\{\}) := & \{\{\}\} \\
\uparrow^c_a((x, tu^a) \oplus t\Gamma^a) := & (x, \uparrow^c_a(tu^a)) \oplus \uparrow^c_a(t\Gamma^a)
\end{array}$$

FIGURE 4.13 – Sémantique abstraite : Définition de la fonction de concrétisation

$$\begin{array}{rll} td^a := & \{l_1; \ldots; l_n\} & t\text{-dépendances} \\ d^a := & \{l_1; \ldots; l_m\} & v\text{-dépendances} \end{array}$$

4.4.1.3 Environnements

De même que pour la sémantique sur-instrumentée et la sémantique instrumentée, on distingue 2 types d'environnements.

Un t-environnement abstrait permet de lier chaque identificateur à une t-valeur abstraite. Ce sont les environnements abstraits dans lesquels sont évalués les programmes.

$$t\Gamma^a := \quad (x_1, tu_1^a); \ldots; (x_n, tu_n^a) \quad t\text{-environnement abstrait}$$

Un v-environnement abstrait permet de lier chaque identificateur à une v-valeur abstraite. Ce sont les environnements abstraite encapsulés dans les fermetures (récursives ou non).

$$\Gamma^a := \quad (x_1, u_1^a); \ldots; (x_n, u_n^a) \quad v\text{-environnement abstraite}$$

4.4.2 Règles d'inférence

Le jugement d'évaluation de la sémantique abstraite prend la forme suivante :

$$t\Gamma^a \vdash^a e \mapsto tu^a$$

où e est l'expression évaluée, $t\Gamma^a$ le t-environnement abstrait dans lequel on effectue l'évaluation et tu^a la t-valeur abstraite résultat de l'évaluation.

Ce jugement est défini par les règles d'inférence présentées en figure 4.14.

Des explications détaillées de ces règles sont données ci-dessous.

A-NUM La règle d'évaluation d'une constante entière est semblable à celle de la sémantique instrumentée. L'ensemble des t-dépendances instrumentées est vide, de même que l'ensemble des v-dépendances instrumentées. Par contre, en ce qui concerne la valeur simple abstraite, elle vaut $\top$ qui est une abstraction de la constante entière.

$$\text{A-NUM}\ \frac{}{t\Gamma^a \vdash^a n \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ \top\ ]\ ]} \qquad \text{A-IDENT}\ \frac{tu^a = t\Gamma^a[x]}{t\Gamma^a \vdash^a x \mapsto tu^a} \qquad \text{A-IDENT-EMPTY}\ \frac{x \notin support(t\Gamma^a)}{t\Gamma^a \vdash^a x \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ \bot\ ]\ ]}$$

$$\text{A-ABSTR}\ \frac{}{t\Gamma^a \vdash^a \lambda x.e \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \lambda x.e, \uparrow^a_{ta}(t\Gamma^a) >\ ]\ ]} \qquad \text{A-ABSTR-REC}\ \frac{}{t\Gamma^a \vdash^a \texttt{rec}f.x.e \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid < \texttt{rec}f.x.e, \uparrow^a_{ta}(t\Gamma^a) >\ ]\ ]}$$

$$\text{A-LETIN}\ \frac{t\Gamma^a \vdash^a e_1 \mapsto tu_1^a \qquad tu_1^a = [\ td_1^a\ \mid\ u_1^a\ ] \qquad (x, tu_1^a) \oplus t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ u_2^a\ ]}{t\Gamma^a \vdash^a let\ x\ =\ e_1\ in\ e_2 \mapsto [\ td_1^a \cup td_2^a\ \mid\ u_2^a\ ]}$$

$$\text{A-APPLY}\ \frac{t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid < \lambda x.e, \Gamma_1^a >\ ]\ ] \qquad t\Gamma^a \vdash^a e_2 \mapsto tu_2^a \qquad tu_2^a = [\ td_2^a\ \mid\ u_2^a\ ] \qquad (x, tu_2^a) \oplus \uparrow^{ta}_a(\Gamma_1^a) \vdash^a e \mapsto [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ]}{t\Gamma^a \vdash^a e_1\ e_2 \mapsto [\ td_1^a \cup td_2^a \cup td^a \cup d_1^a\ \mid\ [\ d_1^a \cup d^a\ \mid\ v^a\ ]\ ]}$$

$$\text{A-APPLY-REC}\ \frac{\begin{array}{c}t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid < \texttt{rec}f.x.e, \Gamma_1^a >\ ]\ ] \qquad tu_1^a = [\ td_1^a\ \mid\ [\ d\_of\_freevars(\texttt{rec}f.x.e, t\Gamma^a)\ \mid\ \top\ ]\ ] \\ t\Gamma^a \vdash^a e_2 \mapsto tu_2^a \qquad tu_2^a = [\ td_2^a\ \mid\ u_2^a\ ] \\ (f, tu_1^a) \oplus (x, tu_2^a) \oplus \uparrow^{ta}_a(\Gamma_1^a) \vdash^a e \mapsto [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ]\end{array}}{t\Gamma^a \vdash^a e_1\ e_2 \mapsto [\ td_1^a \cup td_2^a \cup td^a \cup d_1^a\ \mid\ [\ d_1^a \cup d^a\ \mid\ v^a\ ]\ ]}$$

$$\text{A-APPLY-UNKNOWN}\ \frac{\begin{array}{c}t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid\ v_1^a\ ]\ ] \qquad t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ [\ d_2^a\ \mid\ v_2^a\ ]\ ] \\ \forall (f, x, e, \Gamma_1^a).v_1^a \neq < \lambda x.e, \Gamma_1^a > \wedge v_1^a \neq < \texttt{rec}f.x.e, \Gamma_1^a >\end{array}}{t\Gamma^a \vdash^a e_1\ e_2 \mapsto [\ td_1^a \cup td_2^a \cup d_1^a \cup d_2^a\ \mid\ [\ d_1^a \cup d_2^a\ \mid\ \top\ ]\ ]}$$

$$\text{A-IF}\ \frac{t\Gamma^a \vdash^a e \mapsto [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ] \qquad t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid\ v_1^a\ ]\ ] \qquad t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ [\ d_2^a\ \mid\ v_2^a\ ]\ ]}{t\Gamma^a \vdash^a if\ e\ then\ e_1\ else\ e_2 \mapsto [\ d^a \cup td^a \cup td_1^a \cup td_2^a\ \mid\ [\ d^a \cup d_1^a \cup d_2^a\ \mid\ \top\ ]\ ]}$$

$$\text{A-MATCH}\ \frac{\begin{array}{c}t\Gamma^a \vdash^a e \mapsto tu^a \qquad tu^a = [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ] \qquad tu^a, p \vdash^a_p t\Gamma^a_p \\ t\Gamma^a_p \oplus t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid\ v_1^a\ ]\ ]\end{array}}{t\Gamma^a \vdash^a match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^a \cup td^a \cup td_1^a\ \mid\ [\ d^a \cup d_1^a\ \mid\ v_1^a\ ]\ ]}$$

$$\text{A-MATCH-VAR}\ \frac{\begin{array}{c}t\Gamma^a \vdash^a e \mapsto tu^a \qquad tu^a = [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ] \qquad tu^a, p \vdash^a_p \bot \\ (x, tu^a) \oplus t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ [\ d_2^a\ \mid\ v_2^a\ ]\ ]\end{array}}{t\Gamma^a \vdash^a match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^a \cup td^a \cup td_2^a\ \mid\ [\ d^a \cup d_2^a\ \mid\ v_2^a\ ]\ ]}$$

$$\text{A-MATCH-UNKNOWN}\ \frac{\begin{array}{c}t\Gamma^a \vdash^a e \mapsto tu^a \qquad tu^a = [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ] \qquad tu^a, p \vdash^a_p ?\ t\Gamma^a_p \\ t\Gamma^a_p \oplus t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ [\ d_1^a\ \mid\ v_1^a\ ]\ ] \qquad (x, tu^a) \oplus t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ [\ d_2^a\ \mid\ v_2^a\ ]\ ]\end{array}}{t\Gamma^a \vdash^a match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ d^a \cup td^a \cup td_1^a \cup td_2^a\ \mid\ [\ d^a \cup d_1^a \cup d_2^a\ \mid\ \top\ ]\ ]}$$

$$\text{A-MATCH-ERROR}\ \frac{t\Gamma^a \vdash^a e \mapsto [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ] \qquad tu^a, p \vdash^a_p \times}{t\Gamma^a \vdash^a match\ e\ with\ p \to e_1 \mid x \to e_2 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ \bot\ ]\ ]}$$

$$\text{A-CONSTR-0}\ \frac{}{t\Gamma^a \vdash^a C \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ C\ ]\ ]} \qquad \text{A-CONSTR-1}\ \frac{t\Gamma^a \vdash^a e \mapsto [\ td^a\ \mid\ u^a\ ]}{t\Gamma^a \vdash^a D(e) \mapsto [\ td^a\ \mid\ [\ \emptyset\ \mid\ D(u^a)\ ]\ ]}$$

$$\text{A-COUPLE}\ \frac{t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a\ \mid\ u_1^a\ ] \qquad t\Gamma^a \vdash^a e_2 \mapsto [\ td_2^a\ \mid\ u_2^a\ ]}{t\Gamma^a \vdash^a (e_1, e_2) \mapsto [\ td_1^a \cup td_2^a\ \mid\ [\ \emptyset\ \mid\ (u_1^a, u_2^a)\ ]\ ]} \qquad \text{A-ANNOT}\ \frac{t\Gamma^a \vdash^a e \mapsto [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ]}{t\Gamma^a \vdash^a l\ :\ e \mapsto [\ td^a\ \mid\ [\ l; d^a\ \mid\ v^a\ ]\ ]}$$

FIGURE 4.14 – Sémantique abstraite

$$d\_of\_freevars(e, t\Gamma^a) := \quad dof\_aux(e, t\Gamma^a, \emptyset, \emptyset)$$

$$\begin{aligned}
dof_aux(n, t\Gamma^a, bvars, acc) &:= \quad acc \\
dof_aux(C, t\Gamma^a, bvars, acc) &:= \quad acc \\
dof_aux(D(e), t\Gamma^a, bvars, acc) &:= \quad dof_aux(e, t\Gamma^a, bvars, acc) \\
dof_aux(x, t\Gamma^a, bvars, acc) &:= \quad acc \text{ si } x \in bvars \\
dof_aux(x, t\Gamma^a, bvars, acc) &:= \quad d^a \cup acc \text{ si } x \notin bvars \wedge t\Gamma^a[x] = [\ td^a \ \mid \ [\ d^a \ \mid \ v^a\ ]\ ] \\
dof_aux(\lambda x.e, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e, t\Gamma^a, x; bvars, acc) \\
dof_aux(\texttt{rec}f.x.e, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e, t\Gamma^a, f; x; bvars, acc) \\
dof_aux(e_1\ e_2, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e_2, t\Gamma^a, bvars, acc_1) \\
&\quad \text{pour } acc_1 = dof_aux(e_1, t\Gamma^a, bvars, acc) \\
dof_aux(if\ e\ then\ e_1\ else\ e_2, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e_2, t\Gamma^a, bvars, acc_1) \\
&\quad \text{pour } acc_1 = dof_aux(e_1, t\Gamma^a, bvars, acc_0) \\
&\quad \text{et} \quad acc_0 = dof_aux(e, t\Gamma^a, bvars, acc) \\
dof_aux(match\ e\ with\ p \to e_1 \mid x \to e_2, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e_2, t\Gamma^a, x; bvars, acc_1) \\
&\quad \text{pour } acc_1 = dof_aux(e_1, t\Gamma^a, binders_of(p); bvars, acc_0) \\
&\quad \text{et} \quad acc_0 = dof_aux(e, t\Gamma^a, bvars, acc) \\
dof_aux((e_1, e_2), t\Gamma^a, bvars, acc) &:= \quad dof_aux(e_2, t\Gamma^a, bvars, acc_1) \\
&\quad \text{pour } acc_1 = dof_aux(e_1, t\Gamma^a, bvars, acc) \\
dof_aux(l\ :\ e, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e, t\Gamma^a, bvars, acc) \\
dof_aux(let\ x\ =\ e_1\ in\ e_2, t\Gamma^a, bvars, acc) &:= \quad dof_aux(e_2, t\Gamma^a, x; bvars, acc_1) \\
&\quad \text{pour } acc_1 = dof_aux(e_1, t\Gamma^a, bvars, acc)
\end{aligned}$$

$$\begin{aligned}
binders_of(C) &:= \quad \{\} \\
binders_of(D(x)) &:= \quad \{x\} \\
binders_of((x, y)) &:= \quad \{x; y\}
\end{aligned}$$

FIGURE 4.15 – Sémantique abstraite : v-dépendances des identificateurs libres d'une expression

$$\text{AM-CONSTR-0}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ C\ ]\ ], C \vdash^a_p \{\}} \qquad \text{AM-CONSTR-1}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ D(u^a)\ ]\ ], D(x) \vdash^a_p \{(x,\ [\ \emptyset \ \mid \ u^a\ ])\}}$$

$$\text{AM-COUPLE}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ (u_1^a, u_2^a)\ ]\ ], (x_1, x_2) \vdash^a_p \{(x_1,\ [\ \emptyset \ \mid \ u_1^a\ ]); (x_2,\ [\ \emptyset \ \mid \ u_2^a\ ])\}}$$

$$\text{AM-CONSTR-0-NOT}\ \frac{p \neq C}{[\ td^a \ \mid \ [\ d^a \ \mid \ C\ ]\ ], p \vdash^a_p \bot} \qquad \text{AM-CONSTR-1-NOT}\ \frac{p \neq D(\_)}{[\ td^a \ \mid \ [\ d^a \ \mid \ D(u^a)\ ]\ ], p \vdash^a_p \bot}$$

$$\text{AM-COUPLE-NOT}\ \frac{p \neq (\_, \_)}{[\ td^a \ \mid \ [\ d^a \ \mid \ (u_1^a, u_2^a)\ ]\ ], p \vdash^a_p \bot}$$

$$\text{AM-CONSTR-0-UNKNOWN}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ \top\ ]\ ], C \vdash^a_p ? \{\}} \qquad \text{AM-CONSTR-1-UNKNOWN}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ \top\ ]\ ], D(x) \vdash^a_p ? \{(x,\ [\ \emptyset \ \mid \ [\ \emptyset \ \mid \ \top\ ]\ ])\}}$$

$$\text{AM-COUPLE-UNKNOWN}\ \frac{}{[\ td^a \ \mid \ [\ d^a \ \mid \ \top\ ]\ ], (x_1, x_2) \vdash^a_p ? \{(x_1,\ [\ \emptyset \ \mid \ [\ \emptyset \ \mid \ \top\ ]\ ]); (x_2,\ [\ \emptyset \ \mid \ [\ \emptyset \ \mid \ \top\ ]\ ])\}}$$

$$\text{AM-ERROR}\ \frac{\forall C.v^a \neq C \qquad \forall D.v^a \neq D(\_) \qquad v^a \neq (\_, \_) \qquad v^a \neq \top}{[\ td^a \ \mid \ [\ d^a \ \mid \ v^a\ ]\ ], p \vdash^a_p \times}$$

FIGURE 4.16 – Sémantique abstraite : règles de filtrage

$$\uparrow^a_{ta}(\ [\ td^a\ \mid\ u^a\ ]) = u^a$$

$$\uparrow^a_{ta}(\{\}) = \{\} \qquad \uparrow^a_{ta}((x, tu^a) \oplus t\Gamma^a) = (x, \uparrow^a_{ta}(tu^a)) \oplus \uparrow^a_{ta}(t\Gamma^a)$$

FIGURE 4.17 – Valeurs abstraites : suppression des t-dépendances

A-IDENT Cette règle est identique à celle de la sémantique instrumentée. L'évaluation d'un identificateur se fait de manière habituelle, en allant chercher la valeur correspondante dans l'environnement. Les dépendances de la t-valeur abstraite retournée sont celles qui ont été enregistrées dans l'environnement pour cet identificateur.

A-IDENT-EMPTY Comme dans la sémantique collectrice, une règle a été ajoutée pour permettre l'analyse d'un programme dont certains identificateurs ne sont pas liés dans l'environnement. La valeur simple abstraite retournée est la valeur $\perp$, ce qui correspond à une erreur (l'évaluation d'un tel programme par la sémantique opérationnelle ne retourne aucune valeur).

A-ABSTR L'évaluation abstraite d'une fonction n'a rien de surprenant. Comme dans la sémantique instrumentée, les dépendances sont vides. La valeur simple abstraite est une fermeture encapsulant l'environnement abstrait d'évaluation préalablement nettoyé de ses t-dépendances à l'aide de la fonction $\uparrow^a_{ta}(\bullet)$ définie en figure 4.17.

A-ABSTR-REC Selon le même principe que pour la règle A-ABSTR, on construit une fermeture récursive encapsulant l'environnement abstrait d'évaluation.

A-LETIN La règle d'évaluation d'une liaison est identique à la règle correspondante dans la sémantique instrumentée. Elle évalue e_1 puis ajoute une liaison à l'environnement pour évaluer e_2. Le résultat est une t-valeur abstraite constituée de la concaténation des t-dépendances des sous-expressions évaluées et de la v-valeur abstraite de e_2.

A-APPLY La règle de l'application d'une fonction non-récursive est identique à son homologue de la sémantique instrumentée.

$$\uparrow_a^{ta}(u^a) = [\ \emptyset\ \mid\ u^a\ ]$$

$$\uparrow_a^{ta}(\{\}) = \{\} \qquad \uparrow_a^{ta}((x, u^a) \oplus \Gamma^a) = (x, \uparrow_a^{ta}(u^a)) \oplus \uparrow_a^{ta}(\Gamma^a)$$

FIGURE 4.18 – Valeurs abstraites : ajout de t-dépendances

Comme dans la sémantique instrumentée, nous utilisons une fonction ajoutant des t-dépendances vides à l'environnement encapsulé pour évaluer le corps de la fermeture. Cette fonction, notée $\uparrow_a^{ta}(\bullet)$ est définie en figure 4.18.

A-APPLY-REC La règle d'évaluation abstraite d'une application de fonction récursive ressemble à celle de la sémantique instrumentée. Il y a cependant une différence importante. Lorsque le corps de la fonction récursive est évalué, l'identificateur représentant la fonction récursive est lié à la valeur $\top$. Cette abstraction permet d'empêcher les appels récursifs lors de l'analyse d'un programme. À chaque appel récursif, au lieu de déplier le corps de la fonction une nouvelle fois, on renvoie la valeur $\top$ accompagnée de toutes les dépendances de tous les identificateurs libres présents dans le corps de la fonction.

A-APPLY-UNKNOWN La sémantique abstraite possède une règle d'évaluation d'une application dans le cas où on ne connaît pas le corps de la fonction à appliquer. On caractérise ce cas en spécifiant que la valeur abstraite de e_1 n'est pas une fermeture. Si le programme est bien typé, alors la valeur abstraite de e_1 est forcément $\top$. Cependant, il serait possible de rendre la sémantique abstraite plus précise en utilisant une réprésentation des fonctions ayant un niveau de précision intermédiaire entre la fermeture, contenant le corps de la fonction, et la valeur $\top$ ne fournissant aucune information.

Lorsque l'on ne connaît pas le corps de la fonction à appliquer, on effectue l'approximation suivante : tout label présent dans les v-dépendances de l'argument fait partie à la fois des t-dépendances et des v-dépendances du résultat. Il fait partie des t-dépendances du résultat car une modification de la valeur de l'argument peut provoquer la non-terminaison du corps de la fonction (qui est inconnu). Il fait également partie des v-dépendances du résultat la valeur de l'argument peut être utilisée pour élaborer la valeur de retour de la

fonction.

A-IF Contrairement aux autres sémantiques, la sémantique abstraite ne contient qu'une seule règle d'inférence pour décrire l'évaluation d'une expression conditionnelle. En effet, l'algèbre des valeurs abstraites ne permet pas de réprésenter les valeurs booléennes. Il est donc impossible de déterminer laquelle des deux branches sera évaluée à partir de la valeur abstraite de la condition. Nous évaluons donc les deux branches de l'expression pour recueillir les dépendances du résultat. L'ensemble des t-dépendances du résultat contient les t-dépendances des trois sous-expressions ainsi que les v-dépendances de la condition (car une modification de la valeur de la condition entraîne nécessairement un changement de branche, ce qui peut provoquer la non-terminaison de l'évaluation). L'ensemble des v-dépendances du résultat contient les v-dépendances des trois sous-expressions.

A-MATCH Cette règle décrit l'évaluation abstraite d'un filtrage lorsqu'on sait statiquement que l'évaluation passe toujours par la première branche. Elle ne présente pas de particularité par rapport à la sémantique instrumentée.

A-MATCH-VAR Cette règle décrit l'évaluation abstraite d'un filtrage lorsqu'on sait statiquement que l'évaluation passe toujours par la seconde branche. Elle ne présente pas de particularité par rapport à la sémantique instrumentée.

A-MATCH-UNKNOWN Cette règle décrit l'évaluation abstraite d'un filtrage lorsqu'on ne sait statiquement par quelle branche l'évaluation va passer. Elle correspond à la règle C-MATCH-UNKNOWN de la sémantique collectrice. On évalue alors les deux branches pour récolter les dépendances du filtrage.

A-MATCH-ERROR Cette règle décrit l'évaluation abstraite d'un filtrage dans le cas où la valeur abstraite de l'expression filtrée ne correspond pas à une valeur filtrable. Il s'agit d'une erreur de type. La valeur simple abstraite est donc $\bot$, ce qui indique qu'il n'existe aucune évaluation opérationnelle correspondante.

A-CONSTR-0 A-CONSTR-1 A-COUPLE A-ANNOT Ces quatres règles sont identiques à leurs homologues de la sémantique instrumentée.

4.4.3 Correction

4.4.3.1 Énoncé informel du théorème

La sémantique abstraite est une interprétation abstraite de la sémantique collectrice. C'est-à-dire qu'une évaluation par la sémantique collectrice peut être simulée par une évaluation abstraite (avec l'aide de fonctions d'abstraction et de concrétisation) et que la valeur ainsi obtenue sera moins précise que celle obtenue par une évaluation directe. Les fonctions d'abstraction et de concrétisation sont définies formellement en figures 4.11, 4.12 et 4.13.

Le lien entre l'évaluation directe par la sémantique collectrice et l'évaluation passant par la sémantique abstraite et les fonctions de conversion, est illustré en figure 4.2.

4.4.3.2 Illustration par l'exemple

Exemple 1 : évaluation d'un couple Notons e_1 le programme suivant : (x, l_1:7)

Nous nous intéressons ici à l'évaluation de ce programme par la sémantique collectrice dans l'environnement collecteur $t\Gamma_1^c$:

$$t\Gamma_1^c = (x,\ [\ \emptyset\ \mid\ \{l_2; l_3\}\ \mid\ \{C_1; C_2\}\ ])$$

Voici le jugement d'évaluation :

$$t\Gamma_1^c \vdash^c e_1 \mapsto\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \{\quad (\ [\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ]);\ (\ [\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\quad \}\ ]$$

La sémantique abstraite étant une interprétation abstraite de la sémantique collectrice, on peut simuler toute évaluation collectrice par une évaluation abstraite. Pour cela, on commence par abstraire l'environnement d'évaluation :

$$\uparrow_c^a(t\Gamma_1^c) = (x,\ [\ \emptyset\ \mid\ \ [\ \{l_2; l_3\}\ \mid\ \top\ ]\ ]) = t\Gamma_1^a$$

On peut ensuite évaluer e_1 dans l'environnement abstrait à l'aide de la sémantique abstraite. On obtient le jugement suivant :

$$t\Gamma_1^a \vdash^a e_1 \mapsto\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_2; l_3\}\ \mid\ \top\ ],\ [\ \{l_1\}\ \mid\ \top\ ])\ ]\ ]$$

Il ne reste plus qu'à concrétiser la t-valeur abstraite obtenue pour produire une valeur collectrice qui se trouvera être moins précise que celle obtenue directement par la sémantique collectrice.

$$\begin{array}{c} \uparrow_a^c(\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_2; l_3\}\ \mid\ \top\ ],\ [\ \{l_1\}\ \mid\ \top\ ])\ ]\ ]) \\ =\ [\ \emptyset\ \mid\ \emptyset\ \mid\ \{(\ [\ \{l_2; l_3\}\ \mid\ v_1^i\ ],\ [\ \{l_1\}\ \mid\ v_2^i\ ])\mid \forall v_1^i \in \mathcal{V}^i, \forall v_2^i \in \mathcal{V}^i\}\ ] \end{array}$$

Les ensembles de dépendances sont vides et l'ensemble des valeurs simples obtenu directement est inclus dans l'ensemble des valeurs simples obtenu après abstraction :

$$\begin{array}{l} \{(\ [\ \{l_2; l_3\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ]);\ \ \subseteq^{im}\ \ \{(\ [\ \{l_2; l_3\}\ \mid\ v_1^i\ ],\ [\ \{l_1\}\ \mid\ v_2^i\ ])\mid \forall v_1^i \in \mathcal{V}^i, \forall v_2^i \in \mathcal{V}^i\} \\ \ (\ [\ \{l_2; l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\} \end{array}$$

Voici quelques exemples d'évaluations instrumentées correspondant à l'évaluation collectrice présentée ici :

$$(x,\ [\ \emptyset\ \mid\ \{l_2\}\ \mid\ C_1\ ]) \vdash^i e_1 \mapsto\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_2\}\ \mid\ C_1\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ]$$

$$(x,\ [\ \emptyset\ \mid\ \{l_3\}\ \mid\ C_2\ ]) \vdash^i e_1 \mapsto\ [\ \emptyset\ \mid\ \ [\ \emptyset\ \mid\ (\ [\ \{l_3\}\ \mid\ C_2\ ],\ [\ \{l_1\}\ \mid\ 7\ ])\ ]\ ]$$

4.4.3.3 Énoncé formel du théorème

Ce théorème de correction exprime le fait que la sémantique abstraite est une interprétation abstraite de la sémantique collectrice. Autrement dit, pour tout jugement d'évaluation collectrice, il est possible de simuler cette évaluation en commençant par abstraire l'environnement d'évaluation, puis en évaluant l'expression par la sémantique abstraite pour enfin concrétiser la valeur abstraite. La valeur collectrice ainsi obtenue est alors moins précise que celle obtenue directement par la sémantique collectrice.

Pour exprimer formellement ce théorème, il nous faut d'abord définir formellement la relation d'ordre sur les valeurs collectrices qui exprime le fait qu'une valeur collectrice est plus précise qu'une autre. Cette définition est donnée en figure 4.19.

Théorème 4.4.1 (Correction de la sémantique abstraite)**.**
$\forall (t\Gamma^c, e, tu^c, t\Gamma^a, tu^a).$
$t\Gamma^c \vdash^c e \mapsto tu^c \Rightarrow t\Gamma^a = \uparrow_c^a(t\Gamma^c) \Rightarrow t\Gamma^a \vdash^a e \mapsto tu^a \Rightarrow tu^c \subseteq^c \uparrow_a^c(tu^a)$

$$\frac{\textsc{le-c-val}\quad td_1^c \subseteq td_2^c \qquad d_1^c \subseteq d_2^c \qquad vs_1^i \subseteq^{si} vs_2^i}{[\ td_1^c \mid d_1^c \mid vs_1^i\ ] \subseteq^c [\ td_2^c \mid d_2^c \mid vs_2^i\ ]} \qquad \frac{\textsc{le-si-val}\quad \forall v_1^i \in vs_1^i.\ \exists v_2^i \in vs_2^i.\ v_1^i \subseteq^i v_2^i}{vs_1^i \subseteq^{si} vs_2^i}$$

FIGURE 4.19 – Relation d'ordre sur les valeurs collectrices

4.4.3.4 Preuve de correction

La preuve de correction de ce théorème est similaire à la preuve de correction de la sémantique collectrice. Il s'agit de prouver qu'une sémantique est une interprétation abstraite d'une autre, étant données une fonction d'abstraction et une fonction de concrétisation.

On procède par induction sur le jugement d'évaluation de la sémantique abstraite. Il y a 18 cas à prouver, correspondant aux 18 règles d'inférences de la sémantique abstraite. Nous présentons ici la preuve de quelques uns de ces cas.

cas A-NUM Ce cas est trivial, tout comme le cas A-CONSTR-0. Nous allons tout de même l'expliquer rapidement pour illustrer le principe de la preuve qui est également utilisé dans les autres cas.

Nous avons les hypothèses suivantes :

$$t\Gamma^c \vdash^c n \mapsto [\ \emptyset \mid \emptyset \mid \{n\}\ ]$$
$$t\Gamma^a = \uparrow_c^a(t\Gamma^c)$$
$$t\Gamma^a \vdash^a n \mapsto [\ \emptyset \mid [\ \emptyset \mid \top\ ]\ ]$$

Il reste uniquement à prouver la propriété ci-dessous qui est une conséquence directe de la définition de la relation d'ordre $\subseteq^c$ donnée en figure 4.19 :

$$[\ \emptyset \mid \emptyset \mid \{n\}\ ] \subseteq^c \uparrow_a^c([\ \emptyset \mid [\ \emptyset \mid \top\ ]\ ])$$

cas A-CONSTR-1 Ce cas est légèrement plus complexe que le précédent du fait que l'expression évaluée possède une sous-expression.

Nous avons les hypothèses suivantes dont une hypothèse d'induction :

$$\frac{\textsc{c-constr-1}\quad t\Gamma^c \vdash^c e \mapsto [\ td^c \mid d^c \mid vs^i\ ]}{t\Gamma^c \vdash^c D(e) \mapsto [\ td^c \mid \emptyset \mid \{D([\ d^c \mid v^i\ ]) \mid v^i \in vs^i\}\ ]}$$

$$t\Gamma^a = \uparrow_c^a (t\Gamma^c)$$

$$\frac{t\Gamma^a \vdash^a e \mapsto [\ td^a \ \mid \ [\ d^a \ \mid \ v^a\ ]\ ]}{t\Gamma^a \vdash^a D(e) \mapsto [\ td^a \ \mid \ [\ \emptyset \ \mid \ D(\ [\ d^a \ \mid \ v^a\ ])\ ]\ ]}\ \text{A-CONSTR-1}$$

$$[\ td^c \ \mid \ d^c \ \mid \ vs^i\ ] \subseteq^c \uparrow_a^c ([\ td^a \ \mid \ [\ d^a \ \mid \ v^a\ ]\ ])$$

Nous devons alors montrer cette propriété :

$$[\ td^c \ \mid \ \emptyset \ \mid \ \{D(\ [\ d^c \ \mid \ v^i\ ]) \mid v^i \in vs^i\}\ ] \subseteq^c \uparrow_a^c ([\ td^a \ \mid \ [\ \emptyset \ \mid \ D(\ [\ d^a \ \mid \ v^a\ ])\ ]\ ])$$

Une fois simplifiée à l'aide des définitions de la relation d'ordre et de la fonction de concrétisation, il nous reste à prouver les propriétés suivantes :

$$td^c \subseteq td^a \qquad \{D(\ [\ d^c \ \mid \ v^i\ ]) \mid v^i \in vs^i\} \subseteq^c \uparrow_a^c (D(\ [\ d^a \ \mid \ v^a\ ]))$$

Nous simplifions alors la seconde propriété en plusieurs étapes :

$$\forall v_1^i \in \{D(\ [\ d^c \ \mid \ v^i\ ]) \mid v^i \in vs^i\}.\ \exists v_2^i \in \uparrow_a^c (D(\ [\ d^a \ \mid \ v^a\ ])).\ v_1^i \subseteq^i v_2^i$$
$$\forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c (D(\ [\ d^a \ \mid \ v^a\ ])).\ D(\ [\ d^c \ \mid \ v_1^i\ ]) \subseteq^i v_2^i$$
$$\forall v_1^i \in vs^i.\ \exists v_2^i \in \{D(\ [\ d^a \ \mid \ v^i\ ]) \mid \forall v^i \in \uparrow_a^c (v^a)\}.\ D(\ [\ d^c \ \mid \ v_1^i\ ]) \subseteq^i v_2^i$$
$$\forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c (v^a).\ D(\ [\ d^c \ \mid \ v_1^i\ ]) \subseteq^i D(\ [\ d^a \ \mid \ v_2^i\ ])$$
$$\forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c (v^a).\ d^c \subseteq d^a \wedge v_1^i \subseteq^i v_2^i$$
$$d^c \subseteq d^a \qquad \forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c (v^a).\ v_1^i \subseteq^i v_2^i$$

De notre hypothèse d'induction, nous déduisons les propriétés suivantes, en utilisant simplement les définitions de la relation d'ordre et de la fonction de concrétisation :

$$td^c \subseteq td^a \qquad d^c \subseteq d^a \qquad vs^i \subseteq^c \uparrow_a^c (v^a) \qquad \forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c (v^a).\ v_1^i \subseteq^i v_2^i$$

Ce qui nous permet de conclure immédiatement.

cas A-APPLY Comme d'habitude, le cas de l'application fait partie des cas les plus complexes à prouver.

Nous avons les hypothèses suivantes dont trois hypothèses d'induction :

$$\frac{\begin{array}{c} t\Gamma^c \vdash^c e_1 \mapsto [\ td_1^c \ \mid\ d_1^c \ \mid\ vs_1^i\ ] \qquad t\Gamma^c \vdash^c e_2 \mapsto [\ td_2^c \ \mid\ d_2^c \ \mid\ vs_2^i\ ] \\ v^{im} = multiple\_instrumented\_application(td_1^c, d_1^c, vs_1^i, td_2^c, d_2^c, vs_2^i) \\ (\forall l.\ l \in td^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in td^i \wedge [\ td^i \ \mid\ [\ d^i \ \mid\ v^i\ ]\ ] \in v^{im})) \\ (\forall l.\ l \in d^c \Leftrightarrow (\exists (td^i, d^i, v^i).\ l \in d^i \wedge [\ td^i \ \mid\ [\ d^i \ \mid\ v^i\ ]\ ] \in v^{im})) \\ vs^i = \{v^i \mid \exists (td^i, d^i).\ [\ td^i \ \mid\ [\ d^i \ \mid\ v^i\ ]\ ] \in v^{im}\} \end{array}}{t\Gamma^c \vdash^c e_1\ e_2 \mapsto [\ td^c \ \mid\ d^c \ \mid\ vs^i\ ]}\ \text{C-APPLY}$$

$$t\Gamma^a = \uparrow_c^a(t\Gamma^c)$$

$$\frac{\begin{array}{c} t\Gamma^a \vdash^a e_1 \mapsto [\ td_1^a \ \mid\ [\ d_1^a \ \mid < \lambda x.e, \Gamma_1^a >\ ]\ ] \\ t\Gamma^a \vdash^a e_2 \mapsto tu_2^a \qquad tu_2^a = [\ td_2^a \ \mid\ [\ d_2^a \ \mid\ v_2^a\ ]\ ] \\ (x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a) \vdash^a e \mapsto [\ td^a \ \mid\ [\ d^a \ \mid\ v^a\ ]\ ] \end{array}}{t\Gamma^a \vdash^a e_1\ e_2 \mapsto [\ td_1^a \cup td_2^a \cup td^a \cup d_1^a \ \mid\ [\ d_1^a \cup d^a \ \mid\ v^a\ ]\ ]}\ \text{A-APPLY}$$

$$[\ td_1^c \ \mid\ d_1^c \ \mid\ vs_1^i\ ] \subseteq^c \uparrow_a^c([\ td_1^a \ \mid\ [\ d_1^a \ \mid < \lambda x.e, \Gamma_1^a >\ ]\ ])$$

$$[\ td_2^c \ \mid\ d_2^c \ \mid\ vs_2^i\ ] \subseteq^c \uparrow_a^c([\ td_2^a \ \mid\ [\ d_2^a \ \mid\ v_2^a\ ]\ ])$$

$$\forall (t\Gamma^c, tu^c).\ t\Gamma^c \vdash^c e \mapsto tu^c \Rightarrow (x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a) = \uparrow_c^a(t\Gamma^c) \Rightarrow tu^c \subseteq^c \uparrow_a^c([\ td^a \ \mid\ [\ d^a \ \mid\ v^a\ ]\ ])$$

Nous devons alors montrer cette propriété :

$$[\ td^c \ \mid\ d^c \ \mid\ vs^i\ ] \subseteq^c \uparrow_a^c([\ td_1^a \cup td_2^a \cup td^a \cup d_1^a \ \mid\ [\ d_1^a \cup d^a \ \mid\ v^a\ ]\ ])$$

Après simplification, nous devons donc montrer les trois propriétés suivantes :

$$td^c \subseteq td_1^a \cup td_2^a \cup td^a \cup d_1^a \qquad d^c \subseteq d_1^a \cup d^a \qquad \forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c(v^a).\ v_1^i \subseteq^i v_2^i$$

Prouvons tout d'abord quelques propriétés préliminaires concernant l'évaluation collectrice de e ainsi que son évaluation instrumentée multiple.

Puisque la sémantique collectrice est totale, nous savons qu'il existe un jugement d'évaluation collectrice pour e dans l'environnement $\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a))$. Il existe donc une valeur collectrice $[\ td'^c \ \mid\ d'^c \ \mid\ vs'^i\ ]$ telle que :

$$\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)) \vdash^c e \mapsto [\ td'^c \ \mid\ d'^c \ \mid\ vs'^i\ ]$$

La composition de la fonction de concrétisation et de la fonction d'abstraction est l'identité. Nous en déduisons donc la propriété suivante :

$$\uparrow_c^a(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a))) = (x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)$$

On utilise alors la troisième hypothèse d'induction pour établir la relation d'ordre entre les valeurs collectrice et abstraite de e. Nous obtenons :

$$[\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ] \subseteq^c \uparrow_a^c([\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ])$$

La sémantique instrumentée multiple est elle aussi totale. Il existe donc un jugement de la sémantique instrumentée multiple pour l'évaluation de e dans l'ensemble d'environnements instrumentés suivant : $\uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$. Voici le jugement :

$$\frac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in \uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a))).\ t\Gamma^i \vdash^i e \mapsto tu^i\}}{\uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a))) \vdash^{im} e \mapsto v^{im}}\ \text{I-MULTIPLE}$$

Commençons maintenant par prouver la première des trois propriétés :

$$td^c \subseteq td_1^a \cup td_2^a \cup td^a \cup d_1^a$$

Soit $l \in td^c$ on veut alors montrer que $l \in td_1^a \cup td_2^a \cup td^a \cup d_1^a$

D'après la définition de td^c, on a (td^i, d^i, v^i) tels que : $l \in td^i \wedge [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v^{im}$

Il y a alors 2 cas possibles d'après la définition de *multiple_instrumented_application*. Il s'agit soit de l'application d'une fonction non-récursive, soit de l'application d'une fonction récursive. Nous prouvons plus loin que le second cas est impossible. Commençons par nous placer dans le premier cas, on a alors :

$$\begin{gathered}
< \lambda x.e, \Gamma_1^i > \in vs_1^i \qquad v_2^i \in vs_2^i \qquad tu_2^i = [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2^i\ ]\ ] \\
(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ] \\
[\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] = [\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c\ \mid\ [\ d_1^c \cup d'^i\ \mid\ v'^i\ ]\ ]
\end{gathered}$$

On remarque que $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\} \subseteq^{im} \uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$

Puisque la sémantique instrumentée multiple est croissante, la valeur de e dans l'environnement $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\}$ est plus précise que la valeur de e dans l'environnement $\uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$. Ce qui nous donne :

$$\{\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]\} \subseteq^{im}\ v^{im}$$

D'après le théorème de correction de la sémantique collectrice, on a :

$$v^{im} \subseteq^{im} \uparrow_c^{im}([\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ])$$

On utilise alors la transitivité de la relation d'ordre sur les valeurs instrumentées multiples pour déduire :

$$\{ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]\} \subseteq^{im} \uparrow^{im}_{c}(\ [\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ])$$

En composant cette relation ainsi que la relation d'ordre entre la valeur collectrice de e et sa valeur abstraite ($[\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ] \subseteq^c \uparrow^c_a(\ [\ td^a\ \mid\ [\ d^a\ \mid\ v^a\ ]\ ])$), on obtient :

$$td'^i \subseteq td'^c \subseteq td^a$$

D'autre part, nos hypothèses d'induction nous disent que :

$$td^c_1 \subseteq\ td^a_1 \qquad td^c_2 \subseteq\ td^a_2 \qquad d^c_1 \subseteq\ d^a_1$$

Puisque $l \in td^c_1 \cup td^c_2 \cup td'^i \cup d^c_1$, on déduit donc la propriété voulue pour conclure le cas de l'application d'un fonction non-récursive : $l \in td^a_1 \cup td^a_2 \cup td^a \cup d^a_1$

Montrons maintenant que le cas de l'application d'une fonction récursive est absurde. Dans ce cas, on a les propriétés suivantes :

$$< \texttt{rec}f.x.e, \Gamma^i_1 > \in vs^i_1 \qquad v^i_2 \in vs^i_2$$
$$tu^i_f =\ [\ td^c_1\ \mid\ [\ d^c_1\ \mid < \texttt{rec}f.x.e, \Gamma^i_1 >\ ]\ ] \qquad tu^i_2 =\ [\ td^c_2\ \mid\ [\ d^c_2\ \mid\ v^i_2\ ]\ ]$$
$$(f, tu^i_f) \oplus (x, tu^i_2) \oplus \uparrow^{ti}_i(\Gamma^i_1) \vdash^i e \mapsto\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$$
$$[\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] =\ [\ td^c_1 \cup td^c_2 \cup td'^i \cup d^c_1\ \mid\ [\ d^c_1 \cup d'^i\ \mid\ v'^i\ ]\ ]$$

Or la première propriété $< \texttt{rec}f.x.e, \Gamma^i_1 > \in vs^i_1$ contredit notre première hypothèse d'induction qui implique que $vs^i_1 \subseteq^{im} \uparrow^c_a(< \lambda x.e, \Gamma^a_1 >)$

Ceci nous permet de conclure la preuve de $td^c \subseteq td^a_1 \cup td^a_2 \cup td^a \cup d^a_1$

On procède de même pour montrer que $d^c \subseteq d^a_1 \cup d^a$

Soit $l \in d^c$ on veut alors montrer que $l \in d^a_1 \cup d^a$

D'après la définition de d^c, on a (td^i, d^i, v^i) tels que : $l \in d^i \wedge [\ td^i\ \mid\ [\ d^i\ \mid\ v^i\ ]\ ] \in v^{im}$

Il y a alors 2 cas possibles d'après la définition de *multiple_instrumented_application*. Il s'agit soit de l'application d'une fonction non-récursive, soit de l'application d'une fonction récursive. Pour la même raison que dans le cas précédent, le cas de l'application d'une fonction récursive est absurde. Plaçons nous donc dans le premier cas, on a alors :

$$< \lambda x.e, \Gamma_1^i > \in vs_1^i \qquad v_2^i \in vs_2^i \qquad tu_2^i = [\ td_2^c \ \mid \ [\ d_2^c \ \mid \ v_2^i\ ]\ ]$$
$$(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]$$
$$[\ td^i \ \mid \ [\ d^i \ \mid \ v^i\ ]\ ] = [\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c \ \mid \ [\ d_1^c \cup d'^i \ \mid \ v'^i\ ]\ ]$$

On remarque que $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\} \subseteq^{im} \uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$

Puisque la sémantique instrumentée multiple est croissante, la valeur de e dans l'environnement $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\}$ est plus précise que la valeur de e dans l'environnement $\uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$. Ce qui nous donne :

$$\{\ [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]\} \subseteq^{im} \ v^{im}$$

D'après le théorème de correction de la sémantique collectrice, on a :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ td'^c \ \mid \ d'^c \ \mid \ vs'^i\ ])$$

On utilise alors la transitivité de la relation d'ordre sur les valeurs instrumentées multiples pour déduire :

$$\{\ [\ td'^i \ \mid \ [\ d'^i \ \mid \ v'^i\ ]\ ]\} \subseteq^{im} \uparrow_c^{im}(\ [\ td'^c \ \mid \ d'^c \ \mid \ vs'^i\ ])$$

En composant cette relation ainsi que la relation d'ordre entre la valeur collectrice de e et sa valeur abstraite, on obtient $d'^i \subseteq d'^c \subseteq d^a$

D'autre part, nos hypothèses d'induction nous disent que :

$$d_1^c \subseteq \ d_1^a$$

Puisque $l \in d_1^c \cup d'^i$, on déduit donc la propriété voulue pour conclure : $l \in d_1^a \cup d^a$

Ceci nous permet de conclure la preuve de $d^c \subseteq d_1^a \cup d^a$

Il nous reste alors à montrer cette troisième et dernière propriété :

$$\forall v_1^i \in vs^i.\ \exists v_2^i \in \uparrow_a^c(v^a).\ v_1^i \subseteq^i v_2^i$$

Pour cela, on prend $v_1^i \in vs^i$ et on procède une troisième fois de la même manière, en faisant le lien entre l'évaluation instrumentée correspondant à v_1^i et l'évaluation abstraite de e.

D'après la définition de vs^i, on a (td^i, d^i) tels que : $[\ td^i\ \mid\ [\ d^i\ \mid\ v_1^i\ ]\ ] \in v^{im}$

Il y a là aussi 2 cas possibles d'après la définition de *multiple_instrumented_application*. Il s'agit soit de l'application d'une fonction non-récursive, soit de l'application d'une fonction récursive. Pour la même raison que précédemment, le cas de l'application d'une fonction récursive est absurde. Plaçons nous donc dans le premier cas, on a alors :

$$< \lambda x.e, \Gamma_1^i > \in vs_1^i \qquad v_2^i \in vs_2^i \qquad tu_2^i = [\ td_2^c\ \mid\ [\ d_2^c\ \mid\ v_2^i\ ]\ ]$$
$$(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i) \vdash^i e \mapsto [\ td'^i\ \mid\ [\ d'^i\ \mid\ v'^i\ ]\ ]$$
$$[\ td^i\ \mid\ [\ d^i\ \mid\ v_1^i\ ]\ ] = [\ td_1^c \cup td_2^c \cup td'^i \cup d_1^c\ \mid\ [\ d_1^c \cup d'^i\ \mid\ v'^i\ ]\ ]$$

On remarque que $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\} \subseteq^{im} \uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$

Puisque la sémantique instrumentée multiple est croissante, la valeur de e dans l'environnement $\{(x, tu_2^i) \oplus \uparrow_i^{ti}(\Gamma_1^i)\}$ est plus précise que la valeur de e dans l'environnement $\uparrow_c^{im}(\uparrow_a^c((x, tu_2^a) \oplus \uparrow_a^{ta}(\Gamma_1^a)))$. Ce qui nous donne :

$$\{\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v_1^i\ ]\ ]\} \subseteq^{im}\ v^{im}$$

D'après le théorème de correction de la sémantique collectrice, on a :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(\ [\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ])$$

On utilise alors la transitivité de la relation d'ordre sur les valeurs instrumentées multiples pour déduire :

$$\{\ [\ td'^i\ \mid\ [\ d'^i\ \mid\ v_1^i\ ]\ ]\} \subseteq^{im} \uparrow_c^{im}(\ [\ td'^c\ \mid\ d'^c\ \mid\ vs'^i\ ])$$

De cette dernière relation, nous déduisons qu'il existe une valeur $v_2'^i \in vs'^i$ telle que :

$$v_1^i \subseteq^i v_2'^i$$

Nous utilisons alors la relation d'ordre entre la valeur collectrice de e et sa valeur abstraite pour déduire qu'il existe une valeur $v_2^i \in \uparrow_a^c(v^a)$ telle que :

$$v_2'^i \subseteq^i v_2^i$$

La transitivité de la relation d'ordre nous permet alors de conclure :

$$v_1^i \subseteq^i v_2^i$$

La preuve du cas A-APPLY de notre induction est alors terminée.

4.5 Correction de l'analyse statique

Nous établissons enfin la correction de notre analyse statique. L'analyse statique ainsi que l'ensemble des sémantiques intermédiaires ont été introduites dans le but de simplifier la preuve du théorème présenté ici et de la rendre modulaire.

Cette modularité permettra de réutiliser une grande partie des preuves présentées ici lors d'une modification de l'analyse statique. Par exemple, si on modifie l'algèbre des valeurs abstraites afin de proposer une représentation différente des fonctions ou bien si on modifie les règles d'inférence pour améliorer la précision ou l'efficacité de l'analyse, il suffira de reprendre la preuve de la sémantique abstraite par rapport à la sémantique collectrice sans avoir à refaire les preuves des sémantiques précédentes.

4.5.1 Énoncé informel du théorème

De même que pour la correction de l'analyse dynamique, notre but est de pouvoir affirmer que si un label n'apparaît pas dans le résultat de l'analyse d'un programme alors ce label n'a pas d'impact sur l'évaluation de ce programme.

La différence essentielle entre l'analyse dynamique et l'analyse statique est que cette dernière considère de façon simultanée un ensemble éventuellement infini d'évaluations possibles. En effet, l'environnement abstrait utilisé lors de l'analyse statique correspond à un ensemble éventuellement infini d'environnements possibles alors que l'environnement instrumenté utilisé lors de l'analyse dynamique correspond à un environnement de référence unique.

Plus précisément, après avoir obtenu un jugement d'évaluation abstraite du programme à analyser et après avoir vérifié que le label en question n'apparaît pas dans les dépendances calculées alors on peut affirmer que peu importe l'injection considérée sur ce label, l'évaluation avec injection de ce programme dans n'importe quel environnement possible retournera toujours la valeur de référence du programme (la valeur du programme lors d'une évaluation sans injection).

4.5.2 Illustration par l'exemple

Exemple 1 : évaluation d'un couple

Notons e_1 le programme suivant : (3, l:7)
Son jugement d'évaluation abstraite dans le t-environnent abstrait vide est :

$$\emptyset \vdash^a e_1 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ \emptyset\ \mid\ \top\ ],\ [\ l\ \mid\ \top\ ])\ ]\ ]$$

Seul le label l apparaît dans la t-valeur abstraite de e_1. On peut donc affirmer que tout autre label n'a pas d'impact sur l'évaluation de e_1. On a en effet pour toute injection $(l', v_{l'})$ sur un label l' différent de l :

$$\emptyset \vdash_{l':v_{l'}} e_1 \mapsto (3, 7)$$

Exemple 2 : liaison d'une fonction

Notons e_2 le programme suivant :

```
let f o =
  match o with
  | Some(x) → x + l1 : 8
  | none → d + l2 : 8
in
(f (Some 18), l3 : 17)
```

Définissons le t-environnement abstrait dans lequel nous souhaitons effectuer l'analyse statique du programme e_2 :

$$t\Gamma_2^a = (d,\ [\ td_d^a\ \mid\ [\ d_d^a\ \mid\ \top\ ]\ ])$$

Le jugement d'évaluation abstraite de e_2 dans cet environnement est :

$$t\Gamma_2^a \vdash^a e_2 \mapsto [\ \emptyset\ \mid\ [\ \emptyset\ \mid\ (\ [\ l_1\ \mid\ \top\ ],\ [\ l_3\ \mid\ \top\ ])\ ]\ ]$$

Le label l_2 n'étant pas présent dans la t-valeur abstraite du programme e_2, nous pouvons en conclure qu'aucune injection sur l_2 n'a d'impact sur l'évaluation de e_2 et ceci, quelque

soit la valeur de d dans l'environnement :

$$\forall(v_{l_2}, v_d).(d, v_d) \vdash_{l_2:v_{l_2}} e_2 \mapsto (26, 17)$$

Les dépendances de l'identificateur d présent dans l'environnement ne sont pas présentes non plus, ce qui indique que la valeur de cette identificateur n'a aucune influence sur le résultat du programme.

Par contre, les labels l_1 et l_3 apparaissent dans la t-valeur instrumentée du programme e_2. Il n'est donc pas impossible qu'une injection sur un de ces labels puisse modifier le comportement du programme, comme le montrent les exemples ci-dessous :

$$(d, 26) \vdash_{l_1:24} e_2 \mapsto (42, 17)$$

$$(d, 32) \vdash_{l_3:613} e_2 \mapsto (26, 613)$$

4.5.3 Énoncé formel du théorème

Pour énoncer formellement le théorème de correction, il nous faut tout d'abord définir formellement les notions de non-apparition d'un label dans une valeur abstraite et dans un t-environnement abstrait. On définit pour cela les prédicats $ldna\_in\_eval^a$, $ldna\_in\_val^a$ et $ldna\_in\_env^a$. Leur définition est donnée respectivement en figure 4.20, en figure 4.21 et en figure 4.22.

Théorème 4.5.1 (Correction de l'analyse statique)**.**
$\forall(t\Gamma^a, e, tu^a, l).\ t\Gamma^a \vdash^a e \mapsto tu^a \Rightarrow ldna\_in\_env^a(l, t\Gamma^a) \Rightarrow ldna\_in\_val^a(l, tu^a)$
$\Rightarrow \forall t\Gamma^i \in \uparrow_c^{im}(\uparrow_a^c(t\Gamma^a)).\forall v_l.t\Gamma^i \vdash^i e \mapsto tu^i \Rightarrow \uparrow_{ti}(t\Gamma^i) \vdash_{l:v_l} e \mapsto \uparrow_{ti}(tu^i)$

4.5.4 Preuve de correction

Nous avons les hypothèses suivantes :

$$t\Gamma^a \vdash^a e \mapsto tu^a \qquad ldna\_in\_env^a(l, t\Gamma^a) \qquad ldna\_in\_val^a(l, tu^a)$$
$$t\Gamma^i \in \uparrow_c^{im}(\uparrow_a^c(t\Gamma^a)) \qquad t\Gamma^i \vdash^i e \mapsto tu^i$$

$$\textsc{ldna-e-a-num}\ \frac{}{ldna\_in\_eval^a(l, t\Gamma^a, n)} \qquad \textsc{ldna-e-a-constr-0}\ \frac{}{ldna\_in\_eval^a(l, t\Gamma^a, C)} \qquad \textsc{ldna-e-a-constr-1}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e)}{ldna\_in\_eval^a(l, t\Gamma^a, D(e))}$$

$$\textsc{ldna-e-a-ident}\ \frac{ldna\_in\_val^a(l, t\Gamma^a[x])}{ldna\_in\_eval^a(l, t\Gamma^a, x)} \qquad \textsc{ldna-e-a-ident-unbound}\ \frac{x \notin support(t\Gamma^a)}{ldna\_in\_eval^a(l, t\Gamma^a, x)}$$

$$\textsc{ldna-e-a-abstr}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e)}{ldna\_in\_eval^a(l, t\Gamma^a, \lambda x.e)} \qquad \textsc{ldna-e-a-abstr-rec}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e)}{ldna\_in\_eval^a(l, t\Gamma^a, \texttt{rec} f.x.e)}$$

$$\textsc{ldna-e-a-apply}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e_1) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_2)}{ldna\_in\_eval^a(l, t\Gamma^a, e_1\ e_2)} \qquad \textsc{ldna-e-a-if}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_1) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_2)}{ldna\_in\_eval^a(l, t\Gamma^a, if\ e\ then\ e_1\ else\ e_2)}$$

$$\textsc{ldna-e-a-match}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_1) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_2)}{ldna\_in\_eval^a(l, t\Gamma^a, match\ e\ with\ p \to e_1 \mid x \to e_2)} \qquad \textsc{ldna-e-a-annot}\ \frac{l \neq l' \quad ldna\_in\_eval^a(l, t\Gamma^a, e)}{ldna\_in\_eval^a(l, t\Gamma^a, l\ :\ e)}$$

$$\textsc{ldna-e-a-couple}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e_1) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_2)}{ldna\_in\_eval^a(l, t\Gamma^a, (e_1, e_2))} \qquad \textsc{ldna-e-a-letin}\ \frac{ldna\_in\_eval^a(l, t\Gamma^a, e_1) \quad ldna\_in\_eval^a(l, t\Gamma^a, e_2)}{ldna\_in\_eval^a(l, t\Gamma^a, let\ x\ =\ e_1\ in\ e_2)}$$

FIGURE 4.20 – Prédicat de non-apparition d'un label lors d'une évaluation abstraite

$$\textsc{ldna-v-a-top}\ \frac{}{ldna\_in\_val^a(l, \top)} \qquad \textsc{ldna-v-a-bottom}\ \frac{}{ldna\_in\_val^a(l, \bot)} \qquad \textsc{ldna-v-a-constr-0}\ \frac{}{ldna\_in\_val^a(l, C)} \qquad \textsc{ldna-v-a-constr-1}\ \frac{ldna\_in\_val^a(l, u)}{ldna\_in\_val^a(l, D(u))}$$

$$\textsc{ldna-v-a-couple}\ \frac{ldna\_in\_val^a(l, u_1) \quad ldna\_in\_val^a(l, u_2)}{ldna\_in\_val^a(l, (u_1, u_2))} \qquad \textsc{ldna-v-a-closure}\ \frac{ldna\_in\_eval^a(l, \uparrow_a^{ta}(\Gamma^a), e)}{ldna\_in\_val^a(l, <\lambda x.e, \Gamma^a>)} \qquad \textsc{ldna-v-a-closure-rec}\ \frac{ldna\_in\_eval^a(l, \uparrow_a^{ta}(\Gamma^a), e)}{ldna\_in\_val^a(l, <\texttt{rec} f.x.e, \Gamma^a>)}$$

$$\textsc{ldna-v-a-v-val}\ \frac{l \notin d^a \qquad ldna\_in\_val^a(l, v^a)}{ldna\_in\_val^a(l, [\ d^a\ \mid\ v^a\ ])} \qquad \textsc{ldna-v-a-t-val}\ \frac{l \notin td^a \qquad ldna\_in\_val^a(l, u^a)}{ldna\_in\_val^a(l, [\ td^a\ \mid\ u^a\ ])}$$

FIGURE 4.21 – Prédicat de non-apparition d'un label dans une valeur abstraite

$$\textsc{ldna-env-a-empty}\ \frac{}{ldna\_in\_env^a(l, \emptyset)} \qquad \textsc{ldna-env-a-cons}\ \frac{ldna\_in\_val^a(l, tu^a) \qquad ldna\_in\_val^a(l, t\Gamma^a)}{ldna\_in\_val^a(l, (tu^a) \oplus t\Gamma^a)}$$

FIGURE 4.22 – Prédicat de non-apparition d'un label dans un environnement abstrait

Nous voulons montrer :

$$\uparrow_{ti}(t\Gamma^i) \vdash_{l:v_l} e \mapsto \uparrow_{ti}(tu^i)$$

Tout d'abord, nous utilisons le fait que les sémantiques collectrice et instrumentée multiple sont totales pour déduire les jugements d'évaluation suivants :

$$\uparrow_a^c(t\Gamma^a) \vdash^c e \mapsto tu^c$$

$$\frac{v^{im} = \{tu^i \mid \exists t\Gamma^i \in \uparrow_c^{im}(\uparrow_a^c(t\Gamma^a)).\ t\Gamma^i \vdash^i e \mapsto tu^i\}}{\uparrow_c^{im}(\uparrow_a^c(t\Gamma^a)) \vdash^{im} e \mapsto v^{im}} \text{ I-MULTIPLE}$$

Nous appliquons alors le théorème de correction de la sémantique abstraite (cf. section 4.4.3.3) en utilisant le fait que $\uparrow_c^a(\uparrow_a^c(t\Gamma^a)) = t\Gamma^a$, ce qui nous donne :

$$tu^c \subseteq^c \uparrow_a^c(tu^a)$$

Nous pouvons alors appliquer le théorème de correction de la sémantique collectrice (cf. section 4.3.3.3) en utilisant le fait que $\uparrow_{im}^c(\uparrow_c^{im}(\uparrow_a^c(t\Gamma^a))) = \uparrow_a^c(t\Gamma^a)$, ce qui nous donne :

$$v^{im} \subseteq^{im} \uparrow_c^{im}(tu^c)$$

On remarque que puisque $ldna\_in\_env^a(l, t\Gamma^a)$ et $t\Gamma^i \in \uparrow_c^{im}(\uparrow_a^c(t\Gamma^a))$ alors on a $ldna\_in\_env^{ti}(l, t\Gamma^i)$

D'autre part, on remarque que $tu^i \in v^{im}$.

En utilisant la relation d'ordre issue de la correction de la sémantique collectrice on déduit qu'il existe un $tu_2^i \in \uparrow_c^{im}(tu^c)$ tel que $tu^i \subseteq^i tu_2^i$.

Puis en utilisant la relation d'ordre issue de la correction de la sémantique abstraite on déduit qu'il existe un $tu_3^i \in \uparrow_c^{im}(\uparrow_a^c(tu^a))$ tel que $tu_2^i \subseteq^i tu_3^i$.

Enfin, puisque $ldna\_in\_val^a(l, tu^a)$ alors $ldna\_in\_val^{ti}(l, tu_3^i)$ et $ldna\_in\_val^{ti}(l, tu^i)$.

Nous appliquons enfin le théorème de correction de l'analyse dynamique pour conclure :

$$\uparrow_{ti}(t\Gamma^i) \vdash_{l:v_l} e \mapsto \uparrow_{ti}(tu^i)$$

Chapitre 5

Implémentation et preuve

Nous présentons dans ce dernier chapitre les développements effectués autour de l'analyse de dépendances. D'une part, nous discutons des différents prototypes réalisés. D'autre part, nous exposons le développement au sein de l'assistant à la preuve Coq, qui a permis de formaliser les différentes sémantiques présentées dans cette thèse ainsi que de construire et vérifier leur preuve de correction.

5.1 Prototypes implémentés en OCaml

Trois prototypes ont été développés sous la forme d'une implémantation directe en OCaml. Le premier prototype correspond à l'analyse présentée dans [ABDP12]. Il a permis de cerner les besoins de l'analyse et les améliorations possibles. Un exemple de programme sur lequel nous avons testé ce premier prototype est disponible dans [ABDP12].

Les deux autres prototypes correspondent respectivement à notre analyse dynamique et à notre analyse statique. Cependant, ces derniers ont été développés dans une phase préliminaire, avant de réaliser le développement Coq. Il en résulte qu'ils ne sont pas à jour par rapport aux analyses présentées dans la thèse. En effet, le travail de preuve réalisé en Coq a permis d'identifier différents problèmes qui nous ont conduit à modifier la définition de nos analyses. En particulier, c'est lors de la preuve Coq de la sémantique instrumentée que nous avons identifié la nécessité de faire la distinction entre les t-dépendances et les v-dépendances.

Afin d'obtenir une implémentation à jour par rapport au développement Coq, nous

envisageons une extraction vers OCaml des définitions inductives des sémantiques correspondantes (cf. 5.3).

Il convient de noter que les triplets ont été encodés sous forme de couples imbriqués et le filtrage sur les triplets à été traduit en une imbrication de filtrages de tête à deux branches et d'expressions conditionnelles. Afin de tester des exemples plus complexes, il serait bon d'ajouter aux prototypes un préprocesseur permettant de traduire automatiquement un langage de programmation réaliste vers notre langage noyau. On aurait besoin en particulier d'une traduction d'un filtrage par motifs quelconque vers notre filtrage de tête à deux branches. Ce type de traduction est bien connu, il suffirait de l'intégrer à notre prototype afin de pouvoir accepter des programmes écrits de façon plus naturelle.

5.2 Développement Coq

5.2.1 Contenu du développement

Les sémantiques présentées dans cette thèse permettant de définir et de prouver la correction de nos analyses dynamique et statique ont été formalisées en Coq. Cette formalisation prend la forme d'un ensemble de définitions inductives pour la sémantique opérationnelle, la sémantique avec injection, la sémantique sur-instrumentée, la sémantique instrumentée, la sémantique instrumentée multiple, la sémantique collectrice ainsi que la sémantique abstraite.

Le développement Coq est constitué de la définition de ces 7 sémantiques ainsi que des algèbres de valeurs correspondantes, de 6 théorèmes concernant la correction des sémantiques et 233 lemmes intermédiaires utilisés pour la preuve des théorèmes.

Le code Coq donnant la définition des sémantiques est donné en annexe A. Le code Coq des énoncés des différents théorèmes de correction des sémantiques est donné en annexe B.

Le code Coq complet, disponible en ligne [BVi14] contient 16 621 lignes de code réparties de la manière suivante :

Définition des algèbres de valeurs :	3%
Définition des sémantiques :	7%
Conversions entres les algèbres de valeurs :	3%
Corps des preuves :	30%
Lemmes intermédiaires :	57%

La preuve de correction de la sémantique abstraite par rapport à la sémantique collectrice n'a pas été terminée par manque de temps. Cependant, la preuve a été suffisamment avancée sur papier pour garantir un certain niveau de confiance. Pour compléter le travail de preuve formelle présenté ici, il faudra envisager de terminer cette dernière étape de la preuve.

5.2.2 Intérêt du choix de Coq

Le fait d'avoir choisi Coq pour formaliser et prouver nos différentes sémantiques plutôt que d'opter pour une version papier uniquement offre bien entendu une confiance accrue en la correction de notre analyse. Cependant, ce n'est pas le seul bénéfice. En effet, en tant qu'assistant à la preuve, Coq nous a imposé une certaine rigueur qui nous a permis d'identifier des problèmes dans la définition des sémantiques ainsi que dans l'approche de la preuve.

En particulier, la distinction entre les t-dépendances et les v-dépendances s'est imposée au cours de la preuve Coq de la sémantique instrumentée. Lorsque nous avons entamé cette preuve, il n'y avait qu'une seule notion de dépendance et la sémantique avec injection n'était définie que dans un environnement opérationnel. Nous avons essayé de prouver la propriété suivante : « si un label n'apparaît pas dans la valeur instrumentée d'une expresssion alors deux évaluations quelconques avec injetions sur ce label fournissent forcément la même valeur ». Une preuve par induction a été entamée et lorsque nous sommes arrivés au cas de l'évaluation d'une expression de liaison, il n'a pas été possible d'en faire la preuve. Nous avons rapidement compris la nécessité de définir une algèbre de valeurs contenant l'information nécessaire pour connaître la valeur de l'expression considérée pour toute injection possible. C'est ainsi que l'algèbre des valeurs sur-instrumentées est née, dans le but de définir la sémantique avec injection dans des environnements sur-instrumentés. Il nous a ensuite fallu faire le lien entre cette nouvelle sémantique avec injection et la sémantique

instrumentée, ce qui a donné naissance à la sémantique sur-instrumentée. Et c'est enfin lors de la définition formelle en Coq de la sémantique sur-instrumentée que nous avons identifié la nécessité de séparer les t-dépendances des v-dépendances.

Coq nous a ainsi permis d'identifier rapidement un trou dans notre définition formelle de la notion d'injection, ce qui nous a entraîné dans un processus qui aurait été probablement beaucoup plus long sans l'aide de l'assistant à la preuve formelle.

5.3 Extraction de Coq vers OCaml

Une piste intéressante pour l'implémentation de notre analyse de dépendances est l'extraction à partir de Coq vers OCaml. Les différentes sémantiques que nous avons présentées ont toutes été formalisées en Coq. Leurs jugements d'évaluation sont représentés sous forme de prédicats définis inductivement. Ce style de définition ne fournit pas de fonction d'évaluation. Cependant, les travaux de P.N. Tollitte et al. [TDD12] ont rendu possible l'extraction de code à partir de prédicats définis inductivement.

Leur outil [CRE13] permettrait ainsi d'extraire vers OCaml, à partir de nos définitions Coq, une implémentation de nos analyses dynamique et statique. Ces implémentations auraient l'avantage de correspondre parfaitement à leurs définitions formelle et de pouvoir être mis à jour facilement en cas de nouvelle version de l'analyse.

Il serait éventuellement possible d'obtenir une garantie formelle de la correction de l'implémentation en utilisant leur outil [CRE13] d'extraction d'une relation inductive Coq vers une fonction Coq, puis en utilisant l'extraction [CEC] des fonctions Coq vers OCaml.

Outre les différentes sémantiques intermédiaires non-calculables que nous avons définies pour les besoins de la preuve et qu'il n'est pas question d'implémenter, nous pouvons nous intéresser à l'extraction de la sémantique instrumentée et de la sémantique abstraite. Ces deux sémantiques calculables constituent respectivement la définition de notre analyse dynamique et de notre analyse statique.

Conclusion

Bilan

Nous avons présenté dans cette thèse une analyse statique de programmes ML permettant de cerner l'impact d'une injection (dysfonctionnement ou modification de code) sur le comportement d'un programme et sur sa valeur. Cette analyse de dépendances a été spécialement étudiée pour répondre aux besoins exprimés par des évaluateurs de logiciels critiques. De plus, afin d'offrir un niveau de confiance élevé, nous avons effectué une preuve de correction de notre analyse à l'aide de l'assistant à la preuve Coq.

Afin de parvenir à ce résultat, nous avons commencé par fournir une définition formelle de la notion d'impact. Cette définition, basée directement sur la sémantique opérationnelle du langage, nous a permis de donner un sens précis à la notion de dépendance. Nous nous sommes alors appliqués à prouver la correction de notre analyse vis-à-vis de cette définition formelle.

La preuve formelle s'est déroulée en deux temps. Il a tout d'abord fallu enrichir la sémantique opérationnelle pour qu'elle puisse transporter les informations de dépendance nécessaires à l'analyse, ce qui a donné lieu à la notion de sémantique sur-instrumentée. Ensuite, nous avons procédé par abstractions successives, jusqu'à obtenir une analyse statique permettant d'analyser des programmes dans un environnement partiellement inconnu. La première de ces abstractions a abouti à la définition de la sémantique instrumentée qui constitue une analyse dynamique calculable des dépendances d'un programme. Nous avons ensuite étendu la sémantique instrumentée afin de pouvoir effectuer un ensemble d'évaluations simultanément, étant donné un ensemble d'environnements d'évaluation. C'est ce que nous avons appelé la sémantique instrumentée multiple. À partir de cette dernière, nous avons encore effectué deux abstractions successives afin d'obtenir la sémantique collectrice puis enfin la sémantique abstraite. C'est cette dernière qui constitue l'analyse statique désirée.

Toutes ces sémantiques ont été formalisées en Coq. Le code source, contenant également les preuves de correction, est disponible à l'adresse suivante :
`https://github.com/vincent-benayoun/PhD-thesis`.

Perspectives

Vers un outil utilisable dans l'industrie Ce travail constitue un premier pas vers un outil utilisable en pratique. Il donne une base théorique solide à l'analyse d'impact en définissant formellement la notion d'impact et en proposant une manière efficace de cerner l'impact d'un éventuel dysfonctionnement par l'analyse directe du code source du programme. Il reste encore plusieurs étapes à franchir avant de pouvoir utiliser cette analyse en pratique.

Tout d'abord, il faudra terminer la preuve Coq du théorème de correction de la sémantique abstraite vis-à-vis de la sémantique collectrice. Celle-ci n'a pas été terminée par manque de temps. Cependant, nous avons entamé de façon significative la preuve sur papier 4.4.3.4 en procédant par induction sur le jugement d'évaluation de la sémantique abstraite. Parmi les dix-huit cas de l'induction, nous en avons traité quatre, dont le cas de l'application qui est vraisemblablement un des cas les plus complexes à prouver. Cette preuve permettra de clôturer la preuve complète de l'analyse statique.

Ensuite, il faudra envisager d'étendre le langage sur lequel s'effectue l'analyse pour prendre en compte les fonctionnalités présentes dans les langages ML utilisés dans l'industrie. En particulier il sera nécessaire de gérer les variables mutables et éventuellement les tableaux de valeurs mutables. Cette extension nécessitera probablement une modification de toutes les sémantiques, ce qui aura sans doute un impact conséquent sur les preuves des théorèmes de correction. Une piste intéressante pour éviter la modification de toutes les sémantiques serait d'utiliser une traduction pour transformer tout programme en un programme équivalent ne contenant pas de mutables. Ce type de traduction peut être réalisé de façon naïve en encodant l'environnement des mutables dans le langage et en l'ajoutant en argument et en valeur de retour de toutes les fonctions. On peut aussi utiliser une analyse d'effets pour n'ajouter en argument que les valeurs nécessaires.

Le but de l'analyse est de fournir au développeur ainsi qu'à l'évaluateur de logiciels critiques un outil de spécification et de vérification du flot d'information, à l'image de celui présent dans SPARK. Il faudra donc définir un langage d'annotation permettant de spécifier les relations entre les entrées et les sorties d'un sous-programme. On pourra

ensuite utiliser notre analyse pour inférer les dépendances du sous-programme et vérifier l'adéquation entre les dépendances inférées et la spécification donnée par l'utilisateur.

Optimisation de l'analyse Outre le travail consistant à compléter l'analyse pour en faire un outil utilisable en pratique, différentes pistes d'optimisation de l'analyse sont envisageables. Certaines permettront d'améliorer les performances de l'analyse afin de faciliter le passage à l'échelle pour analyser des programmes de taille importante. D'autres permettront de raffiner l'analyse pour obtenir des résultats plus précis.

En particulier, deux pistes nous semblent prometteuses : l'amélioration de la représentation abstraite des fonctions dans l'algèbre des valeurs et l'ajout d'une représentation abstraite pour les types sommes.

La représentation actuelle des fonctions dans l'algèbre des valeurs abstraites se présente sous forme d'une fermeture. Celle-ci contient le corps de la fonction qui est analysé à chaque endroit du programme où la fonction est appliquée à un argument. L'idée d'amélioration est de trouver une représentation compacte des fonctions ne contenant pas leur corps mais uniquement une information synthétique concernant le lien entre la valeur de retour de la fonction et son paramètre. Une telle représentation ressemblerait probablement aux types des fonctions dans FlowCaml. L'avantage serait double. D'une part le temps d'exécution de l'analyse pourrait être considérablement réduit en analysant le corps de chaque fonction une fois pour toute (probablement à l'aide d'une exécution symbolique) et non à chaque site d'appel de la fonction. D'autre part, l'analyse pourrait gagner en précision puisque l'union de deux fonctions abstraites ne serait plus nécessairement la valeur abstraite $\top$ mais pourrait être une approximation moins brutale dans le treillis des valeurs abstraites.

En ce qui concerne les types sommes, le but serait d'améliorer la précision de l'analyse. Le fait de pouvoir représenter les types sommes dans l'algèbre des valeurs abstraites nous permettrait de rendre plus riche le treillis des valeurs abstraites. On pourrait ainsi profiter de connaissances plus fines sur les valeurs possibles d'une expression filtrée afin d'éliminer certains cas du filtrage.

Bibliographie

[ABDP12] P. Ayrault, V. Benayoun, C. Dubois, and F. Pessaux. ML Dependency Analysis for Assessors. In *International Conference on Software Engineering and Formal Methods (SEFM'12)*, volume 7504 of *LNCS*, pages 278–292, Thessaloniki, Greece, October 2012.

[ABHR99] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In Andrew W. Appel and Alex Aiken, editors, *POPL*, pages 147–160. ACM, 1999.

[ACL03] June Andronick, Boutheina Chetali, and Olivier Ly. Using coq to verify java card tm applet isolation properties. In David Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 335–351. Springer Berlin Heidelberg, 2003.

[ALL96] Martín Abadi, Butler W. Lampson, and Jean-Jacques Lévy. Analysis and caching of dependencies. In Robert Harper and Richard L. Wexelblat, editors, *ICFP*, pages 83–91. ACM, 1996.

[Ayr11] Philippe Ayrault. *Développement de logiciel critique en Focalize. Méthodologie et outils pour l'évaluation de conformité.* PhD thesis, Université Pierre et Marie Curie - LIP6, 2011.

[Bar03] John Barnes. *High Integrity Software : The SPARK Approach to Safety and Security.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[BHA86] Geoffrey L Burn, Chris Hankin, and Samson Abramsky. Strictness analysis for higher-order functions. *Science of Computer Programming*, 7 :249–278, 1986.

[BVi14] Preuve de correction de l'analyse de dépendance en Coq. `https://github.com/vincent-benayoun/PhD-thesis`, 2014.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, pages 238–252, New York, NY, USA, 1977. ACM.

[CC79] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '79, pages 269–282, New York, NY, USA, 1979. ACM.

[CEC] Extraction Coq vers du code fonctionnel. `http://coq.inria.fr/refman/Reference-Manual025.html`.

[CH04] Roderick Chapman and Adrian Hilton. Enforcing security and safety models with an information flow analysis tool. *Ada Lett.*, XXIV(4) :39–46, November 2004.

[CKK+12] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. Frama-c - a software analysis perspective. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *SEFM*, volume 7504 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2012.

[CMP10] Dumitru Ceara, Laurent Mounier, and Marie-Laure Potet. Taint dependency sequences : A characterization of insecure execution paths based on input-sensitive cause sequences. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '10, pages 371–380, Washington, DC, USA, 2010. IEEE Computer Society.

[CRE13] Contribution Coq : RelationExtraction. `http://coq.inria.fr/pylons/contribs/view/RelationExtraction/v8.4`, 2013.

[FHJ+06] Ansgar Fehnker, Ralf Huuck, Patrick Jayet, Michel Lussenburg, and Felix Rauch. Goanna - a static model checker. In Lubos Brim, Boudewijn R. Ha-

verkort, Martin Leucker, and Jaco van de Pol, editors, *FMICS/PDMC*, volume 4346 of *Lecture Notes in Computer Science*, pages 297–300. Springer, 2006.

[Flo03] The Flow Caml System - Documentation and user's manual. `http://www.normalesup.org/~simonet/soft/flowcaml/flowcaml-manual.pdf`, 2003.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[Hun91] Sebastian Hunt. *Abstract Interpretation of Functional Languages : From Theory to Practice*. PhD thesis, Department of Computing, Imperial College of Science Technology and Medicine, 1991.

[JES00] Simon L. Peyton Jones, Jean-Marc Eber, and Julian Seward. Composing contracts : an adventure in financial engineering, functional pearl. In Odersky and Wadler [OW00], pages 280–292.

[Jon94] Neil D. Jones. Abstract interpretation : a semantics-based tool for program analysis, 1994.

[Kni12] Is Knight's 440 million glitch the costliest computer bug ever ? `http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug`, 2012.

[Kri07] Jens Krinke. Information flow control and taint analysis with dependence graphs, 2007.

[MSA80] MIL-STD-1629A. *Procedure for performing a Failure Mode, Effets and Criticality Analysis*. Department of Defense, November 1980.

[MW08] Yaron Minsky and Stephen Weeks. Caml trading - experiences with functional programming on wall street. *J. Funct. Program.*, 18(4) :553–564, 2008.

[Myc81] Alan Mycroft. *Abstract interpretation and optimising transformations for applicative programs*. PhD thesis, University of Edinburgh, 1981.

[OW00] Martin Odersky and Philip Wadler, editors. *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*. ACM, 2000.

[PAC+08] Bruno Pagano, Olivier Andrieu, Benjamin Canou, Emmanuel Chailloux, Jean-Louis Colaço, Thomas Moniot, and Philippe Wang. Certified development tools

implementation in objective caml. In Paul Hudak and David Scott Warren, editors, *PADL*, volume 4902 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008.

[PC00] François Pottier and Sylvain Conchon. Information flow inference for free. In Odersky and Wadler [OW00], pages 46–57.

[PS02] François Pottier and Vincent Simonet. Information flow inference for ml. In John Launchbury and John C. Mitchell, editors, *POPL*, pages 319–330. ACM, 2002.

[RTC92] RTCA, Incorporated. *Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[SR03] Vincent Simonet and Inria Rocquencourt. Flow caml in a nutshell. In *Proceedings of the first APPSEM-II workshop*, pages 152–165, 2003.

[Sta99] Standard Cenelec EN 50128. *Railway Applications - Communications, Signaling and Processing Systems - Software for Railway Control and Protection Systems*, 1999.

[TDD12] Pierre-Nicolas Tollitte, David Delahaye, and Catherine Dubois. Producing certified functional code from inductive specifications. In Chris Hawblitzel and Dale Miller, editors, *CPP*, volume 7679 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.

[X6086] X60510. *Procédures d'analyse des modes de défaillance et de leurs effets (AMDE)*. AFNOR, Décembre 1986.

[XBS06] Wei Xu, Sandeep Bhatkar, and R. Sekar. Taint-enhanced policy enforcement : A practical approach to defeat a wide range of attacks. In *In the Proc. of the 15th USENIX Security Symp*, 2006.

Annexes

Annexe A

Définitions des sémantiques en Coq

A.1 Sémantique opérationnelle

```
Inductive val_of : env → expr → val → Prop :=
| Val_of_num : forall (v : Z) (c : env), val_of c (Num v) (V_Num v)

| Val_of_ident : forall (c : env) (i : identifier) (v : val),
    assoc_ident_in_env i c = Ident_in_env v → val_of c (Var i) v

| Val_of_lambda : forall (c:env) (x:identifier) (e:expr),
    val_of c (Lambda x e) (V_Closure x e c)

| Val_of_rec : forall (c:env) (f x:identifier) (e:expr),
    val_of c (Rec f x e) (V_Rec_Closure f x e c)

| Val_of_apply : forall (c c1 : env) (e1 e2 e : expr) (x: identifier) (v2 v: val),
    val_of c e1 (V_Closure x e c1) → val_of c e2 v2 → val_of (add_env x v2 c1) e v
    → val_of c (Apply e1 e2) v

| Val_of_apply_rec : forall (c c1 : env) (e1 e2 e : expr) (f x: identifier) (v2 v: val),
    val_of c e1 (V_Rec_Closure f x e c1) → val_of c e2 v2
    → val_of (add_env f (V_Rec_Closure f x e c1) (add_env x v2 c1)) e v
    → val_of c (Apply e1 e2) v

| Val_of_let : forall (c : env) (x : identifier) (e1 e2 : expr) (v1 v : val),
    val_of c e1 v1 → val_of (add_env x v1 c) e2 v → val_of c (Let_in x e1 e2) v

| Val_of_If_true : forall (c : env) (e e1 e2 : expr) (v : val),
    val_of c e (V_Bool true) → val_of c e1 v → val_of c (If e e1 e2) v

| Val_of_If_false : forall (c : env) (e e1 e2 : expr) (v : val),
    val_of c e (V_Bool false) → val_of c e2 v → val_of c (If e e1 e2) v

| Val_of_Match : forall (c c_p : env) (e e1 : expr) (p : pattern) (v v_e : val) (br2 : option (identifier*expr)),
    val_of c e v_e → is_filtered v_e p = Filtered_result_Match c_p → val_of (conc_env c_p c) e1 v
    → val_of c (Expr_match e (p,e1) br2) v

| Val_of_Match_var : forall (c : env) (e e1 e2 : expr) (p : pattern) (v v_e : val) (x : identifier),
    val_of c e v_e → is_filtered v_e p = Filtered_result_Match_var → val_of (add_env x v_e c) e2 v
    → val_of c (Expr_match e (p,e1) (Some (x,e2))) v

| Val_of_Constr0 : forall c n, val_of c (Constr0 n) (V_Constr0 n)

| Val_of_Constr1 : forall c n e v, val_of c e v → val_of c (Constr1 n e) (V_Constr1 n v)

| Val_of_Couple : forall c (e1 e2 : expr) (v1 v2 : val),
  val_of c e1 v1 → val_of c e2 v2 → val_of c (Couple e1 e2) (V_Couple v1 v2)

| Val_of_Annot : forall c (l : label) (e : expr) v,
  val_of c e v → val_of c (Annot l e) v.
```

A.2 Sémantique avec injection

```
Inductive val_of_with_injection : label → val → oitenv → expr → val → Prop :=
| Val_of_with_injection_Num : forall (l:label) (vl:val) (v:Z) (c:oitenv),
  val_of_with_injection l vl c (Num v) (V_Num v)

| Val_of_with_injection_Ident :
  forall (l:label) (vl v:val) (c:oitenv) (i:identifier) (uu:oitval),
    assoc_ident_in_oitenv i c = Ident_in_oitenv uu
    → Some v = instantiate_oitval l vl uu
    → val_of_with_injection l vl c (Var i) v

| Val_of_with_injection_Lambda :
  forall (l:label) (vl v:val) (c:oitenv) (c':env) (x:identifier) (e:expr),
    Some c' = instantiate_oitenv l vl c → v = V_Closure x e c'
    → val_of_with_injection l vl c (Lambda x e) v

| Val_of_with_injection_Rec :
  forall (l:label) (vl v:val) (c:oitenv) (c':env) (f x:identifier) (e:expr),
    Some c' = instantiate_oitenv l vl c → v = V_Rec_Closure f x e c'
    → val_of_with_injection l vl c (Rec f x e) v

| Val_of_with_injection_Apply :
  forall (l:label) (vl:val) (c:oitenv) (c1:env) (e1 e2 e:expr) (x:identifier) (v2 v:val),
    val_of_with_injection l vl c e1 (V_Closure x e c1)
    → val_of_with_injection l vl c e2 v2
    → val_of_with_injection l vl (OITEnv_cons x (val_to_oitval v2) (env_to_oitenv c1)) e v
    → val_of_with_injection l vl c (Apply e1 e2) v

| Val_of_with_injection_Apply_rec :
  forall (l:label) (vl:val) (c : oitenv) (c1 : env) (e1 e2 e : expr) (f x : identifier) (v2 v : val),
    val_of_with_injection l vl c e1 (V_Rec_Closure f x e c1)
    → val_of_with_injection l vl c e2 v2
    → val_of_with_injection l vl (env_to_oitenv (add_env f (V_Rec_Closure f x e c1)
                                                           (add_env x v2 c1))) e v
    → val_of_with_injection l vl c (Apply e1 e2) v

| Val_of_with_injection_Let_in :
  forall (l:label) (vl:val) (c : oitenv) (i : identifier) (e1 e2 : expr) (v1 v2 : val),
    val_of_with_injection l vl c e1 v1
    → val_of_with_injection l vl (OITEnv_cons i (val_to_oitval v1) c) e2 v2
    → val_of_with_injection l vl c (Let_in i e1 e2) v2

| Val_of_with_injection_If_true :
  forall (l:label) (vl:val) (c : oitenv) (e e1 e2 : expr) (v : val),
    val_of_with_injection l vl c e (V_Bool true)
    → val_of_with_injection l vl c e1 v
    → val_of_with_injection l vl c (If e e1 e2) v
```

```
| Val_of_with_injection_If_false :
  forall (l:label) (vl:val) (c : oitenv) (e e1 e2 : expr) (v : val),
    val_of_with_injection l vl c e (V_Bool false)
    → val_of_with_injection l vl c e2 v
    → val_of_with_injection l vl c (If e e1 e2) v

| Val_of_with_injection_Match :
  forall (l:label) (vl:val) (c:oitenv) (c_p:env) (e e1 : expr)
         (p : pattern) (v v_e : val) (br2 : option (identifier*expr)),
    val_of_with_injection l vl c e v_e
    → is_filtered v_e p = Filtered_result_Match c_p
    → val_of_with_injection l vl (conc_oitenv (env_to_oitenv c_p) c) e1 v
    → val_of_with_injection l vl c (Expr_match e (p,e1) br2) v

| Val_of_with_injection_Match_var :
  forall (l:label) (vl:val) (c : oitenv) (e e1 e2 : expr) (p : pattern) (v v_e : val) (x : identifier),
    val_of_with_injection l vl c e v_e
    → is_filtered v_e p = Filtered_result_Match_var
    → val_of_with_injection l vl (OITEnv_cons x (val_to_oitval v_e) c) e2 v
    → val_of_with_injection l vl c (Expr_match e (p,e1) (Some (x,e2))) v

| Val_of_with_injection_Constr0 : forall (l:label) (vl:val) c n,
  val_of_with_injection l vl c (Constr0 n) (V_Constr0 n)

| Val_of_with_injection_Constr1 : forall (l:label) (vl:val) c n e v,
  val_of_with_injection l vl c e v
  → val_of_with_injection l vl c (Constr1 n e) (V_Constr1 n v)

| Val_of_with_injection_Couple : forall (l:label) (vl:val) c (e1 e2 : expr) (v1 v2 : val),
  val_of_with_injection l vl c e1 v1
  → val_of_with_injection l vl c e2 v2
  → val_of_with_injection l vl c (Couple e1 e2) (V_Couple v1 v2)

| Val_of_with_injection_Annot_eq : forall (l:label) (vl:val) c (e : expr),
  val_of_with_injection l vl c (Annot l e) vl

| Val_of_with_injection_Annot_neq : forall (l:label) (vl:val) c (l' : label) (e : expr) v,
  val_of_with_injection l vl c e v
  → neq_label l l'
  → val_of_with_injection l vl c (Annot l' e) v.
```

A.3 Sémantique sur-instrumentée

```
Inductive oival_of : oitenv → expr → oitval → Prop :=
| OIVal_of_Num : forall (v:Z) (c:oitenv), oival_of c (Num v) (OIV nil (OIV_ nil (OIV_Num v)))

| OIVal_of_Ident : forall (c:oitenv) (x:identifier) (uu:oitval),
    assoc_ident_in_oitenv x c = Ident_in_oitenv uu → oival_of c (Var x) uu

| OIVal_of_Lambda : forall (c:oitenv) (x:identifier) (e:expr) (uu:oitval),
    uu = OIV nil (OIV_ nil (OIV_Closure x e (oitenv_to_oienv c))) → oival_of c (Lambda x e) uu

| OIVal_of_Rec : forall (c:oitenv) (f x:identifier) (e:expr) (uu:oitval),
    uu = OIV nil (OIV_ nil (OIV_Rec_Closure f x e (oitenv_to_oienv c)))
    → oival_of c (Rec f x e) uu

| OIVal_of_Apply : forall (c:oitenv) (c1:oienv) (e1 e2 e : expr) (x : identifier)
    (uu2 uu:oitval) (u2:oival) (d d' d1:oideps) (td1 td2 td td':oitdeps) (v:oival0),
    oival_of c e1 (OIV td1 (OIV_ d1 (OIV_Closure x e c1)))
    → oival_of c e2 uu2 → uu2 = OIV td2 u2
    → oival_of (OITEnv_cons x uu2 (oienv_to_oitenv c1)) e (OIV td (OIV_ d v))
    → deps_spec_Apply uu2 d1 td' d' (* specification des dependances td' et d' *)
    → uu = OIV (conc_oitdeps td1 (conc_oitdeps td2 (conc_oitdeps td td')))
               (OIV_ (conc_oideps d' d) v) (* resultat de l'application *)
    → oival_of c (Apply e1 e2) uu

| OIVal_of_Apply_Rec :
  forall (c:oitenv) (c1:oienv) (e1 e2 e : expr) (f x:identifier)
    (uu1 uu2 uu:oitval) (u2:oival) (d d' d1:oideps) (td1 td2 td td':oitdeps) (v:oival0),
    oival_of c e1 uu1
    → uu1 = (OIV td1 (OIV_ d1 (OIV_Rec_Closure f x e c1)))
    → oival_of c e2 uu2 → uu2 = OIV td2 u2
    → oival_of (OITEnv_cons f uu1 (OITEnv_cons x uu2 (oienv_to_oitenv c1))) e
               (OIV td (OIV_ d v))
    → deps_spec_Apply uu2 d1 td' d' (* specification des dependances td' et d' *)
    → uu = OIV (conc_oitdeps td1 (conc_oitdeps td2 (conc_oitdeps td td')))
               (OIV_ (conc_oideps d' d) v) (* resultat de l'application *)
    → oival_of c (Apply e1 e2) uu

| OIVal_of_Let_in :
  forall (c:oitenv) (x:identifier) (e1 e2:expr) (uu1 uu2:oitval) (td1 td2:oitdeps) (u1 u2:oival),
    oival_of c e1 uu1 → uu1 = (OIV td1 u1)
    → oival_of (OITEnv_cons x uu1 c) e2 uu2 → uu2 = (OIV td2 u2)
    → oival_of c (Let_in x e1 e2) (OIV (conc_oitdeps td1 td2) u2)

| OIVal_of_If_true :
  forall (c:oitenv) (e e1 e2:expr) (td td1 td':oitdeps) (d d1 d':oideps) (uu1:oitval) (v1:oival0),
    oival_of c e (OIV td (OIV_ d (OIV_Bool true)))
    → oival_of c e1 uu1 → uu1 = (OIV td1 (OIV_ d1 v1))
    → deps_spec_If c e1 e2 d td' d' (* specification des dependances td' et d' *)
    → oival_of c (If e e1 e2) (OIV (conc_oitdeps td' (conc_oitdeps td td1))
                                   (OIV_ (conc_oideps d' d1) v1))
```

```
| OIVal_of_If_false :
  forall (c:oitenv) (e e1 e2:expr) (td td2 td':oitdeps) (d d2 d':oideps) (uu2:oitval) (v2:oival0),
    oival_of c e (OIV td (OIV_ d (OIV_Bool false)))
    → oival_of c e2 uu2 → uu2 = (OIV td2 (OIV_ d2 v2))
    → deps_spec_If c e1 e2 d td' d' (* specification des dependances td' et d' *)
    → oival_of c (If e e1 e2) (OIV (conc_oitdeps td' (conc_oitdeps td td2))
                                   (OIV_ (conc_oideps d' d2) v2))

| OIVal_of_Match :
  forall (c c_p:oitenv) (e e1 e2: expr) (p:pattern) (x:identifier) (uu uu1:oitval)
         (td td1 td':oitdeps) (d d1 d':oideps) (v v1:oival0),
    oival_of c e uu
    → uu = (OIV td (OIV_ d v))
    → is_filtered_oitval uu p = Filtered_oitval_result_Match c_p
    → oival_of (conc_oitenv c_p c) e1 uu1
    → uu1 = (OIV td1 (OIV_ d1 v1))
    → deps_spec_Match c p x e1 e2 d td' d' (* specification des dependances td' et d' *)
    → oival_of c (Expr_match e (p,e1) (Some (x,e2))) (OIV (conc_oitdeps td' (conc_oitdeps td td1))
                                                          (OIV_ (conc_oideps d' d1) v1))

| OIVal_of_Match_var :
  forall (c:oitenv) (e e1 e2: expr) (p:pattern) (x:identifier) (uu uu2:oitval)
         (td td2 td':oitdeps) (d d2 d':oideps) (v v2:oival0),
    oival_of c e uu
    → uu = (OIV td (OIV_ d v))
    → is_filtered_oitval uu p = Filtered_oitval_result_Match_var
    → oival_of (OITEnv_cons x uu c) e2 uu2
    → uu2 = (OIV td2 (OIV_ d2 v2))
    → deps_spec_Match c p x e1 e2 d td' d' (* specification des dependances td' et d' *)
    → oival_of c (Expr_match e (p,e1) (Some (x,e2))) (OIV (conc_oitdeps td' (conc_oitdeps td td2))
                                                          (OIV_ (conc_oideps d' d2) v2))

| OIVal_of_Constr0 : forall (c:oitenv) (n:constr),
    oival_of c (Constr0 n) (OIV nil (OIV_ nil (OIV_Constr0 n)))

| OIVal_of_Constr1 : forall (c:oitenv) (n:constr) (e:expr) (td:oitdeps) (u:oival),
    oival_of c e (OIV td u)
    → oival_of c (Constr1 n e) (OIV td (OIV_ nil (OIV_Constr1 n u)))

| OIVal_of_Couple : forall (c:oitenv) (e1 e2:expr) (td1 td2:oitdeps) (u1 u2:oival),
    oival_of c e1 (OIV td1 u1) → oival_of c e2 (OIV td2 u2)
    → oival_of c (Couple e1 e2) (OIV (conc_oitdeps td1 td2) (OIV_ nil (OIV_Couple u1 u2)))

| OIVal_of_Annot : forall (c:oitenv) (l:label) (e:expr) (td:oitdeps) (d:oideps) (v:oival0),
    oival_of c e (OIV td (OIV_ d v))
    → oival_of c (Annot l e) (OIV td (OIV_ (cons (l, fun x⇒ x) d) v)).
```

A.4 Sémantique instrumentée

```
Inductive ival_of : itenv → expr → itval → Prop :=
| IVal_of_Num : forall (v:Z) (c:itenv),
    ival_of c (Num v) (IV nil (IV_ nil (IV_Num v)))

| IVal_of_Ident : forall (c:itenv) (x:identifier) (uu:itval),
    assoc_ident_in_itenv x c = Ident_in_itenv uu
    → ival_of c (Var x) uu

| IVal_of_Lambda : forall (c:itenv) (x:identifier) (e:expr) (uu:itval),
    uu = IV nil (IV_ nil (IV_Closure x e (itenv_to_ienv c)))
    → ival_of c (Lambda x e) uu

| IVal_of_Rec :
  forall (c:itenv) (f x:identifier) (e:expr) (uu:itval),
    uu = IV nil (IV_ nil (IV_Rec_Closure f x e (itenv_to_ienv c)))
    → ival_of c (Rec f x e) uu

| IVal_of_Apply :
  forall (c:itenv) (c1:ienv) (e1 e2 e : expr) (x : identifier)
    (uu2 uu:itval) (u2:ival) (d d1:ideps) (td1 td2 td:itdeps) (v:ival0),
    (* construction de la valeur *)
    ival_of c e1 (IV td1 (IV_ d1 (IV_Closure x e c1)))
    → ival_of c e2 uu2
    → uu2 = IV td2 u2
    → ival_of (ITEnv_cons x uu2 (ienv_to_itenv c1)) e (IV td (IV_ d v))
    (* resultat de l'application *)
    → uu = IV (conc_itdeps td1 (conc_itdeps td2 (conc_itdeps td d1))) (IV_ (conc_ideps d1 d) v)
    → ival_of c (Apply e1 e2) uu

| IVal_of_Apply_Rec :
  forall (c:itenv) (c1:ienv) (e1 e2 e : expr) (f x:identifier)
    (uu1 uu2 uu:itval) (u2:ival) (d d1:ideps) (td1 td2 td:itdeps) (v:ival0),
    (* construction de la valeur *)
    ival_of c e1 uu1
    → uu1 = (IV td1 (IV_ d1 (IV_Rec_Closure f x e c1)))
    → ival_of c e2 uu2
    → ival_of (ITEnv_cons f uu1 (ITEnv_cons x uu2 (ienv_to_itenv c1))) e (IV td (IV_ d v))
    → uu2 = IV td2 u2
    (* resultat de l'application *)
    → uu = IV (conc_itdeps td1 (conc_itdeps td2 (conc_itdeps td d1))) (IV_ (conc_ideps d1 d) v)
    → ival_of c (Apply e1 e2) uu

| IVal_of_Let_in :
  forall (c:itenv) (x:identifier) (e1 e2:expr) (uu1 uu2:itval) (td1 td2:itdeps) (u1 u2:ival),
    ival_of c e1 uu1
    → uu1 = (IV td1 u1)
    → ival_of (ITEnv_cons x uu1 c) e2 uu2
    → uu2 = (IV td2 u2)
    → ival_of c (Let_in x e1 e2) (IV (conc_itdeps td1 td2) u2)
```

```
| IVal_of_If_true :
  forall (c:itenv) (e e1 e2:expr) (td td1:itdeps) (d d1:ideps) (uu1:itval) (v1:ival0),
    ival_of c e (IV td (IV_ d (IV_Bool true)))
    → ival_of c e1 uu1 → uu1 = (IV td1 (IV_ d1 v1))
    → ival_of c (If e e1 e2) (IV (conc_itdeps d (conc_itdeps td td1)) (IV_ (conc_ideps d d1) v1))

| IVal_of_If_false :
  forall (c:itenv) (e e1 e2:expr) (td td2:itdeps) (d d2:ideps) (uu2:itval) (v2:ival0),
    ival_of c e (IV td (IV_ d (IV_Bool false)))
    → ival_of c e2 uu2 → uu2 = (IV td2 (IV_ d2 v2))
    → ival_of c (If e e1 e2) (IV (conc_itdeps d (conc_itdeps td td2)) (IV_ (conc_ideps d d2) v2))

| IVal_of_Match :
  forall (c c_p:itenv) (e e1 e2: expr) (p:pattern) (x:identifier) (uu uu1:itval)
         (td td1:itdeps) (d d1:ideps) (v v1:ival0),
    ival_of c e uu
    → uu = (IV td (IV_ d v))
    → is_filtered_itval uu p = Filtered_itval_result_Match c_p
    → ival_of (conc_itenv c_p c) e1 uu1
    → uu1 = (IV td1 (IV_ d1 v1))
    → ival_of c (Expr_match e (p,e1) (Some (x,e2))) (IV (conc_itdeps d (conc_itdeps td td1))
                                                         (IV_ (conc_ideps d d1) v1))

| IVal_of_Match_var :
  forall (c:itenv) (e e1 e2: expr) (p:pattern) (x:identifier) (uu uu2:itval)
         (td td2:itdeps) (d d2:ideps) (v v2:ival0),
    ival_of c e uu
    → uu = (IV td (IV_ d v))
    → is_filtered_itval uu p = Filtered_itval_result_Match_var
    → ival_of (ITEnv_cons x uu c) e2 uu2
    → uu2 = (IV td2 (IV_ d2 v2))
    → ival_of c (Expr_match e (p,e1) (Some (x,e2))) (IV (conc_itdeps d (conc_itdeps td td2))
                                                         (IV_ (conc_ideps d d2) v2))

| IVal_of_Constr0 : forall (c:itenv) (n:constr),
    ival_of c (Constr0 n) (IV nil (IV_ nil (IV_Constr0 n)))

| IVal_of_Constr1 : forall (c:itenv) (n:constr) (e:expr) (td:itdeps) (u:ival),
    ival_of c e (IV td u)
    → ival_of c (Constr1 n e) (IV td (IV_ nil (IV_Constr1 n u)))

| IVal_of_Couple : forall (c:itenv) (e1 e2:expr) (td1 td2:itdeps) (u1 u2:ival),
    ival_of c e1 (IV td1 u1)
    → ival_of c e2 (IV td2 u2)
    → ival_of c (Couple e1 e2) (IV (conc_itdeps td1 td2) (IV_ nil (IV_Couple u1 u2)))

| IVal_of_Annot : forall (c:itenv) (l:label) (e:expr) (td:itdeps) (d:ideps) (v:ival0),
    ival_of c e (IV td (IV_ d v))
    → ival_of c (Annot l e) (IV td (IV_ (cons l d) v)).
```

A.5 Sémantique instrumentée multiple

```
Inductive imval_of : imenv → expr → imval → Prop :=
| IMVal_of :
    forall (imc:imenv) (e:expr) (imv:imval),
      imv = (fun itu ⇒
               exists (itc:itenv), In _ imc itc
                                   ∧ ival_of itc e itu)
      → imval_of imc e imv.
```

A.6 Sémantique collectrice

```
Inductive ctval_of : ctenv → expr → ctval → Prop :=
| CTVal_of_Num : forall (ctc:ctenv) (v:Z),
    ctval_of ctc (Num v) (CTVal nil nil (Singleton _ (IV_Num v)))

| CTVal_of_Constr0 : forall (ctc:ctenv) (n:constr),
    ctval_of ctc (Constr0 n) (CTVal nil nil (Singleton _ (IV_Constr0 n)))

| CTVal_of_Constr1 :
  forall (ctc:ctenv) (n:constr) (e:expr) (td:itdeps) (d:ideps) (ivs ivs':Ensemble ival0),
    ctval_of ctc e (CTVal td d ivs)
    → ivs' = SetMap (fun iv ⇒ IV_Constr1 n (IV_ d iv)) ivs
    → ctval_of ctc (Constr1 n e) (CTVal td nil ivs')

| CTVal_of_Apply :
    forall (ctc:ctenv) (e1 e2:expr)
           (td1 td2 td:itdeps) (d1 d2 d:ideps) (ivs1 ivs2:Ensemble ival0)
           (ctv:ctval)
           (imv:Ensemble itval),
    ctval_of ctc e1 (CTVal td1 d1 ivs1)
    → ctval_of ctc e2 (CTVal td2 d2 ivs2)
    → imv = (fun itu ⇒ (exists x e ic1 iv2 itd id iv,
                          In _ ivs1 (IV_Closure x e ic1)
                          ∧ In _ ivs2 iv2
                          ∧ ival_of (ITEnv_cons x (IV td2 (IV_ d2 iv2)) (ienv_to_itenv ic1))
                                    e (IV itd (IV_ id iv))
                          ∧ itu = IV (conc_itdeps td1 (conc_itdeps td2 (conc_itdeps itd d1)))
                                     (IV_ (conc_ideps d1 id) iv))
                       ∨ (exists f x e ic iv2 itd id iv,
                          In _ ivs1 (IV_Rec_Closure f x e ic)
                          ∧ In _ ivs2 iv2
                          ∧ ival_of (ITEnv_cons f (IV td1 (IV_ d1 (IV_Rec_Closure f x e ic)))
                                                  (ITEnv_cons x (IV td2 (IV_ d2 iv2)) (ienv_to_itenv ic)))
                                    e (IV itd (IV_ id iv))
                          ∧ itu = IV (conc_itdeps td1 (conc_itdeps td2 (conc_itdeps itd d1)))
                                     (IV_ (conc_ideps d1 id) iv)))
    → (forall l:label, List.In l td
                       ↔ (exists td', List.In l td'
                          ∧ In _ (SetMap (fun itu ⇒ match itu with
                                                     | IV td (IV_ d iv) ⇒ td end) imv) td'))
    → (forall l:label, List.In l d
                       ↔ (exists d', List.In l d'
                          ∧ In _ (SetMap (fun itu ⇒ match itu with
                                                     | IV td (IV_ d iv) ⇒ d end) imv) d'))
    → ctv = CTVal td d (SetMap (fun itu ⇒ match itu with | IV td (IV_ d iv) ⇒ iv end) imv)
    → ctval_of ctc (Apply e1 e2) ctv

| CTVal_of_Ident : forall (ctc:ctenv) (x:identifier) (ctv:ctval),
    assoc_ident_in_ctenv x ctc = Ident_in_ctenv ctv
    → ctval_of ctc (Var x) ctv
```

```
| CTVal_of_Ident_empty :forall (ctc:ctenv) (x:identifier),
    assoc_ident_in_ctenv x ctc = Ident_not_in_ctenv
    → ctval_of ctc (Var x) (CTVal nil nil (Empty_set _))

| CTVal_of_Lambda :
  forall (ctc:ctenv) (x:identifier) (e:expr) (ctv:ctval) (ivs:Ensemble ival0),
    ivs = SetMap (fun itc ⇒ IV_Closure x e (itenv_to_ienv itc)) (ctenv_to_imenv ctc)
    → ctval_of ctc (Lambda x e) (CTVal nil nil ivs)

| CTVal_of_Rec :
    forall (ctc:ctenv) (f x:identifier) (e:expr) (ctv:ctval) (ivs:Ensemble ival0),
    ivs = SetMap (fun itc ⇒ IV_Rec_Closure f x e (itenv_to_ienv itc)) (ctenv_to_imenv ctc)
    → ctval_of ctc (Rec f x e) (CTVal nil nil ivs)

| CTVal_of_If_true :
    forall (ctc:ctenv) (e e1 e2:expr) (td td1:itdeps) (d d1:ideps) (ivs ivs1:Ensemble ival0),
      ctval_of ctc e (CTVal td d ivs)
      → In _ ivs (IV_Bool true)
      → (not (In _ ivs (IV_Bool false)))
      → ctval_of ctc e1 (CTVal td1 d1 ivs1)
      → ctval_of ctc (If e e1 e2) (CTVal (conc_itdeps d (conc_itdeps td td1))
                                         (conc_ideps d d1) ivs1)

| CTVal_of_If_false :
    forall (ctc:ctenv) (e e1 e2:expr) (td td2:itdeps) (d d2:ideps) (ivs ivs2:Ensemble ival0),
      ctval_of ctc e (CTVal td d ivs)
      → Same_set _ ivs (Singleton _ (IV_Bool false))
      → In _ ivs (IV_Bool false)
      → (not (In _ ivs (IV_Bool true)))
      → ctval_of ctc e2 (CTVal td2 d2 ivs2)
      → ctval_of ctc (If e e1 e2) (CTVal (conc_itdeps d (conc_itdeps td td2))
                                         (conc_ideps d d2) ivs2)

| CTVal_of_If_unknown :
    forall (ctc:ctenv) (e e1 e2:expr)
           (td td1 td2:itdeps) (d d1 d2:ideps) (ivs ivs1 ivs2:Ensemble ival0),
      ctval_of ctc e (CTVal td d ivs)
      → Included _ (Couple _ (IV_Bool true) (IV_Bool false)) ivs
      → ctval_of ctc e1 (CTVal td1 d1 ivs1)
      → ctval_of ctc e2 (CTVal td2 d2 ivs2)
      → ctval_of ctc (If e e1 e2) (CTVal (conc_itdeps d (conc_itdeps td (conc_itdeps td1 td2)))
                                         (conc_ideps d (conc_ideps d1 d2))
                                         (Union _ ivs1 ivs2))

| CTVal_of_If_empty :
    forall (ctc:ctenv) (e e1 e2:expr) (td td1:itdeps) (d d1:ideps) (ivs ivs1:Ensemble ival0),
      ctval_of ctc e (CTVal td d ivs)
      → (not (In _ ivs (IV_Bool true)))
      → (not (In _ ivs (IV_Bool false)))
      → ctval_of ctc (If e e1 e2) (CTVal nil nil (Empty_set _))
```

```
| CTVal_of_Match :
    forall (ctv:ctval) (ctc ctc_p:ctenv) (e e1:expr)
           (br2:option (identifier*expr)) (p:pattern)
           (td td1:itdeps) (d d1:ideps) (ivs ivs1:Ensemble ival0),
      ctval_of ctc e ctv
      → ctv = (CTVal td d ivs)
      → Inhabited _ (Intersection _ ivs ivs_of_matchable)
      → is_filtered_ctval ctv p (Filtered_ctval_result_Match ctc_p)
      → ctval_of (conc_ctenv ctc_p ctc) e1 (CTVal td1 d1 ivs1)
      → ctval_of ctc (Expr_match e (p,e1) br2) (CTVal (conc_itdeps d (conc_itdeps td td1))
                                                      (conc_ideps d d1) ivs1)

| CTVal_of_Match_var :
    forall (ctv:ctval) (ctc:ctenv) (e e1 e2:expr)
           (p:pattern) (x:identifier)
           (td td2:itdeps) (d d2:ideps) (ivs ivs2:Ensemble ival0),
      ctval_of ctc e ctv
      → ctv = (CTVal td d ivs)
      → Inhabited _ (Intersection _ ivs ivs_of_matchable)
      → is_filtered_ctval ctv p Filtered_ctval_result_Match_var
      → ctval_of (CTEnv_cons x ctv ctc) e2 (CTVal td2 d2 ivs2)
      → ctval_of ctc (Expr_match e (p,e1) (Some (x,e2))) (CTVal (conc_itdeps d (conc_itdeps td td2))
                                                                (conc_ideps d d2) ivs2)

| CTVal_of_Match_unknown :
    forall (ctv:ctval) (ctc ctc_p:ctenv) (e e1 e2:expr)
           (p:pattern) (x:identifier)
           (td td1 td2:itdeps) (d d1 d2:ideps) (ivs ivs1 ivs2:Ensemble ival0),
      ctval_of ctc e ctv
      → ctv = (CTVal td d ivs)
      → Inhabited _ (Intersection _ ivs ivs_of_matchable)
      → is_filtered_ctval ctv p (Filtered_ctval_result_Match_unknown ctc_p)
      → ctval_of (conc_ctenv ctc_p ctc) e1 (CTVal td1 d1 ivs1)
      → ctval_of (CTEnv_cons x ctv ctc) e2 (CTVal td2 d2 ivs2)
      → ctval_of ctc (Expr_match e (p,e1) (Some (x,e2)))
                  (CTVal (conc_itdeps d (conc_itdeps td (conc_itdeps td1 td2)))
                         (conc_ideps d (conc_ideps d1 d2))
                         (Union _ ivs1 ivs2))

| CTVal_of_Match_empty :
    forall (ctv:ctval) (ctc ctc_p:ctenv) (e e1:expr)
           (br2:option (identifier*expr)) (p:pattern)
           (td td1:itdeps) (d d1:ideps) (ivs ivs1:Ensemble ival0),
      ctval_of ctc e ctv
      → ctv = (CTVal td d ivs)
      → not (Inhabited _ (Intersection _ ivs ivs_of_matchable))
      → ctval_of ctc (Expr_match e (p,e1) br2) (CTVal nil nil (Empty_set _))
```

```
| CTVal_of_Couple :
  forall (ctc:ctenv) (e1 e2:expr) (td1 td2:itdeps) (d1 d2:ideps) (ivs1 ivs2 ivs':Ensemble ival0),
    ctval_of ctc e1 (CTVal td1 d1 ivs1)
    → ctval_of ctc e2 (CTVal td2 d2 ivs2)
    → ivs' = SetMap2 (fun iv1 iv2 ⇒ IV_Couple (IV_ d1 iv1) (IV_ d2 iv2)) ivs1 ivs2
    → ctval_of ctc (language.Couple e1 e2) (CTVal (conc_itdeps td1 td2) nil ivs')

| CTVal_of_Annot :
  forall (ctc:ctenv) (l:label) (e:expr) (td:itdeps) (d:ideps) (ivs:Ensemble ival0),
    ctval_of ctc e (CTVal td d ivs)
    → ctval_of ctc (Annot l e) (CTVal td (cons l d) ivs)

| CTVal_of_Let_in :
  forall (ctc:ctenv) (x:identifier) (e1 e2:expr)
         (ctv1 ctv2:ctval) (td1 td2:itdeps) (d1 d2:ideps) (ivs1 ivs2:Ensemble ival0),
    ctval_of ctc e1 ctv1
    → ctv1 = (CTVal td1 d1 ivs1)
    → (exists (iv1:ival0), In _ ivs1 iv1)
    → ctval_of (CTEnv_cons x ctv1 ctc) e2 ctv2
    → ctv2 = (CTVal td2 d2 ivs2)
    → ctval_of ctc (Let_in x e1 e2) (CTVal (conc_itdeps td1 td2) d2 ivs2)

| CTVal_of_Let_in_empty :
  forall (ctc:ctenv) (x:identifier) (e1 e2:expr)
         (ctv1:ctval) (td1:itdeps) (d1:ideps) (ivs1:Ensemble ival0),
    ctval_of ctc e1 ctv1
    → ctv1 = (CTVal td1 d1 ivs1)
    → not (Inhabited _ ivs1)
    → ctval_of ctc (Let_in x e1 e2) (CTVal nil nil (Empty_set _)).
```

A.7 Sémantique abstraite

```
Inductive atval_of : atenv → expr → atval → Prop :=
| ATVal_of_Num : forall (v:Z) (atc:atenv),
    atval_of atc (Num v) (ATV nil (AV nil (AV_Top)))

| ATVal_of_Ident : forall (atc:atenv) (x:identifier) (atu:atval),
    assoc_ident_in_atenv x atc = Ident_in_atenv atu
    → atval_of atc (Var x) atu

| ATVal_of_Ident_empty : forall (atc:atenv) (x:identifier) (atu:atval),
    assoc_ident_in_atenv x atc = Ident_not_in_atenv
    → atval_of atc (Var x) (ATV nil (AV nil (AV_Bottom)))

| ATVal_of_Lambda : forall (atc:atenv) (x:identifier) (e:expr) (atu:atval),
    atu = ATV nil (AV nil (AV_Closure x e (atenv_to_aenv atc)))
    → atval_of atc (Lambda x e) atu

| ATVal_of_Rec : forall (atc:atenv) (f x:identifier) (e:expr) (atu:atval),
    atu = ATV nil (AV nil (AV_Rec_Closure f x e (atenv_to_aenv atc)))
    → atval_of atc (Rec f x e) atu

| ATVal_of_Apply :
  forall (atc:atenv) (ac1:aenv) (e1 e2 e : expr) (x : identifier)
    (atu2 atu:atval) (au2:aval) (ad ad1:adeps) (atd1 atd2 atd:atdeps) (av:aval0),
    (* construction de la valeur *)
    atval_of atc e1 (ATV atd1 (AV ad1 (AV_Closure x e ac1)))
    → atval_of atc e2 atu2
    → atu2 = ATV atd2 au2
    → atval_of (ATEnv_cons x atu2 (aenv_to_atenv ac1)) e (ATV atd (AV ad av))
    (* resultat de l'application *)
    → atu = ATV (conc_atdeps atd1 (conc_atdeps atd2 (conc_atdeps atd ad1)))
                (AV (conc_adeps ad1 ad) av)
    → atval_of atc (Apply e1 e2) atu

| ATVal_of_Apply_rec :
  forall (atc:atenv) (ac1:aenv) (e1 e2 e:expr) (f x:identifier)
    (atu2 atu atu_f:atval) (au2:aval) (ad ad1:adeps) (atd1 atd2 atd:atdeps) (av:aval0),
    (* construction de la valeur *)
    atval_of atc e1 (ATV atd1 (AV ad1 (AV_Rec_Closure f x e ac1)))
    → atu_f = ATV atd1 (AV (deps_of_freevars (Rec f x e) (atenv_to_aenv atc)) AV_Top)
    → atval_of atc e2 atu2
    → atu2 = ATV atd2 au2
    → atval_of (ATEnv_cons f atu_f (ATEnv_cons x atu2 (aenv_to_atenv ac1))) e
               (ATV atd (AV ad av))
    (* resultat de l'application *)
    → atu = ATV (conc_atdeps atd1 (conc_atdeps atd2 (conc_atdeps atd ad1)))
                (AV (conc_adeps ad1 ad) av)
    → atval_of atc (Apply e1 e2) atu
```

```
| ATVal_of_Apply_unknown :
  forall (atc:atenv) (e1 e2: expr)
    (atu:atval) (atd1 atd2:atdeps) (ad1 ad2:adeps) (av1 av2:aval0),
    (* construction de la valeur *)
    atval_of atc e1 (ATV atd1 (AV ad1 av1))
    → (match av1 with
          | AV_Closure _ _ _ ⇒ False
          | AV_Rec_Closure _ _ _ _ ⇒ False
          | _ ⇒ True
       end)
    → atval_of atc e2 (ATV atd2 (AV ad2 av2))
    (* resultat de l'application *)
    → atu = ATV (conc_atdeps atd1 (conc_atdeps atd2 (conc_atdeps ad1 ad2)))
                (AV (conc_adeps ad1 ad2) AV_Top)
    → atval_of atc (Apply e1 e2) atu
(* amelioration possible :
   pour une analyse plus precise, si la valeur de e1 ou e2 est Bottom, retourner Bottom *)

| ATVal_of_Let_in :
  forall (atc:atenv) (x:identifier) (e1 e2:expr) (atu1 atu2:atval) (atd1 atd2:atdeps) (au1 au2:aval),
    atval_of atc e1 atu1
    → atu1 = (ATV atd1 au1)
    → atval_of (ATEnv_cons x atu1 atc) e2 atu2
    → atu2 = (ATV atd2 au2)
    → atval_of atc (Let_in x e1 e2) (ATV (conc_atdeps atd1 atd2) au2)

| ATVal_of_If :
  forall (atc:atenv) (e e1 e2:expr) (atd atd1 atd2:atdeps)
         (ad ad1 ad2:adeps) (atu1 atu2:atval) (av av1 av2:aval0),
    atval_of atc e (ATV atd (AV ad av))
    → atval_of atc e1 atu1
    → atu1 = (ATV atd1 (AV ad1 av1))
    → atval_of atc e2 atu2
    → atu2 = (ATV atd2 (AV ad2 av2))
    → atval_of atc (If e e1 e2) (ATV (conc_atdeps ad (conc_atdeps atd (conc_atdeps atd1 atd2)))
                                     (AV (conc_adeps ad (conc_adeps ad1 ad2)) AV_Top))

| ATVal_of_Match :
  forall (atc atc_p:atenv) (e e1 e2: expr) (p:pattern) (x:identifier)
         (atu atu1:atval) (atd atd1:atdeps) (ad ad1:adeps) (av av1:aval0),
    atval_of atc e atu
    → atu = (ATV atd (AV ad av))
    → is_filtered_atval atu p = Filtered_atval_result_Match atc_p
    → atval_of (conc_atenv atc_p atc) e1 atu1
    → atu1 = (ATV atd1 (AV ad1 av1))
    → atval_of atc (Expr_match e (p,e1) (Some (x,e2)))
                (ATV (conc_atdeps ad (conc_atdeps atd atd1))
                     (AV (conc_adeps ad ad1) av1))
```

```
| ATVal_of_Match_var :
  forall (atc atc_p:atenv) (e e1 e2: expr) (p:pattern) (x:identifier)
         (atu atu2:atval) (atd atd2:atdeps) (ad ad2:adeps) (av av2:aval0),
    atval_of atc e atu
    → atu = (ATV atd (AV ad av))
    → is_filtered_atval atu p = Filtered_atval_result_Match_var
    → atval_of (ATEnv_cons x atu atc) e2 atu2
    → atu2 = (ATV atd2 (AV ad2 av2))
    → atval_of atc (Expr_match e (p,e1) (Some (x,e2)))
               (ATV (conc_atdeps ad (conc_atdeps atd atd2))
                    (AV (conc_adeps ad ad2) av2))

| ATVal_of_Match_unknown :
  forall (atc atc_p:atenv) (e e1 e2: expr) (p:pattern) (x:identifier)
         (atu atu1 atu2:atval) (atd atd1 atd2:atdeps) (ad ad1 ad2:adeps) (av av1 av2:aval0),
    atval_of atc e atu
    → atu = (ATV atd (AV ad av))
    → is_filtered_atval atu p = Filtered_atval_result_Match_unknown atc_p
    → atval_of (conc_atenv atc_p atc) e1 atu1
    → atu1 = (ATV atd1 (AV ad1 av1))
    → atval_of (ATEnv_cons x atu atc) e2 atu2
    → atu2 = (ATV atd2 (AV ad2 av2))
    → atval_of atc (Expr_match e (p,e1) (Some (x,e2)))
               (ATV (conc_atdeps ad (conc_atdeps atd (conc_atdeps atd1 atd2)))
                    (AV (conc_adeps ad (conc_adeps ad1 ad2)) AV_Top))

| ATVal_of_Match_bottom :
  forall (atc atc_p:atenv) (e e1 e2: expr) (p:pattern) (x:identifier)
         (atu:atval) (atd:atdeps) (ad:adeps) (av:aval0),
    atval_of atc e atu
    → atu = (ATV atd (AV ad av))
    → is_filtered_atval atu p = Filtered_atval_result_Error
    → atval_of atc (Expr_match e (p,e1) (Some (x,e2)))
               (ATV nil (AV nil AV_Bottom))

| ATVal_of_Constr0 : forall (atc:atenv) (n:constr),
    atval_of atc (Constr0 n) (ATV nil (AV nil (AV_Constr0 n)))

| ATVal_of_Constr1 : forall (atc:atenv) (n:constr) (e:expr) (atd:atdeps) (au:aval),
    atval_of atc e (ATV atd au)
    → atval_of atc (Constr1 n e) (ATV atd (AV nil (AV_Constr1 n au)))

| ATVal_of_Couple : forall (atc:atenv) (e1 e2:expr) (atd1 atd2:atdeps) (au1 au2:aval),
    atval_of atc e1 (ATV atd1 au1)
    → atval_of atc e2 (ATV atd2 au2)
    → atval_of atc (Couple e1 e2) (ATV (conc_atdeps atd1 atd2) (AV nil (AV_Couple au1 au2)))

| ATVal_of_Annot :
  forall (atc:atenv) (l:label) (e:expr) (atd:atdeps) (ad:adeps) (av:aval0),
    atval_of atc e (ATV atd (AV ad av))
    → atval_of atc (Annot l e) (ATV atd (AV (cons l ad) av)).
```

Annexe B

Énoncés des théorèmes en Coq

B.1 Correction de la sémantique avec injection

Dans la version Coq de notre travail, nous avons défini la notion d'injection en formalisant directement la sémantique avec injection dans un t-environnement sur-instrumenté.

Cependant, dans le manuscrit de thèse, nous avons choisi de présenter la sémantique avec injection en deux étapes pour des raisons de clarté dans la présentation. En effet, nous souhaitions commencer par introduire la notion d'injection telle que nous pouvons la concevoir de façon intuitive avant d'entrer dans les détails de la sur-instrumentation des valeurs et des environnements.

Il convient de remarquer ici que la preuve de correction de nos analyses dynamique et statique ne dépend d'aucune manière de la preuve de correction de la sémantique avec injection. En effet, c'est la sémantique avec injection qui pose la définition de la notion d'injection, qui est le fondement de la notion de dépendance. Le théorème de correction que nous apportons n'est donc pas nécessaire à la preuve de nos analyses par rapport à la notion d'injection. Cependant, il constitue une justification de la définition formelle de l'injection par rapport à la notion intuitive que nous en avons.

Voici le théorème de correction de la sémantique avec injection qui a été prouvé en Coq.

```
Theorem injection_correctness :
  forall (l:label) (vl:val) (c:env) (c':oitenv) (e:expr) (v:val),
    label_does_not_appear_in_expr l c' e
    → c = oitenv_to_env c'
    → val_of c e v
    → val_of_with_injection l vl c' e v.
```

B.2 Correction de la sémantique sur-instrumentée

Le théorème présenté dans le manuscrit pour la correction de la sémantique sur-instrumentée (cf. section 3.4.2.3) est identique à celui prouvé en Coq. Nous avons simplement déplié la définition du prédicat `is_instantiable_oitenv`.

```
Theorem oival_of_correctness :
  forall (c:oitenv) (e:expr) (uu:oitval),
    oival_of c e uu
    → forall (l:label) (vl:val),
      is_instantiable_oitenv l vl c
      → forall (v:val),
        Some v = instantiate_oitval l vl uu
        → val_of_with_injection l vl c e v.
```

B.3 Correction de la sémantique instrumentée

Le théorème présenté dans le manuscrit pour la correction de la sémantique instrumentée (cf. section 3.5.3.3) est identique à celui prouvé en Coq.

```
Theorem ival_of_correctness :
  forall (itc:itenv) (e:expr) (itu:itval),
  ival_of itc e itu
  → exists (oitu:oitval),
       (oival_of (itenv_to_oitenv itc) e oitu
        ∧ oitval_to_itval oitu = itu).
```

B.4 Correction de la sémantique collectrice

Le théorème présenté dans le manuscrit pour la correction de la sémantique collectrice (cf. section 4.3.3.3) est identique à celui prouvé en Coq.

On peut peut remarquer une petite différence de notation concernant la fonction de conversion d'un environnement instrumenté multiple en un environnement collecteur. En Coq, cette conversion est définie sous la forme d'un prédicat inductif alors que dans le manuscrit de thèse, nous utilisons une notation fonctionnelle. Il s'agit uniquement d'une différence de notation. La définition donnée dans le manuscrit (cf. figure 4.7) est la même que sa version Coq.

```
Theorem ctval_of_correctness :
  forall (imc:imenv) (e:expr) (imv:imval)
         (ctv:ctval) (ctc:ctenv),
    imval_of imc e imv
    → imenv_to_ctenv imc ctc
    → ctval_of ctc e ctv
    → le_imval imv (ctval_to_imval ctv).
```

B.5 Correction de la sémantique abstraite

Le théorème présenté dans le manuscrit pour la correction de la sémantique abstraite (cf. section 4.4.3.3) est identique à celui prouvé en Coq. La preuve de ce théorème en Coq n'a pas été terminée par manque de temps. La version de la preuve présente dans le manuscrit (cf. section 4.4.3.4) est un peu plus complète puisque le cas de l'application, qui est un des cas les plus complexes, a été prouvé.

```
Theorem atval_of_correctness :
  forall (e:expr) (ctc:ctenv) (ctv:ctval)
         (atc:atenv) (atu:atval),
    ctenv_to_atenv ctc atc
    → ctval_of ctc e ctv
    → atval_of atc e atu
    → le_ctval ctv (atval_to_ctval atu).
```

Index

Jugement d'évaluation

Relation d'ordre

Suppression des t-dépendances

Valeur de référence

le cnam

Vincent BENAYOUN

Analyse de dépendances ML pour les évaluateurs de logiciels critiques

le cnam

Résumé :
Les logiciels critiques nécessitent l'obtention d'une évaluation de conformité aux normes en vigueur avant leur mise en service. Cette évaluation est obtenue après un long travail d'analyse effectué par les évaluateurs de logiciels critiques. Ces derniers peuvent être aidés par des outils utilisés de manière interactive pour construire des modèles, en faisant appel à des analyses de flots d'information. Des outils comme SPARK-Ada existent pour des sous-ensembles du langage Ada utilisés pour le développement de logiciels critiques. Cependant, des langages émergents comme ceux de la famille ML ne disposent pas de tels outils adaptés. La construction d'outils similaires pour les langages ML demande une attention particulière sur certaines spécificités comme les fonctions d'ordre supérieur ou le filtrage par motifs. Ce travail présente une analyse de flot d'information pour de tels langages, spécialement conçue pour répondre aux besoins des évaluateurs. Cette analyse statique prend la forme d'une interprétation abstraite de la sémantique opérationnelle préalablement enrichie par des informations de dépendances. Elle est prouvée correcte vis-à-vis d'une définition formelle de la notion de dépendance, à l'aide de l'assistant à la preuve Coq. Ce travail constitue une base théorique solide utilisable pour construire un outil efficace pour l'analyse de tolérance aux pannes.

Mots clés :
analyse de dépendances, logiciels critiques, langages fonctionnels, Coq, preuve de correction, analyse statique, interprétation abstraite

Abstract :
Critical software needs to obtain an assessment before commissioning in order to ensure compliance to standards. This assessment is given after a long task of software analysis performed by assessors. They may be helped by tools, used interactively, to build models using information-flow analysis. Tools like SPARK-Ada exist for Ada subsets used for critical software. But some emergent languages such as those of the ML family lack such adapted tools. Providing similar tools for ML languages requires special attention on specific features such as higher-order functions and pattern-matching. This work presents an information-flow analysis for such a language specifically designed according to the needs of assessors. This analysis is built as an abstract interpretation of the operational semantics enriched with dependency information. It is proved correct according to a formal definition of the notion of dependency using the Coq proof assistant. This work gives a strong theoretical basis for building an efficient tool for fault tolerance analysis.

Keywords :
dependency analysis, critical software, functional languages, Coq, proof of correctness, static analysis, abstract interpretation