Analysis of probabilistic programs by abstract interpretation

David Monniaux

▶ **To cite this version:**

David Monniaux. Analysis of probabilistic programs by abstract interpretation. Software Engineering [cs.SE]. Université Paris Dauphine - Paris IX, 2001. English. NNT : . tel-00084287

HAL Id: tel-00084287
https://theses.hal.science/tel-00084287

Submitted on 6 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PARIS IX DAUPHINE
UFR MATHÉMATIQUES DE LA DÉCISION

N° attribué par la bibliothèque

_____

ANALYSE DE PROGRAMMES PROBABILISTES PAR INTERPRÉTATION ABSTRAITE

Analysis of probabilistic programs by abstract interpretation

THÈSE

Pour l'obtention du titre de

DOCTEUR EN INFORMATIQUE

(Arrêté du 30 mars 1992)

David MONNIAUX

21 novembre 2001

JURY

Directeur de thèse : M. Patrick COUSOT
Professeur à l'École Normale Supérieure (Paris)

Rapporteurs : M. Prakash PANANGADEN
Professeur à l'Université Mc Gill (Montréal, Québec, Canada)
M. David SANDS
Professeur à l'Université Chalmers (Göteborg, Suède)
M. Alain TROUVÉ
Professeur à l'Université Paris XIII Nord

Suffragants : M. Neil JONES
Professeur à l'Université de Copenhague (Danemark)
M. Alain LICHNEWSKY
Professeur à l'Université Paris XI Sud
M. Vangelis PASCHOS
Professeur à l'Université Paris IX Dauphine

« L'Université n'entend donner aucune approbation ni improbation aux opinions émises dans les thèses : ces opinions doivent être considérées comme propres à leurs auteurs. »

Cet exemplaire contient des corrections de forme (typographie, fautes de frappe, référence manquante) mais aucune correction de fond par rapport à l'exemplaire de soutenance.

Résumé

L'étude de programmes probabilistes intéresse plusieurs domaines de l'informatique : les réseaux, l'embarqué, ou encore la compilation optimisée. C'est tâche malaisée, en raison de l'indécidabilité des propriétés sur les programmes déterministes à états infinis, en plus des difficultés provenant des aspects probabilistes.

Dans cette thèse, nous proposons un langage de formules permettant de spécifier des propriétés de traces de systèmes de transition probabilistes et non-déterministes, englobant celles spécifiables par des automates de Büchi déterministes. Ces propriétés sont en général indécidables sur des processus infinis.

Ce langage a à la fois une sémantique concrète en termes d'ensembles de traces et une sémantique abstraite en termes de fonctions mesurables. Nous appliquons ensuite des techniques d'interprétation abstraite pour calculer un majorant de la probabilité dans le pire cas de la propriété étudiée et donnons une amélioration de cette technique lorsque l'espace d'états est partitionné, par exemple selon les points de programme. Nous proposons deux domaines abstraits convenant pour cette analyse, l'un paramétré par un domaine abstrait non probabiliste, l'autre modélisant les gaussiennes étendues.

Il est également possible d'obtenir de tels majorants par des calculs propageant les mesures de probabilité en avant. Nous donnons une méthode d'interprétation abstraite pour analyser une classe de formules de cette façon et proposons deux domaines abstraits adaptés à ce type d'analyse, l'un paramétré par un domaine abstrait non probabiliste, l'autre modélisant les queues sous-exponentielles. Ce dernier permet de prouver la terminaison probabiliste de programmes.

Les méthodes décrites ci-dessus sont symboliques et ne tirent pas parti des propriétés statistiques des probabilités. Nous proposons d'autre part une méthode de Monte-Carlo abstrait, utilisant des interpréteurs abstraits randomisés.

Abstract

The study of probabilistic programs is of considerable interest for the validation of networking protocols, embedded systems, or simply for compiling optimizations. It is also a difficult matter, due to the undecidability of properties on infinite-state deterministic programs, as well as the difficulties arising from probabilistic aspects.

In this thesis, we propose a formulaic language for the specification of trace properties of probabilistic, nondeterministic transition systems, encompassing those that can be specified using deterministic Büchi automata. Those properties are in general undecidable on infinite processes.

This language has both a concrete semantics in terms of sets of traces, as well as an abstract semantics in terms of measurable functions. We then apply abstract interpretation-based techniques to give upper bounds on the worst-case probability of the studied property. We propose an enhancement of this technique when the state space is partitioned — for instance along the program points —, allowing the use of faster iteration methods. We propose two abstract domains suitable for this analysis, one parameterized by an abstract domain suitable for nondeterministic (but not probabilistic) abstract interpretation, one modeling extended normal distributions.

An alternative method to get such upper bounds works is to apply forward abstract interpretation on measures. We propose two abstract domains suitable for this analysis, one parameterized by an abstract domain suitable for nondeterministic abstract interpretation, one modeling sub-exponential queues. This latter domain allows proving probabilistic termination of programs.

The methods described so far are symbolic and do not make use of the statistical properties of probabilities. On the other hand, a well-known way to obtain informations on probabilistic distributions is the Monte-Carlo method. We propose an abstract Monte-Carlo method featuring randomized abstract interpreters.

Remerciements

Je remercie Patrick COUSOT pour avoir encadré mon travail depuis le DEA et au cours de mes années de thèse. Jeune étudiant, je n'avais aucune expérience de l'écriture de « papiers » scientifiques, peu de connaissances sur l'analyse de programmes et les domaines connexes, et j'ai ainsi pu profiter de son expérience. Je remercie mes collègues Antoine MINÉ et Jérôme FERET pour les échanges d'idées ainsi que la fastidieuse mais ô combien indispensable relecture des publications.

Une des grandes difficultés que j'ai rencontrées dans ma thèse est la nécessité d'aborder plusieurs domaines scientifiques habituellement séparés — l'analyse de programmes, la théorie des processus décisionnels de Markov, la statistique — , avec le risque permanent d'ignorer des résultats déjà établis mais donnés dans un autre formalisme, sous un autre vocabulaire. Heureusement, quelques personnes ont défriché le terrain entre le *model-checking* et les processus décisionnels de Markov. Je remercie en particulier Luca DE ALFARO et Roberto SEGALA de m'avoir fait profiter de leur connaissance des processus décisionnels de Markov, malgré leur emploi du temps chargé.

Le Laboratoire d'Informatique de l'École Normale Supérieure est un cadre agréable pour mener des recherches. Je tiens à remercier Mmes ISNARD et MONGIAT pour leur traitement efficace des problèmes administratifs. Je remercie également la Bibliothèque Interuniversitaire Scientifique de Jussieu de mettre ses riches collections à disposition des chercheurs extérieurs.

Je remercie MM. PANANGADEN, SANDS et TROUVÉ d'avoir accepté d'être mes rapporteurs, et MM. JONES, LICHNEWSKY et PASCHOS de m'avoir fait l'honneur d'être dans le jury de soutenance.

Je remercie les auteurs des différents logiciels dits « libres », dont j'ai eu quotidiennement l'usage au cours de cette thèse, à la fois pour l'implantation de prototypes, la réalisation de figures, l'édition, la typographie et plus généralement l'environnement informatique.

Enfin, je remercie Mai-Linh, pour m'avoir supporté durant ma thèse, ainsi que mes parents.

Contents

Part I

Cadre et résumé de la thèse

Chapter 1

Cadre des recherches développées

Introduction

L'informatique, au cours de ses quelques dizaines d'années d'existence, est passée du stade de technologie expérimentale réservée à quelques grands projets, le plus souvent militaires, au stade de technologie courante, présente à la fois dans les centres de calculs et autres bases de données et dans tous les bureaux modernes, mais aussi dans un grand nombre d'appareils, du plus petit appareil photographique jusqu'aux avions gros porteurs et aux centrales nucléaires. Dans ces derniers cas, où des vies humaines sont en jeu, on attend des systèmes un fonctionnement sûr.

Dans le cas de systèmes non informatiques, il est courant d'estimer des probabilités d'accident, notamment par décomposition d'un système complexe en plusieurs composants, pour autant que les comportements de pannes soient quasiment indépendants d'une pièce à l'autre. Les composants sont souvent décrits par leur « temps moyens entre deux pannes » (*MTBF, mean time between failures*). On peut alors estimer le comportement global du système par une combinaison des probabilités de panne des composants. Cette approche a notamment été popularisée par le rapport Rasmussen sur les risques d'accident dans les centrales nucléaires civiles aux États-Unis [66] et a depuis été appliquée à de nombreux systèmes industriels, ce qui permet, de nos jours, d'imposer réglementairement des probabilités maximales d'accident [21, ch. X]. Une telle approche peut même être employée pour des systèmes naturels : c'est ainsi que l'on estime, par exemple, la probabilité qu'un glissement de terrain se produise au dessus d'une courant pendant une période donnée [33]. Lorsque les résultats de ces analyses sont connus en fonction de certains paramètres du système choisis par l'entrepreneur, on

peut éventuellement optimiser une fonction tenant compte à la fois de la probabilité d'accident et des coûts.

Il est tentant de vouloir étendre de telles analyses à des systèmes comprenant un composant informatique. Prenons un objet courant : une imprimante laser. Celle-ci comprend plusieurs dispositifs électromécaniques et des senseurs, qui permettent de détecter la présence ou l'absence de papier dans les différentes étapes du chemin de papier. Le système est contrôlé par un petit micro-ordinateur. Il est bien connu que ce type d'imprimante a parfois des ratés logiciels : l'informatique de contrôle peut par exemple se persuader que le papier est coincé alors qu'il ne l'est pas. On peut alors vouloir obtenir des données fiables sur la fréquence d'apparition de ce type de pannes, exaspérantes pour les utilisateurs. Dans ce cas, il est possible de conduire des expériences pratiques sur le système complet ; on imagine par contre moins de telles expériences pour des avions de ligne ! Il semble donc souhaitable de pouvoir fournir des données sûres sur le comportement d'un système informatique embarqué, moyennant la connaissance de données sur son environnement. C'est dans ce but que les techniques présentées dans cette thèse ont été développées.

L'expression « probabilité qu'un logiciel soit fiable » peut avoir plusieurs sens très différents. Nous nous intéressons dans cette thèse à un de ces sens, à savoir celui qui étudie le comportement d'un système informatique donné en présence de composants probabilistes, éventuellement incomplètement spécifiés. Nous verrons ici qu'il existe d'autres définitions possibles, comme l'étude de la fiabilité du processus de conception du logiciel lui-même. Nous ne nous intéresserons pas à ces aspects.

1.1 Techniques statistiques d'analyse de logiciels

De nombreuses techniques statistiques ou probabilistes ont été proposées pour mesurer la fiabilité des logiciels, notamment dans le cas des logiciels critiques. Dans ces circonstances, une question naturelle est le rapport qu'il y a entre les techniques présentées dans cette thèse et d'autres familles de techniques qui peuvent paraître proches.

Bien que ces techniques aient en apparence le même but (obtenir des données quantitatives sur la fiabilité des logiciels), nous verrons qu'elles analysent souvent des phénomènes différents et que la solidité de leurs bases mathématiques est très variable.

1.1.1 Ingéniérie logicielle statistique

Dans de nombreux domaines d'ingénierie, la fiabilité d'un système est estimée en se basant sur des données statistiques provenant d'essais, de versions similaires ou de projets du même type (historique). Des modèles établissent alors des corrélations statistiques entre différentes données sur le développement du logiciel (nombre de lignes de codes au total, nombres de lignes de codes modifiées depuis la dernière livraison, nombre de travailleurs sur le projet....) et le nombre d'erreurs trouvées [62]. De la même façon, de tels modèles sont censés permettre d'estimer le temps de développement des logiciels en fonction de différents paramètres de complexité du code. De telles techniques peuvent fournir des indices comme "le nombre estimé d'erreurs pour 100 lignes de code".

Il est important de noter que ce type de technique vise essentiellement à estimer la probabilité d'échec d'un processus social et technique, la programmation, en fonction de différents facteurs, et non pas la probabilité de dysfonctionnement d'un logiciel donné face à un jeu de données de distribution connue. L'estimation de la probabilité qu'un logiciel contienne plus de n bogues est ainsi de même nature que l'estimation de la probabilité qu'une opération chirurgicale réussisse.

Une telle analyse est forcément de nature empirique. Selon une approche courante en sciences expérimentales, on rassemble des données statistiques pertinentes, on construit un modèle paramétrique et on ajuste les paramètres jusqu'à ce que les données et le modèle s'ajustent bien (utilisation de tests statistiques tels que le test du χ^2, estimant quantitativement l'ajustement entre une distribution expérimentale et un modèle). On essaye alors le modèle sur d'autres jeux de données, à des fins de contrôle. Dans le but de mieux adapter de telles estimations au cas particulier d'un logiciel donné, on peut ajuster de nombreux paramètres du modèle, parfois à l'aide de données obtenues par une analyse automatique du code source, le plus souvent essentiellement syntaxiques.

Pour résumer, ces méthodes de *statistical software engineering* ou de *métrique statistique du logiciel* s'intéressent à établir des statistiques répondant plutôt à des questions comme « Si nous mettons un grand nombre d'équipes de programmeurs dans ces conditions de développements, quel sera le nombre moyen de bugs dans le logiciel ? », par opposition à notre approche qui répond à des questions comme « Si nous considérons ce programme dans un environnement possédant ces propriétés statistiques, quelles seront les propriétés statistiques des sorties du programme ? ».

Les techniques proposées dans cette thèse ne sont pas basées sur une analyse du processus de développement du logiciel, mais sur le logiciel lui-même. De plus, nous nous basons sur le fonctionnement du logiciel (exprimé mathématiquement dans une *sémantique*) et non sur des critères syntaxiques.

1.1.2 Test probabiliste

Considérons un système stochastique quelconque, dont le comportement est modélisé par une variable aléatoire V. On classe les comportements du système en un nombre fini de cas par une fonction C. Prenons un exemple: V est la sémantique dénotationnelle d'un programme P, laquelle associe à une entrée (ici aléatoire) la sortie correspondante du programme, $\perp$ si le programme ne termine pas sur cette entrée et err si le programme termine sur une erreur d'exécution (division par zéro...). C pourra alors être la fonction définie par: $C(\perp) = C(\mathsf{err}) = 1$, $C(x) = 0$ où x est toute valeur de retour du programme. $C \circ V$ est donc la valeur aléatoire qui vaut 1 si et seulement si le comportement du programme est indésirable (non terminaison ou erreur d'exécution) et 0 sinon. La probabilité de comportement indésirable du programme est alors l'espérance $\mathbf{E}(C \circ V)$ de la variable aléatoire $C \circ V$.

Il est alors parfaitement loisible d'un point de vue mathématique d'effectuer des statistiques sur $C \circ V$; des techniques bien maîtrisée permettent d'obtenir alors des *intervalles de confiance*, lesquels donnent des informations telles que « Il y a 99,9% de chances que $\mathbf{E}(C \circ V)$ soit compris entre 0,25 et 0,3 », que l'on va souvent résumer par « Avec une bonne assurance, la probabilité d'erreur du système est d'environ 0,28 ».

De tels résultats s'obtiennent par exemple par des méthodes de *test statistique*, où l'on essaye le programme sur un grand nombre de cas d'entrées statistiquement distribués [61, 77]; il s'agit donc d'une *méthode de Monte-Carlo*. On améliore souvent ces méthodes par des techniques de *partition testing*, dans lequel l'espace de test est partitionné selon certains critères. Le programme est alors essayé sur un ou plusieurs éléments tirés au hasard dans chaque élément de la partition. Le choix de la partition se fait souvent sur des critères de couverture du code source ou des chemins d'exécutions possibles dans le flot de contrôle. Le choix du nombre d'essais pour chaque élément de la partition peut se faire selon des estimations de la probabilité d'erreur du système relativement à l'élément de la partition considéré [61]. Il s'agit alors de *stratified sampling* [70, 4.3.4].

Il y a en fait deux utilisations possibles de ces méthodes pour fournir des données quantitatives sur la fiabilité d'un programme. La première est d'appliquer ces méthodes à un programme dont la distribution probabiliste d'entrée est connue, en utilisant cette distribution pour l'échantillonnage. Il s'agit alors d'une méthode mathématiquement bien établie ; c'est la base de notre méthode de *Monte-Carlo abstrait* (chapitre 14). La seconde est d'appliquer cette méthode lorsqu'on ne connaît pas, ou seulement partiellement, la distribution d'entrée ; on utilise alors souvent des distributions uniformes ou d'autres plus ou moins arbitraires (lorsque la distribution uniforme ne s'applique pas, dans le cas d'ensembles infinis). Il faut bien reconnaître que dans ce cas les données numériques de fiabilité obtenues

n'ont pas de valeur mathématique ; quant à leur valeur heuristique, elle dépend beaucoup du programme et des choix plus ou moins arbitraires qui sont faits sur les distributions.

1.2 Méthodes formelles d'analyse de programmes probabilistes

L'analyse de logiciels probabilistes à partir de leur sémantique a fait naître différentes approches.

1.2.1 Méthodes analytiques

Une question que l'on se pose rapidement au vu d'un algorithme est celui de sa complexité. Il y a en fait plusieurs notions de complexité, parmi lesquelles la complexité dans le pire cas et la complexité moyenne. C'est en fait souvent la complexité moyenne qui nous intéresse, car les pires cas arrivent peu en pratique dans bon nombre de problèmes. Qui plus est, une bonne complexité en moyenne suggère l'utilisation d'*algorithmes randomisés*.

Les algorithmes probabilistes ont donc fait l'objet d'études approfondies ; citons notamment les travaux de Knuth [39], Sedgewick et Flajolet [73]. Il s'agit souvent de démonstrations mathématiques combinatoires et probabilistes assez subtiles, établies manuellement. Cependant, certaines techniques permettent d'obtenir automatiquement des données probabilistes en moyenne lorsque les algorithmes opèrent régulièrement sur des structures de données régulières ; des techniques combinatoires automatiques, s'appuyant sur des séries génératrices et des calculs formels sur des fonctions analytiques permettent d'obtenir les résultats escomptés — citons notamment les travaux de Flajolet (une introduction à ce genre d'analyses d'algorithms se trouve dans [73]).

Dans cette thèse, nous ne supposons pas que les structures de données et les algorithmes sont « réguliers »; nous considérons des programmes quelconques.

1.2.2 *Model-checking* probabiliste

Les techniques de *model-checking* [10] permettent de vérifier des propriétés de systèmes de transitions déterministes ou non-déterministes à nombre d'états finis, notamment de propriétés de *sécurité* (le fait que certains états représentatifs de pannes soient inaccessibles). Ces techniques procèdent par calculs exacts ; par exemple, dans le cas de propriétés de sécurité, l'ensemble des états accessibles est

exactement calculé. Bien entendu, dans le cas de systèmes avec de grands nombres d'états, les ensembles d'états considérés sont représentés symboliquement, notamment par des BDD (*binary decision diagrams*) [10, ch. 5]. Des propriétés plus complexes peuvent être exprimées par des formules de logiques adaptées (LTL, CTL, CTL* [10]).

Il est assez naturel d'étendre cette approche fructueuse aux processus probabilistes, voire aux processus à la fois non déterministes et probabilistes, c'est-à-dire aux *processus de décision Markoviens* [64]. Bien que ceux-ci aient été bien étudiés en recherche opérationnelle et en mathématiques financières, on s'est intéressé relativement récemment à leurs applications informatiques [3, 7, 5, 50]. Nous citerons en particulier les remarquables travaux de thèse de Luca de Alfaro [1] et Roberto Segala [74], ainsi que les travaux de l'équipe de Marta Kwiatkowska [7, 3, 4, 34, 34, 43].

Le point délicat dans cette extension est la sémantique assez particulière des systèmes de transitions lorsqu'on permet à la fois des transitions non déterministes (un choix parmi un certain ensemble sans notion de probabilité) et des transitions probabilistes. Ainsi, l'extension des logiques temporelles habituelles [10, chapter 3] n'est pas facile. De plus, la résolution des points fixes demande l'utilisation d'algorithmes plus complexes que les algorithmes habituels.

De même que l'analyse statique par interprétation abstraite non probabiliste permet de dépasser l'obligation de finitude du *model-checking* non probabiliste, les approches décrites dans cette thèse permettent de dépasser l'obligation de finitude du *model-checking* probabiliste.

1.3 Interprétation abstraite probabiliste

Dans cette thèse, nous avons développé deux grandes familles de techniques. La première est celle de l'interprétation abstraite sur les processus de décision markoviens. Au chapitre 7, nous verrons comment ceux-ci sont une extension naturelle à la fois des systèmes de transitions, usuels en informatique, et des chaînes de Markov. Nous verrons ensuite, au chapitre 8, comment analyser ces systèmes de transition par rapport à des formules logiques exprimant des propriétés sur les traces; nous développons alors deux méthodes "duales" d'analyse, l'une représentant des ensembles de mesures de probabilité, l'autre des ensembles de fonctions mesurables selon la théorie de l'interprétation abstraite [16, 18, 15] et nous proposons ainsi une véritable notion d'*interprétation abstraite probabiliste*. Au chapitre 13, nous verrons une approche dénotationnelle de cette même notion.

Nous fournissons au chapitre 9 deux treillis abstraits duaux, permettant d'adapter à ces analyses probabilistes des treillis d'interprétation abstraite non probabiliste et aux chapitres 11 et 12 deux treillis spécifiques visant à représen-

ter exactement des distributions, l'exponentielle et la gaussienne, très utiles pour analyser certains phénomènes. Nous verrons par ailleurs au chapitre 10 que certains formalismes proposés pour l'analyse de systèmes probabilistes reviennent en fait à des cas particuliers de notre notion d'interprétation abstraite.

L'autre famille de méthodes est le *Monte-Carlo* abstrait, que nous développons au chapitre 14. Cette méthode offre l'avantage de pouvoir être implantée facilement sur un interpréteur abstrait préexistant.

La majeure partie des propositions de cette thèse ont été décrites dans diverses conférences internationales à comité de lecture [57, 56, 54, 55]. Par ailleurs, certains des techniques proposées ont été implantées dans un petit analyseur.

Chapter 2

Sémantique et analyse

2.1 Sémantique des systèmes de transitions

2.1.1 Systèmes de transitions probabilistes

Dans cette partie, nous expliquerons progressivement notre notion de systèmes de transitions à la fois non-déterministes et probabilistes, en partant des cas les plus simples jusqu'aux cas les plus complexes. Nous introduirons notamment des *sémantiques de traces* pour ces systèmes.

Dans le cas de programmes déterministes on non-déterministes, sans notion de probabilités, on peut formaliser le comportement pas-à-pas du programme par un *système de transition*. Il s'agit d'un graphe orienté dont les sommets représentent les différents états possibles du programme et les arêtes les transitions possibles d'un état à l'autre. Étant donné un état initial, il est possible de calculer l'ensemble des états accessibles par un simple parcours de graphe.

Dans le cas de programmes probabilistes, on se donne une *probabilité de transition*. Si l'ensemble des états Ω est fini ou dénombrable, cette probabilité est simplement une fonction $T : \Omega \times \Omega \to [0,1]$ telle que pour tout x, $\sum_y T(x,y) = 1$. $T(x,y)$ est alors la probabilité lorsque le système est dans l'état x que le système passe dans l'état y. Un système défini par une probabilité de transition toujours constante, ou *stationnaire*, au cours de son exécution, est appelé une *chaîne de Markov*.

Considérons maintenant des propriétés que nous voulons analyser sur ces systèmes. Dans le cas de la sécurité, on voudra obtenir la probabilité de panne, c'est-à-dire la probabilité d'atteindre tel ou tel ensemble d'états non désirés. Cette propriété, comme d'autres, par exemple impliquant des contraintes d'équité, est une *propriété de trace*, et la probabilité de la propriété est en fait la mesure de probabilité de l'ensemble de traces qu'elle délimite.

Nous construisons donc une mesure de probabilité sur l'ensemble des traces,

c'est à dire l'ensemble $\Omega^{\mathbb{N}}$ des suites infinies d'états, à l'aide du théorème de Ionescu Tulcea [60, proposition V-I-1]: étant donné un état initial x_0 et une probabilité de transition T, nous définissons la mesure désirée sur $\Omega^{\mathbb{N}}$.

2.1.2 Systèmes de transitions non-déterministes et probabilistes

Le cas de programmes à la fois probabilistes et non-déterministes est plus complexe. Nous considérons alors plusieurs probabilités de transition, entre lesquelles un choix est fait à chaque itération. On parle alors non plus de chaînes de Markov, mais de *systèmes décisionnels de Markov* [64]. Ce vocabulaire s'explique par la vision, utile pour la compréhension, d'un individu qui ferait ce choix entre les différentes probabilités en suivant une *politique* ou *stratégie*, par exemple d'un *adversaire* hostile voulant maximiser la probabilité de panne.

Cette notion de décision n'est pas aussi simple qu'il n'y paraît. Nous pouvons tout d'abord distinguer différentes classes de politiques :

- L'adversaire prend sa décision à chaque étape d'exécution au vu du dernier état atteint par le système.
- L'adversaire prend sa décision à chaque étape d'exécution au vu de l'historique des états atteints par le système.
- L'adversaire prend ses décisions une fois que tous les choix aléatoires ont été faits et au vu de ceux-ci.

Décision non-déterministe par rapport au passé et au présent

Là encore, nous utiliserons le théorème de Ionescu Tulcea [60, proposition V-I-1]. Ce théorème dit que si pour tout n nous définissons une probabilité de transition U_n entre les n premiers états et le $n+1$-ième état, nous obtenons une probabilité sur l'espace de traces.

Supposons que le système est défini par une probabilité de transition de $\Omega \times Y$ vers Ω, où Y est le *domaine des choix non-déterministes*. La décision de l'intrus est alors faite selon une probabilité de transition entre Ω^n vers Y. La transition probabiliste exécutée par le système est alors la composition

$$T_n = T \circ \begin{bmatrix} Id \\ U_n \end{bmatrix}, \tag{2.1}$$

une transition de probabilité de Ω^n à Ω ; nous entendons par cette notation la transition définie par :

$$T_n(x_0, \ldots, x_{n-1}; x_n) = T(x_{n-1}, U_n(x_0, \ldots, x_{n-1}); x_n). \tag{2.2}$$

Le théorème de Ionescu Tulcea construit alors à partir de (T_n) une probabilité de transition $G(T,(U_n)_{n\in\mathbb{N}})$ de Ω (l'état initial) vers $\Omega^{\mathbb{N}}$. Notons alors

$$S_T(f,(U_n)_{n\in\mathbb{N}}) = t_0 \mapsto \int \lambda\vec{t}.f(t_0,\vec{t})\ \mathrm{d}[G(T,(U_n)_{n\in\mathbb{N}})(t_0)] \tag{2.3}$$

(on notera S s'il n'y a pas d'ambiguïté) et $R(f)$ l'ensemble des fonctions $S(T,(U_n)_{n\in\mathbb{N}})$, $(U_n)_{n\in\mathbb{N}}$ parcourant l'ensemble des suites de probabilités de transitions, U_n étant une probabilité de transition de Ω^n vers Y (chacune de ces suites constitue une politique).

Soit $E_+(f) = \sup R(f)$ la *sémantique supérieure* et $E_-(f) = \inf R(f)$ la *sémantique supérieure*. Intuitivement, si f est la fonction caractéristique d'un ensemble de traces produisant une panne, E_+ est une analyse dans le pire cas, avec un adversaire poussant à la panne, tandis que E_+ est une analyse dans le meilleur cas, avec un adversaire voulant éviter les pannes. $E_+(f)$ est souvent appelée la *valeur* du processus décisionnel de Markov par rapport à la fonction de récompense f (notons toutefois que nous utilisons un cadre théorique quelque peu différent de celui donné dans [64]).

Décision non-déterministe par rapport au présent

Nous considérons ici des politiques *stationnaires* et *sans mémoire*, c'est-à-dire celles qui restent les mêmes à chaque pas et ne prennent en compte que l'état actuel du système pour prendre une décision. Il s'agit ainsi d'une restriction du modèle décrit au §2.1.2 au cas où $U_n(x_1,\ldots,x_n)$ ne dépend que de x_n.

Ce modèle est particulièrement important, parce que tout en étant considérablement plus simple que le modèle non stationnaire du §2.1.2, il lui est équivalent pour certaines classes de propriétés (rem. 2.2.8).

2.1.3 Discussion

Le point de vue le plus naturel sur les processus non-déterministes et probabilistes est que les décision non-déterministes sont prises au fur et à mesure de l'exécution du programme en prenant en compte à la fois l'état actuel du système et l'historique des états passés. C'est de cette façon que l'on étudie habituellement les processus décisionnels de Markov [64]. Au §2.2, nous verrons des méthodes effectives d'analyse pour ce modèle.

On peut considérer que ce modèle est excessivement pessimiste, en ce que le non-déterminisme, qui simule le comportement d'un environnement inconnu, dépend de l'historique de l'*état interne* du système ; cela est excessif, il faudrait que seule la partie de l'historique observable depuis l'environnement soit prise

en compte. Nous arrivons ainsi à l'étude de *processus décisionnels de Markov à observation partielle*.

Les travaux de Cleaveland [11] s'intéressent plutôt au modèle où les choix non-déterministes sont pris *après* les choix probabilistes. Cela simplifie la théorie, dans une certaine mesure, puisque le produit du processus à analyser et d'un processus d'observation tel qu'un automate de Büchi non-déterministe est alors facilement analysable (voir §2.2.2 pour une discussion sur la difficulté d'utilisation d'automates non-déterministes pour l'analyse dans le modèle où les choix sont faits au fur et à mesure de l'exécution). Nous verrons comment appliquer une méthode de Monte-Carlo pour analyser cette sémantique au §5.

Nous pouvons également envisager l'étude de processus probabilistes à temps continu. Comme d'habitude pour les systèmes à temps continu, il convient de réduire le système vers un système à temps discret [45, 46].

2.2 Analyse abstraite arrière et avant

Comme nous l'avons vu au §2, la sémantique que nous considérons pour les systèmes de transitions probabilistes prend en argument une fonction mesurable définie sur l'ensemble des traces d'exécution (possibles ou impossibles). Cette fonction exprime la propriété que nous voulons tester sur les traces (c'est-à-dire les suites infinies d'états) ; il pourra s'agir, par exemple, de la fonction caractéristique de l'ensemble des traces passant par un certain état (ou ensemble d'états) au moins une fois, ou l'ensemble des traces restant dans un certain ensemble d'états. L'intégrale qui est alors calculée est la probabilité d'atteindre cet état (ou ensemble d'états) ou de rester dans l'ensemble d'états. Nous pouvons par ailleurs choisir la fonction qui à une trace associe le nombre de fois où elle passe par un état (ou ensemble d'états). L'intégrale calculée est alors le temps passé (un pas d'itération comptant pour une unité de temps) passé dans l'état (ou ensemble d'états). Nous verrons ici des généralisations de ces propriétés et comment les analyser.

2.2.1 Les propriétés à analyser

Nous considérons une propriété à analyser sur les traces. À chaque état initial nous associons son *potentiel*, c'est-à-dire l'intégrale de cette propriété sur les traces partant de cet état (ou l'ensemble des intégrales possibles, si l'on prend en compte le non-déterminisme). Les propriétés à analyser sont exprimées sous la forme de fonctions mesurables positives définies sur l'ensemble des traces d'exécution; nous appelons ces fonctions *valuateurs de traces*. Nous considérerons en fait une classe de valuateurs de traces définie syntaxiquement par certaines formules.

Fonctions de potentiel

Soit $I = [0,1]$ ou $[0,+\infty]$. Soit X un ensemble d'états fini ou dénombrable — nous imposons cette contrainte de cardinalité pour éviter des complexités théoriques. $\mathcal{P}(X)$ est l'ensemble des parties de X ; c'est une σ-algèbre.

Soit $X^{\mathbb{N}} \to I$ l'ensemble des fonctions mesurables de X^{ω} dans I, muni de l'ordre produit. Nous appellerons ces fonctions « valuateurs ».

Fonctions booléennes Prenons $I = [0,1]$, voire même $I = \{0,1\}$. Nous nous intéressons à

$$R(V) = \left\{ \lambda t_0 . \int \lambda \langle t_1, \ldots \rangle . V(\langle t_0, t_1, \ldots \rangle) \, \mathrm{d} \left(\bigotimes_{k=1}^{\infty} T_k \right) (x_0) \mid (T_n)_{n \in \mathbb{N}} \in \mathscr{T}^{\mathbb{N}} \right\} \tag{2.4}$$

Nous considérons des formules écrites dans le langage suivant :

formula1 ::=
name
constant
name $+_{set}$ *formula1*
constant $+_{set}$ *formula1*
lfp(*name* $\mapsto$ *formula1*)
gfp(*name* $\mapsto$ *formula1*)
shift(*formula1*)
let *name* = *formula1* in *formula1*

Soit shift : $X^{\mathbb{N}} \to X^{\mathbb{N}}$: $(\text{shift}.t)_k = t_{k+1}$.

Soit env_t l'ensemble des environnements de valuateurs (un environnement de valuateurs associe à chaque *name* un valuateur), muni de l'ordre produit.

$[\![formula]\!]_t : env_t \to (X^\omega \to I)$ est définie par récurrence :

$$[\![name]\!]_t.env = env(name) \tag{2.5}$$
$$[\![constant]\!]_t.env = constant \tag{2.6}$$
$$[\![f_1 +_S f_2]\!]_t.env = \lambda t.\chi_S(t_0).([\![f_1]\!]_t\, env) + \chi_{S^C}(t_0).([\![f_2]\!]_t\, env) \tag{2.7}$$
$$[\![\text{lfp}(name \mapsto f)]\!]_t.env = \text{lfp}(\lambda\phi.[\![f]\!]_t.env[name \mapsto \phi]) \tag{2.8}$$
$$[\![\text{gfp}(name \mapsto f)]\!]_t.env = \text{gfp}(\lambda\phi.[\![f]\!]_t.env[name \mapsto \phi]) \tag{2.9}$$
$$[\![\text{shift}(f)]\!]_t.env = ([\![f]\!]_t.env) \circ \text{shift} \tag{2.10}$$
$$[\![\text{let } name = f_1 \text{ in } f_2]\!]_t.env = [\![f_2]\!]_t.env[name \mapsto [\![f_1]\!]_t.env] \tag{2.11}$$

Nous considérerons en particulier les valuateurs suivants :

Sécurité Soit A un ensemble (mesurable) d'états. La propriété d'*accessibilité* de A définit l'ensemble des traces passant par A. La *sécurité* est son contraire.

$$\text{lfp}(f \mapsto 1 +_A \text{shift}\, f) \tag{2.12}$$

Ce point fixe est atteint en temps dénombrable :

$$\text{lfp}(\underbrace{\phi \mapsto [\![1 +_A \text{shift}\, f]\!]_t.[f \mapsto \phi]}_{\Psi}) = \bigsqcup_n \Psi^n(0)$$

Comme toutes les fonctions $\Psi^n(0)$ sont mesurables (composition de fonctions mesurables), la limite $[\![\text{lfp}(f \mapsto 1 +_A \text{shift}\, f)]\!]_t$ est mesurable.

Vivacité Soit A un ensemble (mesurable) d'états. La propriété de *vivacité* de A définit l'ensemble des traces restant indéfiniment dans A. C'est le dual de la propriété d'accessibilité. Elle correspond à la formule

$$\text{gfp}(f \mapsto \text{shift}\, f +_A 0) \tag{2.13}$$

Comme précédemment, $[\![\text{gfp}(f \mapsto \text{shift}\, f +_A 0)]\!]_t$ est mesurable.

Acceptation de Büchi Soit A un ensemble (mesurable) d'états. La propriété d'*acceptation de Büchi* associée à A définit l'ensemble des traces passant par A infiniment souvent. Nous verrons ici comment définir cet ensemble à l'aide de points fixes.

Considérons W_n l'ensemble des traces passant par A au moins n fois. W_1 est la propriété d'accessibilité $[\![\text{lfp}(f \mapsto 1 +_A \text{shift}\, f)]\!]_t$; $W(2)$ est la propriété d'accessibilité imbriquée $[\![\text{lfp}(f \mapsto \text{shift}\, \text{lfp}(f \mapsto 1 +_A \text{shift}\, f) +_A \text{shift}\, f)]\!]_t$; et, plus généralement, notant $\Psi(\phi) = \text{lfp}(f \mapsto [\![n +_A \text{shift}\, f]\!]_t).[n \mapsto \phi]$, $W_n = \Psi^n(1)$. Clairement, $(W_n)_{n\in\mathbb{N}}$ est décroissante, et la propriété d'acceptation de Büchi W_∞ est la limite de la suite. De plus, W_∞ est un point fixe de Ψ: considérons la trace S; si S passe par A un nombre infini de fois A, alors $W_\infty(S) = \Psi(W_\infty(S)) = 1$, sinon $W_\infty(S) = \Psi(W_\infty(S)) = 0$. W_∞ est alors le plus grand point fixe de Ψ: si nous prenons un point fixe F de Ψ, alors $F \leq 1$; par récurrence sur n, $F = \Psi^n(F) \leq \Psi^n(1) = W_n$ et donc $F \leq W_\infty$. La propriété d'acceptation de Büchi correspond donc à la formule :

$$\text{gfp}(C \mapsto \text{lfp}(R \mapsto \text{shift}(C) +_A \text{shift}(R))) \tag{2.14}$$

Si ϕ est mesurable, alors $\Psi(\phi)$ est mesurable, puisque le plus petit point fixe est atteint en temps dénombrable ; tous les $(W_n)_{n\in\mathbb{N}}$ sont donc mesurables. Leur limite dénombrable est donc mesurable.

Valuateur de sommation Nous verrons maintenant une autre famille de valuateurs, les *valuateurs de sommation*. Le valuateur de sommation associé à une fonction $f : X \mapsto [0, +\infty]$ est la fonction

$$[\![\textstyle\sum A]\!]_t : \left| \begin{array}{ll} X^\omega & \to [0,+\infty] \\ (x_n)_{n\in\mathbb{N}} & \mapsto \sum_{k=0}^{\infty} f(x_k) \end{array} \right. \tag{2.15}$$

Cette fonctions peut évidemment être formulée comme un plus petit point fixe :

$$[\![\Sigma f]\!]_t = \text{lfp}(\phi \to f + \phi \circ \text{shift}) \tag{2.16}$$

Cette construction a deux applications évidentes :

- compter le nombre moyen de fois qu'un programme passe par un ensemble d'états A — f est alors la fonction caractéristique de A ;
- compter le temps moyen utilisé par le processus — f est alors une fonction qui à chaque état associe le temps passé dans cet état (0 pour les états finaux).

2.2.2 Logiques temporelles

Les logiques temporelles [10, chapitre 3] permettent de spécifier des propriétés des systèmes de transition.

Automates de Büchi déterministes

La *logique temporelle linéaire* (LTL) exprime des propriétés définissant des ensembles de traces. Comme les systèmes de transitions probabilistes induisent des probabilités non pas sur les états mais sur les ensembles de traces, LTL semble fort bien convenir pour les analyses probabilistes. Rappelons brièvement la syntaxe de cette logique :

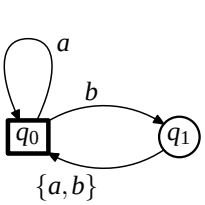
$$
\begin{aligned}
\mathit{LTL\_formula} ::= {} & \mathit{state\_formula} \\
& \mathit{LTL\_formula}_1 \wedge \mathit{LTL\_formula}_2 \\
& \mathit{LTL\_formula}_1 \vee \mathit{LTL\_formula}_2 \\
& \neg \mathit{LTL\_formula} \\
& \mathit{LTL\_formula}_1 \ \textbf{until} \ \mathit{LTL\_formula}_2 \\
& \mathit{LTL\_formula}_1 \ \textbf{release} \ \mathit{LTL\_formula}_2 \\
& \textbf{eventually} \ \mathit{LTL\_formula} \\
& \textbf{always} \ \mathit{LTL\_formula}
\end{aligned}
$$

et sa sémantique ($B = (b_n)_{n\in\mathbb{N}} \in S^{\mathbb{N}}$) :

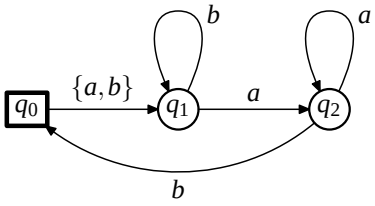
$$
\begin{aligned}
B \models \mathit{state\_formula} &\iff b_0 \models \mathit{state\_formula} \\
B \models \neg \mathit{LTL\_formula} &\iff B \not\models \mathit{LTL\_formula} \\
B \models \%\mathit{textit}\, f_1 \wedge f_2 &\iff (B \models f_1) \wedge (B \models f_2) \\
B \models f_1 \vee f_2 &\iff (B \models f_1) \vee (B \models f_2) \\
B \models f_1 \ \textbf{until} \ f_2 &\iff \exists k \geq 0 \begin{cases} (\forall j < k \ \mathrm{shift}^j(B) \models f_1) \\ (\mathrm{shift}^k(B) \models f_2) \end{cases} \\
B \models f_1 \ \textbf{release} \ f_2 &\iff \forall j \geq 0 \begin{cases} (\forall i < j \ \mathrm{shift}^i(B) \not\models f_1) \\ (\mathrm{shift}^j(B) \models f_2) \end{cases}
\end{aligned}
$$

où $\mathrm{shift}^k(B)$ est le suffixe de la suite B à partir de l'indice k.

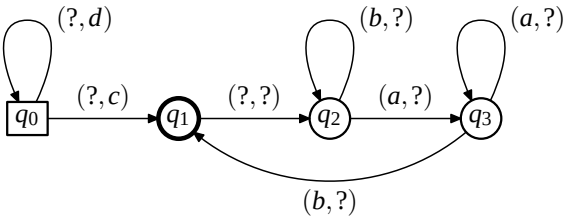
Nous allons maintenant rappeler quelques propriétés des automates de Büchi [10, ch. 9] [76, §I.1]. Les automates de Büchi sont une extension aux mots infinis des classiques automates finis sur les mots finis. Un automate de Büchi non-déterministe sur l'alphabet Σ est défini par un ensemble (fini) d'états Q, une relation de transition $\Delta \subseteq Q \times \Sigma \times Q$, un ensemble d'états initiaux $Q_0 \subseteq Q$ et un ensemble d'états acceptants $A \subseteq Q$. $(s_0, l, s_1) \in \Delta$ veut dire que l'automate peut passer de l'état s_0 à l'état s_1 lorsqu'il lit la lettre l ; nous imposons que pour tout



(a) Équité pour A



(b) Équité pour a et b



(c) Équité pour a et b et accessibilité pour c

Figure 2.1: Automates de Büchi pour diverses conditions. L'alphabet est $\{a,b\} \times \{c,d\}$. L'état initial est encadré ; les états acceptants sont entourés en gras.

état s_0 et lettre l il y ait au moins un état s_1 tel que $(s_0,l,s_1) \in \Delta$. Considérons alors une suite de lettres $L = (l_n)_{n\in\mathbb{N}} \in \Sigma^{\mathbb{N}}$. On dit qu'une suite d'états $R = (r_n)_{n\in\mathbb{N}} \in R^{\mathbb{N}}$ est une *exécution* pour L si $r_0 \in Q_0$ et si pour tout n, $(r_n,l_n,r_{n+1}) \in \Delta$. Une exécution est dite *acceptante* si elle passe un nombre infini de fois par A. Une trace (suite d'états) est *acceptée* s'il existe pour elle une exécution acceptante.

Les automates de Büchi, et c'est là un de leurs grands intérêts, fournissent une méthode élégante pour l'analyse des propriétés de la logique temporelle linéaire (LTL) [10, §3.2]. Considérons une formule LTL F. Appelons Z l'ensemble des formules atomiques sur les états à l'intérieur de F. Il existe un algorithme [10, §9.4] qui construit un automate de Büchi $\mathscr{A}$ sur l'alphabet $\mathcal{P}(Z)$ tel qu'une trace (suite d'états) $(s_n) \in S^{\mathbb{N}}$ est un modèle de F si et seulement si $\mathscr{A}$ accepte $(b_n) \in \left\{0,1\right\}^{Z})^{\mathbb{N}}$ où $z \in b_n$ si et seulement si $s_n \models z$. Vérifier la formule F sur un système de transitions non-déterministe S est alors équivalent à vérifier la condition d'acceptation de Büchi (voir §2.2.1) sur le produit synchrone de S et de $\mathscr{A}$.

Le non-déterminisme du produit synchrone provient de deux sources : le non-déterminisme de S et celui de A. Malheureusement, lorsque l'on rajoute les probabilités, nous ne pouvons plus considérer ces deux non-déterminismes ensemble.

En effet, ce que nous voulons calculer est un majorant de:

$$\sup_{\substack{\textit{nondéterminisme}\\ \textit{du système}}} \mathbf{E}\left(\sup_{\substack{\textit{nondéterminisme}\\ \textit{de l'automate}}} \begin{cases} 1 & \text{si exécution acceptante} \\ 0 & \text{sinon} \end{cases} \right)$$

et nous constatons que l'automate prend ses choix non-déterministes au vu de la suite complète des choix probabilistes. Les méthodes d'analyse expliquées dans ce chapitre ne s'appliquent pas à de tels système (mais la méthode de Monte-Carlo du Ch. 5 s'applique).

D'un autre côté, nous pouvons éviter ce problème en nous restreignant aux automates de Büchi *déterministes*. Un automate de Büchi déterministe est un automate de Büchi dont la relation de transition R est en fait une fonction (pour tout état q_0, il existe exactement un état q_1 tel que $(q_0,q_1) \in R$). Nous considérerons alors une fonction de transition $f : Q \times \Sigma \to Q$.

L'ensemble des automates de Büchi déterministes est clos par produit fini (mais pas par complément [76, example 4.2]). Ceci est intéressant, puisque l'on peut ainsi exprimer des *contraintes d'équité*. On spécifie souvent les systèmes concurrents sans préciser l'ordonnanceur, mais on suppose souvent que celui-ci est *équitable* : aucun processus ne devrait rester infiniment bloqué (d'une façon équivalente, pour tout processus à tout instant, il existe un instant dans le futur où ce processus pourra progresser). Ce que nous voulons mesurer est la probabilité de mauvais fonctionnement du système sous hypothèse d'équité (Fig. 2.1).

Étant donné un automate de Büchi déterministe $\mathscr{A}$ (espace d'états Q, fonction de transition f) et un processus décisionnel de Markov P (espace d'état X, espace d'entrées non-déterministes Y, probabilité de transition T de $X \times Y$ à X), nous définissions leur produit synchrone comme le processus décisionnel de Markov P', d'espace d'état $X \times Q$ et probabilité de transition T' de $(X \times Q) \times Y$ dans $X \times Q$ définie par :

$$T'(((x_1,q_1),y),(x_2,q_2)) = T((x_1,y),x_2) \tag{2.17}$$

Logiques en temps arborescent

La logique en temps arborescent CTL [10, §3.2] est très utilisée pour l'analyse de systèmes non-déterministes (mais néanmoins non probabilistes). Il est donc naturel d'envisager de l'étendre aux systèmes probabilistes. Malheureusement, l'extension L'extension pCTL [30], est malheureusement à la fois plutôt technique et en fait peu utile. Il est toutefois possible d'approximer certains ensembles d'états définis par ces formules par interprétation abstraite.

2.2.3 Analyse supérieure en arrière

Une solution bien connue au problème de la valeur optimale d'un processus décisionnel de Markov est l'*itération de valeur* [64, §7.2.4]. Cette méthode est d'un intérêt principalement théorique pour l'analyse de processus décisionnels de Markov à ensembles d'états finis, puisque l'on ne contrôle pas sa vitesse de convergence alors que de bien meilleurs algorithmes existent. Il s'agit en fait d'une généralisation aux processus décisionnels de Markov de l'analyse d'accessibilité en arrière pour les systèmes non-déterministes ; nous appliquerons donc similairement des techniques d'interprétation abstraite, donnant effectivement des majorants des probabilités des propriétés à analyser.

Sémantique supérieure en arrière

Soit env_e l'ensemble des environnements de fonctions de potentiels (un environnement de fonctions de potentiels associe à chaque *name* une fonction de potentiel), muni de l'ordre produit.

$[\![formula]\!]_{e+} : (X \to I) \to (X \to I)$ est définie par induction :

$$[\![name]\!]_{e+}.env = env(name) \tag{2.18}$$
$$[\![constant]\!]_{e+}.env = \lambda x.constant \tag{2.19}$$
$$[\![f_1 +_S f_2]\!]_{e+}.env = \chi_S.([\![f_1]\!]_{e+} env) + \chi_{S^C}.([\![f_2]\!]_{e+} env) \tag{2.20}$$
$$[\![\mathrm{lfp}(name \mapsto f)]\!]_{e+}.env = \mathrm{lfp}(\lambda\phi.[\![f]\!]_{e+}.env[name \mapsto \phi]) \tag{2.21}$$
$$[\![\mathrm{gfp}(name \mapsto f)]\!]_{e+}.env = \mathrm{gfp}(\lambda\phi.[\![f]\!]_{e+}.env[name \mapsto \phi]) \tag{2.22}$$
$$[\![\mathrm{shift}(f)]\!]_{e+}.env = \sup_{T\in\mathscr{T}} (\overleftarrow{T}([\![f]\!]_{e+}.env)) \tag{2.23}$$
$$[\![\mathrm{let}\ name = f_1\ \mathrm{in}\ f_2]\!]_{e+}.env = [\![f_2]\!]_{e+}.env[name \mapsto [\![f_1]\!]_{e+}.env] \tag{2.24}$$

Quant au valuateur de sommation,

$$[\![\Sigma f]\!]_{e+} = \mathrm{lfp}\left(\phi \mapsto f + \sup_{T\in\mathscr{T}} (\overleftarrow{T}.\phi)\right) \tag{2.25}$$

Calcul approximé effectif

Notre but est, étant donné une formule f, de fournir un majorant (pour l'ordre produit) de $[\![f]\!]_{e+}$, ce qui, comme nous le verrons aux Th. 2.2.3 et 2.2.6, nous donnera un majorant de $E_+([\![f]\!]_t)$. Notre méthode est un cas particulier d'*interprétation abstraite* [18, 16, 15, 14] : nous désirons utiliser des *majorants symboliques* sur lesquels appliquer les opérations. Nous supposons avoir à notre disposition un domaine abstrait tel que ceux décrits aux §4.1 ou §4.2.

- On approxime les plus petits points fixes à l'aide d'*opérateurs d'élargissement*. Il existe par ailleurs dans certains domaines (§4.1.3) d'autres méthodes pour certaines classes de formules, comme par exemple l'accessibilité ou le plus petit point fixe imbriqué dans la formule d'acceptation de Büchi.

- On approxime les plus grands points fixes en s'arrêtant à un nombre d'itérations déterminé heuristiquement. Comme toutes les itérations sont des majorants du plus grand point fixe, cette approche est sûre. Les heuristiques peuvent par exemple utiliser une distance comparant les itérations : les itérations s'arrêtent lorsque la distance passe en dessous d'un certain seuil.

Adversaire basé sur le présent

Dans la définition des adversaires ou des politiques, il est nécessaire de choisir si l'adversaire considère uniquement le dernier état du système ou toute l'histoire de l'exécution. Nous verrons ici les adversaires qui ne regardent que l'état actuel du système, également appelés *politiques sans mémoire*. Si nous fixons la politique, le système devient une chaîne de Markov.

Sémantique concrète Notons $X^{\mathbb{N}}$ l'espace des suites dénombrables d'éléments de X muni de la σ-algèbre produit $\mathcal{P}(X)^{\otimes\mathbb{N}}$. Soit T une probabilité de transition entre $X \times Y$ et X. Considérons l'ensemble des transitions de probabilité

$$\mathscr{T} = \{T \circ (x \mapsto (x, f(x)))_T \mid f \in X \to Y\},$$

en notant g_T la probabilité de transition associée à l'opération déterministe g et $\circ$ la composition des transitions de probabilité. Remarquons que l'opérateur (en arrière) associé sur les fonctions mesurables est

$$\overleftarrow{\mathscr{T}} = \{f \mapsto (\overleftarrow{T}.f) \circ (x \mapsto (x, g(x))) \mid g \in X \to Y\}.$$

La relation d'abstraction entre les sémantiques

Theorème 2.2.1. *Soit f une formule ne contenant ni* lfp *ni* gfp *et où pour toute construction $f_1 +_A f_2$, f_1 est une constante. Soit env un environnement de valuateurs. Notant $E_+(env)$ l'application de E_+ à env coordonnée par coordonnée,*

$$[\![f]\!]_{e+}.(E_+(env)) = E_+([\![f]\!]_t.env) \tag{2.26}$$

Corollaire 2.2.2. *Soit f une formule ne contenant ni* lfp *ni* gfp *et où pour toute construction $f_1 +_A f_2$, f_1 est une constante. Soit env un environnement de valuateurs. Notant $E_+(env)$ l'application de E_+ à env coordonnée par coordonnée,*

$$[\![\text{lfp } f]\!]_{e+}.(E_+(env)) = E_+([\![\text{lfp } f]\!]_t.env) \tag{2.27}$$

Theorem 2.2.3. *Soit f une formule. Soit env un environnement de valuateurs. Supposons que $H \geq E_+(env)$ coordonnée par coordonnée. Alors*

$$[\![f]\!]_{e+}.(H) \geq E_+([\![f]\!]_t.env) \tag{2.28}$$

Adversaire connaissant le présent et le passé

Une autre façon de considérer les adversaires est de leur permettre de prendre leurs décisions en fonction de toute l'historique du calcul. Cela semble en fait mieux modéliser les réactions d'un environnement inconnu.

Theorème 2.2.4. *Soit f une formule ne contenant pas* gfp*. Soit env un environnement de valuateurs. Notant $E_+(env)$ l'application de E_+ à env coordonnée par coordonnée,*

$$[\![f]\!]_{e+}.(E_+(env)) = E_+([\![f]\!]_t.env) \tag{2.29}$$

Theorème 2.2.5. *Les sémantiques vérifient :*

$$E_+([\![\Sigma f]\!]_t) = [\![\Sigma f]\!]_{e+}. \tag{2.30}$$

Theorème 2.2.6. *Soit f une formule. Soit env un environnement de valuateurs. Supposons que $H \geq E_+(env)$ coordonnée par coordonnée ; alors*

$$[\![f]\!]_{e+}.(H) \geq E_+([\![f]\!]_t.env). \tag{2.31}$$

Remarque 2.2.7. Les résultats ci-dessus restent valables même si nous n'utilisons que des politiques déterministes dans la définition de E_+.

Ce fait est remarquable ; en effet, dans des cadres théoriques plus larges, tels que les jeux à deux personnes, cette propriété n'est plus vraie. Prenons par exemple le jeu bien connu rocher–papier–ciseaux [1] du point de vue du premier joueur : il va chercher une stratégie qui maximise le gain minimal (-1 pour une perte, 1 pour une victoire) qu'il puisse obtenir selon la stratégie de l'adversaire (point de vue *minimax*, très utilisé dans la programmation de jeux tels que les échecs). Il est évident que toute stratégie déterministe est mauvaise de ce point de vue, puisque

[1] Il s'agit d'un jeu pour deux personnes, qui jouent des parties successives. À chaque partie, les deux joueurs font simultanément un geste « rocher », « papier » ou « ciseaux » ; « rocher » bat « ciseaux », « papier » bat « rocher » et « ciseaux » bat « papier ».

l'autre joueur a alors à sa disposition une stratégie déterministe qui fait perdre le premier joueur à tout coup, le résultat du minimax sur les stratégies déterministes est -1. Toutefois, une stratégie stationnaire faisant un choix aléatoire uniformément distribué donne un résultat de minimax de 0.

Remarque 2.2.8. Pour les propriétés d'accessibilité (un seul point fixe, un plus petit point fixe), les politiques stationnaires et sans mémoire sont équivalentes aux politiques avec mémoire.

Nous avons constaté que $E_+(\llbracket \mathrm{lfp}(f \mapsto 1 +_A f) \rrbracket_t)$ ne changeait pas selon que l'on se restreigne aux politiques stationnaires sans mémoire dans E_+. Cette propriété n'est malheureusement pas vrai pour la plupart des formules. Prenons d'abord un exemple simple :

et la formule $\mathrm{lfp}\left(\phi \mapsto \mathrm{lfp}(\psi \mapsto 1 +_B \mathrm{shift}(\psi)) +_A \mathrm{shift}(\phi)\right)$ (passage par A puis par B). Toute stratégie probabiliste stationnaire peut être exprimée comme la probabilité α d'aller de S à A (ainsi, la probabilité d'aller de S à B est $1-\alpha$). En tout cas, la probabilité de la formule ci-dessus est strictement inférieure à 1. D'autre part, il est évident que la stratégie déterministe non stationnaire de passer par A au premier pas, puis par B, donne une probabilité 1 à la formule étudiée.

Considérons maintenant une formule avec un plus grand point fixe :

```
while (random() < ndt_[0,1[) { };
```

`ndt_[0,1[()` retourne de façon non-déterministe un réel x dans $[0,1[$ (nous pouvons également choisir un entier positif n et prendre $x = 1 - 1/n$). Nous nous intéressons à la propriété de vivacité de rester indéfiniment dans la boucle. Le choix d'une politique stationnaire est le choix de x, et pour toute valeur fixée de x la probabilité de rester indéfiniment dans la boucle est évidemment 0. Il est par contre possible de fournir $x_1, x_2, \ldots$, changeant à chaque tour de boucle, telle que la probabilité $\prod_i x_i$ soit arbitrairement proche de 1.

Non-déterminisme et plus grands points fixes Pourquoi semblons-nous perdre de la précision si la formule contient des plus grands points fixes ? Nous verrons ici qu'il s'agit d'un problème fondamental avec le non-déterminisme et les traces infinies.

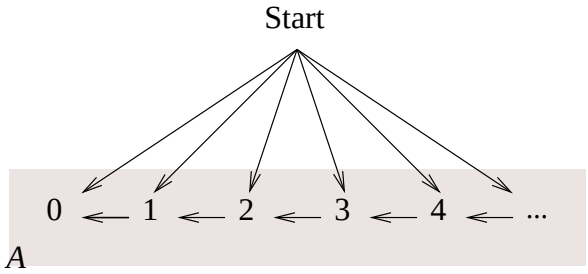


Figure 2.2: Un système de transition pour lequel $E_+(\llbracket \mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0) \rrbracket_t) < \llbracket \mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0) \rrbracket_{e+}$. Tandis que pour tout n il existe une trace passant par A n fois, il n'existe aucune trace passant un nombre infini de fois par A. Pour ce système, l'analyse de plus grands points fixes est intrinsèquement imprécise.

Pas de non-déterminisme Dans ce cas, il n'y absolument pas de perte de précision. Un fait essentiel est que E_+ est alors non seulement continue supérieurement, mais aussi continue inférieurement.

Theorème 2.2.9. *Considérons un système sans non-déterminisme. Soit f une formule. Soit env un environnement de valuateurs. Notant $E_+(env)$ l'application de E_+ à env coordonnée par coordonnée,*

$$\llbracket f \rrbracket_{e+} .(E_+(env)) = E_+(\llbracket f \rrbracket_t .env). \tag{2.32}$$

Non-déterminisme Nous avons vu que, lorsqu'on ajoute le non-déterminisme, E_+ reste continue supérieurement mais non plus inférieurement. Nous n'avons pu prouver que $E_+(\llbracket \mathrm{gfp}\,\Phi \rrbracket_t).env = \llbracket \mathrm{gfp}\,\Phi \rrbracket_{e+} .E_+(env)$, mais seulement l'inégalité $E_+(\llbracket \mathrm{gfp}\,\Phi \rrbracket_t).env \leq \llbracket \mathrm{gfp}\,\Phi \rrbracket_{e+} .E_+(env)$. Malheureusement, l'égalité est fausse, et nous donnerons ici un contre-exemple. Il n'y a d'ailleurs pas besoin de probabilités, le problème se pose sur les systèmes non-déterministes. Voyons maintenant le contre-exemple (Fig. 2.2). Considérons la formule $\phi = \mathrm{shift}(\mathrm{gfp}(f \mapsto \mathrm{shift}(f) +_A 0))$ et les itérations pour gfp, à la fois pour $\llbracket \phi \rrbracket_t$ et $\llbracket \phi \rrbracket_{e+}$. $\llbracket \phi \rrbracket_t$ est l'ensemble des traces passant infiniment par A. Il n'existe évidemment pas de telle trace, donc $E_+(\llbracket \phi \rrbracket_t) = 0$. D'autre part, $\llbracket \phi \rrbracket_{e+}$ est la fonction $(A, n) \mapsto 1$. Il s'agit donc d'une sur-approximation stricte.

Analyse abstraite

Nous verrons ici comment calculer des approximations supérieures de $\llbracket f \rrbracket_{e+}$ par interprétation abstraite.

Cas général Nous introduisons une sémantique abstraite $[\![f]\!]^\sharp_{e+}$, approximation supérieure de f:

- le treillis utilisé fournit une approximation $\mathrm{pre}^\sharp$ de la fonction "shift" ;
- les plus petits points fixes sont approximés supérieurement par la limite stationnaire des itérations accélérées par un opérateur d'élargissement [18, §4.3] ;
- les plus grands points fixes sont approximés supérieurement par itération finie : $\mathrm{gfp}\, f \leq f^n(\top)$ pour tout n.

Partionnement Dans le cas de programmes informatiques, l'espace des états est généralement $P \times M$, où P est l'ensemble des points de programmes, ou, plus généralement, un quelconque *partitionnement* fini du programme — par exemple, suivant le context d'appel des procédures.

Nous prenons ici un modèle où chaque instruction du programme peut se décomposer en:

- un choix nondéterministe ;
- un choix aléatoire ;
- une opération déterministe ;
- un saut conditionnel déterministe selon l'état du programme.

L'équation de transition peut alors s'écrire :

$$F(h) = \mathrm{choice}^*_Y \circ \mathrm{random}^*_{R_l} \circ {F_l}^*_p \left(\sum_{l' \in P} \phi^*_{\tau_l l'} \left(h(l', \bullet) \right) \right) \tag{2.33}$$

en utilisant les opérateurs suivants :

$$\mathrm{choice}^*_Y(h) = m \mapsto \sup_{y \in Y} h(m, y) \tag{2.34}$$

$$\mathrm{random}^*_{R_l}(h) = m \mapsto \int h(m, r)\, \mathrm{d}\mu_{\mathcal{R}_l}(r) \tag{2.35}$$

$${F_l}^*_p(h) = h \circ F_l \tag{2.36}$$

$$\phi^*_A(h) = h.\chi_A \tag{2.37}$$

Considérons maintenant les itérations de F pour atteindre un plus petit point fixe. Nous pouvons écrire l'itération coordonnée par coordonnée sous la forme :

$$\begin{array}{ll} f_1^{(n+1)} & = F_1(f_1^{(n)}, \ldots, f_{|P|}^{(n)}) \\ \vdots & \vdots \\ f_{|P|}^{(n+1)} & = F_{|P|}(f_1^{(n)}, \ldots, f_{|P|}^{(n)}) \end{array} \tag{2.38}$$

Nous pouvons alors appliquer des techniques d'*itérations chaotiques et asynchrones* [16, §2.9], qui permettent une implémentation plus efficace de la recherche de tels points fixes.

2.2.4 Analyse en avant

Nous nous intéresserons ici à des formules closes

- ne contenant pas de lieur « *let* » ;
- pour lesquels les liaisons établies par lfp et gfp ne traversent ni lfp ni gfp.

Nous utiliserons la sémantique abstraite suivante :

$$[\![name]\!]^\sharp_{Fwd(name_R)} = \mu^\sharp \mapsto \mu^\sharp \text{ si } name = name_R \tag{2.39}$$
$$[\![constant]\!]^\sharp_{Fwd(name_R)} = \{0\} \tag{2.40}$$
$$[\![f_1 +_S f_2]\!]^\sharp_{Fwd(name_R)} = [\![f_1]\!]^\sharp_{Fwd(name_R)} \text{ if } name_R \in f_1 \tag{2.41}$$
$$[\![f_1 +_S f_2]\!]^\sharp_{Fwd(name_R)} = [\![f_2]\!]^\sharp_{Fwd(name_R)} \text{ if } name_R \notin f_1 \tag{2.42}$$
$$[\![\text{shift}(f)]\!]^\sharp_{Fwd(name_R)} = [\![f]\!]^\sharp_{Fwd(name_R)} \circ T^\sharp \tag{2.43}$$
$$[\![\text{lfp}(\phi \mapsto f)]\!]^\sharp_{Fwd(name_R)} = 0 \tag{2.44}$$
$$[\![\text{gfp}(\phi \mapsto f)]\!]^\sharp_{Fwd(name_R)} = 0 \tag{2.45}$$
$$\tag{2.46}$$

$$[\![name]\!]^\sharp_{Out} = \{0\} \tag{2.47}$$

$$[\![constant]\!]^\sharp_{Out}.\mu^\sharp = I^\sharp(constant, \mu^\sharp) \tag{2.48}$$

$$\begin{aligned}[\![f_1 +_S f_2]\!]^\sharp_{Out}.\mu^\sharp = \; & [\![f_1]\!]^\sharp_{Out}.\{\phi_S.\mu | \mu \in \mu^\sharp\} + \\ & [\![f_2]\!]^\sharp_{Out}.\{\phi_{S^C}.\mu | \mu \in \mu^\sharp\}\end{aligned} \tag{2.49}$$

$$[\![\text{shift}(f)]\!]^\sharp_{Out}.\mu^\sharp = [\![f]\!]^\sharp_{Out} \circ \mathscr{T}^\sharp(\mu^\sharp) \tag{2.50}$$

$$[\![\text{lfp}(name_R \mapsto f)]\!]^\sharp_{Out}.\mu^\sharp = S^\sharp\left(\mu^\sharp \mapsto [\![f]\!]^\sharp_{Out} \circ [\![f]\!]^\sharp_{Fwd(name_R)}.\mu^\sharp\right) \tag{2.51}$$

$$\begin{aligned}[\![\text{gfp}(name_R \mapsto f)]\!]^\sharp_{Out}.\mu^\sharp = \; & down\big([\![f]\!]^\sharp_{Out} \circ (\nu^\sharp \mapsto \mu^\sharp +^\sharp [\![f]\!]^\sharp_{Fwd(name_R)}(\nu^\sharp))^{N_0(f,\mu^\sharp)} \\ & + I^\sharp([\![f]\!]_{Fwd(name_R)})^{N_0(f,\mu^\sharp)}, \mu^\sharp)\big)\end{aligned} \tag{2.52}$$

Dans cette dernière définition, N_0 est un entier quelconque déterminé par l'analyseur au cours de l'analyse. $down(X)$ est la clôture inférieure de X: $\{y \in \mathbb{R}_+ \mid \exists x \in X \; y \leq x\}$.

Nous avons alors la propriété suivante :

Theorem 2.2.10. *Pour toute formule f, mesure μ et politique non-déterministe $(U_n)_{n\in\mathbb{N}}$,*

$$\int [\![f]\!]_t \; \mathrm{d}(G(T,(U_n)).\mu) \in \gamma(\left[\!\left[\mu^\sharp\right]\!\right]^\sharp_{Out}). \tag{2.53}$$

2.2.5 Discussion

La méthode d'analyse en arrière que nous avons décrite est une généralisation de la méthode d'itération de valeur utilisée en recherche opérationnelle pour calculer la valeur optimale de processus décisionnels de Markov. Notre analyse d'accessibilité est reliée à l'étude de modèles positifs bornés [64, §7.2], où la récompense 1 est donnée la première fois que le processus passe à travers l'ensemble d'états considéré. L'analyse de vivacité est reliée à l'étude de *modèles négatifs* [64, §7.3], où la récompense -1 est donnée la première fois que le processus quitte l'ensemble d'états considéré. Nos remarques du §1 se retrouvent d'ailleurs sous une autre forme dans l'analyse des modèles négatifs. Notre point de vue est toutefois plus général, puisque nous permettons des spécifications de propriétés de traces complexes, avec une sémantique intuitive et une sémantique effective, ces deux sémantiques étant liées par des égalités et des inégalités.

Une extension possible de cet ensemble de propriétés est les *modèles atténués*, dans lesquels l'importance des actions lointaines est atténuée par un facteur multiplicatif $\lambda \in]0,1[$ à chaque itération. Il s'agit alors d'étudier des points fixes d'opérateurs contractants dans des espaces de Banach. L'intérêt de telles formules

pour l'analyse de programmes informatiques n'est pas évident. Une autre extension possible est l'étude de moyennes non seulement dans l'espace, mais aussi dans le temps — le calcul de la moyenne d'une fonction le long de l'exécution d'un processus. Comme la propriété à évaluer est le quotient de deux propriétés de sommation, il n'y a pas de manière évidente de la calculer par itérations.

Nous avons mentionné brièvement (§2.2.3) la différence entre les processus décisionnels de Markov et les jeux à deux joueurs. Ces jeux peuvent modéliser des questions tels que le choix d'une stratégie optimale par le programme pour minimiser la probabilité d'une panne pour tous les environnements possibles. L'étude de méthodes effectives pour ces modèles est très dure [22].

Chapter 3

Treillis abstraits pour l'analyse en avant

3.1 Sommes finies de mesures localisées

Nous avons proposé [54] un treillis permettant de réutiliser à la fois dans sa théorie et dans son implémentation des treillis d'interprétation abstraite non probabiliste.

3.1.1 Définition

Une suite finie A_i de sous-ensembles mesurables de X, deux à deux disjoints, et des coefficients associés $\alpha_i \in \mathbb{R}_+$ représentent l'ensemble des mesures μ telles que :

- μ est concentrée sur $\bigcup A_i$
- pour tout i, $\mu(A_i) \leq \alpha_i$.

En termes d'implantation, les A_i sont des concrétisations d'éléments abstraits, par exemple des polyèdres (Fig. 3.1).

Cette abstraction est intuitive, mais elle est malheureusement difficilement utilisable. En effet, la contrainte de disjonction des ensembles est difficile à conserver avec des sémantiques $[\![c]\!]$ non injectives. Nous utiliserons donc plutôt la définition suivante : une suite finie A_i de sous-ensembles mesurables de X et des coefficients associés $\alpha_i \in \mathbb{R}_+$ représentent l'ensemble des mesures μ telles que :

- $\mu = \sum \mu_i$;
- pour tout i, μ_i est concentrée sur A_i;
- pour tout i, $\mu(A_i) \leq \alpha_i$.

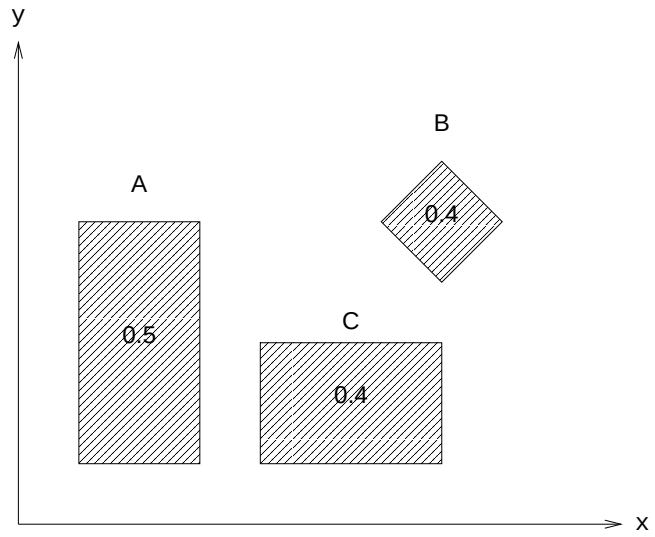


Figure 3.1: Une valeur abstraite représentant les mesures μ telles que $\mu(A) \leq 0.5$, $\mu(B) \leq 0.4$ and $\mu(C) \leq 0.4$.

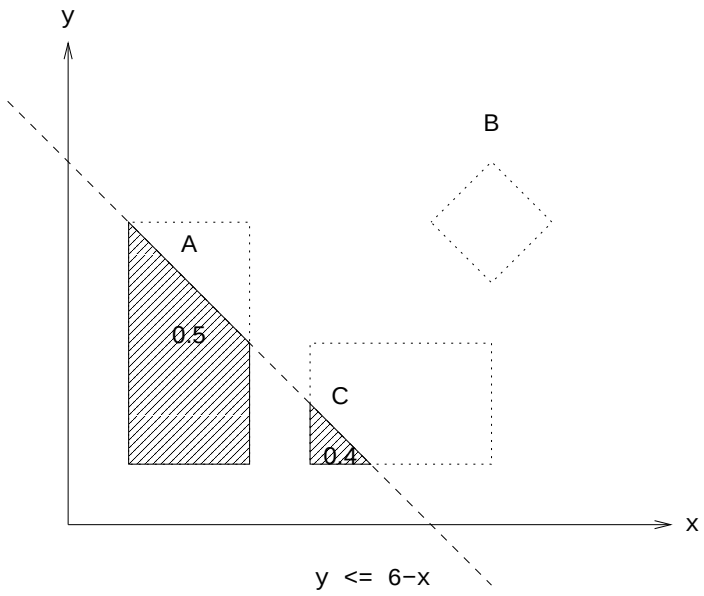


Figure 3.2: La valeur abstraite de la Fig. 9.1 après être passée dans la première branche d'un test `if y<=6-x...`.

Chacun des sous-ensembles A_i est la concrétisation d'un élément d'un treillis abstrait $X^\sharp$, muni d'une fonction monotone de concrétisation $\gamma : (X^\sharp, \sqsubseteq) \to (\mathcal{P}(X), \subseteq)$.

3.1.2 Opérations abstraites

Voyons maintenant les opérations abstraites correspondant à ce domaine:

Constructions déterministes Soit $f : X \to Y$ une opération d'abstraction $f^\sharp : X^\sharp \to Y^\sharp$. L'opération correspondante sur les mesures abstraites est

$$f_p^\sharp : (A^\sharp, \alpha_i)_{1\leq i\leq m} \mapsto (f^\sharp(A^\sharp), \alpha_i)_{1\leq i\leq m}. \tag{3.1}$$

Générateurs non-déterministes Considérons l'opération de choix non déterministe dans un ensemble Y. Nous utilisons l'abstraction :

$$(Z_\lambda, w_\lambda)_{\lambda\in\Lambda} \mapsto (H^\sharp(Z_\lambda), w_\lambda)_{\lambda\in\Lambda} \tag{3.2}$$

où $H^\sharp$ est une abstraction of $A \mapsto A \times Y$.

Choix aléatoires Considérons un générateur de données aléatoires `random` suivant une distribution μ_R. Nous désirons une abstraction de $\mu \mapsto \mu \otimes \mu'$. Supposons que $\mu' = \sum_{j=1}^n \mu'_j$ où μ'_j est concentrée sur $\gamma(B_j^\sharp)$ et de poids β_j au maximum. Prenons maintenant μ d'abstraction $(A_i^\sharp, \alpha_i)_{1\leq i\leq m}$. Alors

$$\mu \otimes \mu' = \sum_{\substack{1\leq i\leq m\\ 1\leq j\leq n}} \mu_i \otimes \mu'_j \tag{3.3}$$

et $\mu \otimes \mu'$ peut être représenté par $(A_i^\sharp \otimes^\sharp B_j^\sharp, \alpha_i\beta_j)_{1\leq i\leq m, 1\leq j\leq n}$.

Tests Rappelons que $\phi_W(\mu)$ est la restriction de μ à W. Supposons que nous disposons d'une abstraction $F_W^\sharp$ de $X \mapsto X \cap W$. L'opération correspondante (voir Fig. 3.2) sur les mesures abstraites est

$$f_p^\sharp : (A^\sharp, \alpha_i)_{1\leq i\leq m} \mapsto (F_W^\sharp(A^\sharp), \alpha_i)_{1\leq i\leq m}. \tag{3.4}$$

3.1.3 Élargissement

Un opérateur d'élargissement simpliste est

$$(A^\sharp, \alpha_i)_{1\leq i\leq m} \nabla (B^\sharp, \beta_j)_{1\leq j\leq n} = (A^\sharp \nabla B^\sharp, \alpha_i \nabla \beta_j)_{1\leq i\leq \max(m,n)} \tag{3.5}$$

suivi d'un rassemblement de certaines zones afin de borner la taille de la structure. Nous proposons des variantes heuristiques autour de cette technique.

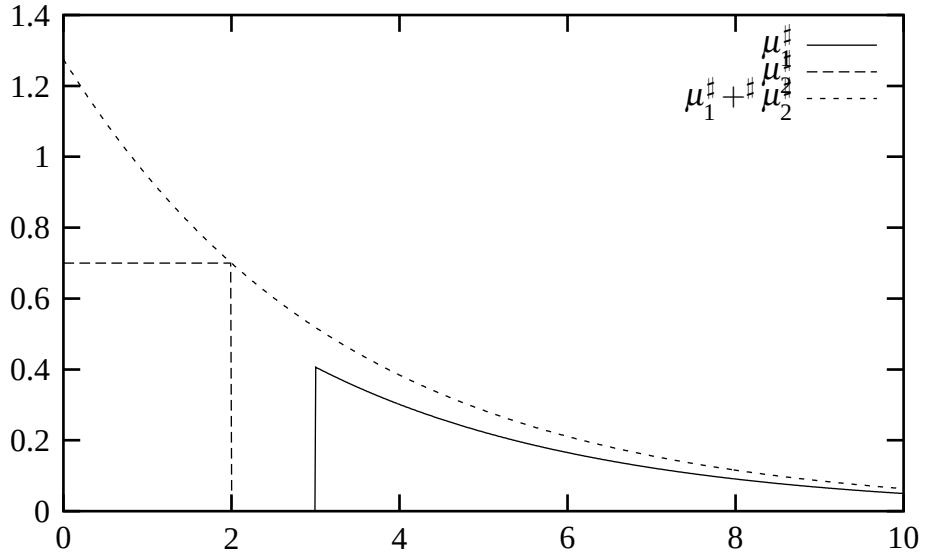


Figure 3.3: Addition abstraite de mesures. $C_x = (1, 0.3)$, $a_x = 3$, $b_x = +\infty$, $W' = 0.7$, $C'_x = \mathsf{none}$, $a'_x = 0$, $b'_x = 2$.

3.2 Queues sous-exponentielles

Nous proposons ici un domaine spécialisé exprimant la décroissance sous-exponentielles des queues de distributions des variables entières. Ce domaine

3.2.1 Domaine de base

Nous définissons ici un domaine représentant des ensembles de mesures μ. Une ensemble symbolique de mesures est soit $\bot$, soit contient d'une part un majorant du poids total de chaque mesure, d'autre associe à chaque variable entière v_i :

- un intervalle entier, avec éventuellement des bornes infinies ;
- une donnée C_i : soit un symbole none, soit deux coefficients (α_i, β_i), où $0 < \alpha_i < 1$, $\beta_i \in \mathbb{R}_+$. (α_i, β_i) impose la contrainte

$$\forall n \in \mathbb{N}\ \mu(\vec{v} \mid v_i = n) \leq \alpha_i \beta_i^n. \tag{3.6}$$

Nous définissons par des majorations aisées des versions abstraites des opérations arithmétiques :

- assignement d'une variable dans une autre ;

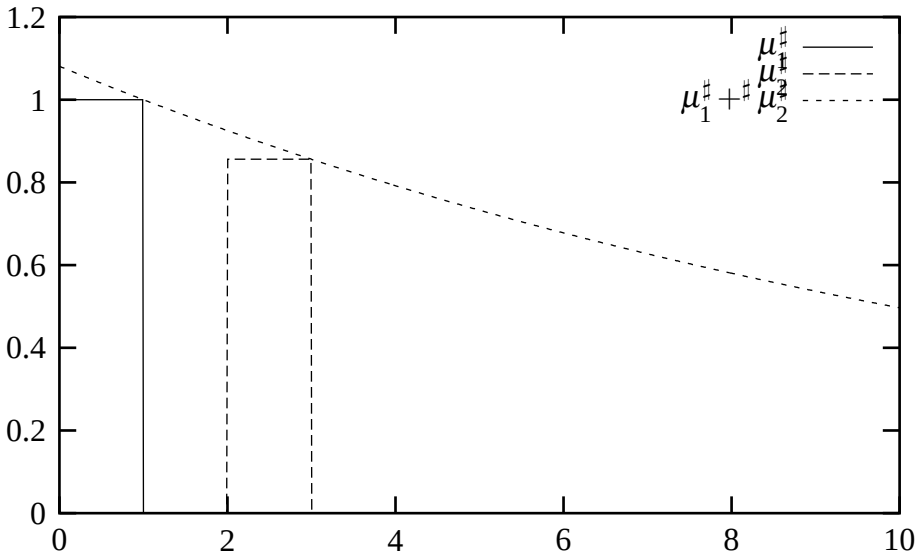


Figure 3.4: * Addition abstraite de mesures. $C_x = \mathsf{none}$, $a_x = 0$, $b_x = 1$, $W = 1$, $a'_x = 2$, $b'_x = 3$, $W' = 0.856$.

- addition d'une constante à une variables ;
- addition de deux variables vers une troisième ;
- générateur aléatoire (simpliste).

ainsi que les opérations suivantes :

- addition de mesures ;
- borne supérieure ;
- élargissement.

L'opérateur d'élargissement consiste essentiellement à appliquer un élargissement sur les réels à α et au logarithme de β. Le problème est que l'usage de réels approchés en virgule flottante peut provoquer des élargissements trop pessimistes. Des heuristiques de seuil pourront être nécessaires.

3.2.2 Sommes finies

Le domaine exposé au §3.2.1 est encore insuffisant, notamment pour le traitement des générateurs aléatoires. Nous appliquons donc une construction de sommes finies semblable à celle du §3.1.

Chapter 4

Treillis abstraits pour l'analyse en arrière

4.1 Fonctions en escalier

Nous avons proposé [57], d'une façon duale du §3.1, de majorer des fonctions mesurables par des fonctions en escalier.

4.1.1 Définition

Prenons un treillis abstrait $X^\sharp$, muni d'une fonction monotone de concrétisation $\gamma : (X^\sharp, \sqsubseteq) \to (\mathcal{P}(X), \subseteq)$. Nous imposons de plus que les images par γ soient mesurables. La suite finie $(A_i^\sharp, \alpha_i)_{1 \leq i \leq m}$ représente l'ensemble des fonctions f inférieures point par point à $\sum_{i=1}^{m} \alpha_i . \chi_{\gamma(A_i^\sharp)}$. Notons qu'il n'y a pas unicité de représentation : on peut représenter le même ensemble de fonctions de plusieurs façons.

Il est toutefois possible de faire un test de comparaison par décomposition des fonctions sur une partition commune de sous-ensembles.

4.1.2 Opérations abstraites

Constructions déterministes Soit $f : X \to Y$ dont l'image inverse est abstraite par $f^{-1\sharp} : Y^\sharp \to X^\sharp$. L'opération correspondante sur les mesures abstraites est

$$f_p^\sharp : (A^\sharp, \alpha_i)_{1 \leq i \leq m} \mapsto (f^{-1\sharp}(A^\sharp), \alpha_i)_{1 \leq i \leq m}. \tag{4.1}$$

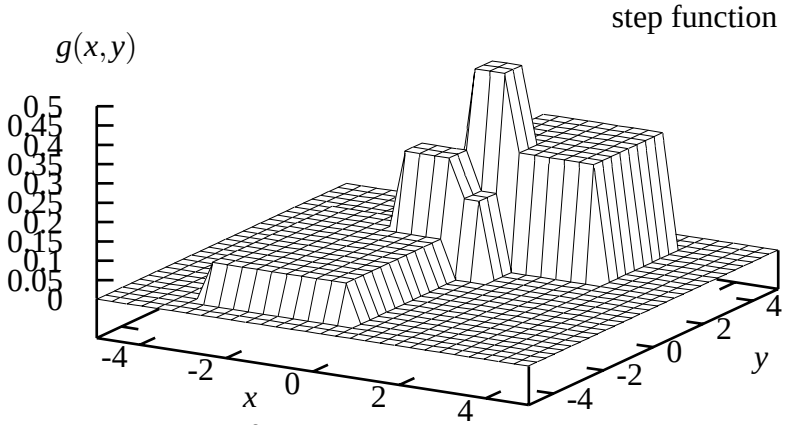


Figure 4.1: Exemple de fonction en escalier :: $0.2\chi_{[-1,1]\times[0,2]} + 0.3\chi_{[0,3]\times[1,4]} + 0.1\chi_{[-3,0]\times[-4,1]}$.

Choix non-déterministe Nous voulons une abstraction de

$$[\![\text{choice}_Y]\!]_p^* : f \mapsto \left(x \mapsto \sup_{y\in Y} f(x,y) \right). \tag{4.2}$$

Supposons que nous avons une abstraction $\pi_1^\sharp$ de la fonction de projection π_1 de $X \times Y$ à X; Alors une abstraction possible est

$$[\![\text{choice}_Y]\!]_p^{*\sharp} : (X_\lambda^\sharp, \alpha_\lambda)_{\lambda\in\Lambda} \mapsto (\pi_1^\sharp(X_\lambda^\sharp), \alpha_\lambda)_{\lambda\in\Lambda}. \tag{4.3}$$

Cette abstraction n'est pas forcément très bonne ; suivant comment la même fonction est décomposée, la projection approximée peut être bonne ou mauvaise (Fig. 4.2). Nous discuterons donc maintenant d'améliorations possibles. Considérons la partition de $X_\lambda^\sharp$ suivant la valeur de $\pi_1^\sharp(X_\lambda^\sharp)$. Considérons un élément de cette partition: $K \subseteq \Lambda$ tel que

$$K = \{\lambda \mid \pi_1^\sharp(X_\lambda^\sharp) = Z^\sharp\}. \tag{4.4}$$

Prenons

$$\alpha_{Z^\sharp} = \sup_{\substack{M\subseteq K \\ \bigcap_{\lambda\in M}\gamma(X_\lambda^\sharp)\neq\emptyset}} \sum_{\lambda\in M} \alpha_\lambda \tag{4.5}$$

En regroupant les $Z^\sharp$ et les $\alpha_{Z^\sharp}$, nous avons maintenant une meilleure approximation (p.ex. pour le cas de la Fig. 4.2). Cette meilleure approximation est bien plus simple si les $\gamma(X_\lambda^\sharp)$ sont deux-à-deux disjoints. Cela arrive par exemple si le treillis abstrait $X^\sharp$ est le treillis plat généré par une partition de X.

(a) Mauvaise projection (b) Bonne projection

Figure 4.2: Projeter directement la décomposition peut donner des résultats très sur-approximés, suivant la décomposition utilisée. Ici, la même fonction est décomposée soit en la fonction caractéristique d'un rectangle, soit comme la somme de cinq fonctions caractéristiques. Dans le premier cas, le résultat est bon, dans le second cas, c'est une très mauvaise approximation.

Notons qu'un domaine abstrait particulièrement intéressant est obtenu quand $X^\sharp$ est $\mathcal{P}(D)$, l'ensemble des parties d'une partition D de X: les valeurs abstraites peuvent alors être mémorisées comme des éléments de $[0,1]^D$ (et non $[0,1]^{2^D}$) après l'étape de simplification suivante :

$$S : \left| \begin{array}{l} [0,1]^{\mathcal{P}(D)} \to [0,1]^D \\ f \qquad\quad \mapsto d \mapsto \sum_{\substack{W \in \mathcal{P}(D) \\ d \in W}} f(W) \end{array} \right. \tag{4.6}$$

Il y a une connexion de Galois $[0,1]^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} [0,1]^D$ où $\alpha(f)(d) = \sup_{x \in d} f(x)$.

Choix aléatoire Nous voulons obtenir ici une abstraction de `R:=random` où `R` est une nouvelle variable et `random` suit la mesure de probabilité μ_R.

Supposons que $\mu_R = \sum_{k=1}^n \mu_k$ où chaque μ_k est concentré sur un sous-ensemble M_k de R. Par exemple, pour $R = \mathbb{R}$, la mesure de probabilité uniforme sur $[0,1]$ peut être décomposée en n mesures μ_k, la mesure de Lebesgue sur $[k/n,(k+1)/n]$. Appelons π_X et π_R les projections de $X \times R$ sur X et R respectivement.

La sémantique abstraite proposée est: $[\![\texttt{r:=random}]\!]^{*\sharp}(A_i^\sharp, \alpha_i)_{1 \le i \le m}$ s'envoie sur $(A_{i,k}^\sharp, \alpha_i.\beta_{i,k})_{1 \le i \le m, 1 \le k \le n}$ où $A_{i,k}^\sharp$ est une abstraction de $\pi_X(\gamma(A_i^\sharp) \cap (X \times M_k))$ et $\beta_{i,k} \ge \mu_k(\pi_R(A))$ (Fig. 4.3 explique comment nous avons construit les approximations de Fig. 4.4).

Tests L'abstraction des tests est aisée, pourvu que nous ayons des abstractions de R_W et $+$:

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]_p^{*\sharp}.f^\sharp = R^\sharp_{[\![b]\!]} \circ [\![c_1]\!]_p^{*\sharp}.f^\sharp +^\sharp R^\sharp_{[\![b]\!]^C} \circ [\![c_2]\!]_p^{*\sharp}.f^\sharp \tag{4.7}$$

où $R_W : f \mapsto f.\chi_W$.

L'abstraction $+^\sharp$ est juste la concaténation des suites finies; quant à R_W:

$$R^\sharp_{W^\sharp} = (X_\lambda^\sharp, \alpha_\lambda)_{\lambda \in \Lambda} \mapsto (X_\lambda^\sharp \cap^\sharp W^\sharp, \alpha_\lambda)_{\lambda \in \Lambda} \tag{4.8}$$

Opérateur d'élargissement Nous utilisons des heuristiques de rapprochement semblables à celles suggérées pour les mesures.

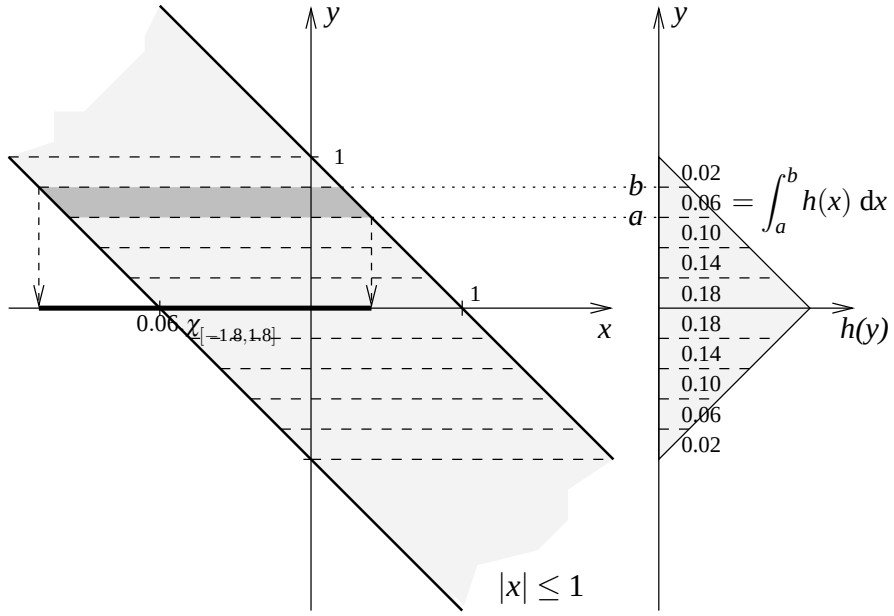


Figure 4.3: Construction de la valeur de sortie de Fig. 4.4 pour $n = 10$. Le domaine abstrait $|x+y| \leq 1$ est découpé en tranches selon l'axe y. Chaque tranche S_k est projetée sur l'axe y et l'intégrale de la fonction de distribution h de `(centered_uniform()+centered_uniform())/2` est prise sur cette projection, donnant un coefficient α_k. La tranche est alors projetée sur l'axe des x et cette projection B_k, associée au coefficient α_k, est un élément de la valeur abstraite $\sum_{k=i}^{n} \alpha_k \cdot \chi_{B_k}$. Les approximations données dans Fig. 4.4 ont été obtenues pour plusieurs nombres de tranches.

4.1.3 Cas d'un domaine non-déterministe fini

Dans le cas d'un domaine non-déterministe $X^\sharp$ fini, les éléments du domaine abstrait pour les fonctions de potentiel s'expriment comme des suites finies dans $[0,1]^{X^\sharp}$. L'analyse de propriétés d'accessibilité revient alors à la recherche de la plus petite solution d'un système d'inéquations linéaires, qui peut être résolu par programmation linéaire [64, §7.2.7]. Dans le cas d'espaces d'états de la forme $\{0,1\}^N$, il existe des algorithmes pour la programmation linéaire basés sur des MTBDDs [23, 44].

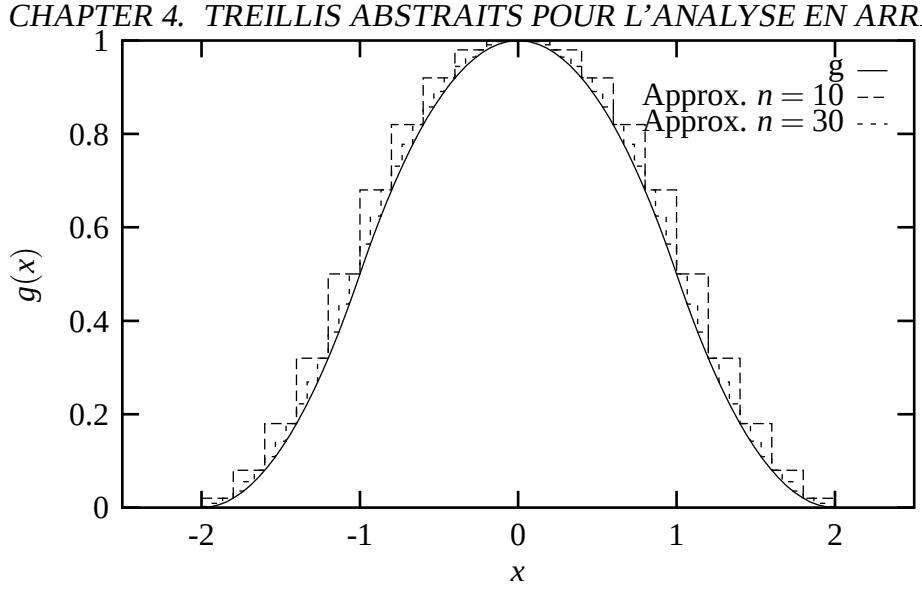


Figure 4.4: Deux abstractions de la même fonction de potentialité

4.2 Gaussiennes

Nous proposons un domaine abstrait de gaussiennes généralisées, pouvant par exemple aider à démontrer le peu d'influence des valeurs extrêmes sur le comportement des programmes.

4.2.1 Définition

Nous considérons des fonctions gaussiennes généralisées, de la forme

$$\begin{array}{rcl} E & \to & \mathbb{R}_+ \\ \vec{v} & \mapsto & \exp(-Q(\vec{v}) + L(\vec{v}) + q_0) \end{array}$$

où Q est une forme quadratique positive et L une forme linéaire sur E telles que $\ker Q \subseteq \ker L$, un $\mathbb{R}$-espace vectoriel de dimension finie représentant le domaine d'évolution des variables, et q_0 est un réel.

4.2.2 Opérations

Borne supérieure et élargissement Nous diagonalisons les deux formes quadratiques dans une base orthogonale commune, et prenons une borne inférieure des polynômes quadratiques coordonnée par coordonnée. Dans le cas de l'opérateur d'élargissement, au bout d'un certain nombre d'itérations, nous approximons inférieurement sur une des coordonnées par le polynôme nul. Comme cela fait décroître strictement le rang de la forme quadratique, la convergence est assurée.

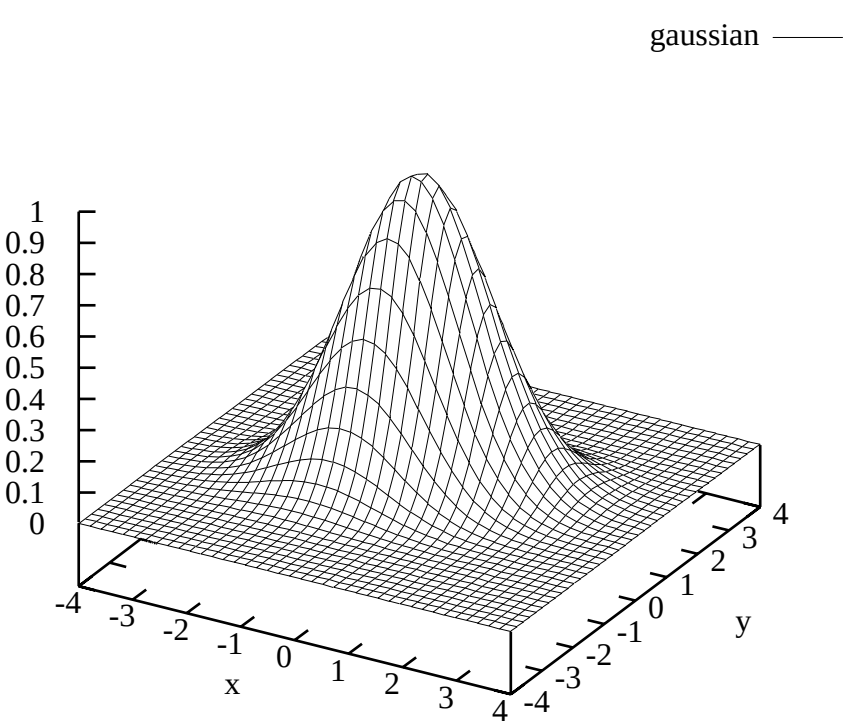


Figure 4.5: Une gaussienne généralisée, de matrice $\left(\begin{smallmatrix} 0.6 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ dans la base orthogonale $\left(\begin{smallmatrix} \cos 0.4 & -\sin 0.4 \\ \sin 0.4 & \cos 0.4 \end{smallmatrix}\right)$.

Générateurs aléatoires Nous voulons l'opérateur abstrait associé à $\rho = \texttt{random}_\lambda$ où ρ est une nouvelle variable.

$$g = [\![\rho = \texttt{random}]\!]^*_p.f = \vec{v} \mapsto \int_x f(\vec{v}+x\vec{e})\,\mathrm{d}\mu_R(x) \tag{4.9}$$

où $\vec{e}$ est un vecteur de base supplémentaire correspondant à ρ et μ_R est la distribution de ρ. Nous supposons de plus que μ_R est donné par la gaussienne $\exp(-(\lambda x^2 + c_1))$, et f par (Q,L,c), alors

$$\begin{aligned} g(\vec{v}) &= \int_{-\infty}^{+\infty} \exp\left(-(Q(\vec{v}+x\vec{e}) + L(\vec{v}+x\vec{e}) + c + \lambda x^2 + c_1)\right)\,\mathrm{d}x \\ &= \exp\left(-\left(\underbrace{Q(\vec{v}) + \frac{1}{4\alpha}Q^*(\vec{v},\vec{e})^2}_{Q'(\vec{v})} + \underbrace{L(\vec{v}) + \frac{1}{2\alpha}Q^*(\vec{v},\vec{e})L(e^\sharp)}_{L'(\vec{v})} + \underbrace{c + c_1 + \frac{L(\vec{e})^2}{4\alpha}}_{c'}\right)\right) \end{aligned} \tag{4.10}$$

Opérations linéaires Nous considérons ici les opérations de programme telles que $v_n := \sum_i \alpha_i v_i$ et plus généralement toute transformation linéaire M associant le vecteur de variables $\vec{V}' = M.\vec{V}$ en sortie de l'instruction au vecteur $\vec{V}$ d'entrée. Alors $(Q',L',c) = ({}^tM\,QM, LM, c)$.

Autres opérations Lorsqu'une variable est affectée par une opération non décrite plus haut, on met à zéro les coefficients de la forme linéaire et de la forme quadratique affectant cette variable.

Chapter 5

Méthode de Monte-Carlo

Les méthodes exposées dans les chapitres précédents se fondent sur la représentation symbolique d'ensembles de mesures ou de fonctions mesurables. Nous n'avons pas encore utilisé les propriétés statistiques des probabilités, en particulier l'utilisation possible de générateurs aléatoires au sein même de l'analyseur. L'approche expérimentale est pourtant classique, s'agissant par exemple d'évaluer les probabilités des différentes faces d'un dé pipé. Nous verrons ici comment combiner cette approche avec l'interprétation abstraite.

5.1 Fondements de la méthode de Monte-Carlo

Nous verrons d'abord les grandes lignes de la méthode de *Monte-Carlo abstrait* proposée.

5.1.1 La méthode de test de Monte-Carlo

« Monte-Carlo », en allusion aux célèbres casinos de cette ville, est un terme général désignant un ensemble de méthodes mettant en jeu des générateurs aléatoires et justifiées par la convergence statistiques. Nous considérerons ici le problème de l'estimation de la probabilité d'un certain événement.

Considérons par exemple le choix d'un point M uniformément dans le carré $[0,1]^2$. Nous voulons déterminer expérimentalement la probabilité de la propriété $P(M)$: « M appartient au quart de disque de centre 0 et de rayon 1 » (Fig. 5.1). Nous pouvons utiliser l'algorithme suivant :

$c \leftarrow 0$
for $i = 1$ to n **do**
  $x \leftarrow$ random
  $y \leftarrow$ random

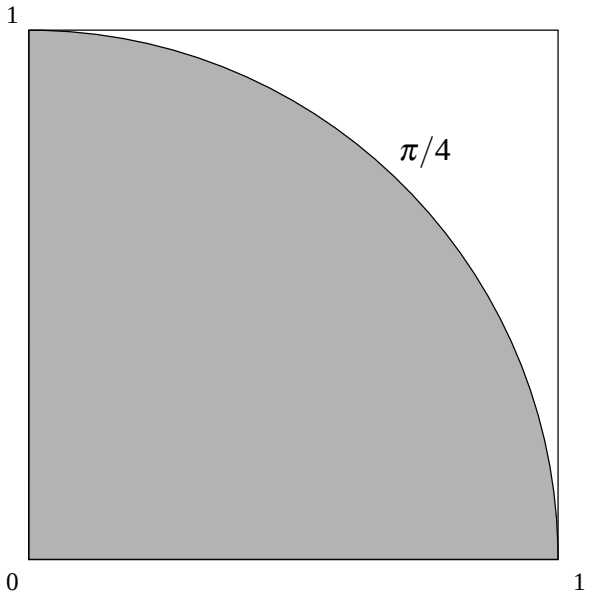


Figure 5.1: Calcul de $\pi/4$ par la méthode de Monte-Carlo.

```
    if x^2 + y^2 ≤ 1 then
      c ← c + 1
    end if
  end for
  p ← c/n
```

Il s'agit d'approximer l'espérance $\mathbf{E}V$ d'une variable aléatoire à image dans $[0,1]$ par la moyenne expérimentale $V^{(n)}$ sur n essais.

Bien entendu, dans cet exemple géométrique simple, cette méthode est peu efficace, car il en existe d'autres largement plus performantes. Elle devient intéressante lorsque l'aire à estimée est délimitée par une fonction caractéristique compliquée. L'analyse de programmes tombe généralement dans ce dernier cas.

Combien devons-nous effectuer d'itérations ? Nous pouvons utiliser la borne de Chernoff-Hoeffding [75, inequality A.4.4] :

$$\Pr\left(\mathbf{E}V \geq V^{(n)} + t\right) \leq e^{-2nt^2} \tag{5.1}$$

Cette borne, justifiée mathématiquement, signifie que la probabilité de sous-estimer V en utilisant $V^{(n)}$ de plus de t est inférieure à e^{-2nt^2}.

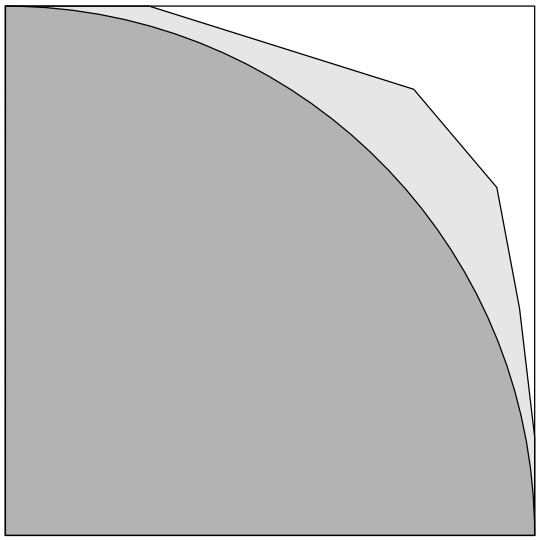


Figure 5.2: Calcul de $\pi/4$ par une méthode de Monte-Carlo approximé.

5.1.2 Méthode de Monte-Carlo abstrait

Nous considérons maintenant un programme dont les entrées peuvent être séparées en deux groupes, représentées chacun par une seule variable : la variable x est prise dans l'ensemble X selon une mesure de probabilité μ, la variable y est dans un ensemble Y. Nous cherchons à majorer la probabilité

$$\mu\{x \in X \mid \exists y \in Y \ [\![c]\!]\langle x,y\rangle \in W\},$$

où $[\![c]\!]$ est la sémantique dénotationnelle du programme c. Notant

$$t_W(x) = \begin{cases} 1 & \text{si } \exists y \in Y \ [\![c]\!]\langle x,y\rangle \in W \\ 0 & \text{sinon,} \end{cases}$$

, la probabilité recherchée est l'espérance de t_W.

Malheureusement, la fonction t_W n'est pas (en général) calculable au sens de Turing, et nous ne pouvons donc pas appliquer directement la méthode de Monte-Carlo.

Imaginons maintenant que nous disposons d'une approximation supérieure et calculable T_W de t_w: pour tout x, $t_W(x) \leq T_W(x)$. Nous pouvons alors nous en servir pour obtenir une approximation supérieure de $\mathbf{E}t_w$ par une méthode de Monte-Carlo. Pour reprendre notre exemple sur $\pi/4$, cela revient par exemple à approcher le disque par un polygone et à faire les tests sur le polygone (Fig. 5.2).

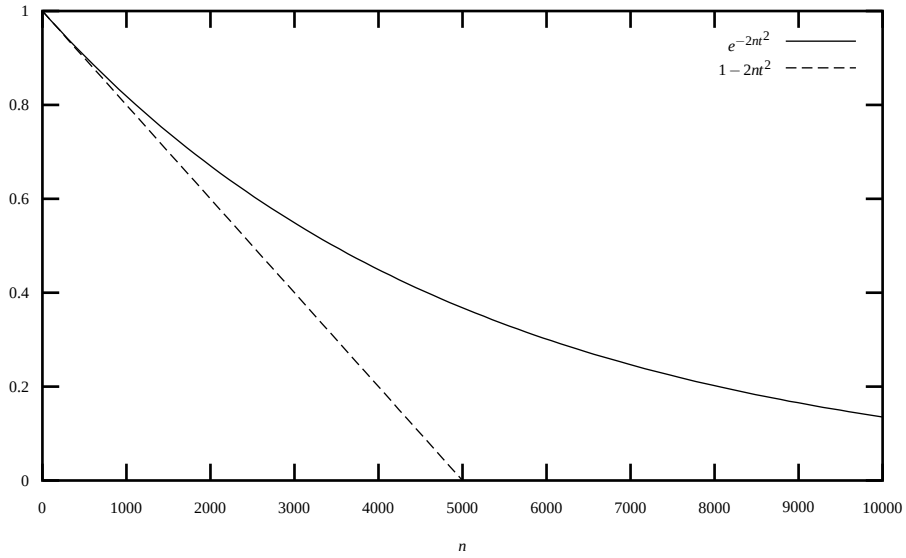


Figure 5.3: Majorant de la probabilité que l'estimation de probabilité calculée excède la vraie valeur de plus de t, pour $t = 0.01$.

Puisque $t_W^{(c)} \leq T_W^{(n)}$, la borne de Chernoff donnée précédemment reste valide en remplaçant t_W par T_W. En particulier, il y a une probabilité d'au moins $1-\varepsilon$ que la vraie espérance $\mathbf{E}t_W$ soit inférieure à $p' = T_W^{(n)} + \sqrt{\frac{-\log \varepsilon}{2n}}$.

5.2 Implantation

Nous randomisons un interpréteur abstrait existant en interprétant l'instruction de choix d'une donnée aléatoire dans le programme analysé comme le choix d'une constante aléatoire dans l'analyseur. Il faut donc bien prendre garde, en cas d'analyse à passe multiples par exemple, de mémoriser les valeurs sur lesquelles on s'est décidé.

Dans le cas de boucles, nous ne randomisons que les premières itérations, et considérons les choix aléatoires ultérieurs comme purement non-déterministes.

Part II

Corps de la thèse (en anglais)

Chapter 6

Introduction

6.1 Probabilistic and nondeterministic programs

Let us begin with a few informal definitions.

- A *deterministic program* is a program whose current state uniquely determines the next state.
- A *nondeterministic* program is a program whose current state uniquely determines a set of possible next states. Of course, computers are basically deterministic machines. But, usually, programs are run in a software and hardware environment that is not totally known at the time of conception due to various factors:
  - input files;
  - user interactions;
  - external hardware for embedded systems;
  - scheduling by the operating system (a same multi-process program may see its processes scheduled differently depending on many factors).

  All these unknown factors can be accounted for by assuming nondeterministic behavior. For instance, the operating system is supposed to choose nondeterministically the process to run at each step. Of course, this means that our model supposes that the system can do more than it can actually do; in this sense, it is a *safe* approach, since any property that has been proved on this pessimistic abstraction of the reality will hold on the real system.

- A *probabilistic* program is a program whose current state uniquely determines a probability distribution on the possible next states. Such probabilistic choices may model the use of random number generators as well as inputs from a stochastic environment. A natural model for such programs is the *Markov chain*.

- A *nondeterministic* and *probabilistic* program is a program whose current state uniquely determines a set of possible probability distributions on the next state. This model thus encompasses both the nondeterministic and probabilistic models. The nondeterminism may be used to model partially unknown probability distributions. A natural model for such programs is the *Markov decision process*.

In chapter 7, we shall describe formally the model of Markov decision processes and the subtle interactions between probabilities and nondeterminism.

6.2 Semantics and abstract semantics

6.2.1 Semantics

Our goal is to develop ways to analyze automatically probabilistic and nondeterministic programs and give safe results about their behaviors. Since our goal is to provide mathematically sound facts about the possible executions of the studied programs, we must associate a mathematical object, called its *semantics* [80, 53] to each program.

Even considering fully deterministic programs, there are several ways to modelize their behavior, depending on the properties that are to be studied. Mathematical relationships of *abstraction* exist between those various semantics [13]. We shall be more particularly interested in:

Small-step operational semantics define the behavior of the systems as transitions from a state to another state. The simplest such semantics is to consider a transition system — a directed graph whose vertices represent the possible global states of the process or set of processes (memory, program counters) and edges represent the possible transitions. In chapter 7, we shall see how to extend this approach to probabilistic and nondeterministic programs, using discrete-time Markov decision processes.

Denotational semantics define the behavior of a sequential program as a function mapping its input to its output. Such semantics are generally defined to be *compositional*, that is, are defined by induction on the syntactic structure of the program when written in a structured programming language such as

Pascal. In chapter 13, we shall see how to derive a denotational semantics for probabilistic programs and how to apply abstract interpretation to it.

There are many issues pertaining to the semantics of Markov decision processes. We shall see in chapter 7 how to specify and analyze *trace properties* of those systems.

There have been many proposal for semantics for probabilistic programs [41, 42, 58] or, even more complex, probabilistic and nondeterministic programs [35, 31, 36, 49, 51]. We shall note here that the goal of this thesis is to provide *analysis methods* for infinite-state probabilistic (and nondeterministic) programs, not to provide domain-theoretic semantics. We hope that abstraction relation such as the ones presented in [13] will be provided between some of the proposed semantics.

6.2.2 Limitations of total static analysis

We shall not give excessively formal definitions of the theory of recursive functions and computability [68]; nevertheless, we shall recall a few facts from it.

A well-known fact about computer programs is that the *halting problem* is undecidable. That is, there exists no algorithm that, given the source code of a program written in a general-purpose programming language, says whether this program is bound to terminate always or it can enter an infinite loop. More generally, the following theorem [68, §2.1], which apply to any "Turing-complete" programming languages, assuming unbounded memory, says in effects that "any non-trivial property of the denotational semantics of programs is undecidable".

Theorem 6.2.1 (Rice). *Let $\Phi(x)$ be the recursive function associated with program x. For any set of recursive functions C, $\Phi^{-1}(C)$ is recursive if and only if C is either the empty set either the set of all recursive functions.*

A *recursive* set is a set whose characteristic function is recursive. A recursive function is a function that can be computed by an always terminating program.

This theorem means in effect that any "infinite horizon" properties on programs running on unbounded state machines cannot be decided by other programs, even with unbounded memory. It could be argued that real-life computer systems have finite memory; nevertheless, solutions using that memory finiteness in the studied system are likely to need exponentially more memory for the analyzer. The state of the art in the techniques of *model checking* such finite systems is about 10^20 to 10^120 states (representing the configurations of about 60 to 360 *bits* of memory) [10, §1.5] memory sizes of hundreds of megabytes, as it is common on today's personal computers, are of course of of reach. It seems therefore impossible to simulate most programs exactly.

A solution to this problem is to *approximate* the properties that we want to analyze. Instead of requiring a "yes or no" answer to questions such as "can the program reach that error state", we shall restrict ourselves to answers like "yes, no or maybe". [1] In building such analyzers, we shall have two goals:

- The analysis should be *safe*; that is, the analysis should not answer that a program cannot reach a certain state whereas it can actually do so. Since this is a formal goal, we shall establish this safety by theorems linking the results of the analysis to properties of the semantics of the programs.
- The analysis should, as much as possible, give interesting results. It is of course possible to make a safe analyzer that answers "maybe" to every question; but it would have no interest to users. We therefore have the heuristic goal that the analyzer provide reasonably precise answers on reasonable programs and properties.

Applications of such analyses exceed the verification of programs. Compiling optimizations often rely on some property of execution of the program (for instance, that two pointers can never point to the same location) that cannot be inferred immediately from its syntax. If the approximate analysis establishes the desired property, the optimization may be applied; otherwise, it is not applied. The taken action is safe in all cases.

6.2.3 Abstract interpretation

Patrick and Radhia Cousot proposed *abstract interpretation* as a way to provide safe and efficient analyses [19, 18, 16, 20, 15, 14]:

> Abstract interpretation is a theory of semantics approximation which is used for the construction of semantics-based program analysis algorithms (sometimes called data-flow analysis), the comparison of formal semantics (e.g.,construction of a denotational semantics from an operational one), the design of proof methods,etc. Automatic program analysers are used for determining statically conservative approximations of dynamic properties of programs.Such properties of the run-time behavior of programs are useful for debugging (e.g., type inference), code optimization (e.g., compile-time garbage collection, useless occur-check elimination), program transformation (e.g.,partial evaluation, parallelization), and even program correctness proofs (e.g., termination proof).

[1] The Company Polyspace Technologies markets analyzers which color the source code in green, red and orange depending on whether the analyzer establishes that that portion of code will, will not or maybe will elicit errors.

The basic idea of abstract interpretation is that of *approximation* in a known direction (greater or lower than the real property). With a view to program analysis, those approximations are chosen so that they can be represented symbolically by efficient data structures. Let us take a simple example: *interval analysis*. An elementary way to check whether a program may try to access arrays using out-of-bound indices is to associate to each integer variable at each program point an interval in which it is known to stay. If that interval for an array accessing instruction is a subset of the permitted range, then that access is safe. Since such an analysis does not allow array sizes that are unknown at the time of analysis of the program (which is the case for most real-life programs), more complex analyses have been introduced such as *convex polyhedra* [20] and other domains [52]. Outside of array bound checking, abstract interpretation has been for instance applied to variable aliasing [25, 24] and other aspects of programs.

6.3 Mathematical prerequisites

6.3.1 Notations

- $A \to B$ is the set of mappings from A to B (also noted B^A);
- $\mathcal{P}(X)$ is the sets of subsets of X (sometimes called its *power-set*).;
- $\chi_B : A \to \{0,1\}$ is the characteristic function of subset B of A.
- δ_x is the Dirac measure at x (§6.3.3).

6.3.2 Ordered sets and lattices

Definition 6.3.1. Let T_1 and T_2 be two ordered sets. A function $f : T_1 \to T_2$ is said to be *monotone* if for all $x, y \in T_1$, $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$. The set of monotone functions (or *monotone operators*) from T_1 to T_2 will be noted $T_1 \xrightarrow{\text{mon}} T_2$.

6.3.3 Measures and integrals

We shall express probabilities using *measures* [71, §1.18]. Measures express the intuitive idea of a "repartition of weight" on the domain; probabilities are a particular case of measures. A measure on a space Ω assigns a "weight" to subsets of Ω. A probability measure is a measure of total weight 1.

Before entering the mathematical definitions, let us see a few familiar examples:

- In the case where Ω is finite or countable, defining a positive measure is just defining a function $f : \Omega \to \mathbb{R}_+$ (the weight of $A \subseteq \Omega$ is then $\sum_{\omega \in A} f(a)$); it is a a probability measure if $\sum_{\omega \in \Omega} f(\omega) = 1$. A measure on a finite space is said to be equidistributed if $\forall \omega \ f(\omega) = \frac{1}{|\Omega|}$.

- In the case of the real field $\mathbb{R}$, the Lebesgue measure μ is so that the measure of a segment $[a,b]$ is its length. The Lebesgue measure on $[0,1]$ formalizes that familiar notion of a real "equidistributed in $[0,1]$". The Lebesgue measure can be defined on $\mathbb{R}^n$, and the measure of an object is its area (for $\mathbb{R}^2$) or volume (for $\mathbb{R}^3$).

- The *unit point mass* (or *Dirac measure*) at x is the measure δ_x defined by: $\delta_x(A) = 1$ if $x \in A$, $\delta_x(A) = 0$ otherwise.

In the general case, for technical reasons, not all sets can be measured and we have to restrict ourselves to some sufficiently large set of measurable subsets of Ω. While in general in computer science the state spaces to be considered are finite or countable, at some point we shall have to consider sets of infinite traces and compute integrals on them (see Ch. 8). We therefore shall do our work in the general theory of Lebesgue integrals.

Let us see now the formal definitions:

Definition 6.3.2. A *σ-algebra* is a set of subsets of a set X that contains $\emptyset$ and is stable by countable union and complementation (and thus contains X and is stable by countable intersection).

In the case of the Lebesgue measure, we shall consider a suitable σ-algebra, such as the Borel or Lebesgue ones [71]. It is sufficient to say that most sets that one can construct are Lebesgue-measurable.

Definition 6.3.3. A set X with a σ-algebra σ_X defined on it is called a *measurable space* and the elements of the σ-algebra are the *measurable subsets*.

We shall often mention measurable spaces by their name, omitting the σ-algebra, if no confusion is possible.

Definition 6.3.4. If X and Y are measurable spaces, $f : X \to Y$ is a *measurable function* if for all W measurable in Y, $f^{-1}(W)$ is measurable in X.

Definition 6.3.5. A *positive measure* is a function μ defined on a σ-algebra σ_X whose range is in $[0,\infty]$ and which is countably additive. μ is countably additive if, taking $(A_n)_{n\in\mathbb{N}}$ a disjoint collection of elements of σ_X, then

$$\mu\left(\bigcup_{n=0}^{\infty} A_n\right) = \sum_{n=0}^{\infty} \mu(A_n). \tag{6.1}$$

To avoid trivialities, we assume $\mu(A) < \infty$ for at least one A. The *total weight* of a measure μ, noted $|\mu|$, is $\mu(X)$. μ is said to be *concentrated* on $A \subseteq X$ if for all B, $\mu(B) = \mu(B \cap A)$. We shall note $\mathcal{M}_+(X)$ the positive measures on X.

Definition 6.3.6. A *σ-finite measure* on X is a measure μ such that there exists a countable family of measurable sets $(A_n)_{n\in\mathbb{N}}$ such that $\forall n\ \mu(A_n) < \infty$ and $\bigcup_n A_n = X$. We note by $\mathcal{M}_+(X)$ the σ-finite measures on X.

Definition 6.3.7. A *probability measure* is a positive measure of total weight 1; a *sub-probability measure* has total weight less or equal to 1. We shall note $\mathcal{M}_{\leq 1}(X)$ the sub-probability measures on X.

Definition 6.3.8. Given two sub-probability measures μ and μ' (or more generally, two σ-finite measures) on X and X' respectively, we note $\mu \otimes \mu'$ the product measure [71, definition 7.7], defined on the product σ-algebra $\sigma_X \times \sigma_{X'}$. The characterizing property of this product measure is that $\mu \otimes \mu'(A \times A') = \mu(A).\mu'(A')$ for all measurable sets A and A'.

These definitions constitute the basis of the theory of Lebesgue integration [71, ch. 1, 2].

Definition 6.3.9. Given a positive measure μ on Ω and a measurable function $f : \Omega \to \mathbb{R}_+$, their product is the measure $f.\mu$ defined as:

$$f.\mu(A) = \int_A f(x)\ \mathrm{d}\mu(x) \tag{6.2}$$

Let us note that we shall not make use of "distributions" in the sense of L. Schwartz's general theory of distributions. A *probability distribution* will then just be a probability measure. Technical theorems on integrals and measures are given in Appendix A.1.

Chapter 7

Transition systems: semantics

In this chapter, we explain gradually our notion of nondeterministic probabilistic transition systems. We shall start from the simplest cases and go to the most complex ones. Most notably, we shall introduce the *trace semantics* of such systems.

7.1 Transition systems

Transition systems formalize the intuitive notion of a system whose state changes with time, assuming a discrete notion of time. We shall begin by the simplest kind, not taking probabilities into account.

Definition 7.1.1. Let Ω be a set of states. Let $\rightarrow$ be a binary relation over Ω. $(\Omega, \rightarrow)$ constitutes a *(nondeterministic) transition system*. $\rightarrow$ is called the *transition relation*.

The *successors* states of a state ω_1 are the states ω_2 such that $\omega_1 \rightarrow \omega_2$; the *predecessors* states of a state ω_2 are the states ω_1 such that $\omega_1 \rightarrow \omega_2$.

If $\omega_0 \in \Omega$ notes an *initial state*, then the *accessible states* are the states $\omega \in \Omega$ such that $\omega_0 \rightarrow^* \omega$ where $\rightarrow^*$ notes the transitive-reflexive closure of $\rightarrow$.

If Ω is finite, the relation relation can be given by a *transition matrix*. Let us assimilate Ω to $\{1, \ldots, N\}$. Then the transition matrix M is defined by $m_{i,j} = 1$ if $i \rightarrow j$, 0 otherwise.

Definition 7.1.2. If for any state $\omega \in \Omega$, then the set of successor states of ω has at exactly one element, we say that the transition system is *deterministic*.

Figure 7.1 represents the transition system associated with the following program:

```
x = ndt_coin_flip();
x = x XOR ndt_coin_flip();
```

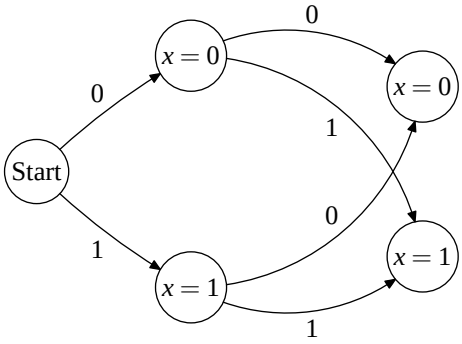


Figure 7.1: A deterministic transition system

where `XOR` is "exclusive or" and `ndt_coin_flip()` nondeterministically flips a coin between 0 and 1.

Now that we have the rules that say whether an execution may proceed from a state to another state, we can define whole properties on executions.

Definition 7.1.3. A *trace of execution* (or simply *trace*) is a sequence $(t_k)_{k\in\mathbb{N}} \in \Omega^{\mathbb{N}}$ of states.

The reader should note that our definition of *trace* does not say whether the actual trace is possible. This is taken care of by the following definition:

Definition 7.1.4. A trace of execution $(t_k)_{k\in\mathbb{N}} \in \Omega^{\mathbb{N}}$ is called *possible* if for all k, $t_k \to t_{k+1}$.

7.2 Probabilistic transition systems

The natural extension of transition systems to the probabilistic case is *probabilistic transition systems*, also known as *Markov chains* or *discrete-time Markov process*.

7.2.1 Discrete case

In this section, we assume that the set of states is *finite* or *countable* so as to introduce the elementary notions of probability transitions while not drowning the reader into the subtleties of measure theory.

The natural extension of the notion of deterministic state is the notion of probability distribution on the set of states, which introduces the probabilistic indeterminacy.

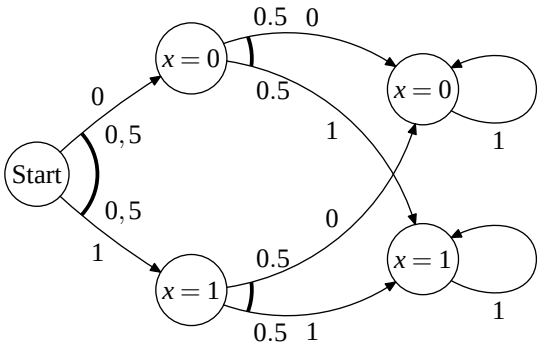


Figure 7.2: A probabilistic transition system

Definition 7.2.1. Let Ω be a finite or countable set of states. A function $f : \Omega \to [0,1]$ is called a *probability distribution* if $\sum_{\omega\in\Omega} f(\omega) = 1$. We shall note $D(\Omega)$ the set of probabilistic distributions on Ω.

Now that we have the probabilistic counterpart of the notion of state, we need to have the counterpart of the notion of transition.

Definition 7.2.2. Let Ω be a finite or countable set of states. Let us consider a function $T : \Omega \times \Omega \to [0,1]$ such that for all $\omega_1 \in \Omega$, $\sum_{\omega_2\in\Omega} T(\omega_1;\omega_2) = 1$. (Ω, T) is called a *probabilistic transition system* (or *Markov chain*) (Fig. 7.2).

Figure 7.2 represents the transition system associated with the following program:

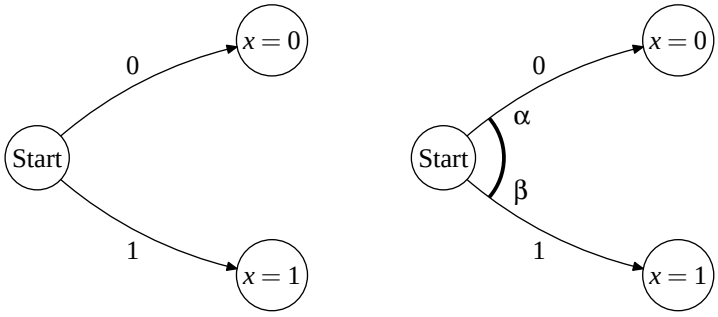
```
x = coin_flip();
x = x XOR coin_flip();
```

where `XOR` is "exclusive or" and `coin_flip()` probabilistically flips a coin between 0 and 1 with probability 0.5 for each possibility.

If Ω is finite, the relation can be given by a *probabilistic transition matrix*. Let us assimilate Ω to $\{1,\ldots,N\}$. Then the transition matrix M is defined by $m_{i,j} = T(i,j)$ if $i \to j$, 0 otherwise.

The intuitive notion of a probabilistic transition is that it maps an *input distribution* to an *output distribution*. This corresponds to the intuitive notion of a successor state.

Definition 7.2.3. Let T be a transition probability between Ω_1 and Ω_2. Let us define $\overrightarrow{T} : D(\Omega_1) \to D(\Omega_2)$ as follows: $\overrightarrow{T}(d)(\omega_2) = \sum_{\omega_1\in\Omega_1} T(\omega_1,\omega_2)d(\omega_1)$.



(a) Nondeterministic

(b) Probabilistic (probabilities α and β)

Figure 7.3: Nondeterminism vs probabilities in graphical notation. Edges are labeled by the value chosen in the probabilistic or nondeterministic choice.

Proof. We need that check that $\overrightarrow{T}(d)$ is indeed a probability distribution on Ω_2; that is, that $\sum_{\omega_2\in\Omega_2}\sum_{\omega_1\in\Omega_1} T(\omega_1,\omega_2)d(\omega_1) = 1$. Since all terms are positive $\sum_{\omega_2\in\Omega_2}\sum_{\omega_1\in\Omega_1} T(\omega_1,\omega_2)d(\omega_1) = \sum_{\omega_1\in\Omega_1} d(\omega_1) \underbrace{\sum_{\omega_2\in\Omega_2} T(\omega_1,\omega_2)}_{1} = 1.$ □

Deterministic and nondeterministic transition systems have a notion of predecessor state. In order to find the probabilistic counterpart to this notion, we must think a bit more about its meaning. When we take the set of predecessors of a set of states A, we think of A as a *condition* and we obtain the set of states that may lead to that condition. As for probabilistic state spaces, we define the probability that condition A holds with respect to distribution d by $\sum_{\omega\in A} d(\omega)$. Equivalently, we can write it $\langle \chi_A, d\rangle$ where χ_A is the characteristic function of A and

$$\langle f, d\rangle = \sum_{\omega\in\Omega} f(\omega)d(\omega). \tag{7.1}$$

A natural extension of this notion of condition is to consider any function $f \in P(\Omega)$ where $P(\Omega) = \Omega \to [0,1]$. We call such functions *condition functions* or *potentiality functions*. Please note that while those functions look similar to distributions, they are quite different in their meaning and will be different mathematical objects when treated in the general, non discrete, case.

Definition 7.2.4. Let T be a transition probability between Ω_1 and Ω_2. Let us define $\overleftarrow{T} : P(\Omega_2) \to P(\Omega_1)$ as follows: $\overleftarrow{T}(f)(\omega_1) = \sum_{\omega_2\in\Omega_2} T(\omega_1,\omega_2) f(\omega_2)$.

Proof. We have to check that $\overleftarrow{T}(f) \in P(\Omega)$. $\sum_{\omega_2 \in \Omega_2} T(\omega_1, \omega_2) \underbrace{f(\omega_2)}_{\leq 1} \leq 1$. □

7.2.2 General case

The extension to non-countable systems is a bit complex. Most notably, one must substitute *integrals* (in the sense of Lebesgue integrals[1] [71]) for sums.

Transition probabilities

The right way to consider probabilistic transitions on possibly uncountable states spaces is to use a *transition probability* [60, 59, definition III.2.1]. This definition generalizes the intuitive notion of a probabilistic transition being a function mapping each state of the first state space to a probability distribution on the second state space.

Definition 7.2.5. Let $(\Omega_1, \mathscr{A}_1)$ and $(\Omega_2, \mathscr{A}_2)$ be two measurable spaces. A *transition probability* P_2^1 is an application from $\Omega_1 \times \mathscr{A}_2$ to $[0,1]$ such that:

1. for all $\omega_1 \in \Omega_1$, $P_2^1(\omega_1, \cdot)$ is a probability on $(\Omega_2, \mathscr{A}_2)$;
2. for all $A_2 \in \mathscr{A}_2$, $P_2^1(\cdot, A_2)$ is a measurable function on $(\Omega_1, \mathscr{A}_1)$.

A probability P_1 on a state space Ω_1 and transition probability P_2^1 between Ω_1 and Ω_2 induce a probability distribution on their product $\Omega_1 \times \Omega_2$ [60, 59, proposition III.2.1]. Intuitively, this product probability describes the probability for a succession of two states $(\omega_1, \omega_2) \in \Omega_1 \times \Omega_2$ such that ω_1 follows P_1 and ω_2 follows P_2^1 relatively to ω_1.

Proposition 7.2.6. *Let $(\Omega_1, \mathscr{A}_1)$ and $(\Omega_2, \mathscr{A}_2)$ two measurable spaces. Let P_1 be a probability on $(\Omega_1, \mathscr{A}_1)$ and P_2^1 a transition probability on $\Omega_1 \times \mathscr{A}_2$. There exists then one unique probability P on $(\Omega_1 \times \Omega_2, \mathscr{A}_1 \otimes \mathscr{A}_2)$ such that*

$$P(A_1 \times A_2) = \int_{A_1} P_2^1(\omega_1, A_2) P_1(\mathrm{d}\omega_1) \tag{7.2}$$

for any $A_1 \in \mathscr{A}_1$ and $A_2 \in \mathscr{A}_2$.

[1] Some people have argued that the full machinery of Lebesgue integrals is overkill. It is indeed quite possible to build up the beginning of the theory of probabilistic transition systems on finite or countable state spaces only using finite or infinite positive sums. However, definition of the properties of *infinite* sequence of events is difficult in any case.

For any real random variable X defined on $(\Omega_1 \times \Omega_2, \mathscr{A}_1 \otimes \mathscr{A}_2, P)$ and positive (resp. quasi-integrable), the function

$$Y(\omega_1) = \int_{\Omega_1} X_{\omega_1}(\omega_2) P_2^1(\omega_1, \mathrm{d}\omega_2) \tag{7.3}$$

is defined everywhere (resp. P_1-almost everywhere), is $\mathscr{A}_1$-measurable on Ω_1, is positive (resp. quasi-integrable with respect to P_1). Furthermore,

$$\int_{\Omega_1 \times \Omega_2} X \, \mathrm{d}P = \int_{\Omega_1} P_1(\mathrm{d}\omega_1) \int_{\Omega_2} X_{\omega_1}(\omega_2) P_2^1(\omega, \mathrm{d}\omega_2) \tag{7.4}$$

Using this proposition, we can build a notion of *composition* of transition probabilities.

Definition 7.2.7. Let $(\Omega_1, \mathscr{A}_1)$, $(\Omega_2, \mathscr{A}_2)$ and $(\Omega_3, \mathscr{A}_3)$ be three measurable spaces, P_2^1 a transition probability on $\Omega_1 \times \mathscr{A}_2$ and P_3^2 one on $\Omega_2 \times \mathscr{A}_3$. Let us define

$$P_3^1(\omega_1, A_3) = \int_{\Omega_2} P_3^2(\omega_2, A_3) P_2^1(\omega_1, \mathrm{d}\omega_2) = \langle P_3^2(\cdot, A_3) P_2^1(\omega_1, \cdot) \rangle; \tag{7.5}$$

the *composition* of both transition probabilities. Then P_3^1 is a transition probability on $\Omega_1 \times \mathscr{A}_3$.

Proof. Let $\omega_1 \in \Omega_1$. Let us consider a family A_n of disjoint elements of $\mathscr{A}_3$.

$$\begin{aligned} P_3^1\left(\omega_1, \bigsqcup_n A_n\right) &= \int_{\Omega_2} \underbrace{P_3^2\left(\omega_2, \bigsqcup_n A_n\right)}_{\sum_n P_3^2(\omega_2, A_n)} . P_2^1(\omega_1, \mathrm{d}\omega_2) \\ &= \sum_n \int_{\Omega_2} P_3^2(\omega_2, A_n) . P_2^1(\omega_1, \mathrm{d}\omega_2) \\ &= \sum_n P_3^1(\omega_1, A_n) \end{aligned} \tag{7.6}$$

$P_3^1(\omega_1, \cdot)$ is thus a positive measure.

$$P_3^1(\omega_1, \Omega_3) = \int_{\Omega_2} \underbrace{P_3^2(\omega_2, \Omega_3)}_{1} P_2^1(\omega_1, \mathrm{d}\omega_2) = 1 \tag{7.7}$$

thus $P_3^1(\omega_1,\cdot)$ is a probabibility.

Let $A_2 \in \mathscr{A}_2$. Let us define $X(\omega_1,\omega_2) = P_3^2(\omega_2,A_3)$. X is a positive random variable on $(\Omega_1\times\Omega_2, \mathscr{A}_1\otimes\mathscr{A}_2, P)$. Applying Prop. 7.2.6 to X yields that $Y = P_3^1(\cdot,A_3)$ is defined everywhere and is measurable. □

We similarly define a notion of *product* of two transition probabilities.

Definition 7.2.8. Let $(\Omega_1,\mathscr{A}_1)$, $(\Omega_2,\mathscr{A}_2)$ and $(\Omega_3,\mathscr{A}_3)$ be three measurable spaces; let P_2^1 be a transition probability between Ω_1 and Ω_2 and P_3^2 a transition probability between Ω_2 and Ω_3. Let us take $\omega_1\in\Omega_1$ and $A\in\mathscr{A}_2\otimes\mathscr{A}_3$; then let us define

$$P_2^1\otimes P_3^2(\omega_1,A) = \int_{\Omega_2}\left(\int_{\Omega_3}\chi_A(\omega_2,\omega_3)P_3^2(\omega_2,\mathrm{d}\omega_3)\right)P_2^1(\omega_1,\mathrm{d}(\omega_2)). \tag{7.8}$$

$P_2^1\otimes P_3^2$ is then a transition probability between Ω_1 and $\Omega_2\times\Omega_3$.

Proof. Let us take $A\in\mathscr{A}_2\otimes\mathscr{A}_3$. Using Prop. 7.2.6 on the random variable $X=\chi_A$, the function $Y:\omega_2\mapsto\int_{\Omega_3}\chi_A(\omega_2,\omega_3)P_3^2(\omega_2,\mathrm{d}\omega_3)$ is a positive random variable on Ω_2, defined everywhere. Using Prop. 7.2.6 on the random variable $X':(\omega_1,\omega_2)\mapsto Y(\omega_2)$, the function $Y':\omega_1\mapsto\int_{\Omega_2}X'(\omega_1,\omega_2)P_3^2(\omega_2,\mathrm{d}\omega_3) = P_2^1\otimes P_3^2(\omega_1)$ is a defined everywhere on Ω_1 and measurable.

Let $\omega_1\in\Omega_1$. Let us consider a family A_n of disjoint elements of $\mathscr{A}_2\otimes\mathscr{A}_3$. Then

$$\begin{aligned}P_2^1\otimes P_3^2\left(\omega_1,\bigcup_n A_n\right) &= \int_{\Omega_2}\left(\int_{\Omega_3}\chi_A(\omega_2,\omega_3)P_3^2(\omega_2,\mathrm{d}\omega_3)\right)P_2^1(\omega_1,\mathrm{d}(\omega_2))\\ &= \int_{\Omega_2}\left(\int_{\Omega_3}\left(\sum_n\chi_{A_n}\right)(\omega_2,\omega_3)P_3^2(\omega_2,\mathrm{d}\omega_3)\right)P_2^1(\omega_1,\mathrm{d}(\omega_2)) = \sum_n P_2^1\otimes P_3^2(A_n)\end{aligned} \tag{7.9}$$

and thus $A\mapsto P_2^1\otimes P_3^2(\omega_1,A)$ is a measure. □

Definition 7.2.9. Let $(E_k,\mathscr{F}_k)_{1\le k\le n}$ be a finite sequence of measurable spaces and, for any $k\in\mathbb{N}$, let T_k be a transition probability between T_{k-1} and T_k. $\bigotimes_{k=1}^n T_k$ will note the transition probability $T_0\otimes(T_1\otimes(\cdots T_n)\cdots)$ between $(E_0,\mathscr{F}_0)$ and $\left(\prod_{k=1}^n E_k,\bigotimes_{k=1}^n\mathscr{F}_k\right)$.

The probabilistic analogue of a discrete potentiality function, which we shall also call potentiality function, will be a measurable function. Let us now consider the probabilistic counterparts of definitions 7.2.3 and 7.2.1.

Definition 7.2.10. Let $(\Omega_1, \mathscr{A}_1)$ and $(\Omega_2, \mathscr{A}_2)$ be two measurable spaces. Let us define a linear operator $\overrightarrow{P_2^1}$ from $\mathcal{M}_+(\Omega_1)$ to $\mathcal{M}_+(\Omega_2)$ as follows:

$$(\overrightarrow{P_2^1}.\mu)(A_2) = \int_{\Omega_1} P_2^1(\omega_1, A_2)\mu(\mathrm{d}\omega_1). \tag{7.10}$$

Let us also define a linear operator $\overleftarrow{P_2^1}$ from $M(\Omega_2, \mathbb{R}_+)$ to $M(\Omega_1, \mathbb{R}_+)$:

$$(\overleftarrow{P_2^1}.f)(\omega_1) = \int_{\Omega_2} f(\omega_2) P_2^1(\omega_1, \mathrm{d}\omega_2). \tag{7.11}$$

Those functions are linked by the following *adjunction relation*:

Proposition 7.2.11. *Let $(\Omega_1, \mathscr{A}_1)$ and $(\Omega_2, \mathscr{A}_2)$ be two measurable spaces. Let μ be a measure on Ω_1 and f be a measurable function on Ω_2. Then*

$$\int f \, \mathrm{d}(\overrightarrow{P_2^1}.\mu) = \int (\overleftarrow{P_2^1}.f) \, \mathrm{d}\mu \tag{7.12}$$

Proof. Let us apply Prop. 7.2.6 to the probability μ on Ω_1 and the transition probability P_2^1. Equation 7.3 applied to the random variable $X(\omega_1, \omega_2) = f(\omega_2)$ yields

$$\begin{aligned} &\int_{\Omega_1} (\overleftarrow{P_2^1}.f) \, \mathrm{d}\mu \\ &\qquad = \int_{\Omega_1} \mu(\mathrm{d}\omega_1) \int_{\Omega_2} P_2^1(\omega_1, \mathrm{d}\omega_2) \, f(\omega_2) \\ &\qquad\qquad = \int_{\Omega_1 \times \Omega_2} d(\omega_2) \, \mathrm{d}P \\ &\qquad\qquad\qquad = \int_{\Omega_2} f(\omega_2) \underbrace{P(\Omega_1 \times \mathrm{d}\omega_2)} \end{aligned} \tag{7.13}$$

where $P(A_1 \times A_2) = \int_{A_1} P_2^1(\omega_1, A_2)\mu(\mathrm{d}\omega_1)$. Replacing P into this equation yields the result. □

Probability measure on traces

One essential fact on the probability of traces is that all trace may have each zero probability; this is not contradictory with the fact that "the probability of a set is the sum of all probabilities of the elements in that set" since that rule applies *per se* for at most countable sets, while there are uncountably infinitely many traces (if the state space contains more than one state, of course). One must consider therefore *sets of traces* to have meaningful probabilities. Let us give a simple example. Let us consider sequences of tosses of a fair coin: the coin has probability

0.5 of giving 0 and 0.5 of giving 1. A trace is then an infinite sequence of zeroes and ones. Let us consider the (regular) set $0^n(0|1)^\omega$ of sequences starting by at least n zeroes. It is obvious that the probability of falling into that set is 2^{-n}. The probability of the singleton containing the sequence of zeroes only is 0.

The following theorem [60, 59, proposition V-I-1] is crucial in defining the probability on traces.

Theorem 7.2.12 (Ionescu Tulcea). *Let $(E_t, \mathscr{F}_t)_{t\in\mathbb{N}}$ be an infinite sequence of measurable spaces and, for any $t \in \mathbb{N}$, let $P_{t+1}^{0,\dots,t}$ be a transition probability relative to the spaces $\left(\prod_{s=0}^t E_s, \bigotimes_{s=0}^t \mathscr{F}_s\right)$ et $(E_{t+1}, \mathscr{F}_{t+1})$. Then there exists for any $x_0 \in E_0$ a unique probability P_{x_0} on*

$$(\Omega, \mathscr{A}) = \prod_t (E_t, \mathscr{F}_t)$$

whose value for all measurable cartesian product $\prod_t F_t$ is given by:

$$P_{x_0}\left[\prod_t F_t\right] = \chi_{A_0}(x_0) \int_{x_1\in F_1} P_1^0(x_0; \mathrm{d}x_1) \int_{x_2\in F_2} P_2^{0,1}(x_0, x_1; \mathrm{d}x_2) \cdots \int_{x_T\in F_T} P_T^{0,\dots,T-1}(x_0, \dots, x_{T-1}; \mathrm{d}x_T) \quad (7.14)$$

as long as T is sufficiently great such that $F_t = E_t$ if $t > T$ (the second member is then independent of the chosen T). For any positive random variable Y on $(\Omega, \mathscr{A})$ only depending on the coordinates up to T, we have:

$$\int_\Omega Y(\omega') P_{x_0}(\mathrm{d}\omega') = \int_{F_1} P_1^0(x_0; \mathrm{d}x_1) \int_{F_2} P_2^{0,1}(x_0, x_1; \mathrm{d}x_2) \cdot \int_{x_T\in F_T} Y(x_0, \dots, x_T) P_T^{0,\dots,T-1}(x_0, \dots, x_{T-1}; \mathrm{d}x_T) \quad (7.15)$$

Furthermore, for any positive random variable Y on $(\Omega, \mathscr{A})$,

$$x_0 \mapsto \int Y(\omega') P_{x_0}(\mathrm{d}\omega') \quad (7.16)$$

is a positive random variable on $(E_0, \mathscr{F}_0)$.

This theorem induces a notion of *infinite* product of transition probabilities.

Definition 7.2.13. Let $(E_t, \mathscr{F}_t)_{t\in\mathbb{N}}$ be an infinite sequence of measurable spaces and, for any $t \in \mathbb{N}$, let T_n be a transition probability between $\mathscr{F}_{n-1}$ and $\mathscr{F}_n$. We note $\bigotimes_{k=1}^\infty T_k$ the transition probability between $(E_0, \mathscr{F}_0)$ and $(\prod_{k=1}^\infty E_k, \bigotimes_{k=1}^\infty \mathscr{F}_k)$ defined as $\left(\bigotimes_{k=1}^\infty T_k\right)(x_0, A) = P_{x_0}(A)$, applying the theorem 7.2.12 to $(P_{t+1}^{0,\dots,t})_{t\in\mathbb{N}}$ defined by $P_{t+1}^{0,\dots,t}(x_0, \dots, x_t; A) = T_t(x_t, A)$.

Proof. From theorem 7.2.12, for any x_0, P_{x_0} is a measure, and for any A, $x_0 \mapsto \int \chi_A P_{x_0}$ is measurable. □

Lemma 7.2.14. *Let $(E_t, \mathscr{F}_t)_{t\in\mathbb{N}}$ be an infinite sequence of measurable spaces and, for any $t \in \mathbb{N}$, let T_n be a transition probability between $\mathscr{F}_{n-1}$ and $\mathscr{F}_n$. Then $T_1 \otimes \left(\bigotimes_{k=2}^{\infty} T_k\right) = \bigotimes_{k=1}^{\infty} T_k$.*

Proof. Let us take $x_0 \in E_0$.

Let us take a finite sequence $(F_t)_{1\leq t\leq T}$ ($t > 1$) such that $F_t \subseteq E_t$. Let us consider $F_1 \times \cdots F_t$. Applying equations 7.8 and 7.14, it follows that

$$\left(T_1 \otimes \left(\bigotimes_{k=2}^{\infty} T_k\right)\right)(x_0, F_1 \times \cdots F_t) = \left(\bigotimes_{k=1}^{\infty} T_k\right)(x_0, F_1 \times \cdots F_t).$$

Since $A \mapsto \left(T_1 \otimes \left(\bigotimes_{k=2}^{\infty} T_k\right)\right)(x_0, A)$ is a probability measure on $\prod_{k=1}^{\infty} T_k$ and $A \mapsto \left(\bigotimes_{k=1}^{\infty} T_k\right)(x_0, F_1 \times \cdots F_t)$ is the unique probability measure having such values on all those finite cartesian products, then the equality of the two measures follows. □

7.3 Nondeterministic and probabilistic transition systems

We shall see how to combine the notions of *nondeterministic choice* (sets of possible choices for which we know no probabilistic properties) and *probabilistic choice* (sets of possible choices for which we know probabilistic properties). This is the notion of *discrete-time Markov decision process* [64] , which has been studied more particularly in the field of operational research and finance mathematics, as well as machine learning.

7.3.1 Different models of probabilistic nondeterminism

Let us now consider the case where the system must be able to do both nondeterministic and probabilistic transitions (examples in Fig. 7.4 and 7.5). The system then has the choice between different transition probabilities, in a set $\mathscr{T}$.

For instance, on Fig. 7.4, in state Q_1, the system has the choice between two partial transition probabilities: the first goes to Q_3 with probability 1, the second goes to Q_4 with probability 1.

For an easier intuition, one may think about this choice as if it were made by an adversary willing to induce certain behaviors. The adversary is supposed to follow a strategy or *policy* [64, §2.1.5]. We shall examine here three subtly different

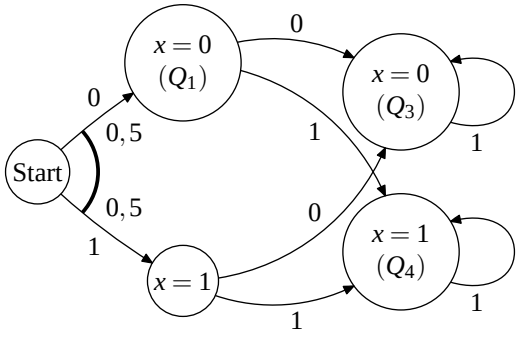


Figure 7.4: A probabilistic-nondeterministic transition system

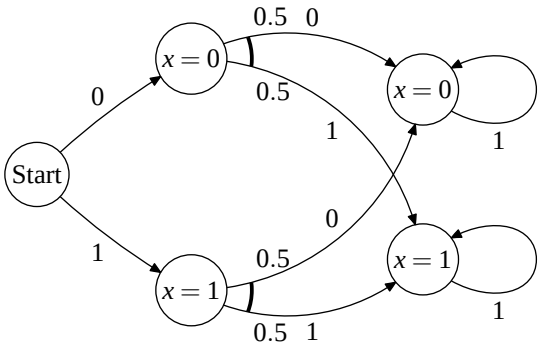
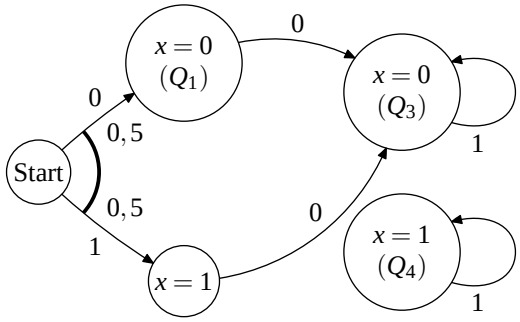
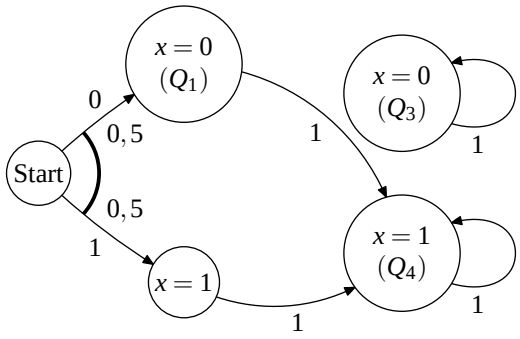


Figure 7.5: Another probabilistic-nondeterministic transition system



(a) Upper transitions



(b) Lower transitions

Figure 7.6: Two purely probabilistic transition systems that define, when composed together nondeterministically, the same probabilistic-nondeterministic transition system as in Fig. 7.4.

conceptions on this notion of policy, which induce three different modelings and different properties:

- The adversary may see only the present state.
- The adversary may see the present state and the past of the execution (the past states). This is the most realistic model of the three, since:
  - a real adversary may well record all past states of the system; if the adversary only records some external view of the system, the model still holds for security analyses, since the capabilities of the adversary are over-estimated by allowing it to look at more than it would be able to;
  - the state of the physical world with which an embedded system interacts depends on all the past actions that were ordered by the embedded system.
- The adversary takes its decisions after all the random choices have been generated, taking them into account. It is as if it could foresee the future of random generators.

7.3.2 Nondeterministic decision based on the past and the present

Let us recall Ionescu Tulcea's theorem (Th. 7.2.12). What this theorem means, in intuitive terms, is that if, for any n, we define a transition probability taking from the first n states to the $n+1$-th one, then we induce a probability on the whole infinite sequence (a transition probability from the initial state to the rest of the sequence, to be precise).

Let us now suppose that our system is defined by a transition probability T from $\Omega \times Y$ to Ω, where Y is the *domain of nondeterministic choices*. For instance, for the systems given in Fig. 7.4 and 7.5, Y is $\{0,1\}$ (choice between the upper and lower arrows, as seen in Fig. 7.6). The operation of the intruder for the transition between states n and $n+1$ is modeled using an unknown transition probability U_n from Ω^n to Y. The whole transition that is executed by the system is then the composition

$$T_n = T \circ \begin{bmatrix} Id \\ U_n \end{bmatrix}, \tag{7.17}$$

which is a transition probability between Ω^n and Ω. By this notation, we mean that, using the notation of Def. 7.2.2,

$$T_n(x_0,\ldots,x_{n-1};x_n) = T(x_{n-1},U_n(x_0,\ldots,x_{n-1});x_n). \tag{7.18}$$

Ionescu Tulcea's theorem then constructs from the (T_n) a transition probability $G(T,(U_n)_{n\in\mathbb{N}})$ from Ω (the initial state space) to $\Omega^{\mathbb{N}}$. We note

$$S_T(f,(U_n)_{n\in\mathbb{N}}) = t_0 \mapsto \int \lambda\vec{t}.f(t_0,\vec{t})\ \mathrm{d}[G(T,(U_n)_{n\in\mathbb{N}})(t_0)] \tag{7.19}$$

(S short for S_T if there is no ambiguity about T) and $R(f)$ the set of all functions $S(T,(U_n)_{n\in\mathbb{N}})$ when $(U_n)_{n\in\mathbb{N}}$ is a sequence of transition probabilities, U_n being a transition probability from Ω^n to Y.

Let $E_+(f) = \sup R(f)$ be the *upper semantics* and $E_-(f) = \inf R(f)$ be the *lower semantics*. Intuitively, if f is the characteristic function of a set of "faulty" traces, E_+ expresses a "worst case" analysis, modeling an adversary willing to make the system err and E_- a "best case" analysis, modeling an adversary willing to prevent the system from erring. $E_+(f)$ is often called the *value* of the Markov decision process with respect to the reward function f (even though we use a slightly different framework as the one given in [64]).

7.3.3 Nondeterministic decision based on the present

We focus here on *memoryless* policies: those only look at the current state of the system to reach a decision. This model can be obtained by restricting the one described in §7.3.2 to the case where $U_n(x_1,\ldots,x_n)$ only depends on x_n.

This model is of particular importance, since it is considerably simpler than the memory-aware model of §7.3.2 and nevertheless is equivalent with respect to certain classes of properties (see remark 8.3.17). A particular case is when the policy is *stationary*, that is, it is the same at each iteration.

7.4 Discussion

The most natural point of view on the nondeterministic-probabilistic processes is that the nondeterministic decisions are taken as the program proceeds, taking into account the current state as well as the previous ones. This how Markov decision processes are usually studied [64]. Chapter 8 will provide effective methods of analysis in this model.

It can be argued that this model is excessively pessimistic. Indeed, if nondeterminism is used to model the environment of an embedded system, then it is excessive to assume that the behavior of this environment depends on the history of the *internal state* of the system; only the part of this history observable from the environment should be taken into account. This leads to the study of *partially observable Markov decision processes* (POMDP); however, their effective analysis is much more complex than that of fully observable processes [48].

Cleaveland's work [11] focuses on the model where the nondeterministic choices are taken *after* the probabilistic ones. This simplifies the theory to some extent, since taking the product of the analyzed process with an nondeterministic "observation" process such as a nondeterministic Büchi automaton is then easy (see §8.2.1 for a discussion on why it is not possible in the model where the nondeterministic choice are taken as the execution proceeds). We shall see how to apply a Monte-Carlo method for such semantics in chapter 14.

Another possible direction is the study of continuous time probabilistic systems. As usual with continuous-time systems, some kind of reduction to discrete time processes is to be done [45, 46].

Chapter 8

Backwards and forward abstract analysis

As we have seen in the preceding chapter, the semantics we consider for probabilistic transition systems take as a parameter a measurable function on the set of (possible or impossible) traces of execution. This function expresses the property we wish to test on the trace (i.e. countable sequence of states); for instance, it may be the characteristic function of the set of sequences that reach a certain state (or a set of states) at least once, or the set of sequences that stay in a certain set of states. The computed integral is then the probability of reaching the state (or the set of states) or of staying in the set of states. Alternatively, the function could be the number of times that the sequence reaches a certain state (or set of states). The computed integral is then the average time (one iteration step counting for one time unit) spent in the state (or set of states). We shall see here generalizations of these properties and how to analyze them.

8.1 The properties to analyze

We consider a property to analyze on the traces. To each initial state we attach its *potentiality*, that is, the integral of the property to analyze over the traces starting from this state (or the set of integrals, if considering nondeterminism). The properties to analyze are expressed as measurable functions from the set of (possible) traces of execution; we call these functions *trace valuators*. We shall actually consider a class of trace valuators defined syntactically by certain formulas.

8.1.1 Potentiality Functions

Let $I = [0,1]$ or $[0,+\infty]$. Let X be a finite or countable set of states — we impose this cardinality constraint so as to avoid theoretical complexities. $\mathcal{P}(X)$ is the set of subsets of X; it is a σ-algebra.

Let $X \to I$ be the set of functions from X to I. These are the *potentiality functions* of our system. The space $X \to I$ of potentiality functions is ordered by $\leq$ point-wise; is a complete lattice

Lemma 8.1.1. *Let X be a finite or countable set of states. Let K be a directed (Def. A.2.1) subset of $X \to I$. Then there exists an ascending sequence of elements of K that converges point-wise to $\sqcup K$, the point-wise upper-bound of K.*

Proof. We shall assimilate X to either the finite set $\{0,\ldots,N-1\}$ or $\mathbb{N}$ (in which case $N = \infty$), depending on its cardinality. Let $F = \sqcup K$.

Let us construct such a sequence $(f_n)_{n\in\mathbb{N}^*}$ of elements of K that converges point-wise to F. Let $n \in \mathbb{N}$. Let us construct by recurrence an ascending sequence $(g_k)_{1\leq k\leq \min(n,N-1)}$ of elements of K:

$n = 1$ There exists a $g_1 \in K$ such that $g_1(0) \geq F(0) - 1/n$ if $F(0) < \infty$ or $g_1(0) \geq n$ if $F(0) = +\infty$.

$n > 1$ There exists a $g \in K$ such that $g(k) \geq F(k) - 1/n$ if $F(k) < \infty$ or $g(k) \geq n$ if $F(0) = +\infty$; since g and g_{k-1} belong to K and K is directed, we can therefore take $g_k \in K$ such that $g_k \geq g$ and $g_k \geq g_{k-1}$.

Let f_n be g_n.

Let us now construct by recurrence an ascending sequence $(\tilde{f}_n)_{n\in\mathbb{N}^*}$ of elements of K:

$n = 1$ Let $\tilde{f}_1 = f_1$.

$n > 1$ There exists a $\tilde{f}_n \in K$ such that $\tilde{n}_n \geq f_n$ and $\tilde{f}_n \geq f_{n-1}$.

Let us show that $(\tilde{f}_n)_{n\in\mathbb{N}^*}$ converges point-wise to F. Let $x \in X$.

- Case when $F(x) < \infty$. If $n \geq x$, then $f_n(x) \geq F(x) - 1/n$. Thus $F(x) - 1/n \leq \tilde{f}_n(x) \leq F(x)$ and $\lim n \to \infty \tilde{f}_n(x) = F(x)$.
- Case when $F(x) = \infty$. If $n \geq x$, then $f_n(x) \geq n$ thus $\lim_{n\to\infty} \tilde{f}_n(x) = F(x)$.

□

Corollary 8.1.2. *If $\alpha : (X \to I) \xrightarrow{mon} Y$ is an ω-upper-continuous operator, then it is an upper-continuous operator.*

Proof. Let F be a totally ordered subset of $X \to I$. F is directed, and thus lemma 8.1.1 implies that there exists an ascending sequence $(x_n)_{n\in\mathbb{N}}$ of elements of F such that $\sqcup F = \bigsqcup_n x_n$. $\alpha(\sqcup F) = \alpha(\bigsqcup_n x_n) = \bigsqcup_n \alpha(x_n) \sqsubseteq \bigsqcup \alpha(F)$. $\alpha(\sqcup F) \sqsupseteq \bigsqcup \alpha(F)$ follows from lemma A.2.2. □

8.1.2 Trace valuators

Let $X^{\mathbb{N}} \to I$ be the set of measurable functions from $X^{\mathbb{N}}$ to I, ordered point-wise. We shall call such functions "valuators".

Boolean trace valuators

We take $I = [0,1]$ or even $I = \{0,1\}$. We shall consider formulas written in the following language:

formula1 ::= | *name*
| *constant*
| *name* $+_{set}$ *formula1*
where *set* $\subseteq X$
| *constant* $+_{set}$ *formula1*
| lfp(*name* $\mapsto$ *formula1*)
| gfp(*name* $\mapsto$ *formula1*)
| shift(*formula1*)
| let *name* = *formula1* in *formula1*

Let shift : $X^{\mathbb{N}} \to X^{\mathbb{N}}$: $(\text{shift}.t)_k = t_{k+1}$.

Let env_t be the set of environments of valuators (an environment of valuators maps each *name* to a valuator), ordered point-wise.

$[\![formula]\!]_t : env_t \to (X^{\mathbb{N}} \to I)$ is defined inductively as follows:

$$[\![name]\!]_t.env = env(name) \tag{8.1}$$
$$[\![constant]\!]_t.env = constant \tag{8.2}$$
$$[\![f_1 +_S f_2]\!]_t.env = \lambda t.\chi_S(t_0).([\![f_1]\!]_t\, env) + \chi_{S^C}(t_0).([\![f_2]\!]_t\, env) \tag{8.3}$$
$$[\![\text{lfp}(name \mapsto f)]\!]_t.env = \text{lfp}(\lambda\phi.[\![f]\!]_t.env[name \mapsto \phi]) \tag{8.4}$$
$$[\![\text{gfp}(name \mapsto f)]\!]_t.env = \text{gfp}(\lambda\phi.[\![f]\!]_t.env[name \mapsto \phi]) \tag{8.5}$$
$$[\![\text{shift}(f)]\!]_t.env = ([\![f]\!]_t.env) \circ \text{shift} \tag{8.6}$$
$$[\![\text{let } name = f_1 \text{ in } f_2]\!]_t.env = [\![f_2]\!]_t.env[name \mapsto [\![f_1]\!]_t.env] \tag{8.7}$$

χ_S is the characteristic function of S and S^C the complement of S. $t_0 \in X$ is the first state of the sequence of states $\vec{t} \in X^{\mathbb{N}}$.

Lemma 8.1.3. *For all formula f, $[\![f]\!]_t$ is monotonic.*

Lemma 8.1.4. *For all formula f without* lfp *or* gfp, $[\![f]\!]_t$ *is continuous.*

Proof. By induction on f. All cases are trivial. □

Lemma 8.1.5. $E_+(\top) = \top$ *and* $E_+(\bot) = \bot$.

Lemma 8.1.6. E_+ *is monotonic and upper-continuous.*

Proof. It is trivial that E_+ is monotonic. Let us now prove it is ω-upper-continuous. Let f_n be an ascending sequence of trace valuators. Then

$$\begin{aligned}&\sup_{f_n} E_+(f_n)(t_0) = \sup_n \int \lambda\langle t_1,\ldots\rangle.\, f_n(\langle t_0,t_1,\ldots\rangle)\,\mathrm{d}\left(\textstyle\bigotimes_{k=1}^{\infty} T_k\right)(t_0)\\ &= \int \lambda\langle t_1,\ldots\rangle.\sup_n \lambda\langle t_1,\ldots\rangle.\, f_n(\langle t_0,t_1,\ldots\rangle)\,\mathrm{d}\left(\textstyle\bigotimes_{k=1}^{\infty} T_k\right)(t_0) = E + (\bigsqcup_n f_n)\end{aligned}$$

From corollary 8.1.2, it follows that E_+ is upper-continuous. □

Some particularly interesting Boolean valuators

We shall consider in this section three very important classes of properties: reachability, liveness and a combination of the two, Büchi acceptance. We shall show that all those properties are measurable.

Reachability Let A be a (measurable) set of states. The *reachability* property associated with A defines the set of traces that pass through A at some point.

$$\mathrm{lfp}(f \mapsto 1 +_A \mathrm{shift}\, f) \tag{8.8}$$

This fixpoint is reached in countable time:

$$\mathrm{lfp}(\underbrace{\phi \mapsto [\![1 +_A \mathrm{shift}\, f]\!]_t.[f \mapsto \phi]}_{\Psi}) = \bigsqcup_n \Psi^n(0)$$

Since all the $\Psi^n(0)$ functions are measurable (composition of measurable functions), the limit $[\![\mathrm{lfp}(f \mapsto 1 +_A \mathrm{shift}\, f)]\!]_t$ of the sequence is also measurable.

Liveness Let A be a (measurable) set of states. The *liveness* property associated with A defines the set of traces that always remain in A. It is the dual situation of safety. It corresponds to the formula

$$\mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0) \tag{8.9}$$

As before, $[\![\mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0)]\!]_t$ is measurable.

Büchi acceptance Let A be a (measurable) set of states. The *Büchi acceptance* property associated with A defines the set of traces that pass through A infinitely often. We shall see how to define it in terms of fixpoints.

Let us consider W_n the set of traces that pass through A at least n times. W_1 is the reachability property $[\![\mathrm{lfp}(f \mapsto 1 +_A \mathrm{shift} f)]\!]_t$; $W(2)$ is the nested reachability property $[\![\mathrm{lfp}(f \mapsto \mathrm{shift}\,\mathrm{lfp}(f \mapsto 1 +_A \mathrm{shift} f) +_A \mathrm{shift} f)]\!]_t$; and, more generally, noting $\Psi(\phi) = \mathrm{lfp}(f \mapsto [\![n +_A \mathrm{shift} f]\!]_t).[n \mapsto \phi]$, $W_n = \Psi^n(1)$. Obviously, $(W_n)_{n\in\mathbb{N}}$ decreases, and the Büchi acceptance property W_∞ is the limit of the sequence. Furthermore, W_∞ is a fixpoint of Ψ: let us take a trace S; if S passes through A an infinite number of times, then $W_\infty(S) = \Psi(W_\infty(S)) = 1$, otherwise $W_\infty(S) = \Psi(W_\infty(S)) = 0$. But then W_∞ is the greatest fixpoint of Ψ: if we take a fixpoint F of Ψ, then $F \leq 1$; by induction over n, $F = \Psi^n(F) \leq \Psi^n(1) = W_n$ and thus $F \leq W_\infty$. The Büchi acceptance property is thus written as:

$$\mathrm{gfp}(C \mapsto \mathrm{lfp}(R \mapsto \mathrm{shift}(C) +_A \mathrm{shift}(R))) \tag{8.10}$$

If ϕ is measurable, then $\Psi(\phi)$ is measurable, since the least fixpoint is reached in countable time; thus all the $(W_n)_{n\in\mathbb{N}}$ are measurable. Their countable limit is thus measurable.

Summation valuator

A related family of trace valuators are the *summing* valuators. The summation valuator associated with a (measurable) function $f : X \mapsto [0, +\infty]$ is the function

$$[\![\sum A]\!]_t : \left| \begin{array}{ll} X^{\mathbb{N}} & \to [0, +\infty] \\ (x_n)_{n\in\mathbb{N}} & \mapsto \sum_{k=0}^{\infty} f(x_k) \end{array} \right. \tag{8.11}$$

Obviously, this function can be formulated as a least fixpoint:

$$[\![\Sigma f]\!]_t = \mathrm{lfp}(\phi \to f + \phi \circ \mathrm{shift}) \tag{8.12}$$

This construct has two obvious applications:

- counting the average number of times the program goes through a certain set of states A; here, f is the characteristic function of A;
- counting the average time spent in the process; here f maps each state to the amount of time spent in that state (0 for states meaning "termination").

8.2 Temporal logics

Temporal logics [10, chapter 3] are expressive means of specifying properties of transition systems.

8.2.1 Deterministic Büchi automata

The *Linear Temporal Logic* (LTL) expresses properties defining sets of traces. Since probabilistic transition systems induce probabilities not on states, but on sets of traces, LTL looks like a natural starting point for probabilistic analyses. We shall recall briefly the syntax of LTL:

$$
\begin{aligned}
\mathit{LTL\_formula} ::= {} & \mathit{state\_formula} \\
& \mathit{LTL\_formula}_1 \wedge \mathit{LTL\_formula}_2 \\
& \mathit{LTL\_formula}_1 \vee \mathit{LTL\_formula}_2 \\
& \neg \mathit{LTL\_formula} \\
& \mathit{LTL\_formula}_1 \textbf{ until } \mathit{LTL\_formula}_2 \\
& \mathit{LTL\_formula}_1 \textbf{ release } \mathit{LTL\_formula}_2 \\
& \textbf{eventually } \mathit{LTL\_formula} \\
& \textbf{always } \mathit{LTL\_formula}
\end{aligned}
$$

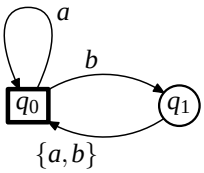
and its semantics ($B = (b_n)_{n\in\mathbb{N}} \in S^{\mathbb{N}}$):

$$
\begin{aligned}
B \models \mathit{state\_formula} &\iff b_0 \models \mathit{state\_formula} \\
B \models \neg \mathit{LTL\_formula} &\iff B \not\models \mathit{LTL\_formula} \\
B \models \%textit\, f_1 \wedge f_2 &\iff (B \models f_1) \wedge (B \models f_2) \\
B \models f_1 \vee f_2 &\iff (B \models f_1) \vee (B \models f_2) \\
B \models f_1 \textbf{ until } f_2 &\iff \exists k \geq 0 \begin{cases} (\forall j < k \ \mathrm{shift}^j(B) \models f_1) \\ (\mathrm{shift}^k(B) \models f_2) \end{cases} \\
B \models f_1 \textbf{ release } f_2 &\iff \forall j \geq 0 \begin{cases} (\forall i < j \ \mathrm{shift}^i(B) \not\models f_1) \\ (\mathrm{shift}^j(B) \models f_2) \end{cases}
\end{aligned}
$$

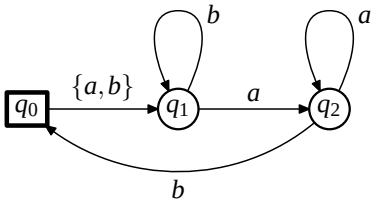
where $\mathrm{shift}^k(B)$ is the suffix of sequence B starting at offset k.

We shall now recall a few facts on Büchi automata [10, Ch. 9] [76, §I.1]. Büchi automata are an extension to infinite words of the familiar finite automata on finite words. A nondeterministic Büchi automaton on the alphabet Σ is defined by a (finite) set of states Q, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, a set of initial states $Q_0 \subseteq Q$ and a set of accepting states $A \subseteq Q$. $(s_0, l, s_1) \in \Delta$ means that the automaton takes a transition from state s_0 to state s_1 on letter l; we require that for any state s_0 and letter l there exists a state s_1 such that $(s_0, l, s_1) \in \Delta$. Let us now consider a sequence $L = (l_n)_{n\in\mathbb{N}} \in \Sigma^{\mathbb{N}}$ of letters. A sequence $R = (r_n)_{n\in\mathbb{N}} \in R^{\mathbb{N}}$ of states is said to be a *run* for L if $r_0 \in Q_0$ and for all n, $(r_n, l_n, r_{n+1}) \in \Delta$. A run is said to be *accepting* if it goes infinitely often through A. A sequence is *accepted* if there exists an accepting run for it.

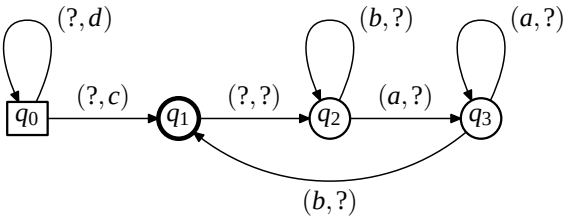
A major interest of Büchi automata is that they give an elegant way to analyze *Linear Temporal Logic* (LTL) [10, §3.2]. Let us consider a LTL formula F. Let



(a) Fairness for *a*



(b) Fairness for *a* and *b*



(c) Fairness for *a* and *b* and reachability for *c*

Figure 8.1: Büchi automata for various conditions. The alphabet is $\{a,b\} \times \{c,d\}$. The initial state is boxed; accepting states have a thicker frame.

us call Z the set of atomic state formulas in F. There exists an algorithm [10, §9.4] to construct a Büchi automaton $\mathscr{A}$ on alphabet $\mathcal{P}(Z)$ such that the trace (sequence of states) $(s_n) \in S^{\mathbb{N}}$ is a model of F if and only if $\mathscr{A}$ accepts $(b_n) \in \left(\{0,1\}^Z\right)^{\mathbb{N}}$ where $z \in b_n$ if and only if $s_n \models z$. Model-checking the formula F on a nondeterministic transition system S is then equivalent to model-checking a Büchi acceptance condition (see §8.1.2) on the synchronous product of S and $\mathscr{A}$.

The nondeterminism of this synchronous product arises from two sources: the nondeterminism of S and the nondeterminism of $\mathscr{A}$. Unfortunately, when adding probabilities, those two nondeterminisms cannot be considered together. Indeed, what we are trying to compute is a bound on:

$$\sup_{\substack{\textit{nondeterminism}\\ \textit{of the system}}} \mathbf{E}\left(\sup_{\substack{\textit{nondeterminism}\\ \textit{of the automaton}}} \begin{cases} 1 & \text{if an accepting run} \\ 0 & \text{otherwise} \end{cases} \right)$$

and we see that the nondeterminism of the automaton is taken knowing the full sequence of probabilistic choices. The analysis methods explained in this chapter do not work for such systems (but the Monte-Carlo method does, see chapter B). On the other hand, an obvious workaround is to restrict ourselves to *deterministic* Büchi automata. . A deterministic Büchi automaton is just like a nondeterministic one except that the transition relation Δ is actually a function (for any state q_0 and letter l, there exists exactly one state q_1 such that $(q_0, l, q_1) \in R$). We then prefer to use a transition function $f : Q \times \Sigma \to Q$.

The class of deterministic Büchi automata is stable by finite product (but not by complementation [76, example 4.2]). This is interesting, since it makes it possible to express *fairness constraints*. Concurrent systems are often specified without precising the scheduler, but the scheduler is often assumed to be *fair*: no process should be indefinitely prevented from running (equivalently, at any process at any point in time, there exists a point in the future when that process will be allowed to proceed). What we want to quantify is the probability of failure of the system given the fairness constraints (Fig. 8.1).

Given a deterministic Büchi automaton $\mathscr{A}$ (state space Q, transition function f) and a Markov decision process P (state space X, space of nondeterministic inputs Y, transition probability T from $X \times Y$ to X), we define their synchronous product as the Markov decision process P', with state space $X \times Q$ and transition probability T' from $(X \times Q) \times Y$ to $X \times Q$ defined by:

$$T'(((x_1, q_1), y), (x_2, q_2)) = T((x_1, y), x_2) \tag{8.13}$$

Lemma 8.2.1. *Let $g : X^{\mathbb{N}} \to I$ be a trace valuator. Let*

$$\begin{array}{rll} \pi_1 : & (X \times Q)^{\mathbb{N}} & \to I \\ & (x_n, q_n)_{n\in\mathbb{N}} & \mapsto \pi_1(x_0, x_1, \ldots) \end{array} \tag{8.14}$$

then we can test g equivalently on P and P':

$$E_{+T}(f) = E_{+T'}(f \circ \pi_1) \tag{8.15}$$

Proof. Let $(U_n)_{n\in\mathbb{N}}$ be a policy for T (U_n is a transition probability from X^n to Y). We define the policy $(U'_n)_{n\in\mathbb{N}}$ where U'_n is a transition probability from $(X \times Q)^n$ to Y as follows:

$$U'_n\left((x_k, q_k)_{0\leq k<n}\right) = U_n\left((x_k)_{0\leq k<n}\right). \tag{8.16}$$

Let $x_0 \in X$. Then

$$\int g \,\mathrm{d}\left(G(T, (U_n)_{n\in\mathbb{N}})(x_0)\right) = \int \pi_1 \circ g \,\mathrm{d}\left(G(T', (U'_n)_{n\in\mathbb{N}})(x_0, q_0)\right). \tag{8.17}$$

Let U'_n be a policy for T' (U'_n is a transition probability from $(X \times Q)^n$ to Y). We recall that f is the transition function of the Büchi automaton. We define the policy $(U_n)_{n\in\mathbb{N}}$ where U_n is a transition probability from $(X \times Q)^n$ to Y as follows:

$$U_n\left((x_k)_{0\leq k<n}\right) = U'_n\left((x_k, q_k)_{0\leq k<n}\right). \tag{8.18}$$

where q_0 is the initial state of the Büchi automaton and $q_{k+1} = f(x_k, q_k)$ for all k.

$$\int g \,\mathrm{d}\left(G(T, (U_n)_{n\in\mathbb{N}})(x_0)\right) = \int \pi_1 \circ g \,\mathrm{d}\left(G(T', (U'_n)_{n\in\mathbb{N}})(x_0, q_0)\right). \tag{8.19}$$

The result follows. □

8.2.2 Branching-time logic: pCTL and pCTL*

The branching-time logic CTL [10, §3.2] has had much success in the analysis of nondeterministic (albeit non probabilistic) systems. It is therefore quite natural to extend this notion to probabilistic systems. The extension described here, pCTL [30], is quite technical in its nature and we invite the reader to consult the relevant literature on CTL* [10, §3] before proceeding. Furthermore, we do not see a clear intuitive meaning of the formulas it allows to write.

CTL* formulas define sets of states as the starting states of sets of traces defined by LTL path formulas (in which state formulas are CTL* state formulas).

CTL, a restricted version of CTL, restrict the kind of path formulas that can be used.

The operation that makes a CTL* state formula out of a LTL path formula is the taking of the initial states: if $[\![\mathtt{f}]\!]_s$ denotes the semantics of f as a state formula and $[\![\mathtt{f}]\!]_p$ its semantics as a path formula, then

$$[\![\mathtt{f}]\!]_s = \{x_0 \in X \mid \exists x_1, \ldots \langle x_0, x_1, \ldots\rangle \in [\![\mathtt{f}]\!]_p\}. \tag{8.20}$$

In the case of probabilistic systems, we do not have sets of starting states but potentiality functions; such potentiality functions are then compared to a threshold value, which gives sets of states. This yields to several ways of going from a state formula, noted as $f_{\bowtie\alpha}$ where f is a trace valuator and $\bowtie$ is $\leq$, $<$, $=$, $>$ or $\geq$. The semantics of this construct is as follows:

$$[\![f_{\bowtie\alpha}]\!] = \{x_0 \in X \mid \forall (U_n)_{n\in\mathbb{N}} S([\![f]\!]_t, (U_n)_{n\in\mathbb{N}})(x_0) \bowtie \alpha\} \tag{8.21}$$

In the case of $<$ and $\leq$, we have an upper bound $[\![f]\!]^\sharp_{e+}$ of $E_+([\![t]\!]_t)$ (see §8.3.5):

$$\forall x_0 \, [\![f]\!]^\sharp_{e+}(x_0) \bowtie \alpha \implies x_0 \notin [\![f_{\bowtie\alpha}]\!]. \tag{8.22}$$

8.3 Backwards Upper Analysis

A well-known solution to the problem of the optimal value of a Markov decision processes is *value iteration* [64, §7.2.4]. This method is mainly of theoretical interest for the analysis of finite state Markov decision processes, since there is little control as to its rate of convergence and much better algorithms are available (see §10.3.1). Since it is actually a kind of generalization to Markov decision processes of the backwards reachability analysis for nondeterministic systems, we shall apply abstract interpretation techniques so as to provide an effective mean to compute upper bounds on the probability of the properties to analyze.

8.3.1 Backwards Upper Semantics

Let env_e be the set of environments of potentiality functions (an environment of potentiality functions maps each *name* to a potentiality function), ordered pointwise.

$[\![\mathit{formula}]\!]_{e+} : (X \to I) \to (X \to I)$ is defined inductively as follows:

$$[\![\mathit{name}]\!]_{e+}.\mathit{env} = \mathit{env}(\mathit{name}) \tag{8.23}$$
$$[\![\mathit{constant}]\!]_{e+}.\mathit{env} = \lambda x.\mathit{constant} \tag{8.24}$$
$$[\![f_1 +_S f_2]\!]_{e+}.\mathit{env} = \chi_S.([\![f_1]\!]_{e+}\,\mathit{env}) + \chi_{S^C}.([\![f_2]\!]_{e+}\,\mathit{env}) \tag{8.25}$$
$$[\![\mathrm{lfp}(\mathit{name} \mapsto f)]\!]_{e+}.\mathit{env} = \mathrm{lfp}(\lambda\phi.[\![f]\!]_{e+}.\mathit{env}[\mathit{name} \mapsto \phi]) \tag{8.26}$$
$$[\![\mathrm{gfp}(\mathit{name} \mapsto f)]\!]_{e+}.\mathit{env} = \mathrm{gfp}(\lambda\phi.[\![f]\!]_{e+}.\mathit{env}[\mathit{name} \mapsto \phi]) \tag{8.27}$$
$$[\![\mathrm{shift}(f)]\!]_{e+}.\mathit{env} = \sup_{T\in\mathscr{T}}(\overleftarrow{T}([\![f]\!]_{e+}.\mathit{env})) \tag{8.28}$$
$$[\![\text{let } \mathit{name} = f_1 \text{ in } f_2]\!]_{e+}.\mathit{env} = [\![f_2]\!]_{e+}.\mathit{env}[\mathit{name} \mapsto [\![f_1]\!]_{e+}.\mathit{env}] \tag{8.29}$$

This semantics is thus some form of μ-calculus, except that "lfp" replaces the μ binder and "gfp" ν; but since we also use μ to note measures, it would have been confusing to also use it in the syntax of the formulas. Similar constructs have been introduced by other authors, such as *quantitative game μ-calculus* [22].

$[\![\mathit{formula}]\!]_{e+}$ is monotonic.

Lemma 8.3.1. *Let f be a formula not containing* gfp. *$[\![f]\!]_{e+}$ is ω-upper-continuous.*

Proof. By structural induction on f. Let env_n be an ascending sequence of environments, whose limit is $\mathit{env}_\sqcup$. The cases for *name*, *constant* and "let" are trivial.

- Let $t_0 \in S$. $\bigsqcup_{n\in\mathbb{N}} \underbrace{([\![f_1 +_S f_2]\!]_{e+}.\mathit{env}_n)(t_0)}_{([\![f_1]\!]_{e+}.\mathit{env})(t_0)} = ([\![f_1]\!]_{e+}.\mathit{env}_\sqcup)(t_0) = ([\![f_1 +_S f_2]\!]_{e+}.\mathit{env}_\sqcup)(t_0)$. Similarly for $t_0 \notin S$.
- The shift case:

$$\begin{aligned}&[\![\mathrm{shift}(f)]\!]_{e+}.\mathit{env}_\sqcup = \bigsqcup_{T\in\mathscr{T}} \overleftarrow{T}([\![f]\!]_{e+}.\mathit{env}_\sqcup))\\ &= \bigsqcup_{T\in\mathscr{T}} \overleftarrow{T}(\bigsqcup_n [\![f]\!]_{e+}.\mathit{env}_n) = \bigsqcup_{T\in\mathscr{T}} \bigsqcup_n \overleftarrow{T}.([\![f]\!]_{e+}.\mathit{env}_n)\\ &= \bigsqcup_n \bigsqcup_{T\in\mathscr{T}} \overleftarrow{T}.([\![f]\!]_{e+}.\mathit{env}_n) = \bigsqcup_n [\![\mathrm{shift}(f)]\!]_{e+}.\mathit{env}_n\end{aligned}$$

- The lfp case follows by applying lemma A.2.13 to the induction hypothesis.

□

As for the summation valuator,

$$[\![\Sigma f]\!]_{e+} = \mathrm{lfp}\left(\phi \mapsto f + \sup_{T\in\mathscr{T}}(\overleftarrow{T}.\phi)\right) \tag{8.30}$$

8.3.2 Present-knowing adversary

When defining adversaries, or policies, one has to decide whether the adversary only looks at the last state of the system or considers or whether it looks at the whole history of the execution. We shall see here the adversaries that only look at the present state of the system, also known as *memoryless policies*. If we fix the policy, the system becomes a Markov chain.

Concrete semantics

Let us note $X^{\mathbb{N}}$ the space of countable sequences of elements of X, with the product σ-algebra $\mathcal{P}(X)^{\otimes\mathbb{N}}$. Let Y be a measurable set of "choices". Let T be a transition probability between $X \times Y$ and X. Let us consider the set of transition probabilities

$$\mathscr{T} = \{T \circ (x \mapsto (x, f(x)))_T \mid f \in X \to Y\},$$

calling g_T the transition probability associated with the deterministic operation g, and $\circ$ the composition of transition probabilities. Let us note that the associated (backwards) operator on measurable functions is

$$\overleftarrow{\mathscr{T}} = \{f \mapsto (\overleftarrow{T}.f) \circ (x \mapsto (x, g(x))) \mid g \in X \to Y\}.$$

Lemma 8.3.2. $\overleftarrow{\mathscr{T}} \subseteq (X \to I) \xrightarrow{mon} (X \to I)$ *is optimizing(see Def. A.2.7).*

Proof. Let $f \in X \to I$. Let $\phi_1 = h \mapsto (\overleftarrow{T}.h) \circ (x \mapsto (x, g_1(x)))$ and $\phi_2 = h \mapsto (\overleftarrow{T}.h) \circ (x \mapsto (x, g_2(x)))$.

Let us define $g_3 : X \to Y$ as follows.
Let x in X. If $(\overleftarrow{T}.h)(x, g_1(x)) \geq (\overleftarrow{T}.h)(x, g_2(x))$ then $g_3(x) = g_1(x)$ else $g_3(x) = g_2(x)$. Let $\phi_3 = h \mapsto (\overleftarrow{T}.h) \circ (x \mapsto (x, g_3(x)))$.

Let us show that $\phi_3(f) \geq \phi_1(f)$ and $\phi_3(f) \geq \phi_2(f)$. Let x in X. $\phi_3(f)(x) = (\overleftarrow{T}.f)(x, g_3(x)) \geq (\overleftarrow{T}.h)(x, g_1(x)) = \phi_1(f)(x)$ and similarly $\phi_3(f)(x) \geq \phi_2(f)(x)$. Thus $\phi_3(f) \geq \phi_1(f)$ and $\phi_3(f) \geq \phi_2(f)$. □

Lemma 8.3.3. *For all transition probability T, $\overleftarrow{T}$ is ω-continuous.*

Proof. Let f_n be an ascending sequence. $(\overleftarrow{T}.f_n)(x) = \int f_n \, \mathrm{d}T_x$. The lemma follows from theorem A.1.1. □

The set according to which we define E_+ is

$$R(f) = \left\{ \lambda t_0. \int \lambda \langle t_1, \ldots \rangle . (\llbracket f \rrbracket_t . env)(\langle t_0, t_1, \ldots \rangle) \, \mathrm{d} \left(\bigotimes_{k=1}^{\infty} T_k \right)(t_0) \mid (T_n)_{n \in \mathbb{N}} \in \mathscr{T}^{\mathbb{N}} \right\} \tag{8.31}$$

The abstraction relation between the semantics

Lemma 8.3.4. *For all t_0 in$X$,*

$$E_+(\lambda t.\chi_S(t_0).V_1(t)+\chi_{S^C}(t_0).V_2(t)).t_0 = \chi_S(t_0).(E_+(V_1).t_0)+\chi_{S^C}(t_0).(E_+(V_2).t_0).$$

Proof. Let $t_0 \in X$.

Let us suppose that $t_0 \in S$. Then $E_+(\underbrace{\lambda t.\chi_S(t_0).V_1(t)+\chi_{S^C}(t_0).V_2(t)}_{V_1}).t_0 =$ $\chi_S(t_0).(E_+(V_1).t_0)+\chi_{S^C}(t_0).(E_+(V_2).t_0)$. Similar proof for $t_0 \notin S$. □

Theorem 8.3.5. *Let f be a formula not containing* lfp *nor* gfp *and where we allow only constants as the left operand of each operator of the form $+_A$. Let env be an environment of valuators. Noting $E_+(env)$ the point-wise application of E_+ to env, we have*

$$[\![f]\!]_{e+}.(E_+(env)) = E_+([\![f]\!]_t.env) \tag{8.32}$$

Proof. We shall demonstrate additionally the property that $R([\![f]\!]_t.env)$ is directed. Proof by induction on the structure of f.

- $[\![name]\!]_{e+}.(E_+(env)) = E_+env(name) = [\![name]\!]_t(E_+(env))$;
  $R([\![\texttt{name}]\!].env)$ is the singleton $\{env(name)\}$ and thus is directed.

- $[\![constant]\!]_{e+}.(E_+(env)) = constant = [\![constant]\!]_t(E_+(env))$;
  $R([\![\texttt{constant}]\!].env)$ is the singleton $\{constant\}$ and thus is directed.

- Let $t_0 \in X$.

$$\begin{array}{ll}
[\![f_1 +_S f_2]\!]_{e+}.(E_+(env)).t_0 & \\
= \chi_S(t_0).([\![f_1]\!]_{e+}.E_+(env).t_0)+\chi_{S^C}(t_0).([\![f_2]\!]_{e+}.E_+(env).t_0) & \\
= \chi_S(t_0).(E_+([\![f_1]\!]_t.env).t_0)+\chi_{S^C}(t_0).(E_+([\![f_2]\!]_t.env).t_0) & \text{(induction)} \\
= E_+(\lambda t.\chi_S(t_0).([\![f_1]\!]_t envt)+\chi_{S^C}(t_0).([\![f_2]\!]_t envt)).t_0 & \text{(lemma 8.3.4)} \\
= E_+([\![f_1 +_S f_2]\!]_t).t_0. &
\end{array}$$

  f_1 must be a *constant*, $R([\![f_1 +_S f_2]\!]_t.env) = f_1 + R([\![f_2]\!]_t.env)$; since $R([\![f_2]\!]_{e+})$ is directed, then so is $R([\![f_1 +_S f_2]\!]_t.env)$.

- Let $t_0 \in X.T^\infty(x)$ is a notation for $\left(\bigotimes_{k=1}^{\infty} T_k\right)(x)$.

$$
\begin{aligned}
&E_+([\![\text{shift}(f)]\!]_t.env).t_0 \\
&\quad = \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda\langle t_1,\ldots\rangle.([\![f]\!]_t.env)\circ\text{shift}(\langle t_0,t_1,\ldots\rangle)\,\mathrm{d}T^\infty t_0 \\
&\quad\quad = \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda\langle t_1,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}T^\infty t_0 \\
&\quad = \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda t_1.\int \lambda\langle t_2,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}T^\infty t_1\,\mathrm{d}T(t_0) \\
&\quad = \sup_{T_1}\sup_{(T_n)_{n\geq 2}} (\overleftarrow{T_1}(\lambda t_1.\int \lambda\langle t_2,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}T^\infty t_1))(t_0) \\
&\quad\quad = \sup_{T_1}(\bigsqcup_{f\in R([\![f]\!]_t.env)} (\overleftarrow{T_1}.f))(t_0).
\end{aligned}
$$

From lemma 8.3.3, $\overleftarrow{T_1}$ is a monotonic, ω-continuous, operator; from the induction hypothesis, $R([\![f]\!]_t.env)$ is directed; from lemma A.2.6,

$$\bigsqcup_{f\in R([\![f]\!]_t.env)} (\overleftarrow{T_1}f) = \overleftarrow{T_1}(\bigsqcup_{f\in R([\![f]\!]_t.env)} f).$$

- The case for "let" is trivial.

□

Corollary 8.3.6. *Let f be a formula not containing* lfp *nor* gfp *and where for all construction $f_1 +_A f_2$, f_1 is a constant. Let env be an environment of valuators. Noting $E_+(env)$ the point-wise application of E_+ to env,*

$$[\![\text{lfp } f]\!]_{e+}.(E_+(env)) = E_+([\![\text{lfp } f]\!]_t.env) \tag{8.33}$$

Proof. $[\![\text{lfp}(name \mapsto f)]\!]_{e+}.env = \text{lfp}(\lambda\phi.[\![f]\!]_{e+}.env[name\mapsto\phi])$.
From lemma 8.3.1, $\lambda\phi.[\![f]\!]_{e+}.env[name\mapsto\phi]$ is ω-upper-continuous.

$[\![\text{lfp}(name \mapsto f)]\!]_t.env = \text{lfp}(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])$.
From lemma 8.1.4, $\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi]$ is ω-upper-continuous.

From the theorem 8.3.5,

$$E_+\circ(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi]) = (\lambda\phi.[\![f]\!]_{e+}.E_+(env)[name\mapsto\phi])).$$

From lemma 8.1.6, E_+ is ω-upper-continuous. The corollary then follows from lemma A.2.17. □

Theorem 8.3.7. *The semantics of the summing operator satisfy:*

$$E_+(\llbracket \Sigma f \rrbracket_t) = \llbracket \Sigma f \rrbracket_{e+} \tag{8.34}$$

Proof. Similar as the proofs of the preceding theorem and corollary. □

Theorem 8.3.8. *Let f be a formula. Let env be an environment of valuators. Let us suppose that $H \geq E_+(env)$ pointwise. Then*

$$\llbracket f \rrbracket_{e+}.(H) \geq E_+(\llbracket f \rrbracket_t.env) \tag{8.35}$$

Proof. Proof by induction on the structure of f.

- $\llbracket name \rrbracket_{e+}.(H) = H(name) \geq E_+(env(name)) = \llbracket name \rrbracket_t (E_+(env))$.
- $\llbracket constant \rrbracket_{e+}.(E_+(env)) = constant = \llbracket constant \rrbracket_t (E_+(env))$.
- Let $t_0 \in X$. Then

$$\begin{array}{ll}
\llbracket f_1 +_S f_2 \rrbracket_{e+}.(H).t_0 & \\
= \chi_S(t_0).(\llbracket f_1 \rrbracket_{e+}.H.t_0) + \chi_{S^C}(t_0).(\llbracket f_2 \rrbracket_{e+}.H.t_0) & \\
\geq \chi_S(t_0).(E_+(\llbracket f_1 \rrbracket_t.env).t_0) + \chi_{S^C}(t_0).(E_+(\llbracket f_2 \rrbracket_t.env).t_0) & \text{(induction)} \\
= E_+(\lambda t.\chi_S(t_0).(\llbracket f_1 \rrbracket_t envt) + \chi_{S^C}(t_0).(\llbracket f_2 \rrbracket_t envt)).t_0 & \text{(lemma 8.3.4)} \\
= E_+(\llbracket f_1 +_S f_2 \rrbracket_t).t_0. &
\end{array}$$

- Let $t_0 \in X$. $T^\infty(x)$ is a notation for $\left(\bigotimes_{k=1}^\infty T_k\right)(x)$. Then

$$\begin{array}{l}
E_+(\llbracket \text{shift}(f) \rrbracket_t.env).t_0 \\
= \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda \langle t_1, \ldots \rangle.(\llbracket f \rrbracket_t.env) \circ \text{shift}(\langle t_0, t_1, \ldots \rangle)\, \mathrm{d}T^\infty t_0 \\
= \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda \langle t_1, \ldots \rangle.(\llbracket f \rrbracket_t.env)(\langle t_1, \ldots \rangle)\, \mathrm{d}T^\infty t_0 \\
= \sup_{(T_n)_{n\in\mathbb{N}}} \int \lambda t_1.\int \lambda \langle t_2, \ldots \rangle.(\llbracket f \rrbracket_t.env)(\langle t_1, \ldots \rangle)\, \mathrm{d}T^\infty t_1\, \mathrm{d}T(t_0) \\
= \sup_{T_1} \sup_{(T_n)_{n\geq 2}} (\overleftarrow{T_1}(\lambda t_1.\int \lambda \langle t_2, \ldots \rangle.(\llbracket f \rrbracket_t.env)(\langle t_1, \ldots \rangle)\, \mathrm{d}T^\infty t_1))(t_0) \\
= \sup_{T_1} (\bigsqcup_{f\in R(\llbracket f \rrbracket_t.env)} (\overleftarrow{T_1}.f))(t_0).
\end{array}$$

Let us apply lemma A.2.2 to $\overleftarrow{T_1}$:

$$\bigsqcup_{f\in R(\llbracket f \rrbracket_t.env)} (T_1^* f) \leq \overleftarrow{T_1}.(\underbrace{\bigsqcup_{f\in R(\llbracket f \rrbracket_t.env)} f}_{E_+(\llbracket f \rrbracket_t.env)\leq \llbracket f \rrbracket_{e+}.H}). \tag{8.36}$$

Taking the least upper bound yields

$$E_+(\llbracket \text{shift}(f) \rrbracket_t.env).t_0 \leq \sup_{T_1} \overleftarrow{T_1}(\llbracket f \rrbracket_{e+}.H) = \llbracket \text{shift}(f) \rrbracket_{e+}.H.$$

- Let us fix *env*. $[\![\text{lfp}(name \mapsto f)]\!]_t.env = \text{lfp}\,\psi_1$ where $\psi_1(\phi_1) = [\![f]\!]_t.env[name \mapsto \phi_1]$. $[\![\text{lfp}(name \mapsto f)]\!]_{e+}.H = \text{lfp}\,\psi_2$ where $\psi_2(\phi_2) = [\![f]\!]_{e+}.H[name \mapsto \phi_2]$. By the induction hypothesis,

$$\psi_2 \circ E_+(\phi_1) = [\![f]\!]_{e+}.H[name \mapsto E_+(\phi_1) \geq [\![f]\!]_t.env[name \mapsto \phi_1] = E_+ \circ \psi_1(\phi_1).$$

  From lemma 8.1.6, E_+ is upper-continuous. From lemma 8.1.5, $E_+(\bot) = \bot$. From lemma A.2.20, $E_+(\text{lfp}\,\psi_1) \sqsubseteq \text{lfp}\,\psi_2$, thus

$$E_+([\![\text{lfp}(name \mapsto f)]\!]_t.env) \sqsubseteq [\![\text{lfp}(name \mapsto f)]\!]_{e+}.H.$$

- Let us fix *env*. $[\![\text{gfp}(name \mapsto f)]\!]_t.env = \text{gfp}\,\psi_1$ where $\psi_1(\phi_1) = [\![f]\!]_t.env[name \mapsto \phi_1]$. $[\![\text{gfp}(name \mapsto f)]\!]_{e+}.H = \text{gfp}\,\psi_2$ where $\psi_2(\phi_2) = [\![f]\!]_{e+}.H[name \mapsto \phi_2]$. By the induction hypothesis,

$$\psi_2 \circ E_+(\phi_1) = [\![f]\!]_{e+}.H[name \mapsto E_+(\phi_1) \geq [\![f]\!]_t.env[name \mapsto \phi_1] = E_+ \circ \psi_1(\phi_1).$$

  From lemma 8.1.6, E_+ is monotone. From lemma 8.1.5, $E_+(\top) = \top$. From lemma A.2.22, $E_+(\text{gfp}\,\psi_1) \sqsubseteq \text{gfp}\,\psi_2$, thus

$$E_+([\![\text{gfp}(name \mapsto f)]\!]_t.env) \sqsubseteq [\![\text{gfp}(name \mapsto f)]\!]_{e+}.H.$$

- The case for "let" is trivial. □

Remark 8.3.9. The above results hold even if we only use deterministic policies in the definition of E_+.

8.3.3 Present and past knowing adversary

Another way to consider adversaries is to allow them to take their decisions according to the full history of the computation. This actually seems a more accurate modeling of an unknown environment.

Concrete semantics

Lemma 8.3.10. *For any trace valuator f, for any g_1 and g_2 in $R(f)$, for any $A \subseteq \Omega$, the function g_3 defined by $g_3(\vec{t}) = g_1(\vec{t})$ if $t_0 \in A$, $g_3(\vec{t}) = g_2(\vec{t})$ otherwise, belongs to $R(f)$.*

Proof. Let $g_1 = S(f,(U^1_n)_{n\in\mathbb{N}})$ and $g_2 = S(f,(U^2_n)_{n\in\mathbb{N}})$. Let us define $(U^3_n)_{n\in\mathbb{N}}$ as follows: $U_3(\vec{t},W) = U_1(\vec{t},W)$ if $t_0 \in A$, $U_2(\vec{t},W)$ otherwise. Let $g_3 = S(f,(U^3_n)_{n\in\mathbb{N}})$. Obviously, if $t_0 \in A$, then $g_3(t_0) = g_1(t_0)$, otherwise $g_3(t_0) = g_2(t_0)$. □

Corollary 8.3.11. *For any x and y in $R(f)$, $\sup(x,y) \in R(f)$ (also said as: $R(f)$ is a sub-upper-lattice).*

Proof. Let $A = \{t_0 \in \Omega \mid g_1(t_0) > g_2(t_0)\}$. By the lemma, $g_3 = \sup(g_1,g_2)$. □

8.3.4 The abstraction relation between the semantics

Lemma 8.3.12. *For all f, t_0 and U_1,*

$$\sup_{(U_n)_{n\geq 2}} \int \lambda\langle t_1,\ldots\rangle.f(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\in\mathbb{N}}).t_0]$$
$$= \sup_{(U_n)_{n\geq 2} U_n \text{ does not depend on } t_0} \int \lambda\langle t_1,\ldots\rangle.f(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\in\mathbb{N}}).t_0]$$

Proof. Let a and b be respectively the left-hand and right-hand sides of the equality to be proven. $a \geq b$ is obvious since b is an upper bound on a smaller set than a. Let us now prove $b \geq a$. Let us take $t_0 \in X$ and $(U_n)_{n\geq 2}$. Let us now consider $(\tilde{U}_n)_{n\geq 2}$ defined by $\tilde{U}_n(\vec{t}';W) = U_n.(t_0,t'_1,\ldots,t'_{n-1};W)$. $\tilde{U}_n(t'_0,\ldots;W)$ does not depend on t'_0. Furthermore,

$$\int \lambda\langle t_1,\ldots\rangle.f(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\in\mathbb{N}}).t_0] = \int \lambda\langle t_1,\ldots\rangle.f(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(\tilde{U}_n)_{n\in\mathbb{N}}).t_0].$$

Thus $a(t_0) \leq b(t_0)$. □

Theorem 8.3.13. *Let f be a formula not containing* gfp. *Let env be an environment of valuators. Noting $E_+(env)$ the point-wise application of E_+ to env,*

$$[\![f]\!]_{e+}.(E_+(env)) = E_+([\![f]\!]_t.env) \tag{8.37}$$

Proof. Proof by induction on the structure of f.

- The cases for "let", *name* and *constant* are trivial.
- For $f_1 +_S f_2$, proof as in Th. 8.3.5.
- Let us first fix U_1. Let us note $T_1 = T \circ \begin{bmatrix} \mathrm{Id} \\ U_1 \end{bmatrix}$ and consider $\overleftarrow{T_1}.E_+([\![f]\!]_t)$. From lemma 8.3.3, $\overleftarrow{T_1}$ is a monotonic, ω-continuous, operator;

from lemma 8.3.10, $R([\![f]\!]_t)$ is directed;
from lemma A.2.6,

$$\bigsqcup_{f\in R([\![f]\!]_t.env)} (\overleftarrow{T_1} f) = \overleftarrow{T_1}(\bigsqcup_{f\in R([\![f]\!]_t.env)} f).$$

It follows that

$$\begin{array}{l}\overleftarrow{T_1}.E_+([\![f]\!]_t).t_0 = \\ \sup_{\substack{(U_n)_{n\geq 2}\\ U_n \text{ not depending on } t_0}} \int\int \lambda\langle t_2,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\geq 2}).t_1]\,T_1(t_0,\mathrm{d}t_1) \\ = \sup_{\substack{(U_n)_{n\geq 2}\\ U_n \text{ not depending on } t_0}} \int \lambda\langle t_2,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\geq 1}).t_0]\end{array}$$

Let us apply lemma 8.3.12 to that last expression. We obtain

$$\overleftarrow{T_1}.E_+([\![f]\!]_t).t_0 = \sup_{(U_n)_{n\geq 2}} \int \lambda\langle t_2,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\geq 1}).t_0] \tag{8.38}$$

Let $t_0 \in X$. Let us now consider all U_1's.

$$\begin{array}{l}E_+([\![\mathrm{shift}(f)]\!]_t.env).t_0 \\ = \sup_{(U_n)_{n\geq 1}} \int \lambda\langle t_1,\ldots\rangle.([\![f]\!]_t.env)\circ\mathrm{shift}(\langle t_0,t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\geq 1}).t_0] \\ = \sup_{U_1}\sup_{(U_n)_{ngeq2}} \int \lambda\langle t_1,\ldots\rangle.([\![f]\!]_t.env)(\langle t_1,\ldots\rangle)\,\mathrm{d}[G(T,(U_n)_{n\geq 1}).t_0] \\ = \left(\sup_{U_1} \overleftarrow{\left(T\circ\begin{bmatrix}\mathrm{Id}\\ U_1\end{bmatrix}\right)}.E_+([\![f]\!]_t)\right).t_0 \text{ (using equ. 8.38)} \\ = [\![\mathrm{shift}(f)]\!]_{e+}.E_+(env)\end{array}$$

- The proof for lfp is the same as the one in corollary 8.3.6. □

□

Theorem 8.3.14. *The semantics of the summing operator satisfy:*

$$E_+([\![\Sigma f]\!]_t) = [\![\Sigma f]\!]_{e+}. \tag{8.39}$$

Proof. Similar as the proof of the preceding theorem. □

Theorem 8.3.15. *Let f be a formula. Let env be an environment of valuators. Let us suppose that $H \geq E_+(env)$ pointwise. Then*

$$[\![f]\!]_{e+}.(H) \geq E_+([\![f]\!]_t.env). \tag{8.40}$$

Proof. Similar as Th. 8.3.8. □

Remark 8.3.16. The above results hold even if we only use deterministic policies in the definition of E_+.

This is quite a noticeable fact. In other, more complex, frameworks, such as two-player games, this is no longer true. Let us analyze the well-known rock–paper–scissors game [1] from the point of view of the first player: he will try to find a strategy that maximizes the minimal gain (-1 for a loss, 1 for a win) he may achieve for all strategies of the adversary (*minimax* point of view, very much used in the programming of computer players for games such as chess). It is quite obvious that any deterministic strategy is bad in that respect, since it is possible to find a corresponding deterministic strategy of the adversary that makes the player lose each time, thus the result of the minimax on the deterministic strategies is -1. On the other hand, a stationary policy taking one of the three possibilities at random with equal probability yields a minimax result of 0.

Remark 8.3.17. For reachability properties (one single fixpoint, a least fixpoint), memoryless and memory-aware adversaries are equivalent.

Let us remark that such equivalences between classes of policies are not always evident. For instance, stationary policies are *not* equivalent to non-stationary policies for certain properties. Let consider an example involving a greatest fixpoint:

```
while (random() < ndt_[0,1[) { };
```

`ndt_[0,1[()` nondeterministically returns a real number x in $[0,1[$ (we could also choose a positive integer n and take $x = 1 - 1/n$). We are interested in the liveness property of staying indefinitely in the loop. The choice of a stationary policy is the choice of x, and for any fixed value of x the probability of staying indefinitely in the loop is 0 (the probability of staying at least n times is x^n). Let us consider, on the other hand, the memory-aware policies consisting in the successive choices $x_1, x_2, \ldots$. The probability of staying in the loop indefinitely is

$$\prod_{k=1}^{\infty} x_k = \exp\left(-\sum_{k=1}^{\infty} \log(1/x_k)\right) \tag{8.41}$$

For any positive real number α, there exists a sequence y_n of positive real numbers such that $\alpha = \sum_k y_n$. Let us take $\Lambda \in]0,1[$, $(y_n)_{n\in\mathbb{N}} \in]0,+\infty[^{\mathbb{N}}$ such that $\sum_n y_n = -\log(\Lambda)$, $x_n = \exp(-y_n)$; then $\prod_{k=1}^{\infty} x_k = \Lambda$. Thus the least upper bound of the probabilities of staying indefinitely in the loop for all memory-aware policies is 1.

[1] This game is played by two people in rounds. In each round, each player makes a gesture, either "rock", "paper" or "scissors"; both gestures are made simultaneously. "Rock" beats "scissors", "paper" beats "rock" and "scissors" beats "paper".

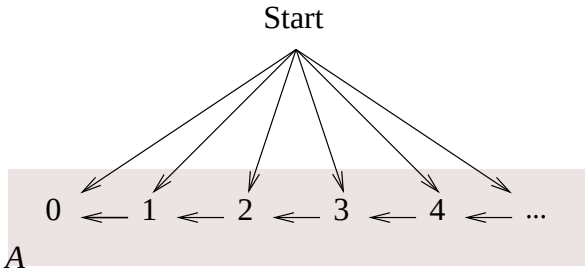


Figure 8.2: A transition system for which $E_+(\llbracket \mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0)\rrbracket_t) < \llbracket \mathrm{gfp}(f \mapsto \mathrm{shift}\, f +_A 0)\rrbracket_{e+}$. While for any n there exists a trace that goes through A for n steps, there does not exist any trace going infinitely through A. On this system, greatest fixpoint analysis is inherently imprecise.

Nondeterminism and greatest fixpoints

Why is that that we seem to lose precision if the formula contains greatest fixpoints? We see that it is an essential problem with nondeterminism and infinite computations.

No nondeterminism In this case, we have absolutely no loss of precision. An essential fact is that E_+ is not only upper-continuous, but fully continuous.

Theorem 8.3.18. *Let us consider a system without non-determinism. Let f be a formula. Let env be an environment of valuators. Noting $E_+(env)$ the point-wise application of E_+ to env,*

$$\llbracket f \rrbracket_{e+}.(E_+(env)) = E_+(\llbracket f \rrbracket_t.env). \tag{8.42}$$

Proof. Proof similar to the proof of Th. 8.3.13, except that we use lemma A.2.18 and the fact that E_+ is continuous to treat the case of gfp. □

Nondeterminism We have seen that, when nondeterminism is added, E_+ is still upper-continuous but not lower-continuous. We have not been able to prove that $E_+(\llbracket \mathrm{gfp}\,\Phi\rrbracket_t).env = \llbracket \mathrm{gfp}\,\Phi\rrbracket_{e+}.E_+(env)$, but only the inequality $E_+(\llbracket \mathrm{gfp}\,\Phi\rrbracket_t).env \leq \llbracket \mathrm{gfp}\,\Phi\rrbracket_{e+}.E_+(env)$. Unfortunately, the equality is false, and we shall give here a counterexample. This problem actually arises as soon as we have nondeterminism, not even needing probabilities, and our counterexample does not use probabilities. Here it is, written in pseudo-C:

```
int n=nondeterministic_nonnegative_integer_choice();
int k;
for(k=0; k<n; j++) /* A */;
```

We actually rather use a symbolic transition system (Fig. 8.2). Let us now consider the formula $\text{shift}(\text{gfp}(f \mapsto \text{shift}(f) +_A 0))$ and look at the iterations for gfp, both for $[\![\phi]\!]_t$ and $[\![\phi]\!]_{e+}$. $[\![\phi]\!]_t$ is the set of traces passing through A an infinite amount of time. Obviously, such traces are not possible, thus $E_+([\![\phi]\!]_t) = 0$. On the other hand, $[\![\phi]\!]_{e+}$ is the function $(A,n) \mapsto 1$. Of course, since $1 > 0$, this is a coarse approximation.

An intuitive vision of this phenomenon is that it is possible for a system to have traces of any integer length, but no trace of infinite length. This problem is inherent in the approach of using $[\![\text{gfp } f]\!]_{e+}$ and is one of the reasons that plead for the development of domains especially suitable for liveness analyses.

8.3.5 Abstract analysis

We shall see here more precisely how to apply abstract interpretation to that backwards semantics.

General case

We compute safe approximations of $[\![f]\!]_{e+}$ by abstract interpretation. We introduce an abstract semantics $[\![f]\!]^\sharp_{e+}$ as an upper approximation of f:

$$\forall env,\ \forall env^\sharp,\ env^\sharp \geq env \implies [\![f]\!]^\sharp_{e+}.env^\sharp \geq [\![f]\!]_{e+}.env. \tag{8.43}$$

The computations for $[\![f]\!]^\sharp_{e+}$ will be done symbolically in an *abstract domain* such as those described in § 9.2 and §12.

- We shall assume that we have an abstract operation for "shift". That is, we have a monotone operation $\text{pre}^\sharp$ such that

  $$\forall f,\ \forall T \in \mathscr{T},\ \text{pre}^\sharp.f^\sharp \geq T^*.f^\sharp. \tag{8.44}$$

  This operation will be supplied by the abstract domain that we use. Then

  $$\forall env,\ \forall env^\sharp,\ env^\sharp \geq env \implies [\![\text{shift}(f)]\!]^\sharp_{e+}.env^\sharp \geq [\![\text{shift}(f)]\!]_{e+}.env. \tag{8.45}$$

  provided that

  $$\forall env,\ \forall env^\sharp,\ env^\sharp \geq env \implies [\![f]\!]^\sharp_{e+}.env^\sharp \geq [\![f]\!]_{e+}.env.$$

  .

- We shall approximate least fixpoints using a *widening operator* [18, §4.3]. A widening operator ∇ is a kind of abstraction of the least upper bound that enforces convergence:

  - $f \nabla g \geq \sup(f,g)$ (pointwise);
  - For any ascending sequence $(v_n)_{n\in\mathbb{N}}$, the sequence $(u_n)_{n\in\mathbb{N}}$ defined inductively by $u_{n+1} = u_n \nabla v_n$ is to be ultimately stationary.

  Then the limit $L^\sharp$ of the sequence defined by $u_0 = 0$ and $u_{n+1} = u_n \nabla f^\sharp(u_n)$, where $f^\sharp$ is an upper approximation of f, is an upper approximation to the least fixpoint of f. More precise upper approximations of the least fixpoint of f can then be reached by iterating $f^\sharp$ over $L^\sharp$. More precise upper approximations of the least fixpoint of f can then be reached by iterating $f^\sharp$ over $L^\sharp$ using a so-called *narrowing operators* [18, §4.3].

- We shall approximate greatest fixpoints using a limited iteration approach: if $f^\sharp$ is an upper approximation of f, then for any $n \in \mathbb{N}$, ${f^\sharp}^n(\top) \geq \text{gfp}\, f$. Here again, we could use *narrowing operators* [18, §4.3].

Partitioning in programs

In the case of programs, the state space is generally $P \times M$, where P is the (finite) set of program points and M the set of possible memory configurations. More generally, P may be a kind of finite *partitioning* of the program — for instance, according to the call context of the procedures. Non-probabilistic analysis generally operates on abstractions of $\mathcal{P}(P \times M) \simeq P \times M \to \{0,1\} \simeq P \to \mathcal{P}(M)$. Given an abstraction of $\mathcal{P}(M)$ by a lattice $L^\sharp$, one obtains a pointwise abstraction of $P \to \mathcal{P}(M)$ by $P \to L^\sharp$. Elements of $P \to L^\sharp$ are just vectors of $|P|$ elements of $L^\sharp$.

This approach can be directly extended to our measurable functions: we shall abstract $P \times M \to I$ (where $I = [0,1]$ or $I = [0,+\infty]$) by $P \to L^\sharp$ if $L^\sharp$ is an abstract domain for $M \to I$.

The first problem is to get an abstraction of the operation used in the "shift" construct:

$$F : \left| \begin{array}{l} P \times M \to I \to P \times M \to I \\ h \qquad\qquad \mapsto (l,m) \mapsto \sup_{y\in Y} \sum_{(l',m')\in P\times M} T\left((l,m),y;(l',m')\right).h(l',m') \end{array} \right. \tag{8.46}$$

Let us take the following form for the program instructions: at program point l, the executed instruction represented by T is the sequence:

1. a nondeterministic choice y is taken in the set Y;
2. a random choice r is taken in set R_l according to distribution $\mathcal{R}_p$;
3. the memory state is combined deterministically with the two choices to form the new memory state using a function $F_l : (M \times Y) \times R_p \to M$;

4. depending on the memory state m, the program takes a deterministic jump to program point $J(l,m)$.

Let us note $\tau_l(l') = \{m \mid J(l,m) = l'\}$ (the set of memory values m that lead to program point l' from program point l; $\tau_l(l')$ is then essentially the condition for a conditional jump). Then we can rewrite the transition equation as follows

$$F(h) = \text{choice}^*_Y \circ \text{random}^*_{R_l} \circ F^*_{l\,p}\left(\sum_{l' \in P} \phi^*_{\tau_l l'}\left(h(l', \bullet)\right)\right) \tag{8.47}$$

using the following building blocks:

$$\text{choice}^*_Y(h) = m \mapsto \sup_{y \in Y} h(m,y) \tag{8.48}$$

$$\text{random}^*_{R_l}(h) = m \mapsto \int h(m,r)\ \mathrm{d}\mu_{\mathcal{R}_l}(r) \tag{8.49}$$

$$F^*_{l\,p}(h) = h \circ F_l \tag{8.50}$$

$$\phi^*_A(h) = h.\chi_A \tag{8.51}$$

The reasons for those notations are explained in §13.3.4.

We shall abstract F as the composition of abstractions for:

- choice^*_Y, nondeterministic choice;
- $\text{random}^*_{R_l}$, probabilistic choice;
- $F^*_{l\,p}$, deterministic run (arithmetic operations and the like);
- ϕ^*_A, tests.

§9.2 will give an abstract domain featuring those operations; Ch. 12 will give another abstract domain where the tests are approximated by the identity function.

Since the function F is ω-upper-continuous, the least fixpoint of F is obtained as the limit of $F^n(0)$ (let us recall that this is the point-wise limit of a sequence of functions from X to I). The expression of the iterates using a partition with respect to P is as follows:

$$\begin{array}{ll} f_1^{(n+1)} & = F_1(f_1^{(n)}, \ldots, f_{|P|}^{(n)}) \\ \vdots & \vdots \\ f_{|P|}^{(n+1)} & = F_{|P|}(f_1^{(n)}, \ldots, f_{|P|}^{(n)}) \end{array} \tag{8.52}$$

In terms of implementation, this means that we update in parallel the $|P|$ elements of the vector representing the iterate. As noted by Cousot [16, §2.9], this parallel

update may be replaced by *chaotic iterations* or *asynchronous iterations*. Chaotic iterations allow us to compute the iterations by taking into account the recently updated elements. All these iteration strategies lead to the same limit (the least fixpoint of F).

Let us consider for instance the following strategy:

$$\begin{array}{ll} f_1^{(n+1)} & = F_1(f_1^{(n)}, \ldots, f_{|P|}^{(n)}) \\ f_2^{(n+1)} & = F_2(f_1^{(n+1)}, \ldots, f_{|P|}^{(n)}) \\ \vdots & \vdots \\ f_{|P|}^{(n+1)} & = F_{|P|}(f_1^{(n+1)}, \ldots, f_{|P|}^{(n)}) \end{array} \tag{8.53}$$

This strategy is itself a monotonic operator whose least fixpoint is to be determined. It has an obvious abstract counterpart yielding an approximate fixpoint in the usual way (§8.3.5).

8.4 Forward analysis

In this section, we only take care of closed formulas of the following form:

- they do not contain any "let" binding; if necessary, inline the bindings;
- the bindings used in lfp and gfp do not traverse any lfp or gfp 4 construct.

Lemma 8.4.1. *For any formula* $\mathrm{lfp}(name_R \mapsto f)$ *of the above form,* $\phi \mapsto [\![f]\!]_t.env[name_R \mapsto \phi]$ *is continuous.*

8.4.1 Absence of nondeterminism

We define a semantics showing the forward progression of the distribution of values inside the program.

$$[\![name]\!]_{Fwd(name_R)} = \mu \mapsto \mu \text{ if } name = name_R \tag{8.54}$$

$$[\![constant]\!]_{Fwd(name_R)} = 0 \tag{8.55}$$

$$[\![f_1 +_S f_2]\!]_{Fwd(name_R)} = [\![f_1]\!]_{Fwd(name_R)} \text{ if } name_R \in f_1 \tag{8.56}$$

$$[\![f_1 +_S f_2]\!]_{Fwd(name_R)} = [\![f_2]\!]_{Fwd(name_R)} \text{ if } name_R \notin f_1 \tag{8.57}$$

$$[\![\mathrm{shift}(f)]\!]_{Fwd(name_R)} = [\![f]\!]_{Fwd(name_R)} \circ \overrightarrow{T} \tag{8.58}$$

$$[\![\mathrm{lfp}(\phi \mapsto f)]\!]_{Fwd(name_R)} = 0 \tag{8.59}$$

$$[\![\mathrm{gfp}(\phi \mapsto f)]\!]_{Fwd(name_R)} = 0 \tag{8.60}$$

$$[\![name]\!]_{Out} = 0 \tag{8.61}$$

$$[\![constant]\!]_{Out}.\mu = \int constant \, \mathrm{d}\mu \tag{8.62}$$

$$[\![f_1 +_S f_2]\!]_{Out}.\mu = [\![f_1]\!]_{Out} \circ \phi_S.\mu + [\![f_2]\!]_{Out} \circ \phi_{S^C}.\mu \tag{8.63}$$

$$[\![\mathrm{shift}(f)]\!]_{Out}.\mu = [\![f]\!]_{Out}.(\overrightarrow{T}.\mu) \tag{8.64}$$

$$[\![\mathrm{lfp}(name_R \mapsto f)]\!]_{Out}.\mu = \sum_{k=0}^{\infty} [\![f]\!]_{Out} \circ [\![f]\!]^k_{Fwd(name_R)}.\mu \tag{8.65}$$

$$[\![\mathrm{gfp}(name_R \mapsto f)]\!]_{Out}.\mu = \sum_{k=0}^{\infty} [\![f]\!]_{Out} \circ [\![f]\!]^k_{Fwd(name_R)}.\mu \tag{8.66}$$

$$+ \lim\downarrow_{n\to\infty} \int \mathrm{d}\left(([\![f]\!]_{Fwd(name_R)})^n.\mu\right) \tag{8.67}$$

Lemma 8.4.2. *For any f and μ, $\int \mathrm{d}\left([\![f]\!]_{Fwd(name_R)}.\mu\right) \leq \int \mathrm{d}\mu$.*

Proof. By induction on the structure of f. All cases are trivial. □

Lemma 8.4.3. *If for all g and μ,*

$$\int \psi.g \, \mathrm{d}(R.\mu) = H_1.\mu + \int g \, \mathrm{d}(R \circ H_2.\mu) \tag{8.68}$$

then for all n,

$$\int \psi^n.g \, \mathrm{d}(R.\mu) = \sum_{k=0}^{n-1} H_1 \circ H_2^k.\mu + \int g \, \mathrm{d}(R \circ H_2^n.\mu). \tag{8.69}$$

Proof. By recurrence on $n \geq 1$. Case $n = 1$ is in the hypotheses. We shall now suppose that the property holds for rank n and prove that it holds for rank $n+1$.

$\int \psi^{n+1}.g \, \mathrm{d}(R.\mu) = \int \psi^n(\psi.g) \, \mathrm{d}(R.\mu)$
$= \sum_{k=0}^{n-1} H_1 \circ H_2^k(\mu) + \int (\psi.g) \, \mathrm{d}(R \circ H_2^n.\mu)$ (induction hypothesis)
$= \sum_{k=0}^{n-1} H_1 \circ H_2^k(\mu) + + H_1(H_2^n.\mu) + \int g \, \mathrm{d}\left(R \circ H_2.(H_2^n.\mu)\right)$ (hypothesis)
$= \sum_{k=0}^{n} H_1 \circ H_2^k(\mu) + \int g \, \mathrm{d}(R \circ H_2^{n+1}.\mu)$ □

Theorem 8.4.4. *For any formula f satisfying the conditions laid at the beginning of §8.4,*

$$\int ([\![f]\!]_t.env) \, \mathrm{d}(T^{\otimes\mathbb{N}}.\mu) = [\![f]\!]_{Out}.\mu. \tag{8.70}$$

Proof. Proof by induction on the structure of f.

- $\int (\llbracket constant \rrbracket_t .env)\ \mathrm{d}(T^{\otimes\mathbb{N}}.\mu) = \int constant\ \mathrm{d}\mu = \llbracket constant \rrbracket_{Out}.\mu$;
- The case of $+_S$ is obtained by splitting μ into $\mu_{|S}$ and $\mu_{|S^C}$:

$$\begin{array}{l}\int (\llbracket f_1 +_S f_2 \rrbracket_t .env)\ \mathrm{d}(T^{\otimes\mathbb{N}}.\mu)\\ = \int \llbracket f_1 \rrbracket_t .env\ \mathrm{d}(T^{\otimes\mathbb{N}}.\mu_{|S} + \int \llbracket f_2 \rrbracket_t .env\ \mathrm{d}(T^{\otimes\mathbb{N}}.\mu_{|S^C}\\ = \llbracket f_1 \rrbracket_{Out}.\mu_{|S} + \llbracket f_2 \rrbracket_{Out}.\mu_{|S^C} = \llbracket f_1 +_S f_2 \rrbracket_{Out}.\mu.\end{array}$$

- Using the induction hypothesis:

$$\int (\llbracket \mathrm{shift}(f) \rrbracket_t .env)\ \mathrm{d}(T^{\otimes\mathbb{N}}.\mu) = \int (\llbracket f \rrbracket_t .env)\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.(\overrightarrow{T}.\mu)\right) = \llbracket f \rrbracket_{Out}.(\overrightarrow{T}.\mu);$$

- From lemmas 8.4.1 and A.2.11, the following least fixpoint is written as a countable least upper bound:

$$\begin{array}{l}\int \llbracket \mathrm{lfp}(name_R \mapsto f).env \rrbracket_t\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \int \sup_n (\phi \mapsto \llbracket f \rrbracket_t .env[name_R \mapsto \phi])^n(0)\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \lim\uparrow_{n\to\infty} \int (\phi \mapsto \llbracket f \rrbracket_t .env[name_R \mapsto \phi])^n(0)\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\end{array}$$

  Let us now apply lemma 8.4.3 to $H_1 = \llbracket f \rrbracket_{Out}$, $H_2 = \llbracket f \rrbracket_{Fwd(name_R)}$, $g = 0$ and $R : \mu \mapsto T^{\otimes\mathbb{N}}.\mu$:

$$\begin{array}{l}\int \llbracket \mathrm{lfp}(name_R \mapsto f).env \rrbracket_t\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \sup_n \left[\sum_{k=0}^{n-1} \llbracket f \rrbracket_{Out} \circ \llbracket f \rrbracket^k_{Fwd(name_R)}.\mu + \int 0\ \mathrm{d}\left(\llbracket f \rrbracket^n_{Fwd(name_R)}.\mu\right)\right]\\ = \llbracket \mathrm{lfp}(name_R \mapsto f).env \rrbracket_{Out}.\mu\end{array}$$

- From lemmas 8.4.1 and A.2.12, the following least fixpoint is written as a countable greatest lower bound:

$$\begin{array}{l}\int \llbracket \mathrm{gfp}(name_R \mapsto f).env \rrbracket_t\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \int \inf_n (\phi \mapsto \llbracket f \rrbracket_t .env[name_R \mapsto \phi])^n(1)\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \inf_n \int (\phi \mapsto \llbracket f \rrbracket_t .env[name_R \mapsto \phi])^n(1)\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\end{array}$$

  Let us now apply lemma 8.4.3 to $H_1 = \llbracket f \rrbracket_{Out}$, $H_2 = \llbracket f \rrbracket_{Fwd(name_R)}$, $g = 1$ and $R : \mu \mapsto T^{\otimes\mathbb{N}}.\mu$:

$$\begin{array}{l}\int \llbracket \mathrm{lfp}(name_R \mapsto f).env \rrbracket_t\ \mathrm{d}\left(T^{\otimes\mathbb{N}}.\mu\right)\\ = \lim_{n\to\infty} \left[\sum_{k=0}^{n-1} \llbracket f \rrbracket_{Out} \circ \llbracket f \rrbracket^k_{Fwd(name_R)}.\mu + \int 1\ \mathrm{d}\left(\llbracket f \rrbracket^n_{Fwd(name_R)}.\mu\right)\right]\\ = \sum_{k=0}^{\infty} \llbracket f \rrbracket_{Out} \circ \llbracket f \rrbracket^k_{Fwd(name_R)}.\mu + \lim\downarrow_{n\to\infty} \int\ \mathrm{d}\left(\llbracket f \rrbracket^n_{Fwd(name_R)}.\mu\right)\end{array}$$

From lemma 8.4.2, the sequence $\left(\int \mathrm{d}\left(\llbracket f \rrbracket^n_{Fwd(name_R)}.\mu\right)\right)_{n\in\mathbb{N}}$ is decreasing.

□

8.4.2 Abstraction, with nondeterminism

We now go to the general case, directly introducing the abstract semantics.

$$\llbracket name \rrbracket^\sharp_{Fwd(name_R)} = \mu^\sharp \mapsto \mu^\sharp \text{ if } name = name_R \tag{8.71}$$

$$\llbracket constant \rrbracket^\sharp_{Fwd(name_R)} = \{0\} \tag{8.72}$$

$$\llbracket f_1 +_S f_2 \rrbracket^\sharp_{Fwd(name_R)} = \llbracket f_1 \rrbracket^\sharp_{Fwd(name_R)} \quad \text{if } name_R \in f_1 \tag{8.73}$$

$$\llbracket f_1 +_S f_2 \rrbracket^\sharp_{Fwd(name_R)} = \llbracket f_2 \rrbracket^\sharp_{Fwd(name_R)} \quad \text{if } name_R \notin f_1 \tag{8.74}$$

$$\llbracket \mathrm{shift}(f) \rrbracket^\sharp_{Fwd(name_R)} = \llbracket f \rrbracket^\sharp_{Fwd(name_R)} \circ T^\sharp \tag{8.75}$$

$$\llbracket \mathrm{lfp}(\phi \mapsto f) \rrbracket^\sharp_{Fwd(name_R)} = 0 \tag{8.76}$$

$$\llbracket \mathrm{gfp}(\phi \mapsto f) \rrbracket^\sharp_{Fwd(name_R)} = 0 \tag{8.77}$$

$$\tag{8.78}$$

$$\llbracket name \rrbracket^\sharp_{Out} = \{0\} \tag{8.79}$$

$$\llbracket constant \rrbracket^\sharp_{Out}.\mu^\sharp = I^\sharp(constant, \mu^\sharp) \tag{8.80}$$

$$\begin{aligned} \llbracket f_1 +_S f_2 \rrbracket^\sharp_{Out}.\mu^\sharp = \; & \llbracket f_1 \rrbracket^\sharp_{Out}.\{\phi_S.\mu | \mu \in \mu^\sharp\} + \\ & \llbracket f_2 \rrbracket^\sharp_{Out}.\{\phi_{S^C}.\mu | \mu \in \mu^\sharp\} \end{aligned} \tag{8.81}$$

$$\llbracket \mathrm{shift}(f) \rrbracket^\sharp_{Out}.\mu^\sharp = \llbracket f \rrbracket^\sharp_{Out} \circ \mathscr{T}^\sharp(\mu^\sharp) \tag{8.82}$$

$$\llbracket \mathrm{lfp}(name_R \mapsto f) \rrbracket^\sharp_{Out}.\mu^\sharp = S^\sharp\left(\mu^\sharp \mapsto \llbracket f \rrbracket^\sharp_{Out} \circ \llbracket f \rrbracket^\sharp_{Fwd(name_R)}.\mu^\sharp\right) \tag{8.83}$$

$$\begin{aligned} \llbracket \mathrm{gfp}(name_R \mapsto f) \rrbracket^\sharp_{Out}.\mu^\sharp = \; & down\left(\llbracket f \rrbracket^\sharp_{Out} \circ (\nu^\sharp \mapsto \mu^\sharp +^\sharp \llbracket f \rrbracket^\sharp_{Fwd(name_R)}(\nu^\sharp))^{N_0(f,\mu^\sharp)}\right. \\ & \left. + I^\sharp(\llbracket f \rrbracket_{Fwd(name_R)})^{N_0(f,\mu^\sharp)}, \mu^\sharp\right) \end{aligned} \tag{8.84}$$

In this last definition, $N_0(f,\mu^\sharp)$ is any integer, determined at run-time by the analyzer. The results exposed here are proved correct for any value of $N_0(f,\mu^\sharp)$, but of course the upper-approximation can be more or less precise depending on this choice (normally, the larger, the better, but also the larger, the lengthier the computation). $down(X)$ is the downwards closure of X: $\{y \in \mathbb{R}_+ \mid \exists x \in X \; y \leq x\}$.

Lemma 8.4.5. *Let us suppose that:*

$$\forall a^\sharp,\ 0^\sharp +^\sharp a^\sharp = a^\sharp \tag{8.85}$$

Then for all $n \in \mathbb{N}$, for all $\mu^\sharp$, $o^\sharp$

$$\mu^\sharp +^\sharp \left(\nu^\sharp \mapsto H_2^\sharp(\mu^\sharp) +^\sharp H_2^\sharp(\mu^\sharp)\right)^{n-1}(o^\sharp) = \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\mu^\sharp)\right)^{n}(o^\sharp) \tag{8.86}$$

Proof. Proof by recurrence. The case $n = 0$ follows from Equ. 8.85. The other cases are proven easily by recurrence. □

Lemma 8.4.6. *Let us suppose that:*

$$\forall a^\sharp,\ \forall b^\sharp,\ H_1^\sharp(a^\sharp) + H_1^\sharp(b^\sharp) \subseteq H_1^\sharp(a +^\sharp b) \tag{8.87}$$

$$\forall a^\sharp,\ 0^\sharp +^\sharp a^\sharp = a^\sharp \tag{8.88}$$

$$\forall g,\ \forall \mu^\sharp,\ I^\sharp(\psi.g, \mu^\sharp) \subseteq H_1^\sharp(\mu^\sharp) + I^\sharp(g, H_2^\sharp(\mu^\sharp)) \tag{8.89}$$

Then for all n,

$$I^\sharp(\psi^n.g, \mu^\sharp) \subseteq H_1^\sharp \circ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\nu^\sharp)\right)^{n-1}(0^\sharp) + I^\sharp\left(g, H_2^{\sharp n}(\mu^\sharp)\right). \tag{8.90}$$

Proof. The case for $n = 1$ is trivial. Let us prove cases for $n > 1$ by recurrence. Let us apply the recurrence hypothesis for n:

$$\begin{aligned}
&I^\sharp(\psi^{n+1}.g, \mu^\sharp) \subseteq H_1^\sharp \circ \left(\nu^\sharp \mapsto H_2^\sharp(\mu^\sharp) +^\sharp H_2^\sharp(\nu^\sharp)\right)^{n-1}(0^\sharp) + I^\sharp\left(g, H_2^{\sharp n}(\mu^\sharp)\right)\\
&\subseteq H_1^\sharp \circ \left(\nu^\sharp \mapsto H_2^\sharp(\mu^\sharp) +^\sharp H_2^\sharp(\nu^\sharp)\right)^{n-1}(0^\sharp) + H_1^\sharp(\mu^\sharp) + I^\sharp\left(g, H_2^{\sharp n+1}(\mu^\sharp)\right)\\
&\subseteq H_1^\sharp\left(\mu^\sharp +^\sharp \left(\nu^\sharp \mapsto H_2^\sharp(\mu^\sharp) +^\sharp H_2^\sharp(\mu^\sharp)\right)^{n-1}(0^\sharp)\right) + I^\sharp\left(g, H_2^{\sharp n+1}(\mu^\sharp)\right) \text{ (Equ. 8.87)}\\
&\subseteq H_1^\sharp\left(\nu^\sharp \mapsto H_2^\sharp(\mu^\sharp) +^\sharp H_2^\sharp(\mu^\sharp)\right)^{n}(0^\sharp) + I^\sharp\left(g, H_2^{\sharp n+1}(\mu^\sharp)\right) \text{ (lemma 8.4.5)}\\
&\sqsubseteq H_1^\sharp \circ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\mu^\sharp)\right)^{n}(0^\sharp) + I^\sharp\left(g, H_2^{\sharp n+1}(\mu^\sharp)\right) \text{ (Equ. 8.4.6).} \quad \square
\end{aligned}$$

Lemma 8.4.7. *Let us suppose the hypotheses of lemma 8.4.6, and additionally the following on $K^\sharp$:*

$$\forall \mu^\sharp,\ I^\sharp(0, \mu^\sharp) = 0 \tag{8.91}$$

$$K^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(K^\sharp) \sqsubseteq K^\sharp \tag{8.92}$$

$$\exists N_0\ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\nu^\sharp)\right)^{N_0}(0^\sharp) \sqsubseteq K^\sharp \tag{8.93}$$

We also suppose the following properties for $H_1^\sharp$:

- $H_1^\sharp$ *is monotonic;*
- *all the elements of the image set of* $H_1^\sharp$ *are topologically closed subsets of the real field with the usual topology;*

We also suppose the following properties for I *and* $I^\sharp$*:*

- $I^\sharp$ *is an abstraction of* I*: if* $\mu \in \gamma(\mu^\sharp)$*, then for all* g $I(g,\mu) \in I^\sharp(g,\mu^\sharp)$*;*
- *for any ascending sequence* $(g_n)_{n\in\mathbb{N}}$, $\lim_{n\to\infty} I(g_n,\mu) = I(\lim_{n\to\infty} g_n,\mu)$.

We then have, for all $\mu \in \gamma(\mu^\sharp)$*:*

$$\lim\nolimits\uparrow_{n\to\infty} \int \psi^n.0 \,\mathrm{d}\mu \in H_1^\sharp(K^\sharp) \tag{8.94}$$

Proof. Let us first apply lemma 8.4.6 for $g = 0$. For all n,

$$I^\sharp(\psi^{n+1}.0,\mu^\sharp) \subseteq H_1^\sharp \circ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\mu^\sharp)\right)^n (0^\sharp) \tag{8.95}$$

For all $n \geq N_0$,

$$H_1^\sharp \circ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\mu^\sharp)\right)^n (0^\sharp) \subseteq H_1^\sharp(K^\sharp) \tag{8.96}$$

Let $\mu \in \gamma(\mu^\sharp)$. Let $n \in \mathbb{N}$. $I(\psi^{n+1}.0,\mu) \in I^\sharp(\psi^{n+1}.0,\mu^\sharp)$ and thus $\int \psi^{n+1}.0 \,\mathrm{d}\mu \in H_1^\sharp(K^\sharp)$. Since $H_1^\sharp(K^\sharp)$ is topologically closed, $\lim\nolimits\uparrow_{n\to\infty} I(\psi^{n+1}.0,\mu) \in H_1^\sharp(K^\sharp)$. □

Theorem 8.4.8. *For all formula* f*, measure* μ *and nondeterministic policy* $(U_n)_{n\in\mathbb{N}}$*,*

$$\int [\![f]\!]_t \,\mathrm{d}\,(G(T,(U_n)).\mu) \in \gamma(\left[\!\left[\mu^\sharp\right]\!\right]^\sharp_{Out}). \tag{8.97}$$

Proof. Proof by induction on the structure of f.

- f cannot be a *name*, since f is supposed to be closed.
- $+_S$ is handled through simple decomposition, using the abstraction properties of $\phi_S^\sharp$, $\phi_{S^C}^\sharp$ and $+^\sharp$:

$$\begin{aligned}
&\int [\![f_1 +_S f_2]\!]_t .env \,\mathrm{d}\,(G(T,(U_n)).\mu) \\
&= \underbrace{\int [\![f_1]\!]_t .env \,\mathrm{d}\left(G(T,(U_n)).(\phi_S.\mu)\right)}_{\in\gamma([\![f_1]\!]^\sharp_{Out}\circ\phi_S^\sharp(\mu^\sharp))} + \underbrace{\int [\![f_2]\!]_t .env \,\mathrm{d}\left(G(T,(U_n)).(\phi_{S^C}.\mu)\right)}_{\in\gamma([\![f_2]\!]^\sharp_{Out}\circ\phi_{S^C}^\sharp(\mu^\sharp))} \\
&\in \gamma([\![f_1 +_S f_2]\!]_{Out}(mu^\sharp))
\end{aligned}$$

- The case for "shift" follows from the abstraction property of $\mathscr{T}^\sharp$:

$$\begin{array}{l}\int [\![\text{shift}(f)]\!]_t.env \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int ([\![f]\!]_t.env)\circ \text{shift}\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int ([\![f]\!]_t.env)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 1}).\underbrace{\left(\left[\begin{smallmatrix}1\\U_0\end{smallmatrix}\right]\mu\right)}_{\in\gamma(\mathscr{T}^\sharp(\mu^\sharp))}\right)\\ \in \gamma([\![f]\!]^\sharp_{Out}(\mu^\sharp)).\end{array}$$

- From lemmas 8.4.1 and A.2.11, the following least fixpoint is written as a countable least upper bound:

$$\begin{array}{l}\int [\![\text{lfp}(name\mapsto f)]\!]_t.env \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int \text{lfp}(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int \lim\uparrow_{n\to\infty}(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])^n(0)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \lim\uparrow_{n\to\infty}\int(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])^n(0)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\end{array}$$

  We wish to apply lemma 8.4.7, with $H_1^\sharp = [\![f]\!]^\sharp_{Out}$ and $H_2^\sharp = [\![f]\!]^\sharp_{Fwd(name)}$. To get the required fixpoint $K^\sharp$, we proceed as follows: we take any N_0 (the larger, the better; but the larger, the lengthier the computation) and compute $K_0^\sharp = \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp H_2^\sharp(\nu^\sharp)\right)^{N_0}(0^\sharp)$. Then we compute an approximate least fixpoint of $abstr\,\nu \mapsto \mu^\sharp +^\sharp H_2^\sharp(\nu^\sharp)$ above $K^\sharp_{N_0}$. We then apply the lemma with $\psi = \phi \mapsto [\![f]\!]_t.env[name\mapsto\phi]$, $I(g,\mu) = \int g\ \mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)$, $I^\sharp(g,\mu^\sharp) = \{I(g,\mu)\mid \mu\in\gamma(\mu^\sharp)\}$. Then

$$\int [\![\text{lfp}(name\mapsto f)]\!]_t.env \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right) \in H_1^\sharp(K^\sharp). \tag{8.98}$$

- From lemmas 8.4.1 and A.2.12, the following greatest fixpoint is written as a countable greatest lower bound:

$$\begin{array}{l}\int [\![\text{gfp}(name\mapsto f)]\!]_t.env \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int \text{gfp}(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \int \lim\downarrow_{n\to\infty}(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])^n(1)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\\ = \lim\downarrow_{n\to\infty}\int(\lambda\phi.[\![f]\!]_t.env[name\mapsto\phi])^n(1)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)\end{array}$$

  We can therefore approximate it safely from above by stopping at a certain

iteration count $N_0(f,\mu^\sharp)$: lemma 8.4.5:

$$\int [\![\mathrm{gfp}(name \mapsto f)]\!]_t .env \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right) \leq \int (\lambda\phi.[\![f]\!]_t .env[name \mapsto \phi])^{N_0(f,\mu^\sharp)}(1)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)$$

From lemma 8.4.6 applied to $H_1^\sharp = [\![f]\!]^\sharp_{Out}$, $H_2^\sharp = [\![f]\!]^\sharp_{Fwd(name)}$, $\psi = \phi \mapsto [\![f]\!]_t .env[name \mapsto \phi]$, $g = 1$, $I(g,\mu) = \int g \,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right)$, $I^\sharp(g,\mu^\sharp) = \{I(g,\mu) \mid \mu \in \gamma(\mu^\sharp)\}$,

$$\begin{aligned} I^\sharp((\lambda\phi.[\![f]\!]_t .env[name \mapsto \phi])^{N_0(f,\mu^\sharp)}(1),\mu^\sharp) \\ \subseteq\ & [\![f]\!]^\sharp_{Out} \circ \left(\nu^\sharp \mapsto \mu^\sharp +^\sharp [\![f]\!]^\sharp_{Fwd(name)}(\nu^\sharp)\right)^{N_0(f,\mu^\sharp)-1}(0^\sharp) \\ & + I^\sharp\left(g, [\![f]\!]^\sharp_{Fwd(name)}{}^{N_0(f,\mu^\sharp)}(\mu^\sharp)\right). \end{aligned}$$

It follows that

$$\int (\lambda\phi.[\![f]\!]_t .env[name \mapsto \phi])^{N_0(f,\mu^\sharp)}(1)\,\mathrm{d}\left(G(T,(U_n)_{n\geq 0}).\mu\right) \in [\![\mathrm{gfp}(name_R \mapsto f)]\!]^\sharp_{Out}.\mu^\sharp$$

and thus the result follows. □

8.5 Discussion

The backwards analysis method we described is a generalization of the value iteration method used in operational research to compute the value of Markov decision processes. Our reachability analysis is related to the study of *positive bounded models* [64, §7.2], where the reward 1 is granted the first time the process runs through the set of states to consider. The liveness analysis is related to the study of *negative models* [64, §7.3], where the reward -1 is granted the first time the process leaves the set of states to consider. Our remarks fro; §4 are reflected in [64]:

> Surprisingly, for negative models, optimal policies exist under weaker conditions than for positive models; however, they are more difficult to find because algorithms need not convergence to [the optimal value].

However, our point of view is more general, since we allow the mixing of least and greatest fixpoints, allowing complex specifications of trace properties. Those specifications are written in a simple formulaic language similar to μ-calculus [10, Ch. 7], for which we give an easy to understand semantics and a more operational one, linked by equalities and inequalities.

A possible extension of these properties is *discounted models* [64, Ch. 6]. In these, the importance of the future decreases exponentially; for instance, λ-discounted reachability would count passing through A for the first time at step n as λ^n instead of 1 (of course, $0 < \lambda < 1$). The theoretical study of those models is considerably easier than that of non-discounted models, since the fixpoints to study are the fixed points of contraction mappings in Banach spaces. While the extension of the techniques exposed in this thesis to discounted models is easy (it suffices to add a multiplication by λ in the semantics of the "shift" operation), the practical interest of such models in computer program checking remains to be seen.

Another possible extension is the computation of averages not only on the trace space, but also on the time: computing the average value of a certain function as long as the program is running. Since this property is the quotient of two summing properties, there is no obvious method to evaluate it iteratively.

We mentioned briefly (§8.3.4) the differences between Markov decision processes and two-player games. Such games can model questions such as the choice of an optimal strategy by the program so as to minimize the probability of a failure for all possible environments. The study of effective methods for such models is very hard [22].

Chapter 9

Finite sums

9.1 Finite sums on measures

As considerable effort has been put into the design and implementation of non-probabilistic abstract domains (see §6.2.3 for examples), it would be interesting to be able to create probabilistic abstract domains from these. In the this section, we shall give such a generic construction.

9.1.1 The Intuition Behind the Method

A finite sequence A_i of pairwise disjoint measurable subsets of X and corresponding coefficients $\alpha_i \in \mathbb{R}_+$, represent the set of measures μ such that:

- μ is concentrated on $\bigcup A_i$
- for all i, $\mu(A_i) \leq \alpha_i$.

For practical purposes, the A_i are concretizations of abstract elements, polyhedra for instance (Fig. 9.1).

This abstraction is intuitive, but lifting operations to it proves difficult: the constraint that sets must be disjoint is difficult to handle in the presence of non injective semantics $[\![c]\!]$. This is the reason why we rather consider the following definition: a finite sequence A_i of (non-necessarily disjoint) measurable subsets of X and corresponding coefficients $\alpha_i \in \mathbb{R}_+$ represent the set of measures μ such that there exist measures μ_i such that:

- $\mu = \sum \mu_i$;
- for all i, μ_i is concentrated on A_i;
- for all i, $\mu(A_i) \leq \alpha_i$.

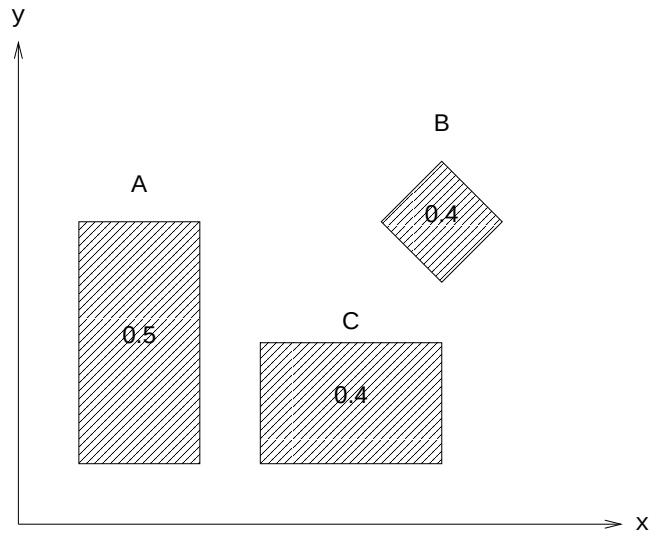


Figure 9.1: An abstract value representing measures μ such that $\mu(A) \leq 0.5$, $\mu(B) \leq 0.4$ and $\mu(C) \leq 0.4$.

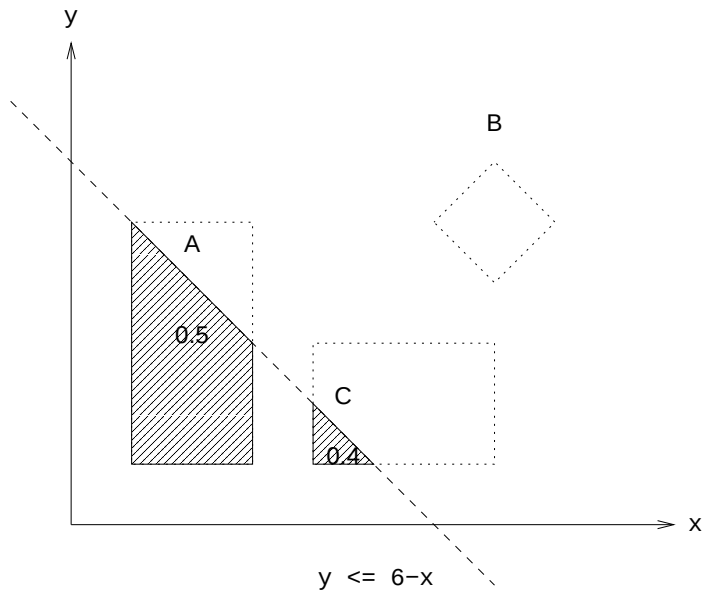


Figure 9.2: The abstract value of Fig. 9.1 after going into the first branch of a `if y<=6-x...` test.

We shall see how to formalize this definition and how program constructs act on such abstract objects.

9.1.2 Theoretical Construction

Let us take an indexing set Λ, an abstraction (see §13.2.1) $\Gamma_X = \langle \sigma_X, X^\sharp, \gamma_X \rangle$ and an abstraction $\Gamma_W = \langle \mathcal{P}([0,1]^\Lambda), W^\sharp, \gamma_W \rangle$. We define an abstraction $\Gamma_{\Lambda,\Gamma_X,\Gamma_W} = \langle \mathscr{C}(X_p), S_{\Lambda,\Gamma_X,\Gamma_W}, \gamma_{\Lambda,\Gamma_X,\Gamma_W} \rangle$. $\mathscr{C}(X_p)$ is the set of closed sets of the topological space X_p for the set-wise topology [27, §III.10] — this is a technical requirement that is easy to fulfill. We wish to define compositionally abstract semantics for our language (defined in §13.1.1). We shall omit the $\Lambda, \Gamma_X, \Gamma_W$ subscript if there is no ambiguity.

Domain

Let $S_{\Lambda,\Gamma_X,\Gamma_W} = X^{\sharp\Lambda} \times W^\sharp$ be our abstract domain. We then define $\gamma_{\Lambda,\Gamma_X,\Gamma_W} : S_{\Lambda,\Gamma_X,\Gamma_W} \to \mathcal{P}(X_p)$ such that $((Z_\lambda)_{\lambda\in\Lambda}, w)$ maps to the set of measures $\mu \in \mathcal{M}_{\leq 1}(X)$ such that there exist measures $(\mu_\lambda)_{\lambda\in\Lambda}$ such that

- for each $\lambda \in \Lambda$, μ_λ is concentrated on $\gamma_X(Z_\lambda)$;
- the family $(\int \mu_\lambda \, \mathrm{d})_{\lambda\in\Lambda}$ of total weights of those measures is in $\gamma_W(w)$.

Deterministic constructs

Given two such constructions $\Gamma_{\Lambda,\Gamma_X,\Gamma_W}$ and $\Gamma_{\Lambda,\Gamma_Y,\Gamma_W}$ and measurable function $f : X \to Y$ such that $f^\sharp$ is an abstraction of f (see formula 13.5), we define

$$f_p^\sharp : \left| \begin{array}{ll} S_{\Lambda,\Gamma_X,\Gamma_W} & \to S_{\Lambda,\Gamma_Y,\Gamma_W} \\ (Z_\lambda, w_\lambda)_{\lambda\in\Lambda} & \mapsto (f^\sharp(Z_\lambda), w_\lambda)_{\lambda\in\Lambda}. \end{array} \right.$$

Theorem 9.1.1. *$f_p^\sharp$ is an abstraction of f_p.*

Nondeterministic generators

We want a function $[\![\mathrm{choice}_Y]\!]^\sharp$ such that

$$\mu \in \Gamma \circ [\![\mathrm{choice}_Y]\!]^\sharp(\mu^\sharp) \implies (A \mapsto \mu(A \times Y)) \in \Gamma(\mu^\sharp). \tag{9.1}$$

The following function works:

$$(Z_\lambda, w_\lambda)_{\lambda\in\Lambda} \mapsto (H^\sharp(Z_\lambda), w_\lambda)_{\lambda\in\Lambda} \tag{9.2}$$

where $H^\sharp$ is an abstraction of $A \mapsto A \times Y$.

Random Inputs or Generators

To accommodate calls to `random`-like instructions, we must be able to abstract a product of two independent random variables knowing an abstraction for each of the variables. More precisely, let us suppose we have two abstractions $S_{\Lambda,\Gamma_X,\Gamma_W}$ and $S_{\Lambda',\Gamma_{X'},\Gamma_{W'}}$. Let us also suppose we have an abstraction $\Gamma_{W_p} = \langle \mathcal{P}([0,1]^{\Lambda\times\Lambda'}), W_p, \gamma_{W_p}\rangle$ and an abstraction $p^\sharp : W \times W' \to W_p$ of

$$p : \left| \begin{array}{ll} [0,1]^\Lambda \times [0,1]^{\Lambda'} & \to [0,1]^{\Lambda\times\Lambda'} \\ ((w_\lambda)_{\lambda\in\Lambda}, (w_{\lambda'})_{\lambda'\in\Lambda'}) & \mapsto (w_\lambda . w_{\lambda'})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}. \end{array} \right.$$

Let us also suppose we have an abstraction $\Gamma_\Pi = \langle \mathcal{P}(X\times X'), \Pi^\sharp, \gamma_\Pi\rangle$ and an abstraction $\times^\sharp : X^\sharp \times X'^\sharp \to X_p$ of $\times : \mathcal{P}(X)\times\mathcal{P}(X') \to \mathcal{P}(X\times X')$ (see formula 13.6). Let us take abstract elements $A = ((Z_\lambda)_{\lambda\in\Lambda}, w) \in S_{\Lambda,\Gamma_X,\Gamma_W}$ and $A' = ((Z'_\lambda)_{\lambda'\in\Lambda'}, w') \in S_{\Lambda',\Gamma_{X'},\Gamma_{W'}}$ then we define

$$A \otimes^\sharp A' = \left((Z_\lambda \times^\sharp Z'_{\lambda'})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}, p^\sharp(W,W') \right)$$

Theorem 9.1.2. *$(A^\sharp, A'^\sharp) \mapsto A^\sharp \otimes^\sharp A'^\sharp$ is an abstraction of $(\mu,\mu') \mapsto \mu\otimes\mu'$. That is, if $\mu \in \gamma_{\lambda,\Gamma_X,\Gamma_W}(A^\sharp)$ and $\mu' \in \gamma_{\lambda,\Gamma_{X'},\Gamma_{W'}}(A'^\sharp)$ then $\mu\otimes\mu' \in \gamma_p(A^\sharp \otimes^\sharp A'^\sharp)$.*

Tests

Lifting equation 13.2 to powersets yields the concrete semantics:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]^\flat_p(W) = [\![e_1]\!]^\flat_p \circ \phi^\flat_{[\![c]\!]}(W) +^\flat [\![e_2]\!]^\flat_p \circ \phi^\flat_{[\![c]\!]^C}(W)$$

which can be abstracted right away by replacing $\flat$'s by $\sharp$'s. All that is therefore needed are suitable $\phi^\sharp_{[\![c]\!]}(W)$ and $+^\sharp$.

We define

$$((Z_\lambda)_{\lambda\in\Lambda}, w) +^\sharp ((Z'_\lambda)_{\lambda'\in\Lambda'}, w') = ((Z_\lambda)_{\lambda\in\Lambda\blacksquare\Lambda'}, w \oplus^\sharp w')$$

where $\oplus^\sharp$ is an abstraction of the canonical bijection between $[0,1]^\Lambda \times [0,1]^{\Lambda'}$ and $[0,1]^{\Lambda\blacksquare\Lambda'}$ where $\Lambda\blacksquare\Lambda'$ is the disjoint union of Λ and Λ'. It is easy to see that such a $+^\sharp$ is an abstraction of $+^\flat$.

Let us suppose we have a suitable abstraction $I^\sharp_{[\![c]\!]} : X^\sharp \to X^\sharp$ of the intersection

function $W^\flat \mapsto W^\flat \cap [\![c]\!]$. We also require that $\forall x^\sharp \in X^\sharp\ I^\sharp_{[\![c]\!]}(x^\sharp) \sqsubseteq x^\sharp$.[1] Then we can define (see Fig. 9.2)

$$\phi^\sharp_{[\![c]\!]}((Z_\lambda)_{\lambda\in\Lambda}, w) = ((I^\sharp_{[\![c]\!]}(Z_\lambda))_{\lambda\in\Lambda}, d^\sharp(w))$$

Theorem 9.1.3. *$\phi^\sharp_{[\![c]\!]}$ is an abstraction of $\phi^\flat_{[\![c]\!]}$.*

9.1.3 Multiplicity of Representations and Coalescing

The reader might have been surprised we consider a preorder on the abstract values, not an order. The reason is that we want to talk of algorithmic representations, and a same concrete set can be represented in several ways. For instance, rational languages can be represented by an infinity of finite automata. Of course, an interesting property is that there is a minimal automaton and thus a canonical form. Yet we point out that this minimal automaton is defined up to state renaming, thus it has several representations.

We propose two *coalescing operations* to simplify representations without loss of precision:

1. If there is a certain Z_0 such that several λ are such that $Z_\lambda = Z_0$, and our numerical lattice enables us to represent exactly a sum, then one could replace all the entries for all these λ's by a single one.

2. Similarly, if Z_{λ_1} and Z_{λ_2} are such that there exists W such that $\gamma_X(Z_{\lambda_1}) \cup \gamma_X(Z_{\lambda_2}) = \gamma_X(W)$, then one can coalesce Z_{λ_1} and Z_{λ_2} into W, with probability $\min(w_{\lambda_1} + w_{\lambda_2}, 1)$.

9.1.4 Practical Constructions

In the previous section, we have given a very parametric construction, depending on parameters and assuming the existence of some operators. In this section we shall give examples of instances of suitable parameters and experimental results on a simple example.

---

[1] One possible construction for this function is $W^\sharp \mapsto W^\sharp \cap^\sharp [\![c]\!]^\sharp$ using an approximation $[\![c]\!]^\sharp$ of the set of environments matched by c and an approximation $\cap^\sharp$ of the greatest lower bound. This does not in general yield optimal results, and it is better to compute $I^\sharp_{[\![c]\!]}(W^\sharp)$ by induction on the structure of c if c is a boolean expression. An example of suboptimality is the domain of integer intervals on one variable, with $W = [0, +\infty[$ and boolean expression $(\mathsf{x} > 2) \vee (\mathsf{x} < -2)$. The abstraction of the domain matched by the expression is $\top$, which gives us the approximate intersection $[0, +\infty[$ while recursive evaluation yields $[2, +\infty[$. Further precision can be achieved by local iterations [29].

Abstract Domain

We shall first define a narrower class of abstract domains for probabilistic applications, for which we shall give algorithms for some operations.

Finite Sequences Let us suppose that $X^\sharp$ has a minimum element $\bot_{X^\sharp}$. We then take $\Lambda = \mathbb{N}$. We note $X^{\sharp(\mathbb{N})}$ the set of sequences with finite support; that is, those that are stationary on the value $\bot_{X^\sharp}$. We shall restrict ourselves to such abstract sequences.

As for the set of numeric constraints, one can for example use polyhedric constraints. Such constraints have the following nice property:

Theorem 9.1.4. *Let us suppose that:*

- *the numeric constraints are expressed as convex polyhedra, and the inclusion of two such polyhedra is decidable;*
- *the intersection test over $X^\sharp$ is computable.*

Then the preorder test on $S(X^\sharp)$ (i.e. the function that, taking $(a,b) \in S(X^\sharp)^2$ as parameter, returns 1 if and only if $a \sqsubseteq_{S(X^\sharp)} b$ and 0 otherwise) is computable.

Proof. Let $a = ((Z_i)_{i<N}, w)$ and $b = ((Z'_i)_{i<N'}, w')$. Let us call $\alpha_i = \mu_i(Z_i)$ and $\alpha'_i = \mu_i(Z'_i)$. Let $(\Xi_i)_{i<M}$ be the set of all nonempty intersections of elements of the sequences Z and Z_i. Let E be the system of equations of the form $\alpha_i = \sum \xi_j$ taking only the ξ_j such that $\Xi_j \subseteq Z_i$, E' the system of equations of the form $\alpha'_i = \sum \xi_j$ taking only the xi_j such that $\Xi_j \subseteq Z_i$. F the system of linear inequations yielded by $(\alpha_i)_{i<N} \in w$ and F' the system of linear inequations yielded by $(\alpha'_i)_{i<N} \in w'$. Given a system of (in)equations σ, we call $\mathcal{S}(\sigma)$ the set of solutions of σ. We claim that

$$a \sqsubseteq_{S(X^\sharp)} b \iff \mathcal{S}(E \cup E' \cup F) \subseteq \mathcal{S}(E \cup E' \cup F').$$

The right-hand side of the equivalence is decidable by hypothesis. □

Our claim is the consequence of the following lemma:

Lemma 9.1.5. *If Z is a nonempty measurable set and $c \in \mathbb{R}_+$, then there exists $\mu \in \mathcal{M}_+(Z)$ such that $\mu(Z) = c$.*

Proof. Let $z_0 \in Z$. Then we define $\mu(A)$ to be equal to c if $z_0 \in A$ and to 0 otherwise. □

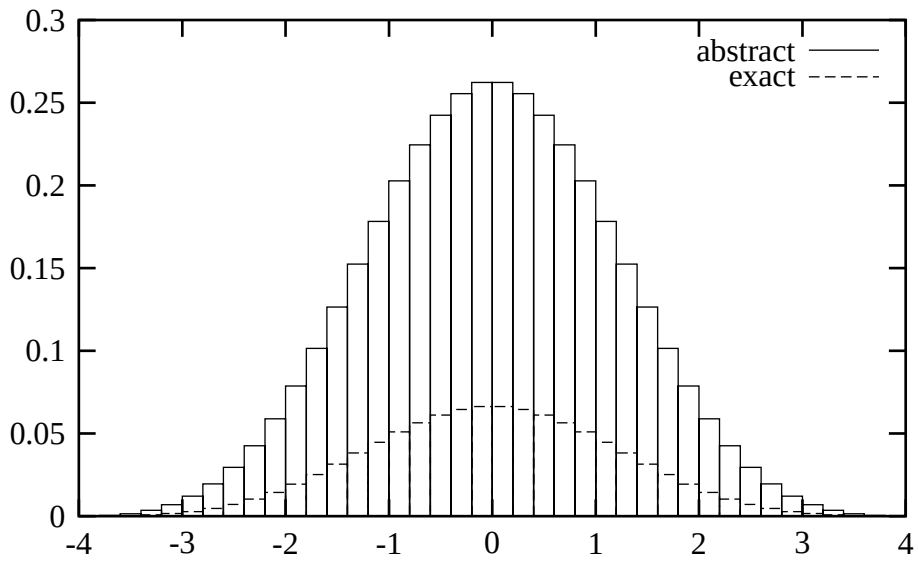


Figure 9.3: Experimental results: $X_1 + X_2 + X_3 + X_4$ where the X_i are independent random variables equidistributed in $[-1,1]$. The approximate simulation divided $[-1,1]$ into 10 sub-segments each of maximal probability 0.1. Estimates on segments of length 0.2.

Simple Constraints We propose a very restricted class of polyhedric constraints, given by finite sequences $(c_n)_{n\in\mathbb{N}} \in [0,1]^{(\mathbb{N})}$, such that

$$(\alpha_n)_{n\in\mathbb{N}} \in \gamma_W((c_n)_{n\in\mathbb{N}}) \iff \forall n \in \mathbb{N}\ \alpha_n \leq c_n.$$

An abstract element is thus stored a finite sequence (Z_n, c_n) of pairs in $X^\sharp \times [0,1]$. Similar *convex hulls* have already been proposed for rules operating on concrete semantics [31].

It is very easy in such a framework to get an upper approximation of the probability of a set W, if we have a function $\tau_W : X^\sharp \to \{\mathsf{true}, \mathsf{false}\}$ so that $\tau_W(X^\sharp) = \mathsf{true} \iff W \cap \gamma_X(X^\sharp) \neq \emptyset$: just take $\sum_{n\in\{n\in\mathbb{N}|\tau_W(Z_n)=\mathsf{true}\}} c_n$.

Experiments

Using our framework, we analyzed the following C program:

```
double x=0.0;
int i;
for (i=0; i<4; i++)
  x += drand48()*2.0-1.0;
```

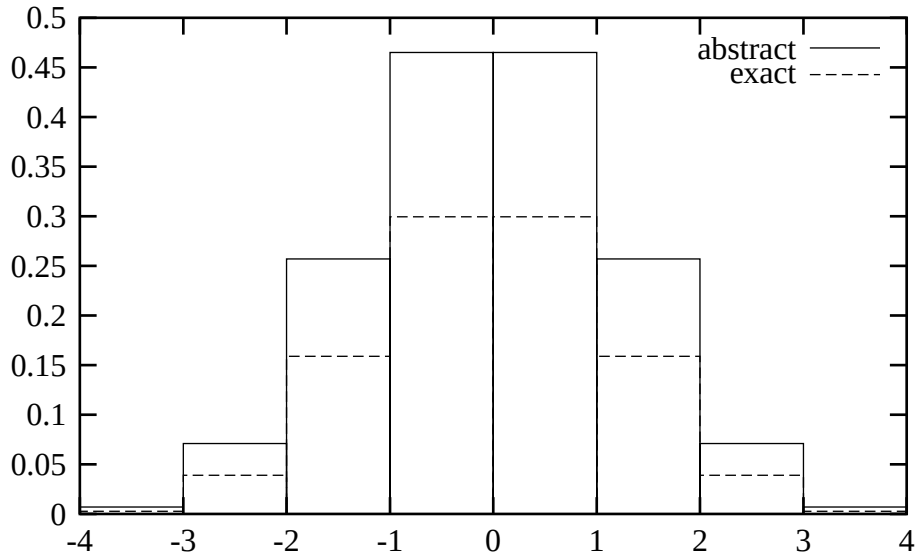


Figure 9.4: Same computations as Fig. 9.3. Approximations on segments of length 1 give more accurate results than approximations on smaller segments.

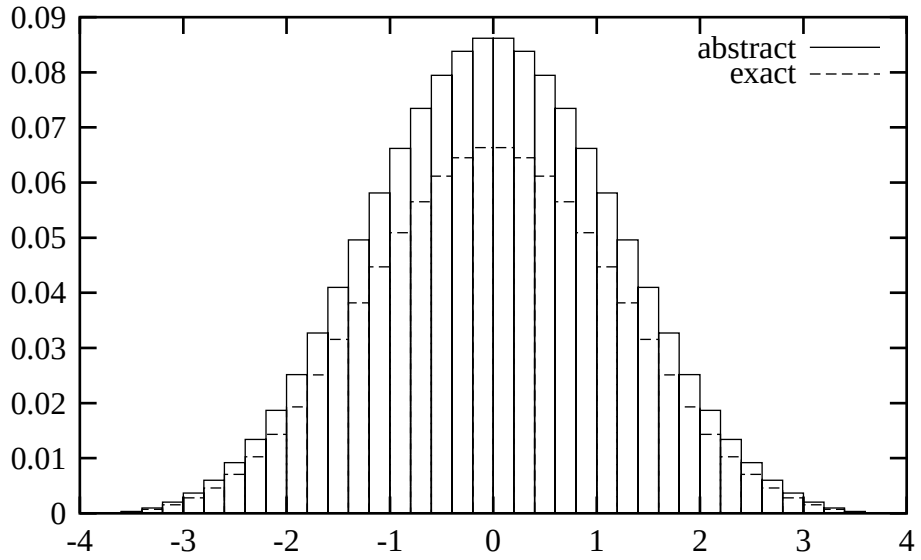


Figure 9.5: Same computation as Fig. 9.3, but the approximate simulation divided $[-1,1]$ into 100 sub-segments each of maximal probability 0.01. The sampled segments are of length 0.2. As with Fig. 9.4, precision is improved is sampling segments are bigger than the segments used in the computation.

The `drand48()` function returns a `double` number equidistributed in $[0,1[$. We chose such a simple program so as to have an easy exact computation.

As an accurate representation of the `double` type would be complex and dependent on the particular C implementation, we rather chose to use an idealized version of this type as the real numbers. Figures 9.3, 9.4 and 9.5 show results of the experiments with different parameters, comparing abstract samples and exact computations.

In those tests, `drand48()` is supposed to return an uniformly distributed real number in $[0,1]$. It is abstracted as $([n/N,(n+1)/N],1/N)_{0\leq n<N}$ where N is a parameter. The "samples" are segments $[\alpha,\beta]$; for each sample segment W, both an upper bound and an exact results of $\mu(W)$ are computed, where μ is the distribution of the final value of `x` in the above program. The upper bound is is computed from the abstract result, by the method described in 9.1.4. The bars displayed in the figure are the chosen segments $[\alpha,\beta]$ in x and their exact and approximate probabilities in y.

Those figures illustrate the following phenomenon: as computations go, the abstract areas Z_n grow bigger. If the samples are not enough bigger than those areas, the approximation are bad (Fig. 9.3). Results improve if N is increased (Fig. 9.5) or the sample size is increased (Fig. 9.4). An intuitive vision of this somewhat paradoxical behavior is that our abstract domain represents masses quite exactly, but loses precision on their exact location. If we ask our analysis to provide information on the probability of a very small area of the output domain (small compared to the precision of the input distribution and of the complexity of the transfer function), it tends to overestimate it (Fig. 9.3) because lots of masses could be located at that point. If we ask on a wider area, the error on the locations of the masses compared to the area becomes small and thus the error on the result becomes acceptable (Fig. 9.4).

Widenings

The crucial problem of the abstract domains not satisfying the ascending chain condition is the "widening" to choose. By widening, we mean some kind of over-approximation that jumps higher in the abstract domain to allow for convergence in finite time even if the abstract domain does not enjoy the property that every ascending sequence is stationary. Let us take a simple example on a nonprobabilistic program, with the domain of intervals: if successive abstract values are $[1,1]$, $[1,2]$, $[1,3]$, $[1,4]$, the system might try jumping to $[1,+\infty[$ for the next iteration. As this overestimates the set, it is safe.

The design of widenings is experimental in order to find a satisfying balance between cost and precision. While it is always possible to give a widening in all abstract domains with a maximum element (just jump to $\top$), it is quite difficult to

design widenings giving interesting results. Here, we shall propose a few ideas:

- Let us suppose we have a widening operator in $X^\sharp$. When successive abstract values in an iteration are $(Z_n, c_n)_{n \leq N}$ such that both Z_n and c_n increase, then try the next iteration with (Z, c_N) where Z is the result of the widening in $X^\sharp$.

- We can also apply widenings on the numerical coefficients c_n. For instance, if we have an increasing sequence $(Z, c_n)_{n \leq N}$, we can jump to (Z, c) where c is slightly above c_N, possibly 1.

Both approaches can be combined.

Another important area is simplification. Each call to `random`-like functions yields a product of measures and multiplies the number of length of the sequence making up the abstract environment by the length of the sequence approximating the measure of the function. This of course can mean fast explosion. While coalescing (see 9.1.3) can help, we might have to consider more energic steps. A possibility is to coalesce several abstract sets that have high probability (let us say, > 0.8) and are "close enough", such as $[0,2]$ and $[1,3]$.

We are currently working on designing on implementing such strategies and testing them on realistic examples.

9.1.5 Conclusions and Prospects

We have given simple probabilistic semantics to a deterministic language supplementing the usual constructions by functions returning random values of known distributions. We have a generic construct to lift usual (that is, non-probabilistic) abstract analyses to probabilistic analyses. The analysis we propose can be used to get upper bounds on the probability of certain events at certain points of a program. We tested it on some simple examples where an exact computation of the probabilities was possible, so as to have early experimental results of the influence of certain parameters over the quality of approximation.

We have proposed heuristics for some operators needed to handle large programs or loops. We expect to be able soon to propose results as to efficient heuristics on certain classes of problems.

9.1.6 Proofs of Convergence and Norm of the Concrete Semantics

This appendix is meant only for the purpose of refereeing. It contains technical proofs necessary for the construction of the concrete semantics. They are not relevant to the main topic of the paper, which is the analysis of this semantics.

Normal and random basic operations

Let us first remark that if H is a continuous operator on measures, using the norm of total variation, then $\|H\| = \sup\{\|H.\mu\| \mid \|\mu\| \leq 1\}$ can be computed on finite positive measures only: let $\|H\|_+ = \sup\{\|H.\mu\| \mid \mu \geq 0 \wedge \|\mu\| \leq 1\}$: if $\mu = \mu^+ - \mu^-$, $\|H.\mu\| = \|H.\mu^+ - H.\mu^-\| \leq \underbrace{\|H.\mu^+\|}_{\leq \|H\|_+ . \|\mu^+\|} + \underbrace{\|H.\mu^-\|}_{\leq \|H\|_+ . \|\mu^-\|} \leq \|H\|_+ . \underbrace{(|\mu^+\| + |\mu^-\|)}_{\|\mu\|}$, so $\|H\| \leq \|H\|_+$. But finite positive measures are signed measures, so $\|H\| \geq \|H\|_+$ and thus $\|H\| = \|H\|_+$.

If $f : X \to Y$ is an application, then $\|f_p.\mu\| = \|\mu\|$: on positive measures, $\|f_p.\mu\| = (f_p.\mu)(Y) = \mu(f^{-1}(Y)) = \mu(X) = \|\mu\|$ and the result extends to signed measures.

If μ_R is a probability measure, then for any μ, $\|\mu \otimes \mu_R\| = \|\mu\|$. This, combined with the norm of assignment, implies that $\left\| [\![\texttt{x:=random}]\!]_p \right\| = 1$.

Flow control

Let us first remark that if W is measurable, then for all μ, $\|\phi_W.\mu\| + \|\phi_{W^C}.\mu\| = \|\mu\|$.

Equation 13.2 defines linear operator $H = [\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p$ as $[\![e_1]\!]_p \circ \phi_{[\![c]\!]} + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}$. Let us suppose that $\left\| [\![e_1]\!]_p \right\| \leq 1$ and $\left\| [\![e_2]\!]_p \right\| \leq 1$. Then

$$\|H.\mu\| \leq \underbrace{\left\| [\![e_1]\!]_p \right\|}_{\leq 1} . \left\| \phi_{[\![c]\!]}.\mu \right\| + \underbrace{\left\| [\![e_2]\!]_p \right\|}_{\leq 1} . \left\| \phi_{[\![c]\!]^C}.\mu \right\|$$
$$\leq \left\| \phi_{[\![c]\!]}.\mu \right\| + \left\| \phi_{[\![c]\!]^C}.\mu \right\| = \|\mu\|,$$

thus $\|H\| \leq 1$.

In equation 13.4, we define a measure as the limit of a sequence of measures:

$$[\![\texttt{while } c \texttt{ do } e]\!]_p.\mu = \sum_{n=0}^{\infty} \phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^n(\mu) \tag{9.3}$$

This limit is taken set-wise: ie, we mean that for any measurable set X,

$$([\![\texttt{while } c \texttt{ do } e]\!]_p.\mu)(X) = \sum_{n=0}^{\infty} (\phi_{[\![c]\!]^C} \circ ([\![e]\!]_p \circ \phi_{[\![c]\!]})^n.\mu)(X).$$

If μ is a positive measure, sum 9.3 indeed defines a positive measure: the partial sums are measures and the set-wise limit of an increasing sequence of positive

measures is a positive measure [27, §III.10]. The result extends to signed measures by splitting the measure into its positive and negative parts. It is also quite evident that equation 9.3 defines a linear operator H, so we have shown that this linear operator maps measures onto measures.

Let us define the partial sums $H_n = \sum_{k=0}^{n} \phi_{[\![c]\!]^C} \circ (\llbracket f \rrbracket_p \circ \phi_{[\![c]\!]})^k$. H_n can be equivalently defined by the following recurrence:

$$\begin{cases} H_0 = \phi_{[\![c]\!]^C} \\ H_{n+1} = \phi_{[\![c]\!]^C} + H_n.\llbracket e \rrbracket.\phi_{[\![c]\!]}. \end{cases}$$

By induction, we prove that for all n, $\|H_n\| \leq 1$, similarly as for the `if` construct.

Let us now consider the sequence of measures $H_n.\mu$. It converges set-wise to $H.\mu$; also, for all n, $\|H_n.\mu\| \leq \|\mu\|$. For all measurable X, $|(H_n.\mu)(X)| \leq \|\mu\|$ and thus, using the set-wise limit, $|(H.\mu)(X)| \leq \|\mu\|$. It follows that $\|H.\mu\| \leq \mu$. This achieves proving $\|H\| \leq 1$.

Remark 9.1.6. In general, H_n does not converge norm-wise to H.

Proof. Let us consider the following C program:

```
/* x is in ]0, 1] */
while (x<=0.5)
  x=x*2;
```

It is easy to see that $\int (H.\mu)\,\mathrm{d} = \int \mu\,\mathrm{d}$. Let us consider the operator π that maps the 1-tuple containing `x` to the 0-tuple and the operators H_n of the loops. $\pi \circ H_n(\mu)$ is then the probability of termination on input measure μ. If $H_n \to H$ in norm, we would have $\pi \circ H_n \to \pi \circ H$ in norm too. Nevertheless, if μ_n is a measure of weight 1 lying in $]0, 2^{-n-1}]$, $(\pi \circ H_n).\mu_n = 0$ while $(\pi \circ H).\mu_n = 1$ so $\|\pi \circ H - \pi \circ H_n\| = 1$. □

Continuity of the various operators enables us to swap them with infinite sums, which is important to get some of the results in equation 13.4.

9.1.7 Proofs of Abstraction

Theorem 9.1.1. *$S(f)$ is an abstraction of $f_p^\sharp$.*

Proof. This amounts to:

$$\forall x \in S(X^\sharp)\ \forall \mu \in S(\gamma_X)(x)\ \ f_p(\mu) \in S(\gamma_Y)(S(f)).$$

Let $((Z_\lambda)_{\lambda\in\Lambda}, w) \in S(X^\sharp)$ and $\mu \in \mathcal{M}_+(X)$. Let us suppose that there exists a family of measures μ_λ each respectively in $\mathcal{M}_+(Z_\lambda)$ such that $\mu = \sum_{\lambda\in\Lambda} \mu_\lambda$ and $(\mu_\lambda(\gamma_X(Z_\lambda)))_{\lambda\in\Lambda} \in \gamma_W(w)$. We want to prove that there exists a family of measures μ'_λ each respectively in $\mathcal{M}_+(f^\sharp(Z_\lambda))$ such that

1. $f_p(\mu) = \sum_{\lambda\in\Lambda} \mu'_\lambda$ and
2. $(\mu'_\lambda(\gamma_Y(f^\sharp(Z_\lambda))))_{\lambda\in\Lambda} \in w$.

Let us take candidates $\mu'_\lambda(U) = \mu_\lambda(f^{-1}(U))$. The first condition is then obviously met. As for the second, $\mu'_\lambda(f^\sharp(Z_\lambda)) = \mu_\lambda(f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda)))$. As $f^\sharp$ abstracts $f^\flat$, $f^\flat \circ \gamma_X(Z_\lambda) \subseteq \gamma_Y \circ f^\sharp(Z_\lambda)$; this implies that $f^{-1}(f^\flat \circ \gamma_X(Z_\lambda)) \subseteq f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))$. As $\gamma_X(Z_\lambda) \subseteq f^{-1}(f^\flat \circ \gamma_X(Z_\lambda))$, by transitivity $\gamma_X(Z_\lambda) \subseteq f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))$. This implies that $\mu'_\lambda(\gamma_Y \circ f^\sharp(Z_\lambda)) = \mu_\lambda(f^{-1}(\gamma_Y \circ f^\sharp(Z_\lambda))) = \mu_\lambda(\gamma_X(Z_\lambda))$, which in turn implies that $(\mu'_\lambda(\gamma_Y \circ f^\sharp(Z_\lambda)))_{\lambda\in\Lambda} \in \gamma_W(w)$. □

Theorem 9.1.2. *$(A^\sharp, A'^\sharp) \mapsto A^\sharp \otimes^\sharp A'^\sharp$ is an abstraction of $(A^\flat, A'^\flat) \mapsto A^\flat \otimes^\flat A'^\flat$ with respect to $\gamma_{S_{\Lambda\times\Lambda',\Gamma_{X_p},\Gamma_{W_p}}}$. That is, if $\mu \in \gamma_X(A^\sharp)$ and $\mu' \in \gamma_X(A'^\sharp)$ then $\mu \otimes \mu' \in \gamma_p(A^\sharp \otimes^\sharp A'^\sharp)$.*

Proof. We have to prove that there exists a family of measures $(\mu^*_{(\lambda,\lambda')})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}$ defined respectively over $(Z_\lambda \times Z_{\lambda'})_{(\lambda,\lambda')\in\Lambda\times\Lambda'}$ such that

1. $\mu^* = \sum_{(\lambda,\lambda')\in\Lambda\times\Lambda'} \mu^*_{(\lambda,\lambda')}$ and
2. $(\mu^*_{(\lambda,\lambda')}(Z_\lambda, Z'_{\lambda'}))_{(\lambda,\lambda')\in\Lambda\times\Lambda'} \in p^\sharp(W, W')$.

We take $\mu^*_{(\lambda,\lambda')} = \mu_\lambda \otimes \mu'_{\lambda'}$ as a candidate. The first point is trivial since $\otimes$ is bilinear. Since $\times^\sharp$ is an abstraction of $\times$, $\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'}) \subseteq \gamma_P(Z_\lambda \times^\sharp Z'_{\lambda'})$. By the definition of the product of measures, since μ_λ is concentrated on $\gamma_X(Z_\lambda)$ and $\mu'_{\lambda'}$ on $\gamma_{X'}(Z'_{\lambda'})$, then $\mu_\lambda \otimes \mu'_{\lambda'}$ is concentrated on $\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})$, which, using the above inclusion, yields

$$\mu^*_{(\lambda,\lambda')}(\gamma_P(Z_\lambda \times^\sharp Z'_{\lambda'}) = \mu^*_{(\lambda,\lambda')}(\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})).$$

By the definition of the product of measures,

$$\mu^*_{(\lambda,\lambda')}(\gamma_X(Z_\lambda) \times \gamma_{X'}(Z'_{\lambda'})) = \mu_\lambda(Z_\lambda).\mu'_{\lambda'}(Z_{\lambda'}).$$

The second point then follows from the fact that $p^\sharp$ is an abstraction of $p^\flat$. □

Theorem 9.1.3. *$\phi^\sharp_{[\![c]\!]}$ is an abstraction of $\phi^\flat_{[\![c]\!]}$.*

Proof. Let us take $((Z_\lambda)_{\lambda\in\Lambda}, w) \in S(X^\sharp)$ and μ one of its concretizations: $\mu = \sum_{\lambda\in\Lambda} \mu_\lambda$ where μ_λ is concentrated on Z_λ and $\int d\mu_\lambda = \alpha_\lambda$ such that $(c_\lambda)_{\lambda\in\Lambda} \in w$. We have to show that $\phi_{[\![c]\!]}(\mu) \in \gamma_{X^\sharp}(\phi^\sharp_{[\![c]\!]})$; that is, that there exists a family $(\mu^*_\lambda)_{\lambda\in\Lambda}$ such that

1. $\lambda \in \Lambda$, μ^*_λ is concentrated on $\gamma_{X^\sharp}(I^\sharp_{[\![\mathtt{c}]\!]}(Z_\lambda))$ and

2. $(\mu^*_\lambda)_{\lambda\in\Lambda} \in \gamma_W(d^\sharp(w))$.

We take $\mu^*_\lambda(X) = \mu_\lambda(X \cap \gamma_X(I^\sharp_{[\![c]\!]}))$.

The first condition is trivial. $I^\sharp_{[\![c]\!]}(Z_\lambda) \sqsubseteq Z_\lambda$ so by monotonicity, $\gamma_X(I^\sharp_{[\![c]\!]}(Z_\lambda) \subseteq \gamma_X(Z_\lambda)$. Therefore $\int \mathrm{d}\mu^*_\lambda \leq \alpha_\lambda$, which proves the second point. □

9.1.8 Nondeterminism

We shall now see with more precision aspects of our analysis pertaining to nondeterminism. Probabilistic nondeterminism means that the outcome of a program can be a set of possible (sub)probability distributions; this encompasses ordinary probabilistic behavior (single distribution in the set) and ordinary nondeterminism (set of Dirac point-mass distributions). To a program c we associate a set $[\![c]\!]^S_p$ of linear continuous operators on measures of norm ≤ 1, the set of all possible denotational semantics for all possible nondeterministic choices:

- Ordinary constructs
$$[\![c]\!]^S_p = \{[\![c]\!]_p\}$$

- Nondeterministic choice
$$[\![c_1 \ \mid \ \cdots \ \mid \ c_n]\!]^S_p = [\![c_1]\!]^S_p \cup \cdots \cup [\![c_n]\!]^S_p$$
Please note that there is no reason why nondeterminism should be finite or countable.

- Sequence
$$[\![c_1\mathtt{;}\ c_2]\!]^S_p = [\![c_2]\!]^S_p \circ^\flat [\![c_1]\!]^S_p$$
where $f^S \circ^S g^S = \{f \circ g \mid f \in f^S \wedge g \in g^S\}$

- Tests
$$[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2]\!]^S_p = [\![c_1]\!]^S_p \circ^S \{\phi_{[\![b]\!]}\} +^S [\![c_2]\!]^S_p \circ^S \{\phi_{[\![b]\!]^c}\}$$
where $f^S +^S g^S = \{f + g \mid f \in f^S \wedge g \in g^S\}$

- Loops Let us consider a sequence $(f_n)_{n\in\mathbb{N}}$ of elements of $[\![c]\!]^S_p$. Let us now consider the sequence of linear operators $H_n = \sum_{k=0}^n \phi_{[\![b]\!]^c} \circ (f_k \circ \phi_{[\![c]\!]})^k$. As

in section 9.1.6, we can show that $H = \lim n \to \infty$ exists and is a continuous linear operator of norm ≤ 1. Furthermore, as in equation 13.4, $H.\mu = \phi_{[\![c]\!]^C}(\lim_{n\to\infty} \mu_n)$ where μ_n is defined inductively:

$$\begin{cases} \mu_0 = \lambda X.0 \\ \mu_{k+1} = \mu + f_k \circ \phi_{[\![c]\!]} \mu_k \end{cases}$$

We then define $[\![\texttt{while } b \texttt{ do } c]\!]^S_p$ to be the set of all such H for all possible $(f_n)_{n\in\mathbb{N}}$.

Let us now define $[\![c]\!]^\flat_p$ an additive map from sets of measures to sets of measures such that $[\![c]\!]^\flat_p.\mu^\flat \supseteq \{f.\mu \mid f \in [\![c]\!]^S_p \wedge \mu \in \mu^\flat\}$:

- Ordinary constructs

$$[\![c]\!]^\flat_p(\mu^\flat) = \{[\![c]\!]_p.\mu \mid \mu \in \mu^\flat\}$$

- Nondeterministic choice

$$[\![c_1 \mid \cdots \mid c_n]\!]^\flat_p(\mu^\flat) = [\![c_1]\!]^\flat_p(\mu^\flat) \cup \cdots \cup [\![c_n]\!]^\flat_p(\mu^\flat)$$

- Sequence

$$[\![c_1; c_2]\!]^\flat_p = [\![c_2]\!]^\flat_p \circ [\![c_1]\!]^\flat_p$$

- Tests

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]^\flat_p = [\![c_1]\!]^\flat_p \circ \phi^\flat_{[\![b]\!]} +^\flat [\![c_2]\!]^\flat_p \circ \phi^\flat_{[\![b]\!]^C}$$

where $\phi^\flat_W(\mu^\flat) = \{\phi_W.\mu \mid \mu \in \mu^\flat\}$ and $\mu^\flat +^\flat \nu^\flat = \{\mu + \nu \mid \mu \in \mu^\flat \wedge \nu \in \nu^\flat\}$.

- Loops

$$[\![\texttt{while } c \texttt{ do } e]\!]^\flat_p(\mu^\flat) = \phi^\flat_{[\![c]\!]^C}(\mathrm{Adh}(\mathrm{lfp}_{\lambda X.0}(\lambda \nu^\flat.\mu +^\flat [\![e]\!]^\flat_p \circ \phi_{[\![c]\!]}(\nu^\flat))))$$

9.1.9 Fast equality test

The purpose of this section is to prove the following property:

Theorem 9.1.4. *We use the following abstraction: $\gamma((X^\sharp_k, \alpha_k)_{1\leq k\leq N})$ is the set of measures μ that are the sums of measures μ_k, each concentrated on $\gamma(X^\sharp_k)$ and of weight less or equal to α_k. Then $\gamma((X^\sharp_k, \alpha_k)_k) = \gamma((Y^\sharp_k, \beta_k)_k)$ if and only if the sequences are a permutation of each other.*

This theorem allows for very fast equality testing if the sequences are stored as finite sets or partial maps with a fast equality testing. We implemented such a scheme as follows:

- abstract areas ($X^\sharp$) are hashed with a collision-resistant hash function h [72, ch. 18]; this means that for all practical purposes, this function behaves as if it were injective;
- partial maps $h(X^\sharp) \mapsto (X^\sharp, \alpha_{x^\sharp})$ over the totally ordered set of hash values (can be implemented as balanced binary trees);
- the concatenation of the $(h(X^\sharp), \alpha)$ couples, ordered according to their first coordinate, is taken and hashed; this yields a hash value $H((X_k^\sharp, \alpha_k)_k)$, well-defined.

This construct allows for the collision-resistance of function H. Comparing two abstract values A and B is therefore equivalent to comparing $H(A)$ and $H(B)$. Using the theorem, we prove that it is also equivalent to comparing $\gamma(A)$ and $\gamma(B)$.

Finite repartitions

The purpose of this subsection is to prove the theorem in the case where Ω is finite.

Let Ω be a finite set. Let $M(\Omega) = (\mathcal{P}(\Omega) \setminus \{\emptyset\}) \to \mathbb{R}_+$. Let us define $c : M(\Omega) \to \mathcal{P}(\Omega \to \mathbb{R}_+)$ such that $w : \Omega \to \mathbb{R}_+$ is in $c(\alpha)$ if and only there exists a repartition function $K : (\mathcal{P}(\Omega) \setminus \{\emptyset\}) \times \Omega \to \mathbb{R}_+$ such that:

$$\forall \omega \in \Omega,\ w(\omega) = \sum_{A \in \mathcal{P}(\Omega) \setminus \{\emptyset\}} K(\omega, A) \tag{9.4}$$

and

$$\forall a \in \mathcal{P}(\Omega) \setminus \{\emptyset\},\ \alpha(A) = \sum_{\omega \in \Omega} K(\omega, A) \tag{9.5}$$

Lemma 9.1.5. *If $c(A) = c(B)$ then for all $\omega \in \Omega$, $A(\{\omega\}) = B(\{\omega\})$.*

Proof. Taking a set $S \subseteq \Omega \to \mathbb{R}_+$, we note $\inf_\omega(S) = \inf S(\{\omega\})$. We then prove that $\inf_\omega(c(A)) = S(\{\omega\})$. Indeed:

- $\inf_\omega(c(A)) \geq S(\{\omega\})$: for any repartition function K, $K(\{\omega\}, \omega) = S(\{\omega\})$ thus the corresponding weight function w is such that $w(\omega) \geq S(\{\omega\})$.
- Let us now consider a repartition function K such that:

- $K(\{\omega\},\omega) = S(\{\omega\})$ (forced)
- for any set $X \in \mathcal{P}(\Omega \setminus \{\omega\}) \setminus \{\emptyset\}$, let us choose $x_X \in X$ and

$$\begin{cases} \forall y \in \Omega \setminus \{\omega, x_X\}, K(X,y) = 0 \\ K(X,x_X) = S(X) \end{cases}$$

This repartition yields a weight function w such that $w(\omega) = S(\omega)$. Therefore, $\inf_\omega(c(A)) \geq S(\{\omega\})$.

Supposing $c(A) = c(B)$, the lemma follows. □

Lemma 9.1.6. *Let us consider $\omega \in \Omega$ and $S \in M(\Omega)$. Let us consider the set*

$$c_\omega(S) = \{w \in c(S) \mid w(\omega) = 0\}.$$

Let us define $S_\omega \in M(\Omega \setminus \{\omega\})$:

$$S_\omega : \left| \begin{array}{ll} \mathcal{P}(\Omega \setminus \{\omega\}) \setminus \{\emptyset\} \to \mathbb{R}_+ \\ X \qquad\qquad\qquad\quad \mapsto S(X) + S(X \sqcup \{\omega\}) \end{array} \right. .$$

Then $c(S_\omega) = c_\omega(S)$.

Proof. Trivial. □

Lemma 9.1.7. *$c(A) = c(B)$ if and only if $A = B$.*

Proof. Let us prove that property by induction on the cardinal of Ω.

- $|\Omega| = 0$ and $|\Omega| = 1$ are trivial cases.
- Let us suppose that $|\Omega| > 1$ and the property is valid for sets of cardinal $|\Omega| - 1$. Let us suppose that $c(A) = c(B)$. Let us now prove that for all $X \in \mathcal{P}(\Omega) \setminus \{\emptyset\}$, $A(X) = B(X)$. Let us proceed by induction on $|X|$.
  - $|X| = 1$: follows from lemma 9.1.5.
  - $|X| > 1$. Let us fix $x \in X$. Using the notations of lemma 9.1.6, $c_x(A) = c_x(B)$; thus following lemma 9.1.6, $c(A_x) = c(B_x)$. Applying the induction hypothesis to set $X \setminus \{x\}$ and functions A_x and B_x, we obtain that for all $Y \in \mathcal{P}(X \setminus \{x\}) \setminus \{\emptyset\}$, $A_x(Y) = B_x(Y)$; in particular, $A_x(X \setminus \{x\}) = B_x(X \setminus \{x\})$. Using lemma 9.1.6, this gets rewritten as $A(X) + A(X \setminus \{x\}) = B(X) + B(X \setminus \{x\})$. By the induction hypothesis, $A(X \setminus \{x\}) = B(X \setminus \{x\})$, thus $A(X) = B(X)$. □

Let us now define $c_{\leq}$ similarly as c, replacing equation 9.5 by the following inequality:

$$\forall a \in \mathcal{P}(\Omega) \setminus \{\emptyset\},\ \alpha(A) \geq \sum_{\omega \in \Omega} K(\omega, A) \tag{9.6}$$

Lemma 9.1.8. *If $c_{\leq}(A) = c_{\leq}(B)$ then $A = B$.*

Proof. Obviously, for any $S \in M(\Omega)$,

$$c(S) = \{w \in c_{\leq}(S) \mid \sum_{\omega \in \Omega} w(s) = \sum X \in \mathcal{P}(\Omega) \setminus \{\emptyset\} S(X)\}.$$

Thus $c(A) = c(B)$. Applying lemma 9.1.7, the result follows. □

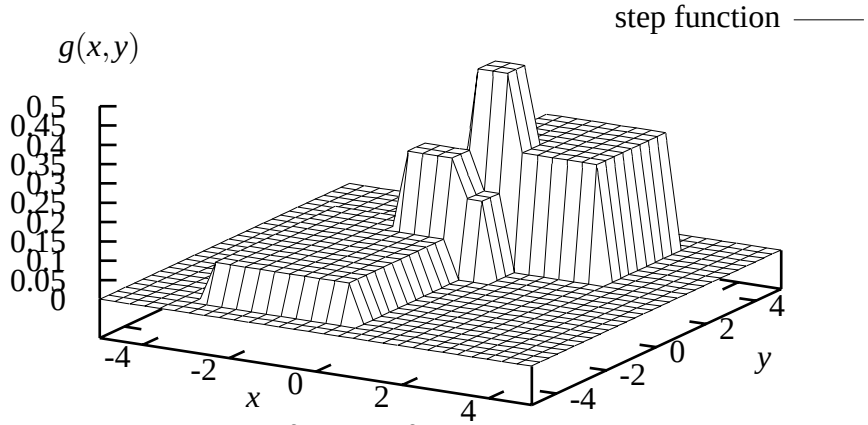


Figure 9.6: An example of a step function: $0.2\chi_{[-1,1]\times[0,2]} + 0.3\chi_{[0,3]\times[1,4]} + 0.1\chi_{[-3,0]\times[-4,1]}$. The non-vertical slopes are of course an artefact from the plotting software.

9.2 Step functions

Our goal now is to have an efficient way of representing sets of weight functions. In this section we propose an abstract lattice based on *step functions*. As usual in Lebesgue integration theory, a step function is a finite linear combination of characteristic functions of (measurable) subsets of the domain (see Fig. 9.6); this generalizes the usual definition when the domain is $\mathbb{R}$. χ_M will denote the characteristic function of subset M — that is, the function mapping x onto 1 if $x \in M$ and 0 elsewhere.

9.2.1 Representation

Let us take an "ordinary" abstract interpretation lattice $X^\sharp$ for the domain X. This means we have a non-decreasing function $\gamma : (X^\sharp, \sqsubseteq) \to (\mathcal{P}(X), \subseteq)$. We shall only consider the set $X^\sharp_w$ of step functions of the form $\sum_k \alpha_k \cdot \chi_{\gamma A^\sharp_k}$ where $A^\sharp_k \in X^\sharp$. This function can be represented in machine by a finite list of couples $(A^\sharp_k, \alpha_k)_{1\leq k\leq n}$.

The set $X^\sharp_w$ is pre-ordered by the usual pointwise ordering: $(A^\sharp_k, \alpha_k) \sqsubseteq (B^\sharp_k, \beta_k)$ if and only if for all $x \in X$, then $\sum_k \alpha_k \cdot \chi_{\gamma A^\sharp_k}(x) \leq \sum_k \beta_k \cdot \chi_{\gamma B^\sharp_k}(x)$. Please note that while the pointwise ordering $\leq$ on step functions is indeed antisymmetric, $\sqsubseteq$ is only a preorder since representation is not unique: $\{([0,1],1),(]1,2],1)\}$ and $\{([0,2],1)\}$ both represent $\chi_{[0,2]}$. Let us define

$$\gamma_w : \left| \begin{array}{l} (X^\sharp_w, \sqsubseteq) \to (\mathcal{P}(M(X,\mathbb{R}_+)), \subseteq) \\ (A^\sharp_k, \alpha_k) \mapsto \{ f \in M(X,\mathbb{R}_+) \mid f \leq \sum_k \alpha_k \cdot \chi_{\gamma A^\sharp_k} \} \end{array} \right. \tag{9.7}$$

$(X_w^\sharp, \sqsubseteq)$ is therefore a suitable abstract domain for weight functions.

9.2.2 Comparison

Our abstract lattice does not have unicity of representation, as noted above. Yet comparison and equivalence testing are easily computable, provided the underlying abstract domain provides an intersection test — a computable function $(A^\sharp, B^\sharp) \mapsto \begin{cases} 1 & \text{if } \gamma(A) \cap \gamma(B) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$

Let us take two abstract values $A_w^\sharp = ((A_i^\sharp, \alpha_i)_{1 \leq i \leq m})$ and $B_w^\sharp = ((B_j^\sharp, \alpha_j)_{1 \leq j \leq n})$. Let us consider the set C of nonempty intersections $\bigcap \gamma(A_i^\sharp)_{i \in I} \cap \bigcap \gamma(B_j^\sharp)_{j \in J} \neq \emptyset$ where I is a subset of the indices $1..m$ and J is a subset of the indices $1..n$: each element of C is denoted by a couple (I, J).
Let us define

$$w : \left| \begin{array}{ll} C & \to \mathbb{R} \\ (I,J) & \mapsto \sum_{i \in I} \alpha_i - \sum_{j \in J} \beta_i. \end{array} \right.$$

Then $A_w^\sharp \sqsubseteq B_w^\sharp \iff \forall c \in C\ w(c) \leq 0$.

9.2.3 Abstract operations

Deterministic constructs

Let us now suppose we have an abstraction $[\![c]\!]^{-1^\sharp}$ of the function $[\![c]\!]^{-1} : \mathcal{P}(Y) \to \mathcal{P}(X)$. Then an abstraction of $[\![c]\!]_p^*$ is

$$[\![c]\!]_p^{*\sharp} = (X_\lambda^\sharp, \alpha_\lambda)_{\lambda \in \Lambda} \mapsto ([\![c]\!]^{-1^\sharp}(X_\lambda^\sharp), \alpha_\lambda)_{\lambda \in \Lambda}. \tag{9.8}$$

Nondeterministic choice

We want an abstraction of

$$[\![\text{choice}_Y]\!]_p^* : f \mapsto \left(x \mapsto \sup_{y \in Y} f(x,y) \right). \tag{9.9}$$

Let us suppose we have an abstraction $\pi_1^\sharp$ of the projection function π_1 from $X \times Y$ to X; then a possible abstraction is

$$[\![\text{choice}_Y]\!]_p^{*\sharp} : (X_\lambda^\sharp, \alpha_\lambda)_{\lambda \in \Lambda} \mapsto (\pi_1^\sharp(X_\lambda^\sharp), \alpha_\lambda)_{\lambda \in \Lambda}. \tag{9.10}$$

This abstraction is not necessarily very good. For instance, depending on how the same function is decomposed, the projection may be good or bad (Fig. 9.7).

(a) Bad projection (b) Good projection

Figure 9.7: Projecting the decomposition straightforwardly may lead to very bad approximations depending on the original decomposition. Here, the same function is decomposed either as the characteristic function of a rectangular area, either as the sum of the characteristic functions of five areas. In the former case, the result is good; in the latter, we have a very bad approximation.

We shall now discuss improvements. Let us consider $(X^\sharp_\lambda), \alpha_\lambda)_{\lambda\in\Lambda}$. Let us partition the $X^\sharp_\lambda$ according to the value of $\pi^\sharp_1(X^\sharp_\lambda)$. Let us consider an element of that partition: $K \subseteq \Lambda$ so that

$$K = \{\lambda \mid \pi^\sharp_1(X^\sharp_\lambda) = Z^\sharp\}. \tag{9.11}$$

Let us take

$$\alpha_{Z^\sharp} = \sup_{\substack{M\subseteq K \\ \bigcap_{\lambda\in M}\gamma(X^\sharp_\lambda)\neq\emptyset}} \sum_{\lambda\in M} \alpha_\lambda \tag{9.12}$$

Collecting all the $Z^\sharp$ and the $\alpha_{Z^\sharp}$, we now have a better approximation (for instance, in the case in Fig. 9.7). Obviously, this better approximation is much simpler if all $\gamma(X^\sharp_\lambda)$ are disjoint pairwise. This happens, for instance, if the abstract lattice $X^\sharp$ is the flat lattice generated by a partition of X.

Let us additionally note that a particularly interesting domain is constructed when $X^\sharp$ is $\mathcal{P}(D)$, the power-set generated by a partition D of X: the abstract values can be memorized as elements of $[0,1]^D$ (and not $[0,1]^{2^D}$) after the following "simplification step":

$$S: \left| \begin{array}{ll} [0,1]^{\mathcal{P}(D)} & \to [0,1]^D \\ f & \mapsto d \mapsto \sum_{\substack{W\in\mathcal{P}(D)\\ d\in W}} f(W) \end{array} \right. \tag{9.13}$$

There is a Galois connection $[0,1]^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} [0,1]^D$ where $\alpha(f)(d) = \sup_{x\in d} f(x)$.

Random number generation

We shall obtain here an abstraction of `R:=random` where `R` is a new variable and `random` follows the probability measure μ_R. Let us suppose the random variable lies in a set R. $[\![\texttt{r:=random}]\!]^*$ is therefore a linear operator from $M(X\times R, \mathbb{R}_+)$ to $M(X, \mathbb{R}_+)$.

Let us suppose that $\mu_R = \sum_{k=1}^n \mu_k$ where each μ_k is concentrated on a subset M_k of R. For instance, taking $R = \mathbb{R}$, the uniform probability measure on $[0,1]$ can be split into n measures μ_k, the Lebesgue measure on $[k/n, (k+1)/n]$. Let us call π_X and π_R the projections of $X\times R$ onto X and R respectively.

Using prop. 13.3.14,

$$[\![\texttt{r:=random}]\!]^*.\chi_A = x \mapsto \sum_{k=1}^n \int \chi_A(x,y)\ \mathrm{d}\mu_k(y). \tag{9.14}$$

But $\int \chi_A(x,y)\ \mathrm{d}\mu_k(y) \leq \mu_k(\pi_R(A))$, and $\int \chi_A(x,y)\ \mathrm{d}\mu_k(y) = 0$ if $x \notin \pi_X(A)$. Therefore

$$[\![\texttt{r:=random}]\!]^*.\chi_A \leq \mu_k(\pi_R(A)).\chi_{\pi_X(A)}. \tag{9.15}$$

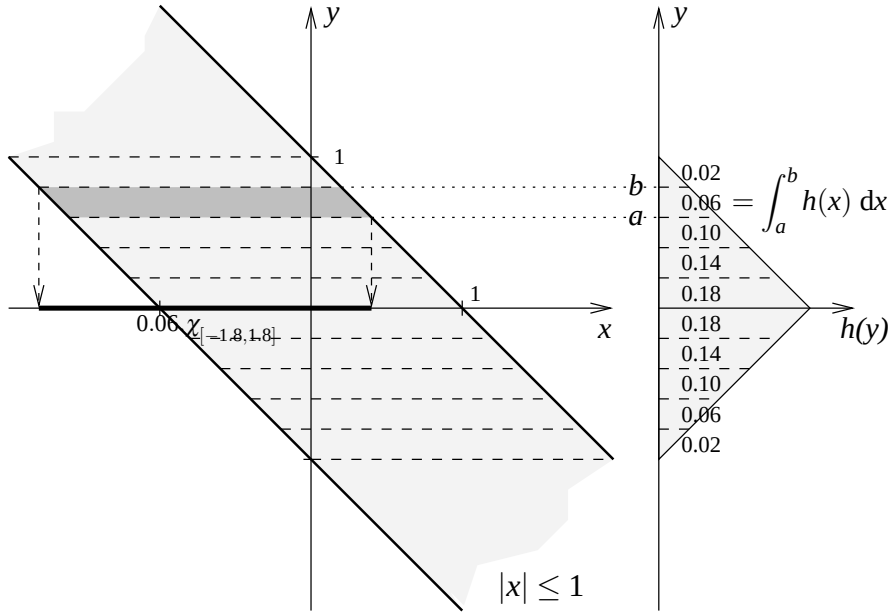


Figure 9.8: Construction of the output value of Fig. 9.9 for $n = 10$. The $|x+y| \leq 1$ abstract area is sliced along divisions along the y axis. Each slice S_k is projected onto the y axis and the integral of the distribution function h of `centered_uniform()+centered_uniform()` is taken on this projection, yielding a coefficient α_k. The slice is then projected on the x axis and this projection B_k, with the α_k coefficient is an element of the abstract value $\sum_{k=i}^{n} \alpha_k . \chi_{B_k}$. The approximations plotted in Fig. 9.9 are those sums, with various numbers of slices.

Lifting to abstract semantics is then easy: $[\![\texttt{r:=random}]\!]^{*\sharp}(A_i^\sharp, \alpha_i)_{1\leq i\leq m}$ maps to $(A_{i,k}^\sharp, \alpha_i.\beta_{i,k})_{1\leq i\leq m, 1\leq k\leq n}$, where $A_{i,k}^\sharp$ is an abstraction of $\pi_X(\gamma(A_i^\sharp) \cap (X \times M_k))$ and $\beta_{i,k} \geq \mu_k(\pi_R(A))$ (Fig. 9.8 explains how we built the approximations for Fig. 9.9). Both the $A_{i,k}^\sharp$ and the $\beta_{i,k}$ can be computed easily for many underlying abstract domains, such as the nondependent product of intervals [14].

Of course, there is some amount of choice in how to cut μ into μ_k. We suggest to cut into measures of equal weight. Of course, the higher the number of μ_k's, the better the immediate results (Fig. 9.9), nevertheless a high number of elements in abstract values may necessitate an early use of widenings. We hope the forthcoming implementation will help adjust such heuristic parameters.

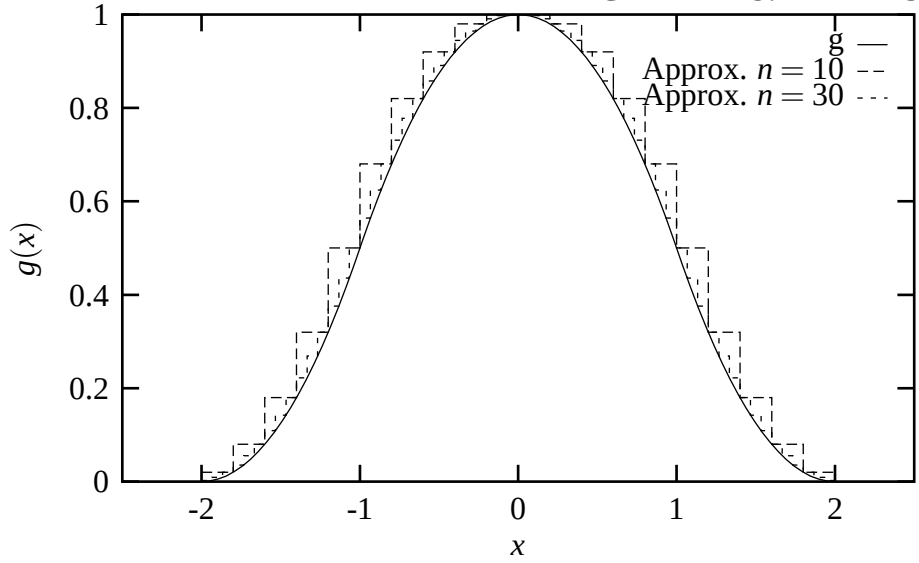


Figure 9.9: Two approximations of the actual potentiality function of Fig. 13.2, resulting from the abstract interpretation of the program of Fig. 13.1. Our generic lattice is parameterized by the product lattice of intervals. Different discretizations of the probability measure $h(x)\,\mathrm{d}x$ of `centered_uniform()+centered_uniform()` yield more or less precise abstractions. Here, the interval $[-1,1]$ where h is nonzero is divided into n segments of equal size, yielding n elements in the output abstract value.

Tests

The semantics for tests gets straightforwardly lifted to the abstract semantics, provided we have abstract operators for R_W and $+$:

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]^{*\sharp}_p.f^\sharp = R^\sharp_{[\![b]\!]} \circ [\![c_1]\!]^{*\sharp}_p.f^\sharp +^\sharp R^\sharp_{[\![b]\!]^C} \circ [\![c_2]\!]^{*\sharp}_p.f^\sharp \tag{9.16}$$

where $R_W : f \mapsto f.\chi_W$.

Abstracting $+$ is easy: $+^\sharp$ is the concatenation operator on sequences (or families); as for R_W:

$$R^\sharp_{W^\sharp} = (X^\sharp_\lambda, \alpha_\lambda)_{\lambda\in\Lambda} \mapsto (X^\sharp_\lambda \cap^\sharp W^\sharp, \alpha_\lambda)_{\lambda\in\Lambda} \tag{9.17}$$

Widening operators

We shall here propose a few heuristics for widenings. Widenings are not only useful in the computation of fixpoints, they also provide a way to "simplify" abstract elements, to save memory, even if such a simplification results in a loss of precision.

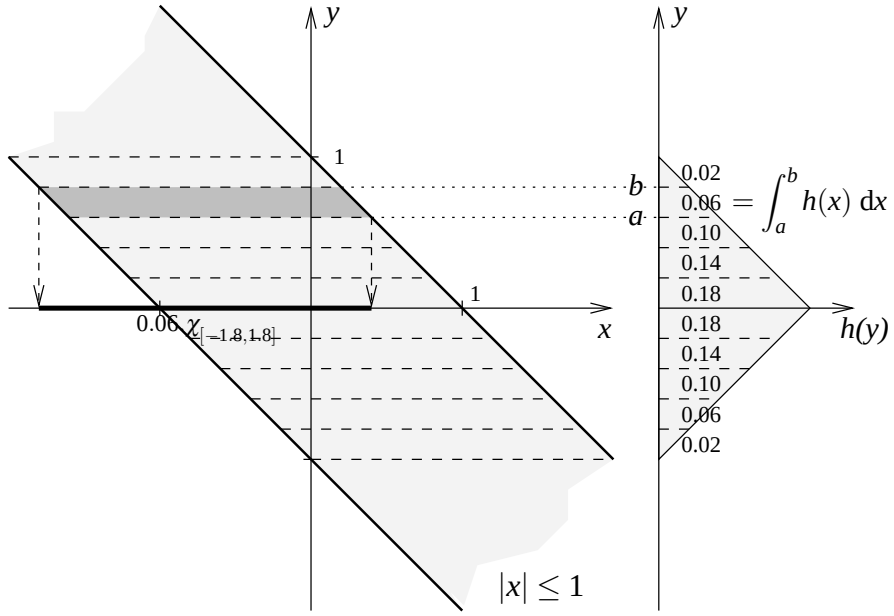


Figure 9.10: Construction of the output value of Fig. 9.9 for $n = 10$. The $|x+y| \leq 1$ abstract area is sliced along divisions on the y axis. Each slice S_k is projected onto the y axis and the integral of the distribution function h of `centered_uniform()+centered_uniform()` is taken on this projection, yielding a coefficient α_k. The slice is then projected on the x axis and this projection B_k, with the α_k coefficient is an element of the abstract value $\sum_{k=i}^{n} \alpha_k.\chi_{B_k}$. The approximations plotted in Fig. 9.9 are those sums, with various numbers of slices.

Let us suppose we have a widening sequence ∇_k on $X^\sharp$. We shall now give heuristics for computing $x\nabla_k y$ where $x = (X_i^\sharp, \alpha_i)_{1\leq i\leq a}$ and $y = (Y_j^\sharp, \beta_j)_{1\leq j\leq b}$. For the sake of simplicity, we shall suppose we aim at keeping Λ index sets less than n elements for a certain fixed n. We shall suppose that $a \leq n$.

The main idea is coalescing. For each element $(X_i^\sharp)$, find "close" elements $(Y_{j_{i,1}}^\sharp), \ldots, (Y_{j_{i,m}}^\sharp)$, the closeness criterion being largely heuristic and dependent on the chosen lattice; this criterion does not influence the correctness of the method, only its precision and efficiency. The $j_{i,m}$ are supposed to be pairwise distinct and to cover the whole range $1, \ldots, n$. We then define

$$x\nabla_k y = (X_i^\sharp \nabla_k \left(Y_{j_{i,1}}^\sharp \cup \cdots \cup Y_{j_{i,m}}^\sharp, \max(\alpha_i, \beta_{j_{i,1}} + \ldots + \beta_{j_{i,m}}) \right). \tag{9.18}$$

Let us now take a sequence $(x^{(k)})_{k\in\mathbb{N}}$ such that $x^{(k+1)} = x^{(k)}\nabla y^{(k)}$. Then for all $1 \leq$

$i \leq n$, $X_i^{(k+1)} = X_i^{(k)} \nabla v_i^{(k)}$ for some $v_i^{(k)}$, so the sequence $(X_i^{(k)})_{k\in\mathbb{N}}$ is stationary. Since this holds for all i, this means that $(x^{(k)})$ is stationary.

9.2.4 Using a finite nondeterministic domain

In the case where $X^\sharp$ is finite, we can consider our computations as a linear programming problem. First, instead of considering elements of $(X_w^\sharp, \sqsubseteq)$, we can consider them as finite vectors in $\left([0,1]^{X^\sharp}, \leq\right)$. Let us now notice that apart from the nondeterministic choice construct §9.2.3, all our operations are linear operations on those vectors (which is not surprising since they abstract linear operations on the weight functions). In §8.3.5, we saw a model for which the function to be iterated is

$$F(h) = \text{choice}^*_{Y_l} \circ \text{random}^*_{R_l} \circ (F_l)^*_p \left(\sum_{l'\in P} \phi^*_{\tau_l l'} \left(h(l', \bullet) \right) \right) \tag{9.19}$$

and thus, the abstract version is

$$(F^\sharp.h)(l) = \text{choice}^{*\,\sharp}_{Y_l} \circ \underbrace{\text{random}^{*\,\sharp}_{R_l} \circ (F_l)^{*\sharp}_p \left(\sum^\sharp_{l'\in P} \phi^{*\,\sharp}_{\tau_l l'} \left(h(l', \bullet) \right) \right)}_{L^\sharp} \tag{9.20}$$

where $L^\sharp$ is thus a linear operator. Let us now rewrite Equ. 9.12:

$$(\text{choice}^{*\,\sharp}_{Y_l}.f^\sharp)(Z^\sharp) = \sup_{\substack{M\subseteq K \\ \bigcap_{\lambda\in M} \gamma(X^\sharp_\lambda)\neq\emptyset}} \sum_{\lambda\in M} \alpha_\lambda \tag{9.21}$$

where

$$K = \{\lambda \mid \pi_1^\sharp(X^\sharp_\lambda) = Z^\sharp\}. \tag{9.22}$$

The whole $F^\sharp$ function can therefore be rewritten as the point-wise least upper bound of a finite number of linear functions $(L_i^\sharp)_i$:

$$F^\sharp(f^\sharp) = \sup_i L_i^\sharp(f^\sharp). \tag{9.23}$$

The abstract fixpoint problems can then be solved exactly using linear programming, as in §10.3.1. In the case of abstract state spaces of the form $\{0,1\}^N$, there exist algorithms for linear programming using MTBDDs [23, 44].

Let us additionally note that a particularly interesting construct is made when $X^\sharp$ is $\mathcal{P}(D)$, the power-set generated by a partition D of X : the abstract values can

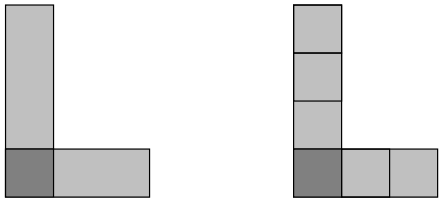


Figure 9.11: Simplification step. Instead of the sum of the characteristic functions of two elements of $\mathcal{P}([\![0,3]\!]^2)$, we rather consider a function $[\![0,3]\!]^2 \to [0,1]$ (thus assigning a value to each square of the grid).

be memorized as elements of $[0,1]^D$. A "simplification step" (Fig. 9.11) can be applied after each operation:

$$S : \left| \begin{array}{l} [0,1]^{\mathcal{P}(D)} \to [0,1]^D \\ f \qquad\quad \mapsto d \mapsto \sum_{\substack{W \in \mathcal{P}(D) \\ d \in W}} f(W) \end{array} \right. \tag{9.24}$$

There is a Galois connection $[0,1]^X \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} [0,1]^D$ where $\alpha(f)(d) = \sup_{x \in d} f(x)$.

9.2.5 Comparison with other methods

Let us first remark that our method is a natural extension of conventional backwards abstract interpretation. Indeed, let us consider only programs containing no `random`-like operations; any program, even including `random`-like operations, can be transformed into a program containing none by moving the streams of random numbers into the environment of the program (this corresponds to the first semantics proposed by Kozen [41, 42]).

With such programs, our framework is equivalent to computing reverse images of sets: $[\![c]\!]^*_p.\mu.\chi_W = \mu([\![c]\!]^{-1}(W))$ and our proposed abstract domain just expresses that $[\![c]\!]^*_p.\mu.\chi_W \leq \mu \circ \gamma \circ [\![c]\!]^{-1^\sharp}(W^\sharp)$. There is nevertheless a difference that makes our abstract domain more interesting: in the presence of streams of random numbers, our abstract domain just makes use of an ordinary abstract domain, while computing approximate reverse images in the presence of infinite streams requires an abstract domain capable of abstracting infinite sequences such that the results remain interesting.

9.3 Discussion

The abstract domains proposed in this chapter are highly generic; they are internally use non-probabilistic abstract interpretation domains. Their limitations are twofold:

- In order to get good precision, long tuples must be used, which leads to long computation times.
- While widening operators are available, their performance is highly heuristic.

Chapter 10

Related formalisms

10.1 Probabilistic Bisimulation

Larsen and Skou [47] proposed a notion of *probabilistic bisimulation*.

Definition 10.1.1 (probabilistic bisimulation). Let X be a countable set of states and $\rho : X \times X \to \mathbb{R}_+$ be a transition probability. An equivalence relation $\sim$ on X is said to be a probabilistic bisimulation if for all states c and d such that $c \sim d$ and any equivalence class S in ${}^{S}\!/_{\sim}$ then $\rho(c,S) = \rho(d,S)$. Let us then define ${}^{\rho}\!/_{\sim} : {}^{S}\!/_{\sim} \times {}^{S}\!/_{\sim} \to \mathbb{R}_+$ as follows: ${}^{\rho}\!/_{\sim}(S,S') = \rho(s,S')$ where $s \in S$.

Lemma 10.1.2. *Let ρ and τ be two transition probabilities and $\sim$ an adapted probabilistic bisimulation. Then $\rho \circ \tau$ is also a transition probability such that $\sim$ is an adapted probabilistic bisimulation and ${}^{\rho}\!/_{\sim} \circ {}^{\tau}\!/_{\sim} = {}^{\rho \circ \tau}\!/_{\sim}$.*

Proof. Let c and d be two $\sim$-equivalent states. Let $S \in {}^{X}\!/_{\sim}$. We then have

$$\begin{aligned}
&\rho \circ \tau(c,S) \\
&\qquad = \sum_{s' \in X} \tau(c,s').\rho(s',S) \\
&\qquad = \sum_{S' \in {}^{X}\!/_{\sim}} \sum_{s' \in S'} \tau(c,s') \underbrace{\rho(s',S)}_{{}^{\rho}\!/_{\sim}(S',S)} \\
&\qquad = \sum_{S' \in {}^{X}\!/_{\sim}} {}^{\rho}\!/_{\sim}(S',S). \underbrace{\sum_{s' \in S'} \tau(c,s')}_{\tau(c,S') = \tau(d,S')} \\
&\qquad = \rho \circ \tau(d,S) \qquad (10.1)
\end{aligned}$$

Therefore we can define $\rho \circ \tau_{/\sim}$ a transition probability on $X_{/\sim}$.

Let $s \in S$. Then

$$\begin{aligned}
&\rho_{/\sim} \circ \tau_{/\sim}(S,S') \\
&\qquad = \sum_{S'' \in X_{/\sim}} \underbrace{\rho_{/\sim}(S,S'')}_{\rho(s,S'')} . \tau_{/\sim}(S'',S') \\
&\qquad = \sum_{S'' \in X_{/\sim}} \sum_{s'' \in S''} \rho(s,s'') . \underbrace{\tau_{/\sim}(S'',S')}_{\tau(s'',S')} \\
&\qquad\qquad = \sum_{s'' \in X} \rho(s,s'') . \tau(s'',S') \\
&\qquad\qquad = \rho \circ \tau(s,S') = \rho \circ \tau_{/\sim}(S,S')
\end{aligned} \tag{10.2}$$

which establishes that $\rho_{/\sim} \circ \tau_{/\sim} = \rho \circ \tau_{/\sim}$. □

10.1.1 Probabilistic bisimulation as a form of abstract interpretation

Let us suppose there exists a probabilistic bisimulation $\sim$ on X compatible with a probabilistic transition relation ρ. Let us consider abstract elements in the flat lattice on the set $\mathbb{R}_+^{X/\sim}$. The concretization of an element $(\alpha_S)_{S \in X_{/\sim}}$ is the set of probability distributions ρ such that for all $S \in X_{/\sim}$, $\sum_{s \in S} \rho(s) = \alpha(S)$.

Let us define $\rho^\sharp \left((\alpha_S)_{S \in X_{/\sim}} \right) = \left(\sum_{S' \in X_{/\sim}} \rho_{/\sim}(S',S).\alpha_{S'} \right)_{S \in X_{/\sim}}$. Then $\rho^\sharp$ is an abstraction of $\rho^\flat$.

10.2 Probabilistic forward data-flow analysis

Ramalingam [65] introduced *data flow frequency analysis*. Here, traditional data-flow analysis (a particular case of abstract interpretation) is extended to cases where the control flow of the program is a Markov chain. We shall see in this chapter how to express such analyzes within our probabilistic abstract interpretation framework and how to generalize this approach.

10.2.1 Finite distributive non probabilistic data flow analysis

Let X be the set of states and $D \subseteq \mathcal{P}(X)$ a (finite) set of properties. We abstract $\mathcal{P}(X)$ by $\mathcal{P}(D)$ as follows:

$$\gamma_X : \left| \begin{array}{l} \mathcal{P}(D) \to \mathcal{P}(X) \\ P \quad \mapsto \bigcup_{p \in P} p \end{array} \right. \tag{10.3}$$

Obviously, elements of $\mathcal{P}(D)$ can be implemented as bit-vectors of length $|D|$.

We consider *distributive* abstract operations: $f^\sharp(P) = \bigcup_{p \in P} f^\sharp(\{p\})$. Let us note that if the elements of D are pairwise disjoint, or more generally if D is a (complete) lower sub-lattice of $\mathcal{P}(X)$, then $\mathcal{P}(D)$ is a (complete) lower semi-lattice and there exists a Galois connection $(\mathcal{P}(X), \subseteq) \underset{\alpha_X}{\overset{\gamma_X}{\leftrightarrows}} (\mathcal{P}(D), \subseteq)$. In this conditions, for any concrete distributive operation $f : \mathcal{P}(X) \to \mathcal{P}(X)$ there exists a distributive best abstraction $f^\sharp : \mathcal{P}(D) \to \mathcal{P}(D)$.

10.2.2 Probabilistic data-flow analysis

Abstract lattice

We wish to abstract the lattice $L = \mathcal{P}(\mathcal{M}_{\leq 1}(X))$ of sets of positive measures on X. We build a lattice $L^\sharp$ similar to the one in §9.1: $L^\sharp = [0, +\infty]^D$ where $+^\sharp$ is the point-wise sum, $\sqsubseteq$ is the point-wise comparison, $\sqcup$ is the point-wise maximum, $\sqcup$ is the point-wise minimum.

The abstraction between L and $L^\sharp$ is defined as follows: $\gamma(f)$ is the set of positive measures μ such that there exist measures $(\mu_d)_{d \in D}$ such that:

- the total weight of μ_d is less than $f(d)$;
- μ_d is located in d.

If the elements of D are pairwise disjoint, this condition is equivalent to $\forall d \in D, \mu(d) \leq f(d)$.

Semantics will be defined as pseudo-linear operators over vectors in $L^\sharp = D \to [0,+\infty]$: by "pseudo-linear" we mean that all operators H are such that for any μ and μ' in $L^\sharp$ and $\lambda \in \mathbb{R}_+$, $H.(\lambda.\mu^\sharp + \mu'^\sharp) = \lambda.H(\mu^\sharp) + H(\mu'^\sharp)$. As the vectors are finite, those operators can be represented as square matrices whose elements belong to the semiring $(\mathbb{R}_+, +, ., 0, 1)$ (no infinite elements appear in the operators).

Deterministic Operations

Let $f : X \to X$ be a deterministic operation and $f^\sharp : \mathcal{P}(D) \to \mathcal{P}(D)$ a (non-probabilistic) abstraction of f. The operator on measures associated to f is

$$f_p : \left| \begin{array}{ll} \mathcal{M}_{\leq 1}(X) \to \mathcal{M}_{\leq 1}(X) \\ \mu \qquad\quad \mapsto W \mapsto \mu(f^{-1}(W)) \end{array} \right. \tag{10.4}$$

An abstraction of this operator is

$$f_p^\sharp : \left| \begin{array}{l} L^\sharp \to L^\sharp \\ \mu^\sharp \mapsto d \mapsto \sum_{d' \in D | d \in f^\sharp(\{d'\})} \mu^\sharp(d) \end{array} \right. \tag{10.5}$$

Tests

Let $C \subseteq \mathcal{P}(X)$ be a (measurable) Boolean condition. We wish to abstract the function

$$\phi_C : \left| \begin{array}{ll} \mathcal{M}_{\leq 1}(X) \to \mathcal{M}_{\leq 1}(X) \\ \mu \qquad\quad \mapsto W \mapsto \mu(W \cap C). \end{array} \right. \tag{10.6}$$

An obvious abstraction is

$$\phi_C^\sharp : \left| \begin{array}{l} L^\sharp \to L^\sharp \\ \mu^\sharp \mapsto d \mapsto \begin{cases} 1 & \text{if } d \cap C \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \end{array} \right. \tag{10.7}$$

Combinations using least upper bounds

The problem is that, due to nondeterminism, we sometimes would like to consider a common upper approximation over several operators T_y. Let us suppose that $T_y^\sharp$ are respective linear approximations of forward transition probabilities $\overrightarrow{T_y}$. Of course, a safe common approximation is

$$\mu^\sharp \mapsto \bigsqcup_y T_y^\sharp(\mu^\sharp) \tag{10.8}$$

but this is not a linear operation. On the other hand, an obvious linear upper approximation is the coefficient-wise least upper bound of the $T_y^\sharp$: if the T_y are represented by matrices $t_{i,j}^{(y)}$, then we take the matrix whose coefficients are $(\sup_y t_{i,j}^{(y)})_{i,j}$.

Least fixpoints

As noted by Ramalingam [65, §5], the least fixpoint problem (used to solve reachability) within this framework amount tos solving a linear equation $A.\mu^\sharp + B = \mu^\sharp$ for its smallest solution in $D \to [0,+\infty]$. This is an algebraic path problem over the closed semiring $(\mathbb{R}_+, +, ., *, 0, 1)$ and can be solved by standard algorithms [69].

10.3 Backwards data-flow analysis

Discussion of positive bounded models [64, §7.2].

10.3.1 Linear programming

The technique of *value iteration* [64, §7.2.4] does convergence, but we do not quite know the speed of convergence: there are not iteratively improving bounds [64, §7.2.4]. We shall see here a techniques dealing with least-fixpoints that applies better since we are on a finite-dimensional space [12, 44].

Surprisingly, our problems of least fixpoints can also be solved using *linear programming* [64, §7.2.7], that is, finding effectively the minimum of a linear form on an extended convex polyhedron [37]. Our reachability problem $[\![\mathrm{lfp}(f \mapsto H +_A \mathrm{shift}(f))]\!]_{e+}$ (where H is some function) is the least solution in $[0,1]^D$ of the following *Bellman equation*:

$$X = \chi_A + \chi_{A^C} \sup_{T \in \mathscr{T}} \overleftarrow{T}.X \tag{10.9}$$

From theorem A.2.10, this solution is also the minimal element X of $[0,1]^D$ that satisfies the following inequation:

$$X \geq \chi_A + \chi_{A^C} \sup_{T \in \mathscr{T}} \overleftarrow{T}.X \tag{10.10}$$

coordinate-wise. Let us recall that we can restrict ourselves to deterministic policies: $\mathscr{T}$ is the set of transition probabilities Δ from D to D such that there exists a function $f : D \to Y$ such that $\Delta(x,x') = T(x, f(x); x')$. We can therefore rewrite the above inequation as follows:

$$\forall d \in D \; X_d \geq 0 \tag{10.11}$$

$$\forall d \in D \; \forall y \in Y \; X_d \geq \chi_A(d) + \chi_{A^C}(d) \sum_{d' \in D} T(d,y;d').X_{d'} \tag{10.12}$$

Let us remark that there are $|D| + |Y|.|D|$ inequations in the system.

Chapter 11

Sub-Exponential Queues

11.1 Basic Abstract Domain and Abstract Operations

We wish to represent sets of (sub)probability measures symbolically. In this section, we give a basic abstract domain expressing exponentially decreasing tails of probabilistic distributions on integers. This abstract domain is not very satisfactory by itself (it is very crude when it comes to loops), but is a basis for the definition of a more complex domain (section 11.2).

11.1.1 Abstract Domain

Let V be the set of variables. Each element of the abstract domain E is a tuple of coefficients. Those coefficients will represent three kinds of known facts on the measures:

- an upper bound W on their total weight;
- upper bounds on the intervals of probable variation for the integer variables: for any variable v, the probability that v is outside $[a_v, b_v]$ is 0; of course, a_b and/or b_v can be infinite ($a_v \leq b_v$);
- for each integer variable v, some data C_v on its exponential decreasing: either none or a pair $(\alpha_v, \beta_v) \in \mathbb{R}_+ \times [0,1[$ meaning that the probability that variable v is k is bounded by $\alpha_v \beta_v^k$.

$\gamma_E(W, (a_v, b_v)_{v\in V}, (C_v)_{v\in V})$ is the set of all measures matching the above conditions. We shall note $\mu(condition)$ for the application of the measure μ to the set

of environments matching condition *condition*. The three conditions above then get written as:

$$\mu \in \gamma_E(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}) \iff \begin{cases} \mu(\mathsf{true}) \leq W \\ \forall v \in V\ \mu(v \notin [a_v,b_v]) = 0 \\ \forall v \in V\ C_v = (\alpha_v,\beta_v) \Rightarrow \forall k \in \mathbb{Z}\ \mu(v=k) \leq \alpha_v \beta_v^k \end{cases} \tag{11.1}$$

11.1.2 Arithmetic operations

We do not provide an abstract operator for each of the basic operations that a program may encounter; for instance, we say nothing of multiplication. In cases that are not described, we just apply interval propagation [15] and set $C_v = \mathsf{none}$ for every modified variable v. In some case, we shall provide only for some cases, while some others can be handled by using the symmetry of the operation and reverting to a described case.

We focus on the operations that will be most useful for our analysis goals (number of iterations taken in a loop, number of used CPU cycles). For instance, we consider the case of the arithmetic plus since it will be used to count loop iterations or program cycles, whereas multiplication is of little use for such tasks.

Arithmetic plus

We define here the abstract operation $(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}) \mapsto (W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V}) = \mathsf{z := x+y}^\sharp_p.(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V})$.

The distribution after applying an arithmetic plus obeys the following convolution equation:

$$([\![\mathsf{x+y}]\!]_p.\mu)(\mathsf{z}=t) = \sum_{k\in\mathbb{Z}} \mu(\mathsf{x}=k \wedge \mathsf{y}=t-k). \tag{11.2}$$

Let us suppose that $\mu \in \gamma_E(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V})$; we want to produce $(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})$ such that $([\![\mathsf{x+y}]\!]_p.\mu \in \gamma_E(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})$.

Obviously we can take $W'=W$, $a'_\mathsf{z} = a_\mathsf{x} + a_\mathsf{y}$, $b'_\mathsf{z} = b_\mathsf{x} + b_\mathsf{y}$, and $b'_v = b_v$ and $C'_v = C_v$ for all $v \neq \mathsf{z}$.

We therefore have four cases:

- $C_x = \mathsf{none}$ and $C_y = \mathsf{none}$, then $C'_z = \mathsf{none}$;
- $C_x = \mathsf{none}$ and $C_y = (\alpha_y,\beta_y)$. We then have $\mu(x=k\wedge y=t-k) \leq \alpha_y\beta_y^{t-k}$ if $k \in [a_x,b_x]$ $\mu(x=k\wedge y=t-k)=0$ otherwise. Inequality 11.2 then yields $\mathbb{P}(x+y=t) \leq \alpha_y \sum_{k=a_x}^{b_x} \beta_y^{t-k}$.

Let $\alpha'_z = \alpha_y \beta_y^{-b_x} \frac{\beta_y^{b_x - a_x + 1} - 1}{\beta_y - 1}$ and $\beta'_z = \beta_y$. In particular, if $b_x = a_x$ (variable x is actually a constant), then $\alpha'_z = \alpha_y \beta_y^{-b_x}$.

Then $([\![\mathsf{x+y}]\!]_p.\mu)(\mathsf{z} = t) \leq \alpha'_z \beta'^t_z$. If $\alpha'_z = \infty$, we take $C'_z = \mathsf{none}$ else we take $C'_z = (\alpha'_z, \beta'_z)$.

- $C_x = (\alpha_x, \beta_x)$ and $C_y = \mathsf{none}$; this is *mutatis mutandis* the previous case.
- $C_x = (\alpha_x, \beta_x)$ and $C_y = (\alpha_y, \beta_y)$; we then apply the previous cases and take the greatest lower bound of both.

11.1.3 Random Generation

We define here the abstract operation $(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \mapsto (W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V}) = \rho \texttt{ := random}^\sharp_p.(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$.

Let us recall that $\rho\texttt{:=random}_p.\mu = \mu \otimes \mu_R$ where μ_R is the distribution of the generator. Let us note W_R the total weight of μ_R (it can be less than 1, see §11.2.3). We take $W' = W_R.W$, and for any variable v except ρ, $a'_v = a_v$ and $b'_v = b_v$; if $C_v = (\alpha_v, \beta_v)$ then $C'_v = (W_R.\alpha_v, \beta_v)$, else $C'_v = \mathsf{none}$. If the generator has an integer output and is bounded in $[a_R, b_R]$, then $a'_\rho = a_R$ and $b'_\rho = b_R$. $C'_\rho = \mathsf{none}$.

11.1.4 Flow control

As with preceding operations, we only define the cases that will be actually used for our analysis goals. Other cases are handled by simple interval propagation and setting C_v to none for every modified variable v.

Least upper bound

We define here the abstract operation

$$\big((W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}), (W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V})\big) \mapsto (W'', (a''_v, b''_v)_{v \in V}, (C''_v)_{v \in V})$$

noted as

$$(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \sqcup^\sharp (W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V}).$$

Given $(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V})$ and $(W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V})$, we want to obtain $(W'', (a''_v, b''_v)_{v \in V}, (C''_v)_{v \in V})$ such that $\gamma_E(W, (a_v, b_v)_{v \in V}, (C_v)_{v \in V}) \cup \gamma_E(W', (a'_v, b'_v)_{v \in V}, (C'_v)_{v \in V}) \subseteq \gamma_E(W'', (a''_v, b''_v)_{v \in V}, (C''_v)_{v \in V})$ with little loss of precision. Let us take $W'' = \max(W, W')$ and for all v:

- $a''_v = \min(a_v, a'_v)$ and $b''_v = \max(b_v, b'_v)$

- if $C_v = (\alpha_v, \beta_v)$ and $C'_v = (\alpha'_v, \beta'_v)$ then we take $\beta''_v = \max(\beta_v, \beta'_v)$ and $\alpha''_v = \max(\alpha_v \beta_v^{a''_v}, \alpha'_v \beta_v'^{a''_v}).\beta''^{-a''_v}_v$.
- if $C_v = \mathsf{none}$ or $C'_v = \mathsf{none}$ then $C''_v = \mathsf{none}$.

Widening

We shall define here the abstract operation mapping $\big((W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}),(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})\big)$ to $(W'',(a''_v,b''_v)_{v\in V},(C''_v)_{v\in V})$ noted as

$$(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V})\nabla_E(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V}).$$

Given $(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V})$ and $(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})$, we want $(W'',(a''_v,b''_v)_{v\in V},(C''_v)_{v\in V})$ such that

$$\gamma_E(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V})\cup\gamma_E(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})\subseteq\gamma_E(W'',(a''_v,b''_v)_{v\in V},(C''_v)_{v\in V}).$$

We shall note this operation $\sqcup^\sharp$. We also want that for any sequence $(\mu_n^\sharp)_{n\in\mathbb{N}}$, the sequence defined inductively by $\mu'_{n+1} = \mu'_n \sqcup^\sharp \mu_n^\sharp$ is stationary.

We shall use a widening operator $\nabla_{\mathbb{R}}$ on the reals:

- $x\nabla_{\mathbb{R}}y = \infty$ if $x < y$;
- $x\nabla_{\mathbb{R}}y = y$ otherwise.

Let us take $W'' = \max(W, W')$ and for all v:

- if $a_v > a'_v$, $a''_v = -\infty$ else $a''_v = a_v$;
- if $b_v < b'_v$, $b''_v = +\infty$ else $b''_v = a_v$;
- two cases:
  - If $a''_v = +\infty$ or $C_v = \mathsf{none}$ or $C'_v = \mathsf{none}$ then $C''_v = \mathsf{none}$.
  - Otherwise, if $C_v = (\alpha_v, \beta_v)$ and $C'_v = (\alpha'_v, \beta'_v)$ then we take

$$\beta''_v = \exp(-(-\ln\beta_v)\nabla_{\mathbb{R}}(-\ln\beta'_v))$$

and

$$\alpha''_v = (\alpha_v\beta_v^{a''_v}\nabla_{\mathbb{R}}\alpha'_v\beta_v'^{a''_v})).\beta''^{-a''_v}_v.$$

If $\alpha''_v < \infty$ then $C''_v = (\alpha''_v, \beta''_v)$, otherwise $C''_v = \mathsf{none}$.

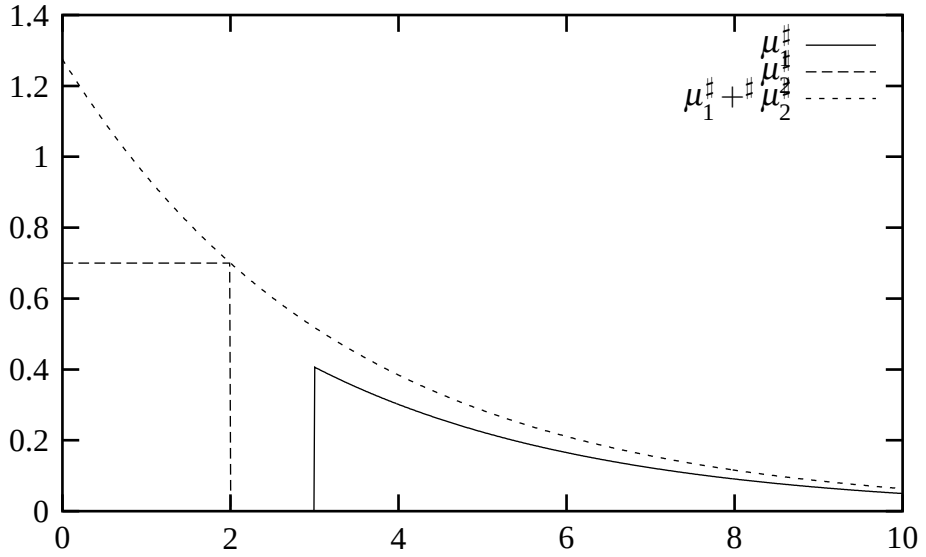


Figure 11.1: Abstract addition of measures. $C_x = (1, 0.3)$, $a_x = 3$, $b_x = +\infty$, $W' = 0.7$, $C'_x = \mathsf{none}$, $a'_x = 0$, $b'_x = 2$. For the sake of readability, the discrete distributions are extended to continuous ones. The near-vertical slopes are artefacts of the plotting software replacing vertical slopes.

Addition

We shall define here the abstract operation mapping $\big((W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}),(W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V})\big)$ to $(W'',(a''_v,b''_v)_{v\in V}, (C''_v)_{v\in V})$ noted as

$$(W,(a_v,b_v)_{v\in V},(C_v)_{v\in V}) +^\sharp (W',(a'_v,b'_v)_{v\in V},(C'_v)_{v\in V}).$$

Two cases are of particular interest:

- For all $t \in \mathbb{Z}$, $\mu(v=t) \leq \alpha_v \beta_v^t$, $t \notin [a_v,b_v] \Rightarrow \mu(v=t)$, $t \notin [a'_v,b'_v] \Rightarrow \mu'(v=t)$ and $\mu'(\text{true}) \leq W'$.

  Two cases:

  - $b'_v < a'_v$; then let us take $\alpha''_v = \max(\alpha_v, W'.\beta_v^{-b'_v})$ and $\beta''_v = \beta_v$, then $\mu + \mu(t=v) \leq \alpha''_v \beta''^t_v$ (see Fig. 11.1 for an example);
  - otherwise, let us take $\alpha''_v = \alpha_v, W' + \beta_v^{b'_v}$ and $\beta''_v = \beta_v$, then $\mu + \mu(t=v) \leq \alpha''_v \beta''^t_v$.

- For all $t \in \mathbb{Z}$, $t \notin [a_v,b_v] \Rightarrow \mu(v=t)$, $t \notin [a'_v,b'_v] \Rightarrow \mu'(v=t)$, where $a'_v > b_v$.

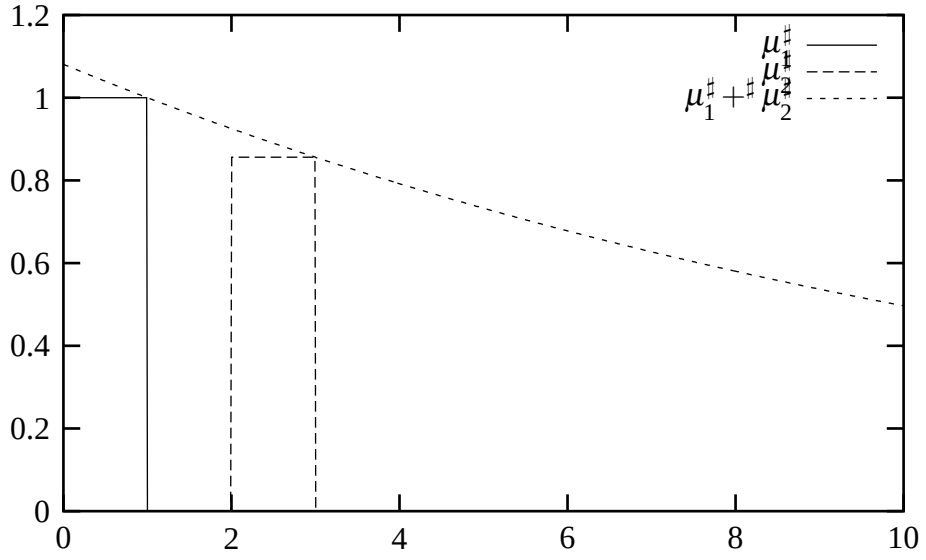


Figure 11.2: * Abstract addition of measures. $C_x = \mathsf{none}$, $a_x = 0$, $b_x = 1$, $W = 1$, $a'_x = 2$, $b'_x = 3$, $W' = 0.856$.

Let us take $\beta''_v = (W/W')^{\frac{1}{b'_v - b_v}}$ and $\alpha''_v = W.\beta''^{b_v}_v$, then $\mu + \mu(t = v) \leq \alpha''_v \beta''^t_v$ (see Fig. 11.2 for an example).

11.1.5 Machine Reals in our Abstract Domain

Our abstract domain makes ample use of real numbers: each coefficient α_v or β_v is *a priori* a real number. A possible implementation of these coefficients is machine reals (IEEE 754 or similar); another is rational numbers as quotients of arbitrary-precision integers. We discuss here the possible consequences of the use of machine reals for the implementation of those coefficients.

The use of machine reals leads to two implementation problems. The first problem is that it is difficult to ascertain the loss of precision induced by machine reals throughout computations: how can the user be convinced that the output of the analyzer is sound? This can be worked around by using directed rounding modes.

A more annoying problem, relevant in implementation, is the fact that accrued imprecisions may lead to "drift", especially when using directed rounding modes. By "drift", we mean that a sequence of real numbers that, mathematically speaking, should appear to be stationary may be strictly ascending in the floating-point approximation. In that case, our widening operator on the reals $\nabla_{\mathbb{R}}$ (§11.1.4) may jump prematurely to $+\infty$.

It may be desirable to consider that small changes in some real coefficients do not indicate that the coefficient is actually changing but rather indicate some loss of precision. We expect the current work on abstract domains for real numbers [67] to provide better solutions to that problem.

11.2 Abstract Domain of Finite Sums

The preceding domain is not yet very suitable to handle random generation. In this section, we lift it to the domain of its finite sums. This is similar to [54, section 4].

11.2.1 Definition

We shall define another domain S and another concretization function γ_S. S consists of finite tuples of elements of E or, more interestingly, of a reduced product [18, §4.2.3.3] P of E and another domain D. For our subsequent examples, we shall use for P the product of E and the lattice of real intervals (for real variables). γ_P is the concretization function for this reduced product: $\gamma_P(e^\sharp, d^\sharp) = \gamma_E(e^\sharp) \cap \gamma_D(d^\sharp)$.

As a running example we shall use for D the lifted domain of real intervals: to each real variable v we attach an interval $[a_v, b_v]$, with possibly infinite bounds. The probability that variable v is outside $[a_v, b_v]$ is zero.

Items of S are therefore finite tuples of elements of P. The concretization function γ_S is defined as follows: $\mu \in \gamma_S(p_1^\sharp, ..., p_n^\sharp)$ if and only if there exist $\mu_1 \in \gamma_P(p_1)$, ..., $\mu_n \in \gamma_P(p_n)$ such that $\mu = \sum_{k=1}^{n} \mu_k$.

11.2.2 Arithmetic operations

Deterministic operations

Basic operations are handled by linearity: since for any program construct P its semantics $[\![P]\!]_p$ is a linear operator, it follows that if $\mu \in \gamma_S(p_1^\sharp, ..., p_n^\sharp)$ then $[\![P]\!]_p.\mu \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp)$.

The abstract operator is therefore constructed:

$$[\![P]\!]_p^\sharp(p_1^\sharp, ..., p_n^\sharp) = ([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp).$$

11.2.3 Random operations

Let us consider a random generator G operating according to the distribution μ_R. The semantics of the random operation is $[\![\texttt{random}]\!]_p.\mu = \mu \otimes \mu_R$ (Equ. 13.1). We shall suppose that our underlying domain P has an abstract operation $\otimes^\sharp$.

Let us suppose that $\mu_R = \sum_{i=1}^{N} \mu_i$. Then $\mu \otimes \mu_R = \sum_{i=1}^{N} \mu \otimes \mu_i$ thus if $\mu \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp)$ then $\mu \otimes \mu_R \in \gamma_S([\![P]\!]_p^\sharp.p_1^\sharp \otimes^\sharp \mu_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_1^\sharp \otimes^\sharp \mu_N^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp \otimes^\sharp \mu_1^\sharp, \ldots, [\![P]\!]_p^\sharp.p_n^\sharp \otimes^\sharp \mu_N^\sharp)$.

As an example, let us consider the case of an uniform random generator in $[0,1]$. Let us "cut" it into N equal sub-segments: let us note μ_R the uniform measure on $[0,1]$. $\mu_R = \sum_{i=1}^{N} \mu_i$ with $\mu_i(X) = \mu_R(X \cap [(i-1)/N; i/N])$. For the abstraction of $\mu \mapsto \mu \otimes \mu_i$ over our example for the domain P: $(e, (d_1, \ldots, d_n)) \mapsto (e, (d_1, \ldots, d_n, [(i-1)/N; i/N])$.

11.2.4 Flow control

$[\![\texttt{while } c \texttt{ do } e]\!]^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}(\mathrm{lfp}^\sharp X^\sharp \mapsto W^\sharp \sqcup [\![e]\!]^\sharp(\phi^\sharp_{[\![c]\!]}(X^\sharp)))$.

Addition

Addition is very simple:

$$(p_1, \ldots, p_n) +^\sharp (p'_1, \ldots, p'_{n'}) = (p_1, \ldots, p_n, p'_1, \ldots, p'_{n'}) \tag{11.3}$$

Loops

For accuracy reasons, we wish $[\![\texttt{while } b \texttt{ do } e]\!]^\sharp.(p_1^\sharp, \ldots, p_n^\sharp)$ to operate separately on each component $p_1^\sharp, \ldots, p_n^\sharp$. We thus define an auxiliary function $H^\sharp$ such that $[\![\texttt{while } b \texttt{ do } e]\!]^\sharp.(p_1^\sharp, \ldots, p_n^\sharp) = (H^\sharp.p_1^\sharp, \ldots, H^\sharp.p_n^\sharp)$. $H^\sharp$ is defined following equation 13.4:

$$H^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}(\mathrm{lfp}^\sharp X^\sharp \mapsto \mathsf{merge}(W^\sharp \sqcup W^\sharp +^\sharp [\![e]\!]^\sharp(\phi^\sharp_{[\![c]\!]}(X)))).$$

where $\mathsf{merge}(p_1, \ldots, p_n) = (p_1 +^\sharp \cdots +^\sharp p_n)$.

We construct the "approximate least fixpoint operation" $\mathrm{lfp}^\sharp(f^\sharp)$ as follows: we consider the sequence $u_0^\sharp = (0)$, $u_{n+1}^\sharp = u_n^\sharp \nabla_P f^\sharp(u_n^\sharp)$. All elements of the sequences are 1-tuples of elements of P. If ∇_P is a widening operator for the domain P, then this sequence is stationary, and its limit is an approximation of the least fixpoint of $f^\sharp$.

Such a ∇_P operator can be defined as follows: $(e,d)\nabla_P(e',d') = (e\nabla_E e', d\nabla_D d')$ where ∇_E is the widening operator defined at §11.1.4.

11.3 Examples and Applications

11.3.1 Proof of Probabilistic Termination

Let us consider the following program:

```
double x, y;
int k = 0;
do
{
  x = uniform()+uniform();
  y = uniform();
  k++;
}
while(x < 1.4 || y < 0.2)
```

`uniform()` is assumed to be a random generator uniformly distributed in $[0,1]$. In this simple case, the probability that the loop is taken is independent of the input data and of the number of iterations done so far. Furthermore, we can compute it mathematically (0.856), with our knowledge of the distribution of the random variables and the computation of an integral.

Let us examine here what happens if we apply the above method, dividing the random generator into $N = 20$ sub-segments of equal length (§11.2.3). At the end of the first iteration, after the merge, the analyzer establishes that the probability of reaching 1 iteration is less than 0.8805.[1] Applying the last case of §11.1.4, we obtain $a_k = 1$, $b_k = 2$, $\beta_k = 0.8805$, $\alpha_k \approx 1.1357$. After another iteration, we obtain $a_k = 1$, $b_k = 3$, $\beta_k = 0.8805$, $\alpha_k \approx 1.1357$. The analyzer widens to $a_k = 1$, $b_k = \infty$, $\beta_k = 0.8805$, $\alpha_k \approx 1.1357$, which is stable.

We have therefore established that the probability that $k = x$ at the beginning of the body of the loop is less than 0.8805^{k-1}. That is of course not as good as the exact computation, but still offers a reasonable bound.

11.3.2 Statistical Inference of Average Running Times

It is often needed to derive statistical facts such as average running times for real-time systems. Results achieved by the preceding method can be too rough to give precise estimates; nevertheless, they can help in making mathematically accurate some experimental statistical results.

Intuitively, to get the average running time of a system, we should take a large number n of random samples $(x_k)_{1 \le k \le n}$ for the input (distributed according to the

[1] This result has been computed automatically by the Absinthe abstract interpreter `http://cgi.dmi.ens.fr/cgi-bin/monniaux/absinthe`.

supposed or inferred distribution on the inputs of the system). Let us call $R(x)$ the running time on input x; we then hope to have an approximation of the means $\bar{R}$ by taking $\frac{1}{n}\sum_{k=1}^{n} R(x_k)$.

Unfortunately, this method is not mathematically sound, essentially because R is not bounded, or rather because we have no information as to the very large values of R (the "tail" of the distribution). For instance, let us suppose that R is 0 for 99.99% of the input values and V for 0.01% of the input values. With 90% probability, the experimental average obtained by the Monte-Carlo method using 1000 samples will be 0, whereas the correct value is $V/10000$. As V can be arbitrarily large, this means we cannot gain any confidence interval on the result.

On the other hand, if, using the analysis described in this paper, we can show that the tail of the distribution decreases exponentially, we can get a confidence bound. Let us note $\mathbb{P}(A)$ the probability of an event A. Let us suppose that our analysis has established that the probability of running a loop at least k times is less than $\alpha.\beta^k$. Let $N \geq 1$. Let us run the program n times, stopping each time when either the loop terminates or it has been executed N times. Let us consider the random variable R where $R(x)$, the number of iterations the program takes when run on input x. We wish to determine $\bar{R} = \sum_{k=0}^{\infty} k.\mathbb{P}(R=k)$. To achieve this, we split the sum in two parts: $\bar{R} = \bar{R}_{\leq N} + \bar{R}_{>N}$ where $\bar{R}_{\leq N} = \sum_{k=0}^{N} k.\mathbb{P}(R=k)$ and $\bar{R}_{>N} = \sum_{k=N+1}^{\infty} k.\mathbb{P}(R=k)$.

We wish to obtain an upper bound on $\bar{R}_{\geq N}$.

$$\sum_{k=a}^{b} k.\beta^k = \frac{1}{(\beta-1)^2}.\left[\beta^{b+1}\left((b+1)(\beta-1)-\beta\right) - \beta^a\left(a(\beta-1)-\beta\right)\right] \quad (11.4)$$

and thus

$$\sum_{k=a}^{\infty} k\beta^k = \frac{\beta^a}{(\beta-1)^2}.(\beta - a(\beta-1)) \quad (11.5)$$

Therefore $\bar{R}_{>N} \leq \alpha.\frac{\beta^{N+1}}{(\beta-1)^2}.(\beta-(N+1)(\beta-1))$.

On the other hand, $\bar{R}_{<N}$ can be estimated experimentally [56]. Let us consider the random variable R^*: $R^*(x) = 0$ if $R(x) > N$ and $R^*(x) = R(x)/N$ if $R(x) < N$; this is a random variable whose range is a subset of $[0,1]$. It is therefore possible to apply to R^* the various techniques known for estimating the expectation of such random variables, such as Chernoff-Hoeffding bounds (see [32] [75, inequality A.4.4] or inequation B.1) or simple Gaussian confidence intervals, using the Central Limit Theorem. We then obtain $\bar{R}_{<N} = N.\bar{R}^*$.

We therefore obtain a confidence upper bound on $\bar{R}$.

Chapter 12

Gaussian Distributions

12.1 Parabolas

12.1.1 Common lower bound

Let $P_1(x) = a_1x^2 + b_1x + c_1$ and $P_2(x) = a_2x^2 + b_2x + c_2$. Let us find a quadric polynomial $P_3(x) = a_3x^2 + b_3x + c_3$ less than P_1 and P_2. Obviously, a_3 must be less than both a_1 and a_2, else P_3 is above P_1 or P_2 near $\pm\infty$.

Let us first suppose that neither $P_1 \leq P_2$ nor $P_2 \leq P_1$, since those cases have an obvious solution. Let us remark that this condition holds if and only if P_1 and P_2 intersect, that is, when $\text{discr}(P_1 - P_2) > 0$.

Let us first note that there is in general no "greatest lower bound" among quadratic polynomials. Indeed, let us consider $P_1(x) = (x-1)^2$ and $P_2(x) = (x+1)^2$. If we try $a_3 \to a_1 = a_2 = 1$, we get parabolas whose lower point goes to ∞ (see Fig. 12.2.

The first choice, an arbitrary one, will thus be of a_3 less than a_1 and a_2.

We choose P_3 to be tangent to both P_1 and P_2. This means that $P_3 - P_1$ and $P_3' - P_1'$ have a common root (resp. for $P_3 - P_2$ and $P_3' - P_2'$). This is equivalent to $P_3 - P_1$ and $P_2 - P_1$ each having a double root. That property is ensured by the conditions on the discriminants of the polynomials: $\text{discr}(P_3 - P_1) = 0$ and $\text{discr}(P_3 - P_2) = 0$, that is:

$$(b_3 - b_1)^2 = 4(a_3 - a_1)(c_3 - c_1) \tag{12.1}$$
$$(b_3 - b_2)^2 = 4(a_3 - a_2)(c_3 - c_2) \tag{12.2}$$

Let us suppose for now that $a_1 > a_2$. Solving this 2-unknown, 2-equation

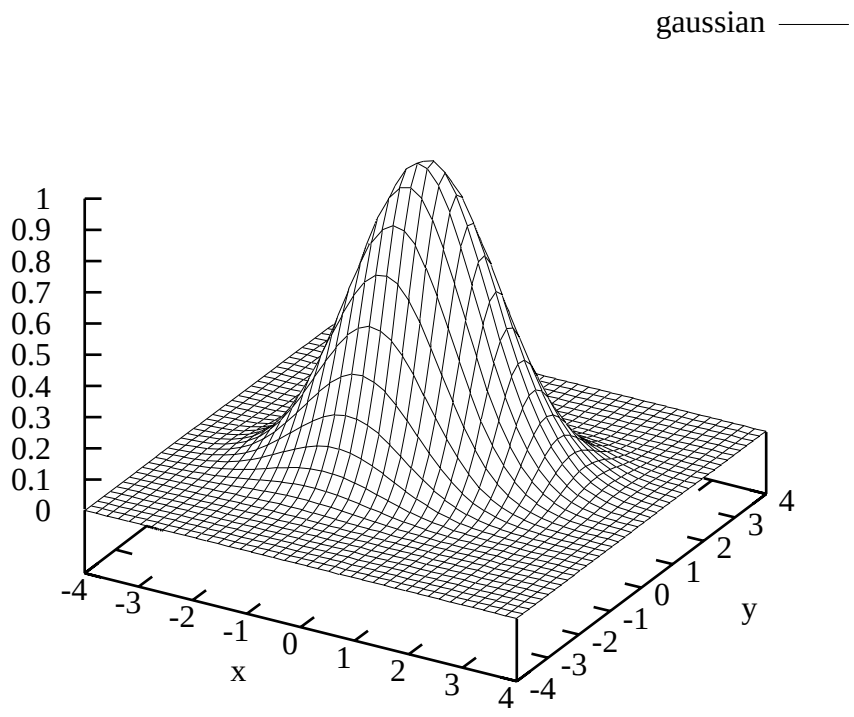


Figure 12.1: An extended Gaussian distribution. Its matrix is $\begin{pmatrix} 0.6 & 0 \\ 0 & 1 \end{pmatrix}$ in the orthogonal basis $\begin{pmatrix} \cos 0.4 & -\sin 0.4 \\ \sin 0.4 & \cos 0.4 \end{pmatrix}$.

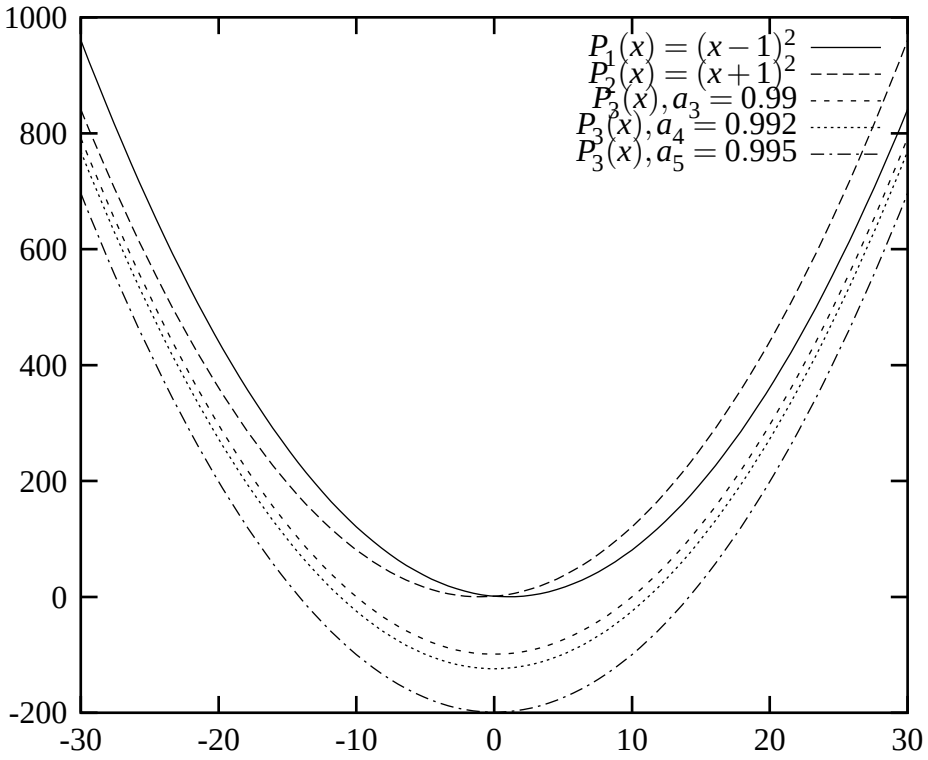


Figure 12.2: No greatest lower bound in parabolas for $P_1(x) = (x-1)^2$ and $P_2(x) = (x+1)^2$. It is impossible to optimize both for the infinite branches and the middle of the curve.

system yields:

$$b_3 = \frac{-a_2 b_1 + a_3(b_1 - b_2) + a_1 b_2 \pm \sqrt{\Delta}}{a_1 - a_2} \tag{12.3}$$

$$\Delta = (a_1 - a_3)(-a_2 + a_3)(-(b_1 - b_2)^2 + 4(a_1 - a_2)(c_1 - c_2)) \tag{12.4}$$

We aim to maximize $\inf P_3 = -\frac{b_3}{2a_3}$, therefore we minimize b. Since $a_1 > a_2$, we choose

$$b_3 = \frac{-a_2 b_1 + a_3(b_1 - b_2) + a_1 b_2 - \sqrt{\Delta}}{a_1 - a_2} \tag{12.5}$$

$$c_3 = c_1 + \frac{(b_3 - b_1)^2}{4(a_3 - a_1)} \tag{12.6}$$

The case $a_1 < a_2$ is treated *mutatis mutandis*.

Let us now treat $a_1 = a_2$, which is a degenerate case.

$$b_3 = \frac{b1 + b2}{2} - 2\frac{(a1 - a3)(c1 - c2)}{b1 - b2} \tag{12.7}$$

$$c_3 = c_1 + \frac{(b_3 - b_1)^2}{4(a_3 - a_1)} \tag{12.8}$$

12.2 Positive quadratic forms

12.2.1 Elementary facts

Proposition 12.2.1 (Cauchy-Schwarz). *Let Q be a positive quadratic form. Then for any x and y,*

$$Q^*(x,y) \leq \sqrt{Q(x)Q(y)}.$$

Proof. Let $\lambda \in \mathbb{R}$. $Q(x + \lambda y) = \lambda^2 Q(y) + 2\lambda Q^*(x,y) + Q(x)$. Since Q is positive, this quadric polynomial is positive for all λ. This polynomial therefore cannot have separate zeroes and thus its discriminant $4Q^*(x,y)^2 - 4Q(x)Q(y)$ is no greater than zero. □

Corollary 12.2.2. *Let Q be a positive quadratic form. Its isotropic cone Iso$Q$ is then equal to its kernel* $\ker Q$.

Proof. Obviously $\ker Q \subseteq \mathrm{Iso}Q$. Let us now prove the reverse inclusion. Let $x \in \mathrm{Iso}Q$ and $y \in E$. By the Cauchy-Schwarz inequality, $|Q^*(x,y)| \leq \sqrt{\underbrace{Q(x)}_{0} Q(y)}$ and thus $Q^*(x,y) = 0$. Therefore $x \in \ker Q$. □

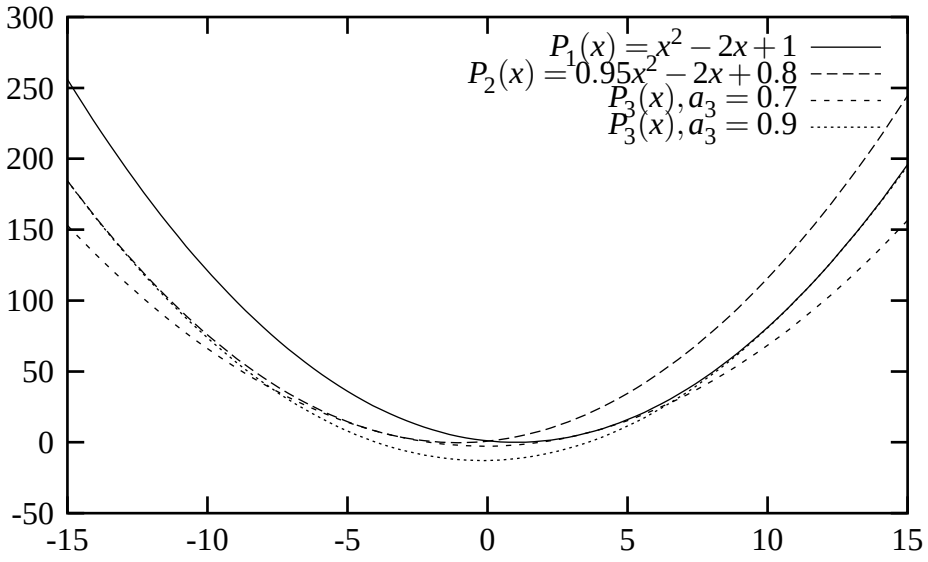


Figure 12.3: An example of common lower bounds in quadratic polynomials (P_4 and P_3 for P_1 and P_2).

Lemma 12.2.3. *Let E be a vector space. Let Q be a quadratic form over E. Let F be a finite dimension subspace of E so that $F \cap IsoQ = \{0\}$. Then F has an orthonormal basis.*

Lemma 12.2.4. *Let E be a finite dimension vector space. Let Q be a quadratic form over E. Let F be a subspace of E so that $F \cap IsoQ = \{0\}$. Then $E = F \oplus F^\perp$ where $F^\perp$ is the orthogonal of F with respect to Q.*

Proof. Let us first show that $F \cap F^\perp = \{0\}$. Let x in $F \cap F^\perp$. In particular $x \perp x$, and thus $x \in \text{Iso}Q$. Since $F \cap \text{Iso}Q = \{0\}$, $x = 0$.

Let us now define the orthogonal projection π over F. From lemma 12.2.3, there exists an orthonormal basis $(v_i)_{1 \leq i \leq k}$ of F. Let us now define

$$\pi(x) = \sum_{i=1}^{k} Q^*(x, v_i) v_i$$

It is easy to see that π is linear, that $\pi \circ \pi = \pi$ and that $\Im \pi = F$. Let us take $x \in E$,

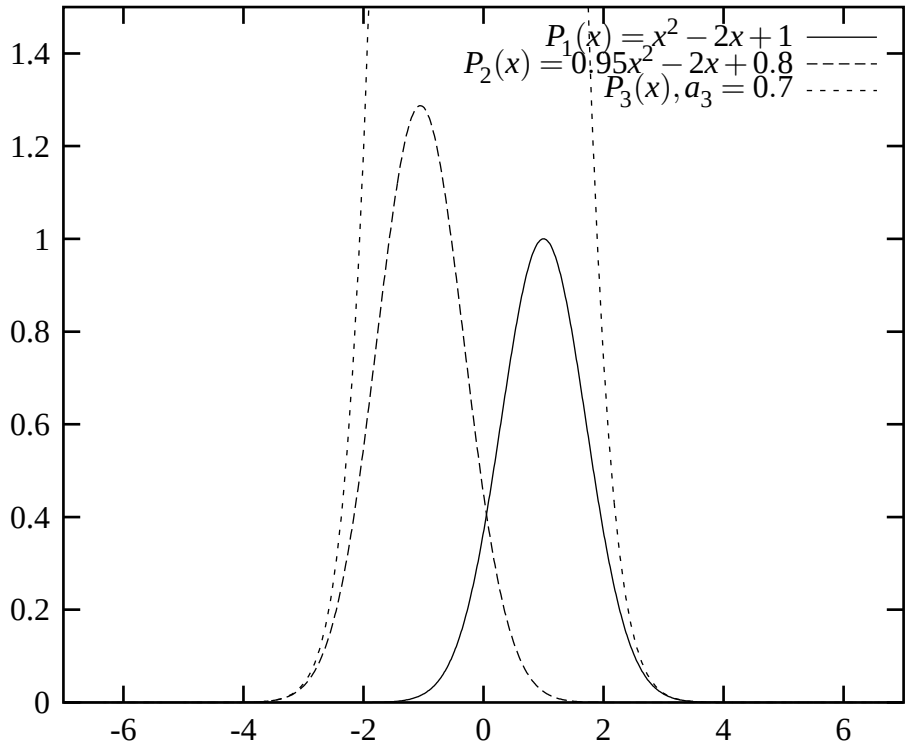


Figure 12.4: The Gaussians corresponding to the preceding figure. Note that while the central part is grossly overestimated (and truncated in the figure), the queues are finely approximated.

let us show that $x-\pi(x) \perp F$. For this, let us show that $x-\pi(x) \perp v_j$ for any j.

$$\begin{aligned} Q^*(x-\pi(x),v_j) &= Q^*(x-\sum_i Q^*(x,v_i)v_i,v_j) \\ &= Q^*(x,v_j)-\sum_i Q^*(x,v_i)\underbrace{Q^*(v_i,v_j)}_{\delta_i^j} \\ &= Q^*(x,v_j)-Q^*(x,v_j)=0 \end{aligned}$$

For any $x \in E$, $x = \underbrace{(x-\pi(x))}_{\in F^\perp} + \underbrace{\pi(x)}_{\in F}$. □

12.2.2 Common diagonalization

We shall often have to consider two different quadratic forms at the same time. It is then very useful to consider both of them in the same orthogonal basis. The following theorem guarantees that it is possible provided at least one of the quadratic forms is positive definite (or negative definite) [2, th. III.3.1]:

Theorem 12.2.5. *Let E be a finite dimension vector space. Let Q_1 be a quadratic form over E and Q_2 be a positive definite (or negative definite) quadratic form over E. Then there exists a base where both Q_1 and Q_2 are diagonal.*

When Q_2 is the canonical dot product over $\mathbb{R}^n$, this theorem rephrases into a well-known matrix formulation:

Corollary 12.2.6. *Let M be a symmetric matrix. Then there exists a diagonal matrix D and an orthogonal matrix P such that $M = PDP^{-1}$.*

We shall suppose that we dispose of certain numerical algorithms (algorithms 1, 2, 3). Those are available in the general literature (see [9, chapter 6] for algorithms on how to find eigenvalues and vectors) as well as free and commercial software [28].

Algorithm 1 QUADDIAG0, diagonalize a symmetric matrix in an orthonormal basis

Require: M a symmetric matrix
Ensure: $[P,D]$ where D is a diagonal matrix and P is an orthogonal matrix such that $M = PDP^{-1}$.
{Use your favorite numerical analysis algorithm here.}

As for the more generic operation where a second quadratic form is chosen, its matrix formulation is as follows:

Algorithm 2 ORTH, get an orthnormal basis of the image space of a matrix

Require: M a matrix

Ensure: B where $\Im B = \Im M$ and the columns of B form an orthonormal free family

{Use your favorite numerical analysis algorithm here.}

Algorithm 3 NULL, get an orthnormal basis of the null space of a matrix

Require: M a matrix

Ensure: B where $\Im B = \ker M$ and the columns of B form an orthonormal free family

{Use your favorite numerical analysis algorithm here.}

Algorithm 4 QUADDIAG1, common diagonalization of a quadratic form and a positive definite quadratic form

Require: $[Q_1, Q_2]$ where Q_1 a symmetric matrix, Q_2 a positive definite symmetric matrix

Ensure: $[P, I, d, D_1]$ where P is an invertible matrix, I its inverse, $d = \det P$, D_1 is a diagonal matrices such that $Q_1 = {}^t I\, D_1 I$ and $Q_2 = {}^t I\, I$

$[P_2, D_2] \leftarrow \text{QUADDIAG0}[Q_2]$

$Z \leftarrow D_2^{-1/2}$

$H \leftarrow P_2 Z$

$G \leftarrow {}^t H\, Q_2 H$

$[P_1, D_1] \leftarrow \text{QUADDIAG0}[G]$

$I \leftarrow {}^t P_1 \sqrt{D_2}\, {}^t P_2$

$P \leftarrow P_2 Z P_1$

$d \leftarrow \det Z$

Corollary 12.2.7. *Let Q_1 be a symmetric matrix, let Q_2 be a positive definite symmetric matrix. Then there exists an invertible matrix P and a diagonal matrix D_1 such that $Q_1 = {}^tP^{-1} D_1 P^{-1}$ and $Q_2 = {}^tP^{-1} P^{-1}$. The columns of P are an orthogonal base for both Q_1 and Q_2.*

Algorithm QUADDIAG1 (Alg. 4) computes effectively the new basis P, as well as P^{-1} and $\det P$, which will be of future use.

Unfortunately, in our case, we have to handle quadratic forms that have isotropic vectors. On the other hand, we only consider positive forms, and we thus have a second theorem:

Theorem 12.2.8. *Let E be a finite dimension vector space. Let Q_1 and Q_2 be two positive quadratic forms over E. Then there exists a base where both Q_1 and Q_2 are diagonal.*

Proof. Let us take a supplementary subspace F of $\ker Q_1 \cap \ker Q_2$. Obviously, $F \perp \ker Q_1 \cap \ker Q_2$ for both Q_1 and Q_2.

Let us consider the restrictions $Q_{1|F}$ and $Q_{2|F}$ of those quadratic forms over F. Both are positive quadratic forms. Furthermore, $Q_1 + Q_{2|F}$, the restriction of their sum, is a positive definite quadratic form: if for $x \in F$, $Q_1(x) + Q_2(x) = 0$, then since $Q_1(x) \geq 0$ and $Q_2(x) \geq 0$, $Q_1(x) = Q_2(x) = 0$. x is thus in $\mathrm{Iso}Q_1 \cap \mathrm{Iso}Q_2$. Since Q_1 and Q_2 are positive, $\mathrm{Iso}Q_1 = \ker Q_1$ and $\mathrm{Iso}Q_2 = \ker Q_2$ (lemma 12.2.2) and thus $x \in F \cap (\ker Q_1 \cap \ker Q_2) = \{0\}$, $x = 0$.

Let us apply theorem 12.2.5 to $Q_{1|F}$ and $Q_1 + Q_{2|F}$. There exists a base $\vec{b}_1, \ldots, \vec{b}_{\dim F}$ of F and eigenvalues $\lambda_1, \ldots, \lambda_{\dim F}$ and $\nu_1, \ldots, \nu_{\dim F}$ such that $Q_1(\sum_i x_i \vec{b}_i) = \sum_i \lambda_i x_i^2$ and $Q_2(\sum_i x_i \vec{b}_i) = \sum_i \nu_i x_i^2$. Let us complete this base by a base of $\ker Q_1 \cap \ker Q_2$, let us take $\lambda_i = \nu_i = 0$ for $i > \dim F$, $\mu_i = \lambda_i - \nu_i$ for any i. Then $Q_1(\sum_i x_i \vec{b}_i) = \sum_i \lambda_i x_i^2$ and $Q_2(\sum_i x_i \vec{b}_i) = \sum_i \mu_i x_i^2$. □

Let us now develop an effective algorithm for this theorem (Alg. 5). For effectiveness reasons, we choose F to be the orthogonal of $\ker Q_1 \cap \ker Q_2$ for the canonic dot product. Since $\ker Q_1 \cap \ker Q_2^{\perp} = \Im Q_1 + \Im Q_2$ (lemmas 12.2.9 and 12.2.10), we obtain an orthonormal basis of F by orthogonalizing a generating family of $\Im Q_1$ (the columns of Q_1), extending that basis to an orthonormal basis of $\Im Q_1 + \Im Q_2$ using a generating family of $\Im Q_2$ (the columns of Q_2). We then have an orthonormal basis of F, which can be extended to a basis B of $\mathbb{R}^n$ using a generating family of $\mathbb{R}$ (the canonical basis).

We consider both quadratic forms Q_1 and Q_2 on that basis B. Their matrices are of the form

$$Q_{1/2} = {}^tB\, Q_1 B = \begin{pmatrix} Q_1' & 0 \\ 0 & 0 \end{pmatrix} \tag{12.9}$$

where Q_1' and Q_2' are square matrices of size $\dim F$. We diagonalize Q_1' with respect to the definite positive matrix $Q_1' + Q_2'$ and output the results with respect to the right bases.

Algorithm 5 QUADDIAG2, common diagonalization of two positive quadratic forms

Require: Q_1 and Q_2 two positive symmetric matrices

Ensure: $[P, I, d, D_1, D_2]$ where P is an invertible matrix, I its inverse, $d = \det P$, D_1 and D_2 two diagonal matrices such that $Q_1 = {}^tI\, D_1 I$ and $Q_2 = {}^tI\, D_2 I$

$F \leftarrow \text{ORTH}\left[\begin{pmatrix} Q1 & Q2 \end{pmatrix}\right]$

$K \leftarrow \text{NULL}\left[\begin{pmatrix} Q1 \\ Q2 \end{pmatrix}\right]$

$[P', I', d, D_1'] \leftarrow \text{QUADDIAG1}({}^tF\, Q_1 F, {}^tF\, (Q_1 + Q_2) F)$

$D_1 \leftarrow \begin{pmatrix} D_1' & 0 \\ 0 & 0 \end{pmatrix}$

$D_2 \leftarrow \begin{pmatrix} 1 - D_1' & 0 \\ 0 & 0 \end{pmatrix}$

$P \leftarrow \begin{pmatrix} FP' & K \end{pmatrix}$

$I \leftarrow \begin{pmatrix} I' {}^tF \\ {}^tK \end{pmatrix}$

Lemma 12.2.9. *Let M be a symmetric matrix. Then $(\ker M)^\perp = \Im M$.*

Proof. $\Im M \subseteq (\ker M)^\perp$: let us take $M.x \in \Im M$, $y \in \ker M$, $\langle M.x, y\rangle = \langle x, \underbrace{{}^tM\, .y}_{0}\rangle =$ 0. But those spaces have the same dimension $n - \dim\ker M$ and are thus equal. □

Lemma 12.2.10. *Let E be a vector space and A and B two subspaces. Then $(A+B)^\perp = A^\perp \cap B^\perp$.*

Proof. Since $A \subseteq A+B$, $A^\perp \supseteq (A+B)^\perp$, similarly $B^\perp \supseteq (A+B)^\perp$ and thus $A^\perp \cap B^\perp \supseteq (A+B)^\perp$.

Let us take $x \in A^\perp \cap B^\perp$ and $y = \underbrace{a}_{\in A} + \underbrace{b}_{\in B}$, then $\langle x, y\rangle = \underbrace{\langle x, a\rangle}_{0} + \underbrace{\langle x, b\rangle}_{0}$. □

12.3 Extended normal distributions

12.3.1 Construction

Definition 12.3.1. Let E be a finite dimensional real vector space. Let us consider a positive quadratic form Q and a linear form L over E such that $\ker Q \subseteq \ker L$. q_0 is a real number. The function

$$\begin{array}{rcl} E & \to & \mathbb{R}_+ \\ \vec{v} & \mapsto & \exp(-Q(\vec{v}) + L(\vec{v}) + q_0) \end{array}$$

is called an *extended normal distribution*. It shall be noted G_{Q,L,q_0}.

Definition and proposition 12.3.2. Let $\phi : \vec{v} \mapsto \exp(-Q(\vec{v}) + L\vec{v} + q)$ be an extended normal distribution over an euclidean space E. Then there exists an orthonormal basis $(\vec{v}_i)_{1\leq 1\leq n}$, a positive real number K, coefficients $(\lambda_i)_{1\leq 1\leq n}$ and coordinates $(p_i)_{1\leq 1\leq n}$ such that

$$\phi(\sum x_i\vec{v}_i) = K\exp(-\sum_i \lambda_i(x_i - p_i)^2).$$

The point P, whose coordinates in the basis $(\vec{v}_i)_{1\leq 1\leq n}$ are $(p_i)_{1\leq 1\leq n}$, is called the *center* of the distribution.

Proof. Let us apply theorem 12.2.5 to Q and the canonic dot product (a positive definite quadratic form). There exists an orthonormal basis $(\vec{v}_i)_{1\leq 1\leq n}$ so that Q is diagonal $(\lambda_1, \ldots, \lambda_n)$ in this basis: $Q(\sum_i x_i\vec{v}_i) = \sum \lambda_i x_i^2$.

Let us write the linear form L in this basis: $L(\sum_i x_i\vec{v}_i) = \sum \alpha_i x_i$.

$$\begin{aligned}
-Q(\sum_i x_i\vec{v}_i) + L(\sum_i x_i\vec{v}_i) + q_0 & \\
&= q_0 - \sum_i (\lambda_i x_i^2 - \alpha_i x_i) \\
&= \left(q_0 + \sum_{i/\alpha_i\neq 0} \frac{\alpha_i^2}{4\lambda_i}\right) - \sum_{i/\alpha_i\neq 0} \lambda_i (x_i^2 - 2x_i\frac{\alpha_i}{2\lambda_i} + \frac{\alpha_i^2}{4\lambda_i^2}) \\
&= \left(q_0 + \sum_{i/\alpha_i\neq 0} \frac{\alpha_i^2}{4\lambda_i}\right) - \sum_i \lambda_i (x_i - \frac{\alpha_i}{2\lambda_i})^2
\end{aligned}$$

Noting $p_i = \frac{\alpha_i}{2\lambda_i}$ and $K = \exp\left(q_0 + \sum_{i/\alpha_i\neq 0} \frac{\alpha_i^2}{4\lambda_i}\right)$, the result follows. □

12.3.2 Least upper bound and widening

Let (Q_1, L_1, q_1) and (Q_2, L_2, q_2) be two extended normal distributions. We wish to get a common upper bound for them.

Let us note that, in general, there is no least upper bound in Gaussian distributions. Let us simply consider the case where $\dim E = 2$ and the Gaussians are centered and unscaled: $\vec{v} \mapsto \exp(-Q_1(\vec{v}))$ and $\vec{v} \mapsto \exp(-Q_2(\vec{v}))$ where Q_1 and Q_2 are quadratic forms; let us remark that in this case, the point-wise ordering of the Gaussians coincides with the inclusion ordering of the ellipses $Q_1(\vec{v}) = 1$ and $Q_2(\vec{v}) = 1$. On Fig. 12.5 are two ellipses A and B for which we want to obtain

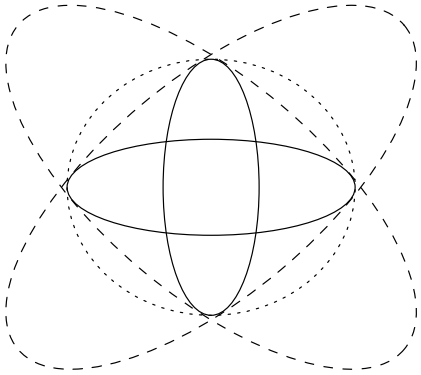


Figure 12.5: Common upper bounds for two ellipses. One of them is the result of the procedure defined in §12.3.2 (least upper bound in the common orthogonal base). Other common upper bounds demonstrate the absence of a least upper bound in the ellipses.

a common upper bound (with axes parallel to those of the figure for the sake of readability), the common upper bound output by the procedure described in this section and two common upper bounds that demonstrate the impossibility of a least upper bound of A and B in the ellipses.

We note $(Q_1, L_1, q_1) \sqcup (Q_2, L_2, q_2)$ the extended normal distribution defined as follows. Since Q_1 and Q_2 are positive, we diagonalize them in the same base $(\vec{v}_i)_{1 \leq 1 \leq n}$ (theorem 12.2.8). Then $Q_1(\sum_i x_i \vec{v}_i) = \sum \lambda_i x_i^2$ and $Q_2(\sum_i x_i \vec{v}_i) = \sum \mu_i x_i^2$. Let us write the linear forms L_1 and L_2 in this basis: $L_1(\sum_i x_i \vec{v}_i) = \sum \alpha_i x_i$ and $L_2(\sum_i x_i \vec{v}_i) = \sum \beta_i x_i$.

Let σ_i and τ_i be partitions of q_1 and q_2 respectively ($\sum_i \sigma_i = q_1$ and $\sum_i \tau_i = q_2$). We can take $\sigma_i = q_1/n$ and $\tau_i = q_2/n$.

Let $a_i X^2 + b_i X + c_i = (\lambda_i X^2 + \alpha_i X + \sigma_i) \sqcap (\mu_i X^2 + \beta_i X + \tau_i)$ (as defined in §12.1).

Let $Q(\sum_i x_i \vec{v}_i) = \sum_i a_i x_i^2$, $L(\sum_i x_i \vec{v}_i) = \sum_i b_i x_i$ and $q = \sum_i c_i$.

Let us check that $\ker Q \subseteq \ker L$. Since $\ker Q$ is the isotropic cone of Q and the $\vec{v}_i$ form a diagonal basis for Q, it follows that a subset of the $\vec{v}_i$ form a basis of $\ker Q$. For any index i such that $\vec{v}_i$ is in that basis, $a_i = 0$ and thus $b_i = 0$ by construction, so $\vec{v}_i \in \ker L$.

We define

$$(Q_1, L_1, q_1) \sqcup (Q_2, L_2, q_2) = (Q, L, q). \tag{12.10}$$

Let us remark that

$$\dim\ker Q \geq \max(\dim\ker Q_1, \dim\ker Q_2) \tag{12.11}$$

since for all i such that $\lambda_i = 0$ or $\mu_i = 0$, $a_i = 0$.

We define the widening operator similarly, except that this time we need to ensure convergence of the ascending sequences $u_{n+1} = u_n \nabla v_n$. We shall modify the least upper bound operator in two respects to obtain the widening operator:

1. Intuitively, when $a_i < \lambda_i$, this means that along that particular vector $\vec{v}_i$ the Gaussian gets flatter and flatter. The natural widening is to upper-approximate it by a flat line. In this case, we take $a'_i = b'_i = 0$ and

$$c'_i = \min_x a_i x^2 + b_i + c = \frac{-b^2}{4a} + c. \tag{12.12}$$

2. If all $a_i = 0$ and c is still decreasing, we take the last resort of jumping to $\top$.

The convergence of the method is ensured by the fact that whenever step 1 is applied, $\dim\ker Q$ strictly increases. Since in non-widening steps, $\dim\ker Q$ increases or stays constant, it follows that at most $\dim E$ step 1 may be applied. After this, the a_i stay constant. The only way the sequence can still go on ascending is by an increase in c. Then step 2 ensures termination.

12.3.3 Random generators

Let us recall Equ. 13.1 and Prop. 13.3.14: the backwards operation associated with $\rho = \texttt{random}_\lambda$ where ρ is a fresh real variable

$$g = [\![\rho = \texttt{random}]\!]^*_\rho . f = \vec{v} \mapsto \int_x f(\vec{v} + x\vec{e})\, \mathrm{d}\mu_R(x) \tag{12.13}$$

where $\vec{e}$ is an additional basis vector corresponding to the fresh variable ρ and μ_R is the distribution of the new variable. If μ_R is given by the Gaussian $\exp(-(\lambda x^2 +$

$c_1)$), and f is given by (Q,L,c), then

$$g(\vec{v}) = \int_{-\infty}^{+\infty} \exp\left(-(Q(\vec{v}+x\vec{e})+L(\vec{v}+x\vec{e})+c+\lambda x^2+c_1)\right)\,\mathrm{d}x$$

$$= \int_{-\infty}^{+\infty} \exp\left(-(Q(\vec{v})+L(\vec{v})+c+c_1)\right) \exp\left(-(\underbrace{(Q^*(\vec{v},\vec{e})+L(\vec{e}))}_{\beta}x+\underbrace{(Q(\vec{e})+\lambda)}_{\alpha}x^2)\right)\,\mathrm{d}x$$

$$= \exp\left(-(Q(\vec{v})+L(\vec{v})+c+c_1)\right)\int_{-\infty}^{+\infty}\exp(-(\beta x+\alpha x^2))\,\mathrm{d}x$$

$$= \exp\left(-(Q(\vec{v})+L(\vec{v})+c+c_1)\right)\sqrt{\frac{\pi}{a}}\exp\left(\frac{\beta^2}{4\alpha}\right)$$

$$= \exp\left(-\left(\underbrace{Q(\vec{v})+\frac{1}{4\alpha}Q^*(\vec{v},\vec{e})^2}_{Q'(\vec{v})}+\underbrace{L(\vec{v})+\frac{1}{2\alpha}Q^*(\vec{v},\vec{e})L(e^\sharp)}_{L'(\vec{v})}+\underbrace{c+c_1+\frac{L(\vec{e})^2}{4\alpha}}_{c'}\right)\right) \tag{12.14}$$

Because of the definition of g as the integral of a bounded function versus a measure of finite total weight, g is bounded; thus Q is positive and $\ker L \subseteq \ker Q$.

12.3.4 Linear operations

We shall deal here with program statements such as $v_n := \sum_i \alpha_i v_i$ and more generally any linear transformation M where the vector of variables $\vec{V}'$ after the instruction is $M.\vec{V}$ where $\vec{V}$ is the vector of variable before the instruction. Following Prop. 13.3.10,

$$\left[\!\left[\vec{V} := M.\vec{V}\right]\!\right]^*_p.f = f \circ M \tag{12.15}$$

and thus $(Q',L',c) = ({}^tM\,QM, LM, c)$.

12.3.5 Other operations

We shall approximate other operations by releasing all constraints on the variables affected by them: forgetting variable in set V is achieved as follows:

- $q'_{i,j} = q_{i,j}$ if $i \notin V$ and $j \notin V$, $q'_{i,j} = 0$ otherwise;
- $L'_i = L_i$ if $i \notin V$, $L_i = 0$.

It is quite obvious that if $f : E \mapsto E$ leaves all coordinates outside of V intact, $G_{Q,L,q_0} \circ f \leq G_{Q',L',q_0}$ point-wise.

12.4 Discussion

The domain presented in this chapter purports at representing exactly one of the most widely used distributions, the Gaussian distribution. It actually represents Gaussian measurable functions with a view to backwards analysis. As shown in Fig. 12.4, this analysis yields coarse results in the center of the functions; on the other hand, it leads to very precise results in the queues of the distribution. It therefore seems desirable to use it as a way to bound the influence of the queues of the random generators while using other methods, including abstract Monte-Carlo, for the center of the distribution.

The main problem with this domain is that it does not interact well with precise bounds, obtained for instance with a test with respect to an interval. A possible direction of research is an abstract domain covering both precise bounds and Gaussian bounds.

An application of this Gaussian analysis could be the study of the propagation of computational inaccuracies introduced by floating-point arithmetics, modeled by random choices.[1] We hope to contribute to that recently opened field [67] of abstract interpretation applied to round-off errors.

[1] This idea of modeling inaccuracies by random choices is the basis of the CESTAC method [78, 8, 79].

Chapter 13

Denotational approaches

Following earlier models, we lift standard deterministic and nondeterministic semantics of imperative programs to probabilistic semantics. This semantics allows for random external inputs of known or unknown probability and random number generators.

We then propose a method of analysis of programs according to this semantics, in the general framework of abstract interpretation. This method lifts an "ordinary" abstract lattice, for non-probabilistic programs, to one suitable for probabilistic programs.

13.1 Probabilistic Concrete Semantics

Throughout this paper, we shall define compositionally several semantics and expose relationships between them. We shall use as an example some simple Pascal-like imperative language, but we do not mean that our analysis methods are restricted to such languages.

13.1.1 Summary of Non-probabilistic Concrete Semantics

We shall here consider denotational semantics for programs. (equivalent operational semantics could be easily defined, but we shall mostly deal with denotational ones for the sake of brevity).

The language is defined as follows: the compound program instructions are

instruction ::= *elementary*
instruction `;` *instruction*
`if` *boolean_expr* `then` *instruction* `else` *instruction* `endif`

`while` *boolean_expr* `do` *instruction* `done`

and the boolean expressions are defined as

boolean_expr ::= *boolean_atomic*
boolean_expr `and` *boolean_expr*
boolean_expr `or` *boolean_expr*
`not` *boolean_expr*

elementary instructions are deterministic, terminating basic program blocks like assignments and simple expression evaluations. *boolean_atomic* boolean expressions, such as comparisons, have semantics as sets of "acceptable" environments. For instance, a *boolean_atomic* expression can be `x < y + 4`; its semantics is the set of execution environments where variables `x` and `y` verify the above comparison. If we restrict ourselves to a finite number n of integer variables, an environment is just a n-tuple of integers.

The denotational semantics of a code fragment c is a mapping from the set X of possible execution environments before the instruction into the set Y of possible environments after the instruction. Let us take an example. If we take environments as elements of $\mathbb{Z}^3$, representing the values of three integer variables `x`, `y` and `z`, then $[\![\texttt{x:=y+z}]\!]$ is the function $\langle x,y,z\rangle \mapsto \langle y+z,y,z\rangle$. Semantics of basic constructs (assignments, arithmetic operators) can be easily dealt with this way; we shall now see how to deal with flow control.

The semantics of a sequence is expressed by simple composition

$$[\![e_1\texttt{;}\ e_2]\!] = [\![e_2]\!] \circ [\![e_1]\!]$$

Tests get expressed easily, using as the semantics $[\![c]\!]$ of a boolean expression c the set of environments it matches:

$$[\![\texttt{if}\ c\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!](x) = \text{if } x \in [\![c]\!] \text{ then } [\![e_1]\!](x) \text{ else } [\![e_2]\!](x)$$

and loops get the usual least-fixpoint semantics (considering the point-wise extension of the Scott flat ordering on partial functions)

$$[\![\texttt{while}\ c\ \texttt{do}\ f]\!] = \text{lfp}\,\lambda\phi.\lambda x.\text{if } x \in [\![c]\!] \text{ then } \phi \circ [\![f]\!](x) \text{ else } x.$$

Non-termination shall be noted by $\perp$.

13.1.2 Our Framework for Probabilistic Concrete Semantics

We shall give a semantics based on measures (§6.3.3). Our semantics shall be expressed as continuous linear operators between measure spaces, of norm less than 1, using the Banach norm of total variation on measures. This is necessary to ensure the mathematical well-formedness of certain definitions, such as the concrete semantics of loops. As the definitions for these concepts and some mathematical proofs for the definition of the concrete semantics are quite long and not relevant at all to the analysis, we shall omit them from this paper and refer the reader to an extended version. As a running example for the definitions of the semantics, we shall use a program with real variables `x`, `y` and `z`; the set of possible environments is then $\mathbb{R}^3$.

General form

Let us consider an elementary program statement c such that $[\![c]\!] : X \to Y$, X and Y being measurable spaces. We shall also suppose that $[\![c]\!]$ is measurable. Let us first remark that this condition happens, for instance, for any continuous function from X and Y if both are topological spaces and σ_Y is the Borel σ-algebra [71, §1.11]. $[\![\texttt{x := y+z}]\!] = \langle x,y,z\rangle \mapsto \langle y+z,y,z\rangle$ is continuous.

To $[\![\texttt{c}]\!]$ we associate the following linear operator $[\![\texttt{c}]\!]_p$:

$$[\![\texttt{c}]\!]_p : \left| \begin{array}{ll} \mathcal{M}_{\leq 1}(X) \to \mathcal{M}_{\leq 1}(Y) \\ \mu \qquad\quad \mapsto \lambda W.\mu([\![c]\!]^{-1}(W)) \end{array} \right. .$$

We shall see that all flow control constructs "preserve" measurability; i.e., if all sub-blocks of a complex construct have measurable semantics, then the construct shall have measurable semantics. We shall then extend the framework to programs containing `random`-like operators; their semantics will be expressed as linear operators of norm less than 1 on measure spaces.

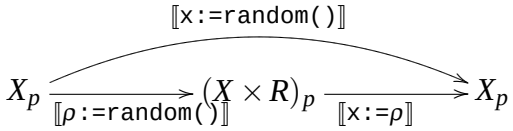
Random Inputs or Generators

An obvious interest of probabilistic semantics is to give an accurate semantics to assignment such as `x:=random();`, where `random()` is a function that, each time it is invoked, returns a real value equidistributed between 0 and 1, independently of previous calls.[1] We therefore have to give a semantics to constructs such as `x:=random();`, where `random` returns a value in a measured space R

[1] Of course, functions such as the POSIX C function `drand48()` would not fulfill such requirements, since they are pseudo-random generators whose output depends on an internal state that changes each time the function is invoked, thus the probability laws of successive invocations

whose probability is given by the measure μ_R and is independent of all other calls and previous states.

We decompose this operation into two steps:[2]



The second step is a simple assignment operator, addressed by the above method. The first step boils down to measure products:

$$\llbracket \rho \texttt{:=random()} \rrbracket : \left| \begin{array}{l} X_p \to (X \times R)_p \\ \mu \ \mapsto \mu \otimes \mu_R \end{array} \right. . \tag{13.1}$$

Tests and Loops

We restrict ourselves to test and loop conditions b such that $\llbracket \texttt{b} \rrbracket$ is measurable. This condition is fulfilled if all the *boolean_atomic* sets are measurable since the σ-algebra is closed by finite union and intersection. For instance, $\llbracket \texttt{x < y} \rrbracket = \{\langle x,y,z\rangle \mid x < y\}$ is measurable.

The deterministic semantics for tests are:

$$\llbracket \texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2 \rrbracket (x) = \text{if } x \in \llbracket x \rrbracket \text{ then } \llbracket e_1 \rrbracket (x) \text{ else } \llbracket e_2 \rrbracket (x).$$

Let us first compute

$$\llbracket \texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2 \rrbracket^{-1} (W) = (\llbracket e_1 \rrbracket^{-1} (W) \cap \llbracket c \rrbracket) \cup (\llbracket e_2 \rrbracket^{-1} (W) \cap \llbracket c \rrbracket^C).$$

$\llbracket \texttt{c} \rrbracket$ is the set of environments matched by condition c. It is obtained inductively from the set of environment matched by the atomic tests (e.g. comparisons):

- $\llbracket c_1 \texttt{ or } c_2 \rrbracket = \llbracket c_1 \rrbracket \cup \llbracket c_2 \rrbracket$
- $\llbracket c_1 \texttt{ and } c_2 \rrbracket = \llbracket c_1 \rrbracket \cap \llbracket c_2 \rrbracket$
- $\llbracket \texttt{not } c \rrbracket = \llbracket c \rrbracket^C$

---

are not independent. However, ideal random generators are quite an accurate approximation for most analyses.

[2] Another equivalent way, used by Kozen [41, 42], is to consider random values as countable streams in the input environment of the program.

Using our above framework to lift deterministic semantics to probabilistic ones, we get

$$\begin{aligned}\llbracket \texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2 \rrbracket_p(\mu) &= X \mapsto \mu(\llbracket \texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2 \rrbracket^{-1}(X)) \\ &= X \mapsto \mu((\llbracket e_1 \rrbracket^{-1}(X) \cap \llbracket c \rrbracket) \cup (\llbracket e_2 \rrbracket^{-1}(X) \cap \llbracket c \rrbracket^C) \\ &= X \mapsto \mu(\llbracket e_1 \rrbracket^{-1}(X) \cap \llbracket c \rrbracket) + \mu(\llbracket e_2 \rrbracket^{-1}(X) \cap \llbracket c \rrbracket^C) \\ &= \llbracket e_1 \rrbracket_p \circ \phi_{\llbracket c \rrbracket}(\mu) + \llbracket e_2 \rrbracket_p \circ \phi_{\llbracket c \rrbracket^C}(\mu) \end{aligned} \quad (13.2)$$

where $\phi_W(\mu) = \lambda X.\mu(X \cap W)$.

We lift in the same fashion the semantics of loops (we note $\sqcup$ an union of pairwise disjoint subsets of a set):

$$\begin{aligned}&\llbracket \texttt{while } c \texttt{ do } e \rrbracket^{-1}(X) \\ &\quad = (\mathrm{lfp}\, \lambda \phi.\lambda x.\text{if } x \in \llbracket c \rrbracket \text{ then } \phi \circ \llbracket e \rrbracket(x) \text{ else } x)^{-1}(X) \\ &\quad = \bigsqcup_{n \in \mathbb{N}} (\lambda Y. \llbracket e \rrbracket^{-1}(Y) \cap \llbracket c \rrbracket)^n (X \cap \llbracket c \rrbracket^C) \end{aligned} \quad (13.3)$$

We therefore derive the form of the probabilistic semantics of the `while` loop:

$$\begin{aligned}\llbracket \texttt{while } c \texttt{ do } e \rrbracket_p(\mu) &= \lambda X.\mu \left(\bigsqcup_{n \in \mathbb{N}} (\lambda Y. \llbracket e \rrbracket^{-1}(Y) \cap \llbracket c \rrbracket)^n (X \cap \llbracket c \rrbracket^C) \right) \\ &= \lambda X. \sum_{n \in \mathbb{N}} \mu \left((\lambda Y. \llbracket e \rrbracket^{-1}(Y) \cap \llbracket c \rrbracket)^n (X \cap \llbracket c \rrbracket^C) \right) \\ &= \sum_{n=0}^{\infty} \phi_{\llbracket c \rrbracket^C} \circ (\llbracket e \rrbracket_p \circ \phi_{\llbracket c \rrbracket})^n(\mu) \\ &= \phi_{\llbracket c \rrbracket^C} \left(\sum_{n=0}^{\infty} (\llbracket e \rrbracket_p \circ \phi_{\llbracket c \rrbracket})^n(\mu) \right) \\ &= \phi_{\llbracket c \rrbracket^C} \left(\lim_{n \to \infty} (\lambda \mu'.\mu + \llbracket e \rrbracket_p \circ \phi_{\llbracket c \rrbracket}(\mu'))^n (\lambda X.0) \right) \end{aligned} \quad (13.4)$$

Limits and infinite sums are taken according to the set-wise topology. We refer the reader to an extended version of this paper for the technical explanations on continuity and convergence.

13.1.3 Probabilities and Nondeterminism

It has been pointed out [49, 31] that we must distinguish deterministic and non-deterministic probabilistic semantics. Deterministic, non-probabilistic semantics

embed naturally into the above probabilistic semantics: instead of a value $x \in X$, we consider the Dirac measure $\delta_x \in \mathcal{M}_{\leq 1}(X)$ defined by $\delta_x(X) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise.} \end{cases}$ How can we account for nondeterministic non-probabilistic semantics?

We move from deterministic to nondeterministic semantics by lifting to powersets. It is possible to consider nondeterministic probabilistic semantics: the result of a program is then a set of probability measures. Of course, nondeterministic non-probabilistic semantics get embedded naturally: to $A \in \mathcal{P}(X)$ we associate $\{\delta_a \mid a \in A\} \in \mathcal{P}(\mathcal{M}_+(X))$. We therefore consider four semantics:

determinism	nondeterminism
probabilistic	nondeterministic probabilistic

The advantage of probabilistic nondeterminism is that we can consider programs whose inputs are not all distributed according to a distribution, or whose distribution is not exactly known. Our analysis is based on probabilistic nondeterminism and thus handles all cases.

13.2 Abstract Semantics

We shall first give the vocabulary and notations we use for abstractions in general. We shall then proceed by giving an example of an domain that abstracts probabilistic semantics as defined in the previous section. This domain is parametric in multiple ways, most importantly by the use of an abstract domain for the non-probabilistic semantics of the studied system.

13.2.1 Summary of Abstraction

Let us consider a preordered set $X^\sharp$ and a monotone function $\gamma_X : X^\sharp \to \mathcal{P}(X)$. $x^\sharp \in X^\sharp$ is said to be an *abstraction* of $x^\flat \subset X$ if $x^\flat \subseteq \gamma_X(x^\sharp)$. γ_X is called the *concretization function*. The triple $\langle \mathcal{P}(X), X^\sharp, \gamma_X \rangle$ is called an *abstraction*. $\mathcal{P}(X)$ is the *concrete domain* and $X^\sharp$ the *abstract domain*. Such definitions can be extended to any preordered set $X^\flat$ besides $\mathcal{P}(X)$.

Let us now consider two abstractions $\langle \mathcal{P}(X), X^\sharp, \gamma_X \rangle$ and $\langle \mathcal{P}(Y), Y^\sharp, \gamma_Y \rangle$ and a function $f : X \to Y$. $f^\sharp$ is said to be *an abstraction of* f if

$$\forall x^\sharp \in X^\sharp \; \forall x \in X \; x \in \gamma_X(x^\sharp) \Rightarrow f(x) \in \gamma_Y(f^\sharp(x^\sharp)) \tag{13.5}$$

More generally, if $\langle X^\flat, X^\sharp, \gamma_X \rangle$ and $\langle Y^\flat, Y^\sharp, \gamma_Y \rangle$ are abstractions and $f^\flat : X^\flat \to Y^\flat$ is a monotone function, then $f^\sharp$ is said to be *an abstraction of* $f^\flat$ if

$$\forall x^\flat \in X^\flat \; \forall x^\sharp \in X^\sharp \; x^\flat \sqsubseteq \gamma_X(x^\sharp) \Rightarrow f^\flat(x^\flat) \sqsubseteq \gamma_X(f^\sharp(x^\sharp)) \tag{13.6}$$

Algorithmically, elements in $X^\sharp$ will have a machine representation. To any program construct c we shall attach an effectively computable function $[\![\mathtt{c}]\!]^\sharp$ such that $[\![\mathtt{c}]\!]^\sharp$ is an abstraction of $[\![\mathtt{c}]\!]$. Given a machine description of a superset of the inputs of the programs, the abstract version yields a superset of the outputs of the program. If a state is not in this superset, this means that, for sure, the program cannot reach this state.

Let us take an example, the *domain of intervals*: if $X^\sharp = Y^\sharp = T^3$ where $T = \{(a,b) \in \mathbb{Z} \cup \{-\infty,+\infty\} \mid a \leq b\} \cup \{\bot\}$, $\gamma(a,b) = \{c \in \mathbb{Z} \mid a \leq c \leq b\}$ and γ induces a preorder $\sqsubseteq_T$ over T and, pointwise, over $X^\sharp$, then we can take $[\![\mathtt{x:=y+z}]\!]^\sharp((a_\mathtt{x},b_\mathtt{x}),(a_\mathtt{y},b_\mathtt{y}),(a_\mathtt{z},b_\mathtt{z})) = ((a_\mathtt{y}+a_\mathtt{z},b_\mathtt{y}+b_\mathtt{z}),(a_\mathtt{y},b_\mathtt{y}),(a_\mathtt{z},b_\mathtt{z}))$.

13.2.2 Probabilistic Abstraction

The concrete probabilistic domains given in 13.1.2 can be abstracted as in the previous definition. Interesting properties of such an abstraction would be, for instance, to give an upper bound on the probability of some subsets of possible environments at the end of a computation.

13.2.3 Turning Fixpoints of Affine Operators into Fixpoints of Monotone Set Operators

Equation 13.4 shows that the semantics of loops are given as infinite sums or, equivalently, as fixpoints of some affine operators. In non-probabilistic semantics, the semantics of loops is usually the fixpoint of some monotone operator on the concrete lattice, which get immediately abstracted as fixpoints on the abstract lattice. The approximation is not so evident in the case of this sum; we shall nevertheless see how to deal with it using fixpoints on the abstract lattice.

Defining μ_n recursively, as follows: $\mu_0 = \lambda X.0$ and $\mu_{n+1} = \psi\mu_n$, with $\psi(\nu) = \mu + [\![e]\!]_p \circ \phi_{[\![c]\!]}(\nu)$, we can rewrite equation 13.4 as $[\![\mathtt{while}\ c\ \mathtt{do}\ e]\!]_p(\mu) = \phi_{[\![c]\!]^C}(\lim_{n\to\infty}\mu_n)$. We wish to approximate this limit in the measure space by an abstract element.

We shall use the following method: to get an approximation of the limit of a sequence $(u_n)_{n\in\mathbb{N}}$ defined recursively by $u_{n+1} = f(u_n)$, we can find a closed set S stable by f such that $u_N \in S$ for some N; then $\lim_{n\to\infty} u_n \in S$. Let us note than finding such a set does not prove that the limit exists; we have to suppose that f is such that this limit exists. In our case, this condition is necessarily fulfilled.

Let us take $\mu^\sharp$ and $\mu_0^\sharp$ respective abstractions of μ. Let us call $\psi^\sharp(\nu^\sharp) = \mu^\sharp +^\sharp [\![e]\!]_p^\sharp \circ \phi_{[\![c]\!]}^\sharp(\nu^\sharp)$. Let us take a certain $N \in \mathbb{N}$ and call $L^\sharp = \mathrm{lfp}\,\lambda\nu^\sharp.\psi^{\sharp N}(\mu_0^\sharp) \sqcup \psi^\sharp(\nu^\sharp)$; then by induction, for all $n \geq N$, $\mu_n \in \gamma(L^\sharp)$. As $\gamma(L^\sharp)$ is topologically

closed, $\lim n \to \infty \in \gamma(L^\sharp)$. Therefore $L^\sharp$ is an approximation of the requested limit.

Let us suppose that we have an "approximate least fixpoint" operation $\mathrm{lfp}^\sharp : (X^\sharp \xrightarrow{\text{monotonic}} X^\sharp) \to X^\sharp$. By "approximate least fixpoint" we mean that if $f^\sharp : X^\sharp \to X^\sharp$ is monotonic, then, noting $x_0^\sharp = \mathrm{lfp}^\sharp(f)$, $f^\sharp(x_0^\sharp) \sqsubseteq x_0^\sharp$. The justification of our appellation, and the interest of such a function, lies in the following well-known result:

Lemma 13.2.1. *If $f^\sharp : X^\sharp \to X^\sharp$ is an abstraction of $f^\flat : \mathcal{P}(X) \to \mathcal{P}(X)$ and $f^\sharp(x_0^\sharp) \sqsubseteq x_0^\sharp$, then* $\mathrm{lfp}\, f^\flat \subseteq \gamma_X(x_0^\sharp)$.

Of course, building such an operation is not easy. Trying the successive iterations of $f^{\sharp n}$ until reaching a fixpoint does not necessarily terminate. One has to use special tricks and widening operators to build such a function (see 9.1.4).

Provided we have such an operation, abstraction follows directly:

$$[\![\texttt{while } c \texttt{ do } e]\!]^\sharp(W^\sharp) = \phi^\sharp_{[\![c]\!]^C}(\mathrm{lfp}^\sharp X^\sharp \mapsto W^\sharp \sqcup [\![e]\!]^\sharp(\phi^\sharp_{[\![c]\!]}(X))).$$

As usual in abstract interpretation, it might be wise to do some semantics-preserving transformations on the program, such as unrolling the first few turns of the loop, before applying this abstraction. This is likely to yield better results.

13.3 Adjoint semantics

In his seminal papers, Kozen proposed semantics of probabilistic programs as continuous linear operators on measure spaces. We shall see that operators representing the semantics of probabilistic programs have adjoints, in a certain sense that we shall define. These adjoints are the basis of our analysis techniques; furthermore, their existence gives a straightforward proof of Kozen's theorem.

13.3.1 Intuition

Let us consider a very simple C program (Fig. 13.1) where `centered_uniform()` is a random generator returning a `double` uniformly distributed in $[-1,1]$, independent of previous calls. We are interested in the relationship between the probability of executing `B` depending on the probability distribution generated in `x` by `A`. What we would therefore like is a (linear) function f mapping the probability measure μ generated at A to the probability $f(\mu)$ that program point `B` is executed. It will be proved that there exist a "weight function" g such that $f(\mu) = \int g \, \mathrm{d}\mu$. We shall see how to compute such a g.

```
double x, y;
... /* A */
x += centered_uniform()+centered_uniform();
...
if (fabs(x) <= 1)
{
  ... /* B */
}
```

Figure 13.1: A simple probabilistic program

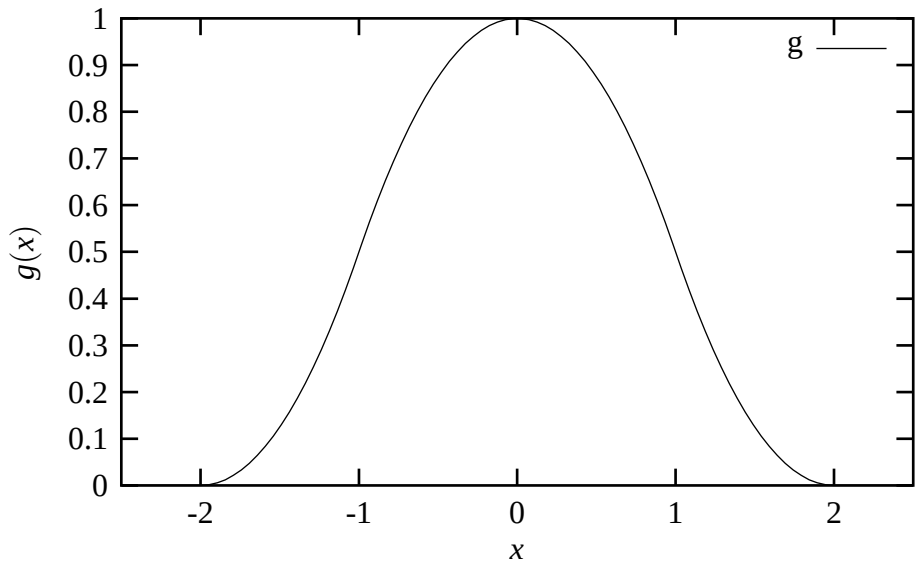


Figure 13.2: Weight function g such that the probability of outcome of step B (see Fig. 13.1) given the probability measure μ at step A is $\int g \, \mathrm{d}\mu$. x is the value of variable x.

```
double x, y;
... /* A */
if (x+y >= -1)
{
  x += 2;
}
y = centered_uniform()+centered_uniform();
x += y/2;
...
if (fabs(x) <= 1)
{
  ... /* B */
}
```

Figure 13.3: Another probabilistic program

A plotting of g is featured in figure 13.2. Let us give a few examples of the use of this function:

- The probability that B will happen if A drives x according to some uniform distribution in $[a,b]$ is $\int_a^b g(x)\,\mathrm{d}x$.
- The probability that B will happen if A sets x to a constant C is $g(C)$.

The set $g^{-1}(0)$ is the set of values at step A that have no chance of starting a program trace reaching step B. Please note that this is slightly different from the set of values that cannot start a program trace reaching step B. This is the difference between "impossible" and "happening with 0 probability". Let us see the following example:

```
while (centered_uniform() > 0) { };
```

Nontermination happens with probability 0 — that is, it is extremely unlikely, negligible. However, it is possible, however infinitely unlikely, that the random number generator outputs an infinite stream of reals less than 0, thus producing nonterminating behavior.

13.3.2 Summary of semantics according to Kozen

The semantics of a probabilistic program c can be seen as a linear operator $[\![c]\!]_p$ mapping the input probability distribution (measure μ) to an output measure

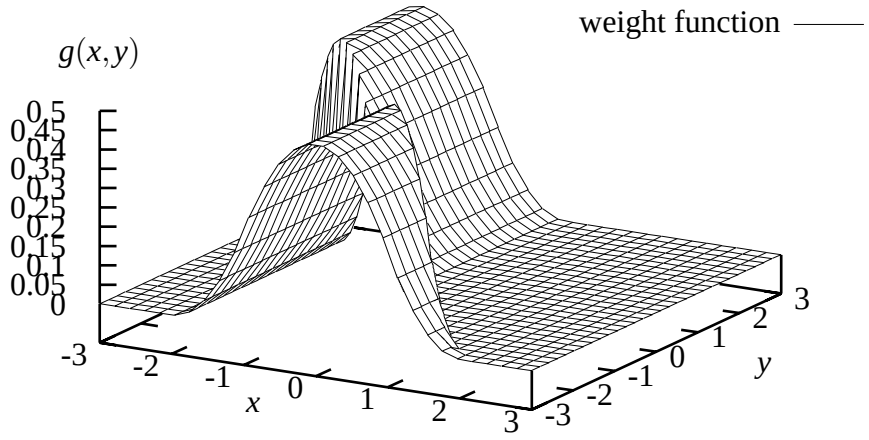


Figure 13.4: Weight function g such that the probability of outcome of step B (see Fig. 13.3) given the probability measure μ at step A is $\int g \, d\mu$. x and y are the respective values of variables x and y.

$[\![c]\!]_p.\mu$ Values given by random number generators can either be seen as successive reads from streams given as inputs or are handled internally in the semantics [41, 42]; here we shall use the second approach, though both approaches are equivalent. We shall not discuss here the technical details necessary to ensure continuity of operators, convergences etc... and we shall refer the reader to [54][extended version].

The semantics of a program c whose initial environment lies in the measurable set X and whose final environment lies in the measurable set Y shall be given as a linear operator (of norm less than 1 for the norm of total variation on measures). If c contains no calls to random number generators, $[\![\mathsf{c}]\!]$ is just a measurable function (a function f is said to be *measurable* if and only if for any measurable set X, $f^{-1}(X)$ is measurable).

We shall base ourselves on an ordinary denotational semantics: $[\![c]\!]$ is a function from set X to set Y if c has type $X \to Y$. For the sake of simplicity, we shall not deal with nontermination here so no $\perp$ value is needed. To make meaningful probabilities, X and Y are measurable sets (for instance, countable sets) and $[\![c]\!]$ is assumed to be a measurable function. These restrictions are of a technical nature and do not actually restrict the scope of the analysis in any way; the reader shall refer to [54] for more details.

Let us summarize the probabilistic semantics:

Elementary constructs (assignments etc...) get simple semantics: $[\![\mathtt{c}]\!]_p.\mu = \lambda X.\mu([\![\mathtt{c}]\!]^{-1}(X))$.

Random number generation Let us suppose each invocation of `random` yields a value following distribution μ_R, each invocation being independent from another, and stores the value into a fresh variable. Then $[\![\mathtt{c}]\!]_p.\mu = \mu \otimes \mu_R$ where $\otimes$ is the product on measures.

Tests Let us define $\phi_W(\mu) = \lambda X.\mu(X \cap W)$. Then

$$[\![\mathtt{if}\ c\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2]\!]_p(\mu) = [\![e_1]\!]_p \circ \phi_{[\![c]\!]}(\mu) + [\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}(\mu). \tag{13.7}$$

Loops The semantics for loops amounts to summing inductively the semantics of the tests:

$$[\![\mathtt{while}\ c\ \mathtt{do}\ e]\!]_p(\mu) = \sum_{n=0}^{\infty} \phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^n(\mu), \tag{13.8}$$

the limit being taken set-wise [27, §III.10].

13.3.3 Adjoints and pseudo-adjoints

In this section, we shall recall the definition of an adjoint of a linear operator and give a definition of a pseudo-adjoint. We shall also give some easy properties, without proofs for the sake of brevity.

Let us consider two measurable sets (X, σ_X) and (Y, σ_Y). Let us first define, for f a measurable function and μ a measure,

$$\langle f, \mu \rangle = \int f \, \mathrm{d}\mu.$$

Proposition 13.3.1. *Taking f in the bounded functions (norm $\|\cdot\|_\infty$) and μ in the signed measures (norm of the total variation $\|\cdot\|$), this defines a continuous bilinear scalar form. Moreover, this form has the following properties:*

- *for all f and μ, $|\langle f, \mu \rangle| \leq \|f\|_\infty.\|\mu\|$;*
- *$\langle f, \cdot \rangle = \mu \mapsto \langle f, \mu \rangle$ has norm $\|f\|_\infty$;*
- *$\langle \cdot, \mu \rangle = f \mapsto \langle f, \mu \rangle$ has norm $\|\mu\|$.*

Corollary 13.3.2. *If $\langle f, \cdot \rangle = 0$ then $f = 0$. If $\langle \cdot, \mu \rangle = 0$ then $\mu = 0$.*

Let us consider a linear operator H from the signed measures on X to the signed measures on Y, and we can consider whether it admits an *adjoint operator* R:

$$\int (R.f)\ \mathrm{d}\mu = \int f\ \mathrm{d}(H.\mu) \tag{13.9}$$

or

$$\langle R.f, \mu \rangle = \langle f, H.\mu \rangle. \tag{13.10}$$

Remark 13.3.3. If an operator has an adjoint, then this adjoint is unique.

Lemma 13.3.4. *If R is the adjoint of H:*

- *R is continuous if and only H is continuous;*
- $\|R\| = \|H\|$.

Corollary 13.3.5. *The operator mapping an operator onto its adjoint is therefore a linear isometry.*

The reason why we consider such adjoints is the following lemma:

Lemma 13.3.6. *If $H \in \mathscr{L}(\mathscr{M}(X), \mathscr{M}(Y))$ has an adjoint operator $R \in \mathscr{L}(\mathscr{B}(Y, \mathbb{R}), \mathscr{B}(X, \mathbb{R}))$ and H is zero on all the Dirac measures, then H is zero.*

The implications of this lemma on probabilistic program semantics is that if we can prove, which we shall do later, that linear operators representing program semantics have adjoints, then the semantics of two programs will be identical on all input measures if and only if they are identical on discrete measures [42, th. 6.1].

Remark 13.3.7. In general, not all positive continuous linear operators on measures have adjoints in the above sense.

For technical reasons, we shall also have to use a notion of pseudo-adjoint. Let H be a function from $\mathcal{M}_+(X)$ to $\mathcal{M}_+(Y)$. Let us suppose there exists a function R such that for all measurable function $f : Y \to [0, \infty]$ $R(f) : X \to [0, \infty]$, $\langle f, H.\mu \rangle = \langle R.f, \mu \rangle$. We shall then call R the *pseudo-adjoint* of H. As previously, we have:

Remark 13.3.8. An operator has an unique pseudo-adjoint.

Adjoints and pseudo-adjoints are identical notions in well-behaved cases. A continuity condition ensures that we do not get undefined cases, e.g. $\infty - \infty$.

Lemma 13.3.9. *If H is a continuous positive linear operator on measures (positive meaning that $\mu \geq 0 \Rightarrow H.\mu \geq 0$) and R is a positive linear operator that is the pseudo-adjoint of H, then R is the adjoint of H and $\|R\| = \|H\|$.*

13.3.4 Program semantics have adjoints

A few facts are easily proved:

Proposition 13.3.10. *Operators on measures that are lifted from functions (e.g. f_p where $f_p.\mu$ is the measure $X \mapsto \mu(f^{-1}(X))$) have (pseudo-)adjoints: the (pseudo-)adjoint of f_p is $g \mapsto g \circ f$.*

Proposition 13.3.11. *If H_1 and H_2 have (pseudo-)adjoints R_1 and R_2, then $R_2 \circ R_1$ is the adjoint of $H_1 \circ H_2$.*

Proposition 13.3.12. *ϕ_W has (pseudo-)adjoint $R_W = f \mapsto f.\chi_W$ where χ_W is the characteristic function of W.*

Proposition 13.3.13. *If H_1 and H_2 have respective (pseudo-)adjoint R_1 and R_2, then $H_1 + H_2$ has (pseudo-)adjoint $R_1 + R_2$.*

Proposition 13.3.14. *If μ_R is a σ-finite positive measure, $\mu \mapsto \mu \otimes \mu_R$ has pseudo-adjoint $f \mapsto (x \mapsto \int f(x,\cdot)\,\mathrm{d}\mu_R)$.*

This is an application of the Fubini-Tonelli theorem [27, VI.10].

Lemma 13.3.15. *If $f : X \to [0;\infty]$ is a positive measurable function and $(\mu_n)_{n\in\mathbb{N}}$ is a sequence of positive measures, then*

$$\int f \,\mathrm{d}\left(\sum_{n=0}^{\infty} \mu_n\right) = \sum_{n=0}^{\infty} \int f \,\mathrm{d}\mu_n. \tag{13.11}$$

The sum of measures is taken set-wise.

Corollary 13.3.16. *If $(H_n)_{n\in\mathbb{N}}$ are operators on measures with respective pseudo-adjoints $(R_n)_{n\in\mathbb{N}}$, then $\sum_{n=0}^{\infty} H_n$ has pseudo-adjoint $\sum_{n=0}^{\infty} R_n$ (these sum being taken as simple convergences).*

Lemma 13.3.17. *Let c be a probabilistic program. Then the linear operator $[\![c]\!]_p$ has a pseudo-adjoint.*

Corollary 13.3.18. *Since program semantics operator are continuous, of norm less than* 1*, then they have adjoints of norm less than* 1.

Kozen proved [41, 42] the following theorem:

Theorem 13.3.19. *Semantics of probabilistic programs differ if and only if they differ on point masses.*

Proof. This theorem follows naturally from the preceding corollary and lemma 13.3.6. □

We shall often note T^* the adjoint of T. We therefore have defined an adjoint semantics for programs: $[\![c]\!]_p^* \in L(M(Y,\mathbb{R}_+), M(X,\mathbb{R}_+))$ where $M(X,\mathbb{R}_+)$ is the set of measurable functions from X to $\mathbb{R}_+$.

13.4 Abstract interpretation of reverse probabilistic semantics

We wish to apply the usual framework of abstract interpretation to the above semantics; that is, for any program c, we want to build a monotonic function $[\![c]\!]_p^{*\sharp}$: $M(Y,\mathbb{R}_+)^\sharp \to M(X,\mathbb{R}_+)^\sharp$ such that if $f \in \gamma_Y(f^\sharp)$ then $[\![c]\!]_p{}^*.f \in \gamma_X([\![c]\!]_p^{*\sharp}(f^\sharp))$. γ_X (respectively γ_Y) is a *concretization function* that maps an element $f^\sharp$ (with a finite, manageable machine representation) to a subset of X (resp. Y).

13.4.1 Backwards abstract interpretation

We shall suppose we have an abstraction of the normal, non-probabilistic, semantics, suitable for backwards analysis: for any elementary construct c such that $[\![c]\!] : X \to Y$, we must have a monotonic function $[\![c]\!]^{-1\sharp} : Y^\sharp \to X^\sharp$ such that for all $A^\sharp$, $[\![c]\!]^{-1}(\gamma_Y(A^\sharp)) \subseteq \gamma_X([\![c]\!]^{-1\sharp}(A))$. We also must have abstractions for the $\phi_{[\![\mathtt{c}]\!]}$ functions.

Let us note the following interesting property of the "usual" assignment operator: $[\![x \texttt{ := } e]\!]^{-1} = \overline{\pi}_x \circ \phi_{[\![x \texttt{ = } e]\!]}$ where $\overline{\pi}_x$ is the "projection parallel to x": $\overline{\pi}_x(W) = \{\vec{v} \mid \exists \vec{v}' \; \forall x' \; x' \neq x \Rightarrow v_{x'} = v_x\}$. It is therefore quite easy to build reverse abstractions for the elementary constructs.

13.4.2 General form of abstract computations

Let us now suppose we have an abstract domain with appropriate abstract operators for the elementary constructs of the language (we shall give an example of construction of such domains in the next section). We shall see in this section how to deal with the flow-control constructs: the sequence, the test and the loop.

Sequence

Since $[\![e_1\texttt{;}e_2]\!]_p^* = [\![e_1]\!]_p^* \circ [\![e_2]\!]_p^*$ then $[\![e_1\texttt{;}e_2]\!]_p^{*\sharp} = [\![e_1]\!]_p^{*\sharp} \circ [\![e_2]\!]_p^{*\sharp}$.

Tests

Let us recall the reverse semantics of the `if` construct:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p^* = R_{[\![\mathtt{c}]\!]} \circ [\![e_1]\!]_p^* + R_{[\![\mathtt{c}]\!]^C} \circ [\![e_2]\!]_p^* \qquad (13.12)$$

This equation gets straightforwardly lifted to the abstract domain:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p^{*\sharp} = R^\sharp_{[\![\mathsf{c}]\!]} \circ [\![e_1]\!]_p^{*\sharp} + R^\sharp_{[\![\mathsf{c}]\!]^C} \circ [\![e_2]\!]_p^{*\sharp} \qquad (13.13)$$

is a valid abstraction.

Loops

Let us recall the reverse semantics of the `while` construct:

$$[\![\texttt{while } c \texttt{ do } e]\!]_p^* = \sum_{n=0}^{\infty} \left(R_{[\![c]\!]} \circ [\![e]\!]_p^* \right)^n \circ R_{[\![\mathsf{c}]\!]^C} \qquad (13.14)$$

Defining f_n recursively, as follows: $f_0 = \lambda x.0$ and $f_{n+1} = \psi f_n$, with

$$\psi(g) = R_{[\![\mathsf{c}]\!]^C}.f + R_{[\![c]\!]} \circ [\![e]\!]_p^*.g,$$

we can rewrite equation 13.14 as $[\![\texttt{while } c \texttt{ do } e]\!]_p^*.f = \lim_{n\to\infty} f_n$. We wish to approximate this limit in the measure space by an abstract element.

ψ gets lifted straightforwardly to an abstract operator: $\psi^\sharp(g^\sharp) = R^\sharp_{[\![\mathsf{c}]\!]^C}.f^\sharp +^\sharp R^\sharp_{[\![c]\!]} \circ [\![e]\!]_p^{*\sharp}.g^\sharp$. Let us define $f_0^\sharp$ to be an abstraction of the set $\{f_0\}$ and $f_{n+1}^\sharp = \psi^\sharp(f_n^\sharp)$. Obviously, for all n, $f_n \in \gamma(f_n^\sharp)$. If there exists an N such that $\forall n \geq N$, then $\lim_{n\to\infty} f_n \in \gamma f_N^\sharp$ since $\gamma f_N^\sharp$ is topologically closed. We have therefore found an abstraction of $[\![\texttt{while } c \texttt{ do } e]\!]_p^*.f$.

If the lattice $T^\sharp$ is such that all ascending sequences are stationary, then such a N will necessarily exist. In general, such a N does not exist and we are forced to use so-called *widening operators* [18, §4.3], as follows: we replace the sequence f_n by the sequence defined by $\hat{f}_0^\sharp = f_0^\sharp$ and $\hat{f}_{n+1}^\sharp = \hat{f}_n^\sharp \nabla_n \psi^\sharp(\hat{f}_n^\sharp)$ where ∇_n are widening operators:

- for all a and b, $a \sqsubseteq a\nabla b$ and $b \sqsubseteq a\nabla b$;
- for all sequence $(u_n)_{n\in\mathbb{N}}$ and any sequence $(v_n)_{n\in\mathbb{N}}$ defined recursively as $v_{n+1} = v_n \nabla u_n$, then (v_n) is stationary.

Obviously, for all n, $f_n \sqsubseteq \gamma(\hat{f}_n^\sharp)$. Since $(\hat{f}_n^\sharp)_{n\in\mathbb{N}}$ is stationary after rank N, and $\gamma(\hat{f}_N^\sharp)$ is topologically closed, this implies that $\lim_{n\to\infty} f_n \in \gamma f_N^\sharp$ as above.

13.5 Discussion

Our forward denotational semantics for purely probabilistic programs is essentially the same as the one proposed by Kozen [41,42]. We on the other hand introduced the adjoin semantics, which gives rise to a particularly important backwards analysis while providing an elegant proof to some results that were tediously proved in Kozen's papers.

It should be noted that while both semantics are equally accurate when dealing with purely probabilistic programs (containing no nondeterminism), they are very different when dealing with programs containing nondeterministic choices. The problem is that the forward semantics on the tests is a very coarse approximation when there is nondeterminism; for instance, considering the following program:

```
x = nondeterministic_choice_01();
if (x)
{
} else
{
}
```

then, applying our formulas, the measured upper bound on the probability of termination of the program is... 2!

Let us see of such a coarse result can be produced. The state space of our program is $X = \{0,1\}$, depending on the value of variable `x`. We use $\mathcal{P}(\mathcal{M}_{\leq 1}(X))$ as the abstract domain. After the first step of execution, the set of measures is the set of measures on X of total weight 1. Then the abstract version of equation 13.2 is:

$$\begin{aligned}
&[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]^\sharp_p(\mu^\sharp) \\
&\qquad = [\![e_1]\!]_p \circ \phi^\sharp_{[\![c]\!]}(\mu^\sharp) +^\sharp [\![e_2]\!]_p \circ \phi^\sharp_{[\![c]\!]^C}(\mu^\sharp) \\
&= \{\mu \mid \mu(1) \leq 1 \wedge \mu(0) = 0\} +^\sharp \{\mu \mid \mu(0) \leq 1 \wedge \mu(1) = 0\} \\
&\qquad \ni \{\mu \mid \mu(0) = 1 \wedge \mu(1) = 1\}. \quad (13.15)
\end{aligned}$$

That latter measure has total weight 2.

Informally, the weakness of the forward analysis is that it is not relational with respect to the branches of the test: it does not conserve the information that the total weight of the measures on both sides must be the same as the weight of the measure before the test. The obvious solution to this problem is to use the forward analysis of the operational semantics (§ 8.4), which conserves that relation by integrating the program counter into the machine state.

On the other hand, the backwards analysis is a direct counterpart of the backwards analysis for Markov decision processes (§ 8.3.1). We conclude that the backwards analysis is to be preferred when following the denotational point of view.

Chapter 14

The Abstract Monte-Carlo Method

The methods exposed in the preceding chapters have focused on symbolically representing measures, or measurable functions. We have not so far made use of possibilities of the probabilities, among which the possibility of using some probabilistic choice in the analyzer itself. This is after all a very familiar way of proceeding: to study the respective probabilities of the faces of an imperfect die, one can roll it a great number of times and count the frequencies of the outcomes. We shall see here how to extend that approach using abstract interpretation.

14.1 The Monte-Carlo Testing Method

Monte-Carlo sampling is a widely used techniques in the resolution of probability problems for which no analytical solution is known. Its name comes from it being based on randomization, just like gambling in casinos. While its name would make Monte-Carlo sampling look like an unsound, hazardous method, it actually is a mathematically sound method. We shall begin by a summary on Monte-Carlo sampling.

14.1.1 An Intuitive View

Monte-Carlo is a general term for a wide range of methods, all having in common the use of random generators and statistical convergence. Here we consider the problem of finding the probability that a certain event holds.

For instance, let us consider the random choice of a point M, uniformly distributed in $[0,1]^2$. We want to determine experimentally the probability of

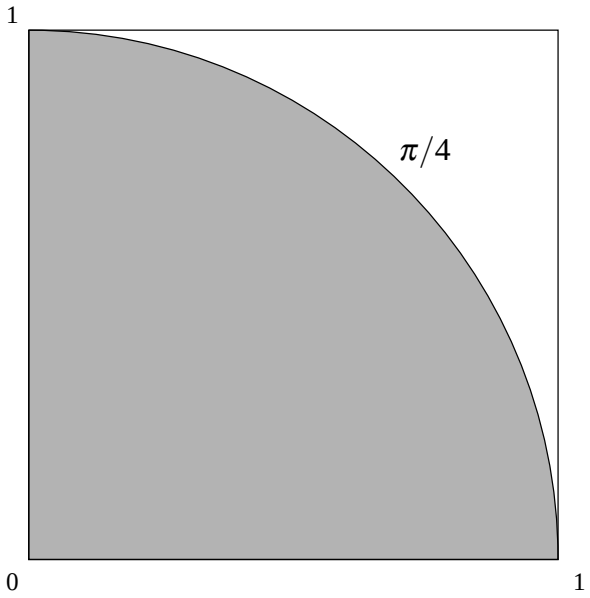


Figure 14.1: Computing $\pi/4$ using the Monte-Carlo method.

the property $P(M)$: "M belongs to the quarter-disc of center 0 and radius 1" (Fig. 14.1). We can use the following algorithm:

$c \leftarrow 0$
for $i = 1$ to n **do**
  $x \leftarrow$ random
  $y \leftarrow$ random
  if $x^2 + y^2 \leq 1$ **then**
    $c \leftarrow c + 1$
  end if
end for
$p \leftarrow c/n$

Of course, in our simple geometric case, using such a method would be very inefficient, since there are faster methods of computing the desired area. Yet such a method is interesting if the shape the area of which is to be estimated is described by a complex characteristic function. Program analysis is generally in this latter case.

A lenghtier explanation of the Monte-Carlo method is featured in appendix B.

14.1.2 The Monte-Carlo Testing Method Applied to Programs

The reader unfamiliar with probability theory is invited to consult appendix 6.3.3.

Let us consider a deterministic program c whose input x lies in X and whose output lies in Z. For the sake of simplicity, we shall suppose in this sub-section that c always terminates. We shall note $[\![c]\!] : X \mapsto Z$ the semantics of c (such that $[\![\mathsf{c}]\!](x)$ is the result of the computation of c on the input x). We shall take X and Z two measurable spaces and constrain $[\![c]\!]$ to be measurable. These measurability conditions are technical and do not actually restrict the scope of programs to consider, since they are fulfilled as long as all elementary operations (assignments, arithmetic) have measurable semantics [54] — which is always the case for countable domains such as the integers, and is the case for elementary operations over the real numbers.

Let us consider that the input x to the program is a random variable whose probability measure is μ and that $W \subseteq Z$ is a measurable set of final states whose probability we wish to measure. The probability of W is therefore $\mu([\![c]\!]^{-1}(W))$. Noting

$$t_W(x) = \begin{cases} 1 & \text{if } [\![c]\!](x) \in W \\ 0 & \text{otherwise,} \end{cases}$$

this probability is the expectation $\mathbf{E}t_W$.

Let us apply the Monte-Carlo method for averages to this random variable t_W (see appendix B). $\mathbf{E}t_W$ is then approximated by n random trials:

$c \leftarrow 0$
for $i = 1$ to n **do**
  $x \leftarrow \text{random}(\mu)$
  run program c on input x.
  if program run ended in a state in W **then**
    $c \leftarrow c + 1$
  end if
end for
$p \leftarrow c/n$

A confidence interval can be supplied, for instance using the Chernoff-Hoeffding bound (appendix B, inequation B.1): there is at least a $1 - \varepsilon$ probability that the true expectation $\mathbf{E}t_W$ is less than $p' = p + \sqrt{\frac{-\log \varepsilon}{2n}}$ (We shall see the implications in terms of complexity of these safety margins in more detail in section 14.3; see also Fig. 14.3).

This method suffers from two drawbacks that make it unsuitable in certain

cases:

- It supposes that all inputs to the program are either constant or driven according to a known probability distribution. In general, this is not the case: some inputs might well be only specified by intervals of possible values, without any probability measure. In such cases, it is common [61] to assume some kind of distribution on the inputs, such as an uniform one for numeric inputs. This might work in some cases, but grossly fail in others, since this is mathematically unsound.

- It supposes that the program terminates every time within an acceptable delay.

We propose a method that overcomes both of these problems.

14.1.3 Abstract Monte-Carlo

We shall now consider the case where the inputs of the program are divided in two: those, in X, that follow a random distribution μ and those that simply lie in some set Y. Now $[\![c]\!] : X \times Y \to Z$. The probability we are now trying to quantify is $\mu\{x \in X \mid \exists y \in Y \ [\![c]\!]\langle x,y\rangle \in W\}$.

Mathematical conditions

Some technical conditions must be met such that this probability is well-defined; namely, the spaces X and Y must be standard Borel spaces [38, Def. 12.5]. Since countable sets, $\mathbb{R}$, products of sequences of standard Borel spaces are standard Borel [38, §12.B], this restriction is of no concern to most semantics.

Let us first prove that the probability we want to evaluate is well-defined.

Lemma 14.1.1. *If $[\![c]\!] : X \times Y \to Z$ is measurable, then for any measurable $W \subseteq Z$, $\{x \in X \mid \exists y \in Y \ [\![c]\!]\langle x,y\rangle \in W\}$ is measurable for the completed measure.*

Proof. Let us suppose X and Y are standard Borel spaces [38, §12.B]. $X \times Y$ is thus a Polish space [38, §3.A] such that the first projection π_1 is continuous. Let $A = \{x \in X \mid \exists y \in Y \ [\![c]\!]\langle x,y\rangle \in W\}$; then $A = \pi_1([\![c]\!]^{-1}(W))$. Since $[\![c]\!]$ is a measurable function and W is a measurable set, $[\![c]\!]^{-1}(W)$ is a Borel subset in the Polish space $X \times Y$. A is therefore analytic [38, Def. 14.1]; from Lusin's theorem [38, Th. 21.10], it is universally measurable. In particular, it is μ-measurable [38, §17.A]. $\mu(A)$ is thus well-defined. □

Noting

$$t_W(x) = \begin{cases} 1 & \text{if } \exists y \in Y \ [\![c]\!] \langle x,y \rangle \in W \\ 0 & \text{otherwise,} \end{cases}$$

this probability is the expectation $\mathbf{E}t_W$.

While it would be tempting, we cannot use a straightforward Monte-Carlo method since, in general, t_W is not computable, even if we restrict ourselves to programs that always terminate. Indeed:

Lemma 14.1.2. *Let us consider a Turing-complete language L, and its subset L_T of programs $\mathbb{N} \to \{0,1\}$ that terminate on all inputs. Let us consider $f : L_T \to \{0,1\}$ such that $f(P) = 1$ if and only if $\exists b \in \mathbb{N} \ [\![P]\!](b) = 1$. Then f is not computable.*

Proof. Let us take a Turing machine (or program in a Turing-complete language) F. There exists an algorithmic translation taking F as input and outputting the Turing machine $\tilde{F}$ computing the total function $\varphi_{\tilde{F}}$ such that

$$\varphi_{\tilde{F}} \langle x,y \rangle = \begin{cases} 1 & \text{if } F \text{ terminates in } y \text{ or less steps on input } x \\ 0 & \text{otherwise.} \end{cases}$$

Let us remark that $\tilde{F}$ terminates on all inputs.

Let us take $X = Y = \mathbb{N}$ and $Z = \{0,1\}$ and the program $\tilde{F}$, and define $t_{\{1\}}$ as before. $t_{\{1\}}(x) = 1$ if and only if F terminates on input x. It is a classical fact of computability theory that the $t_{\{1\}}$ function is not computable for all F [68]. □

Abstract Interpretation Can Help

Abstract interpretation (see §6.2.3) is a general scheme for approximated analyses of safety properties of programs. We use an abstract interpreter to compute a function $T_W : X \to \{0,1\}$ testing the following safety property:

- $T_W(x) = 0$ means that no value of $y \in Y$ results in $[\![c]\!](x,y) \in W$;
- $T_W(x) = 1$ means that some value of $y \in Y$ *may* result in $[\![c]\!](x,y) \in W$.

This means that for any x, $t_W(x) \leq T_W(x)$.

Let us note the intuitive meaning of such an approximation: since we cannot check algorithmically whether some points belong to a certain area A ($t_W^{-1}(\{1\})$), we check whether they belong to some area that is guaranteed to be a superset of A. This is analogous to what would happen if we tried to compute the area of our quarter disc (Fig. 14.1) but we were restricted to testing the belonging to,

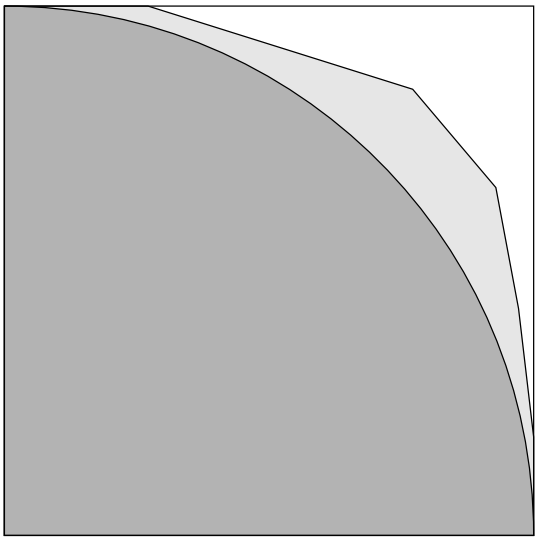


Figure 14.2: Computing $\pi/4$ using an approximate Monte-Carlo method.

say, convex polygones. We could then use the following scheme: find a convex polygone enclosing the disc (Fig. 14.2) and compute the area of this polygon using the Monte-Carlo method.

Let us use the following algorithm:

$c \leftarrow 0$
for $i = 1$ to n **do**
  $x \leftarrow \text{random}(\mu)$
  $c \leftarrow c + T_W(x)$
end for
$p \leftarrow c/n$

With the same notations as in the previous sub-section: $t_W^{(c)} \leq T_W^{(n)}$.

As in the preceding section, we wish to obtain a confidence interval. Let us take $t > 0$. Using the Chernoff-Hoeffding bound or some other bound, we obtain an upper bound B of $\Pr\left(\mathbf{E}t_W \geq t_W^{(n)} + \alpha\right)$. Since $t_W^{(c)} \leq T_W^{(n)}$, the event $\mathbf{E}t_W \geq T_W^{(n)} + \alpha$ is a sub-event of $\mathbf{E}T_W \geq T_W^{(n)} + \alpha$, thus we can meaningfully declare that $\Pr\left(\mathbf{E}t_W \geq T_W^{(n)} + \alpha\right) \leq B$. Thus *any confidence upper bound obtained for the concrete average is valid if considering the abstract average*. In particular, the Chernoff-Hoeffding bound establishes that there is at least a $1 - \varepsilon$ probability that the true expectation $\mathbf{E}t_W$ is less than $p' = T_W^{(n)} + \sqrt{\frac{-\log \varepsilon}{2n}}$.

We shall see in the following section how to build abstract interpreters with a view to using them for this Monte-Carlo method.

14.2 Analysis Block-Structured Imperative Languages

From the previous section, it would seem that it is easy to use any abstract interpreter in a Monte-Carlo method. Alas, we shall now see that special precautions must be taken in the presence of calls to random generators inside loops or, more generally, fixpoints.

14.2.1 Concrete Semantics

We have for now spoken of deterministic programs taking one input x chosen according to some random distribution and one input y in some domain. Calls to random generators (such as the POSIX `drand48()` function [63]) are usually modeled by a sequence of independent random variables. If a bounded number of calls ($\leq N$) to such generators is used in the program, we can consider them as input values: x is then a tuple $\langle x_1, \ldots, x_N, v \rangle$ where x_1, ..., x_n are the values for the generator and v is the input of the program. If an unbounded number of calls can be made, it is tempting to consider as an input a countable sequence of values $(x_n)_{n\in\mathbb{N}}$ where x_1 is the result of the first call to the generator, x_2 the result of the second call...; a formal description of such a semantics has been made by Kozen [41, 42].

Such a semantics is not very suitable for program analysis. Intuitively, analyzing such a semantics implies tracking the number of calls made to number generators. The problem is that such simple constructs as:

```
if (...)  { random(); } else {}
```

are difficult to handle: the countings are not synchronized in both branches.

Denotational Semantics for Non-Random Programs

We shall now propose another semantics, identifying occurrences of random generators by their program location and loop indices. The Backus-Naur form of the programming language we shall consider is:

instruction ::= *elementary*
| *instruction* `;` *instruction*
| `if` *boolean_expr*
`then` *instruction*
`else` *instruction*
`endif`
| `while` *boolean_expr*

`do` *instruction*
`done`

We leave the subcomponents largely unspecified, as they are not relevant to our method. *elementary* instructions are deterministic, terminating basic program blocks like assignments and simple expression evaluations. *boolean_expr* boolean expressions, such as comparisons, have semantics as sets of acceptable environments. For instance, a *boolean_expr* expression can be `x < y + 4`; its semantics is the set of execution environments where variables `x` and `y` verify the above comparison. If we restrict ourselves to a finite number n of integer variables, an environment is just a n-tuple of integers.

The denotational semantics of a code fragment c is a mapping from the set X of possible execution environments before the instruction into the set Y of possible environments after the instruction. Let us take an example. If we take environments as elements of $\mathbb{Z}^3$, representing the values of three integer variables `x`, `y` and `z`, then $[\![\texttt{x:=y+z}]\!]$ is the strict function $\langle x,y,z\rangle \mapsto \langle y+z,y,z\rangle$. Semantics of basic constructs (assignments, arithmetic operators) can be easily dealt with this forward semantics; we shall now see how to deal with flow control.

The semantics of a sequence is expressed by simple composition

$$[\![e_1\texttt{;}\ \ e_2]\!] = [\![e_2]\!] \circ [\![e_1]\!] \tag{14.1}$$

Tests get expressed easily, using as the semantics $[\![c]\!]$ of a boolean expression c the set of environments it matches:

$$[\![\texttt{if}\ \ c\ \ \texttt{then}\ \ e_1\ \ \texttt{else}\ \ e_2]\!](x) \quad = \quad \text{if } x \in [\![c]\!] \text{ then } [\![e_1]\!](x) \text{ else } [\![e_2]\!](x) \tag{14.2}$$

and loops get the usual least-fixpoint semantics (considering the point-wise extension of the Scott flat ordering on partial functions)

$$[\![\texttt{while}\ \ c\ \ \texttt{do}\ \ f]\!] \quad = \quad \text{lfp}(\lambda\phi.\lambda x.\text{if } x \in [\![c]\!] \text{ then } \phi \circ [\![f]\!](x) \text{ else } x). \tag{14.3}$$

Non-termination shall be noted by $\perp$.

Our Denotational Semantics For Random Programs

We shall consider a finite set G of different generators. Each generator g outputs a random variable r_g with distribution μ_g; each call is independent from the precedent calls. Let us also consider the set P of program points and the set $\mathbb{N}^*$ of finite sequences of positive integers. The set $C = P \times \mathbb{N}^*$ shall denote the possible times

in an execution where a call to a random generator is made: $\langle p, n_1 n_2 ... n_l \rangle$ notes the execution of program point p at the n_1-th execution of the outermost program loop, ..., n_l-th execution of the innermost loop at that point. C is countable. We shall suppose that inside the inputs of the program there is for each generator g in G a family $(\hat{g}_{\langle p,w \rangle})_{\langle p,w \rangle \in C}$ of random choices.

Our goal is to express the semantics of a program P as a function $[\![P]\!] : X \times Y \to Z$ where $X = \prod g \in G R_i^C$, R_i being the range of the random generator $r^{(i)}$. The semantics of the language then become:

For composition:

$$[\![e_1 \texttt{;}\ \ e_2]\!]_r = [\![e_2]\!] \circ [\![e_1]\!] \tag{14.4}$$

Tests get expressed easily, using as the semantics $[\![c]\!]$ of a boolean expression c the set of environments it matches:

$$[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_r . \langle w, x \rangle = \\ \text{if } x \in [\![c]\!] \text{ then } [\![e_1]\!] . \langle w, x \rangle \text{ else } [\![e_2]\!] . \langle w, x \rangle \tag{14.5}$$

Loops get the usual least-fixpoint semantics (considering the point-wise extension of the Scott flat ordering on partial functions):

$$[\![\texttt{while } c \texttt{ do } f]\!]_r . \langle w_0, x_0 \rangle = \\ \text{lfp}\,(\lambda \phi . \lambda \langle w, x \rangle . \text{if } x \in [\![c]\!] \text{ then } \phi \circ S \circ [\![f]\!] \langle w, x \rangle) \text{ else } x) . \langle 1 . w_0, x_0 \rangle \tag{14.6}$$

where $S . \langle c . w, x \rangle = \langle (c+1) . w, x \rangle$. The only change is that we keep track of the iterations of the loop.

As for random expressions,

$$[\![p : \texttt{random}_g]\!]_r . \langle w, x \rangle = \hat{g}_{\langle p,w \rangle} \tag{14.7}$$

where p is the program point.

This semantics is equivalent to the denotational semantics proposed by Kozen [41, 42, 2nd semantics] and Monniaux [54], the semantic of a program being a continuous linear operator mapping an input measure to the corresponding output. The key point of this equivalence is that two invocations of random generators in the same execution have different indices, which implies that a fresh output of a random generator is randomly independent of the environment coming to that program point.

14.2.2 Analysis

Let us apply the Monte-Carlo abstract interpretation technique (§14.1.3) to our concrete semantics (§14.2.1). Let us first recall that in this case, the random input

x is actually a tuple $(\hat{g}^{(1)},\ldots,\hat{g}^{(n)})$. Each $\hat{g}^{(i)}$ corresponds to a random number generator named i (for instance, i can be textttdrand48() or `random()`)...); for a generator g_i, $\hat{g}_i$ is a family $\left(\hat{g}^{(i)}_{\langle p,w\rangle}\right)_{\langle p,w\rangle\in C}$ of random choices.

We shall suppose we have a normal abstract interpreter for the target language at our disposal. To adapt it to our probabilistic requirements, we shall first add the semantics of the random generators. Formally speaking, the semantics of an operation $[\![\texttt{x:=random()}]\!]$ is $\left[\!\!\left[\texttt{x:=}\hat{g}^{(\texttt{random()})}_{\langle l_1,\ldots,l_n\rangle}\right]\!\!\right]$ where $\langle l_1,\ldots,l_n\rangle$ is the tuple of numbers of fixpoint iterations. This equality is lifted to abstract operations: $[\![\texttt{x:=random()}]\!]^\sharp_r = \left[\!\!\left[\texttt{x:=}\hat{g}^{(\texttt{random()})}_{\langle l_1,\ldots,l_n\rangle}\right]\!\!\right]^\sharp$. This simply means that when the abstract interpreter encounters a call to a random generator, it effectively computes a random number C and uses it as if it were interpreting the instruction `x:=C`.

The problem now is that the abstract semantics $[\![\texttt{x:=random()}]\!]^\sharp_r$ may change from iteration to iteration. How can we do fixpoint computations? The answer is to provide a second possible abstract semantics, not depending on the current number of iterations, $[\![\texttt{x:=random()}]\!]^\sharp_{\text{fix}}$. This semantics treats the call to the random generator as purely nondeterministic, it just asserts that the variable `x` lies in the range $R_{\texttt{random()}}$ of the random generator `random()`.

The Monte-Carlo abstract interpreter should therefore run as follows:

- initial iterations of fixpoints should be randomized;
- when trying to stabilize the fixpoint (computing successive iterations, possibly with widenings, until a fixpoint is reached), treat the random generators as purely nondeterministic.

14.2.3 Arbitrary control-flow graphs

The abstract interpretation framework can be extended to logic languages, functional languages and imperative languages with recursion and other "complex programming mechanisms (call-by-reference, local procedures passed as parameters, non-local gotos, exceptions)" [6]. In such cases, the semantics of the program are expressed as a fixpoint of a system of equations over parts of the domain of environments. The environment in that case includes the program counter, call stack and memory heap; of course a suitable abstract lattice must be used.

Analyzing a program P written in a realistic imperative language is very similar to analyzing the following interpreter:

$s \leftarrow$ initial state for P
while s is not a termination state **do**
  $s \leftarrow N(s)$

end while

where $N(s)$ is the next-state function for P (operational semantics). The abstract analyzer analysis that loop using an abstract state and an abstract version of N. Most analyses partition the domain of states according to the program counter, and the abstract interpreter then computes the least fixpoint of a system of semantic equations.

Such an analysis can be randomized in exactly the same fashion as the one for block-structured programs presented in the previous section. It is all the same essential to store the generated values in a table such that backwards analysis can be used.

14.2.4 Ineffective or unsound suggestions

In many cases, we shall analyze programs whose source of randomness is a pseudo-random generator [40, chapter 3]. It is therefore possible to use ordinary abstract interpretation to analyze them. This is a perfectly sound approach, albeit an ineffective one. Abstract interpretation is designed to exhibit structures and invariants in programs. A pseudo-random generator is designed to avoid exhibiting any structure in its output. A pseudo-random generator whose behavior could be analyzed by ordinary abstract interpretation would therefore be a low-quality, predictible one.

It has also been suggested that we should randomize the end of the iterations (that is, the final part of terminating loops) as well as the initial sequence. Alas, there does not seem to be a way to make this work easily. Let us take the following program, where `random()` is 0 or 1 with probability 0.5:

```
n=0;
while (random()==0) n++;
```

Since the program terminates if and only if the last `random()` call is 1, a naive randomization may conclude that the program terminates only half of the time. This is false, since the program terminates with probability 1.

We could think of improving this method by randomizing taking into account reachability conditions: termination of the program should be reachable from the chosen random value. This is unfortunately not doable: there is no algorithmic means to compute the set of random values that allow termination. Furthermore, over-approximating this set would under-estimate the probability, while our aim is to over-estimate the probability. It might be possible to achieve an over-estimation result using backwards under-approximating abstract interpretation, but in any case it would be difficult and imprecise.

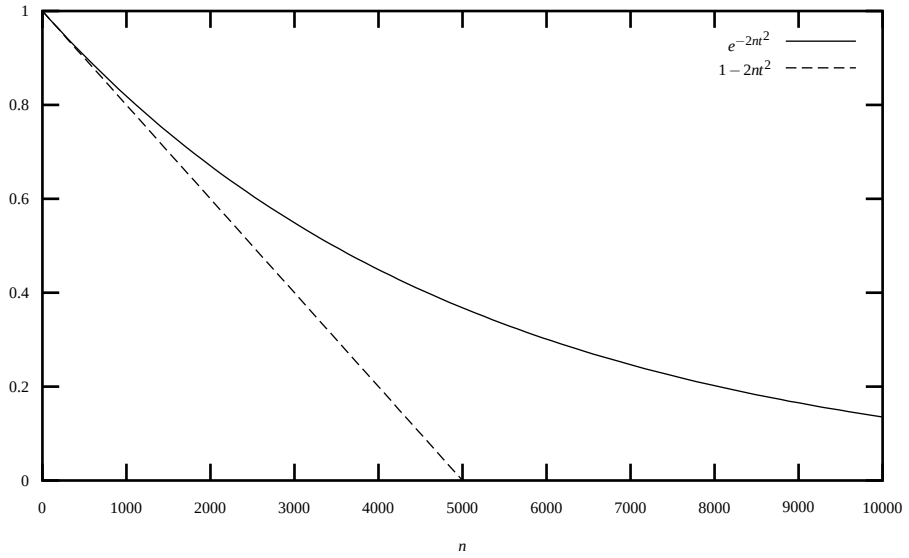


Figure 14.3: Upper bound on the probability that the probability estimate exceeds the real value by more than t, for $t = 0.01$.

14.3 Complexity

The complexity of our method is the product of two independent factors:

- the complexity of one ordinary static analysis of the program; strictly speaking, this complexity depends not only on the program but on the random choices made, but we can take a rough "average" estimate that depends only on the program being analyzed;
- the number of iterations, that depends only on the requested confidence interval; the minimal number of iterations to reach a certain confidence criterion can be derived from inequalities [75, appendix A] such as inequation (B.1) and does not depend on the actual program being analyzed.

We shall now focus on the latter factor, as the former depends on the particular case of analysis being implemented.

Let us recall inequation (B.1): $\Pr\left(\mathbf{E}t_W \geq \overline{t}_W^{(n)} + t\right) \leq e^{-2nt^2}$. *It means that to get with* $1-\varepsilon$ *probability an approximation of the requested probability* μ*, it is sufficient to compute an experimental average over* $\left\lceil -\frac{\log \varepsilon}{2t^2} \right\rceil$ *trials.*

This exponential improvement in quality (Fig. 14.3) is nevertheless not that interesting. Indeed, in practice, we might want ε and t of the same order of magnitude as μ. Let us take $\varepsilon = \alpha t$ where α is fixed. We then have $n \sim -\frac{\log t}{t^2}$, which

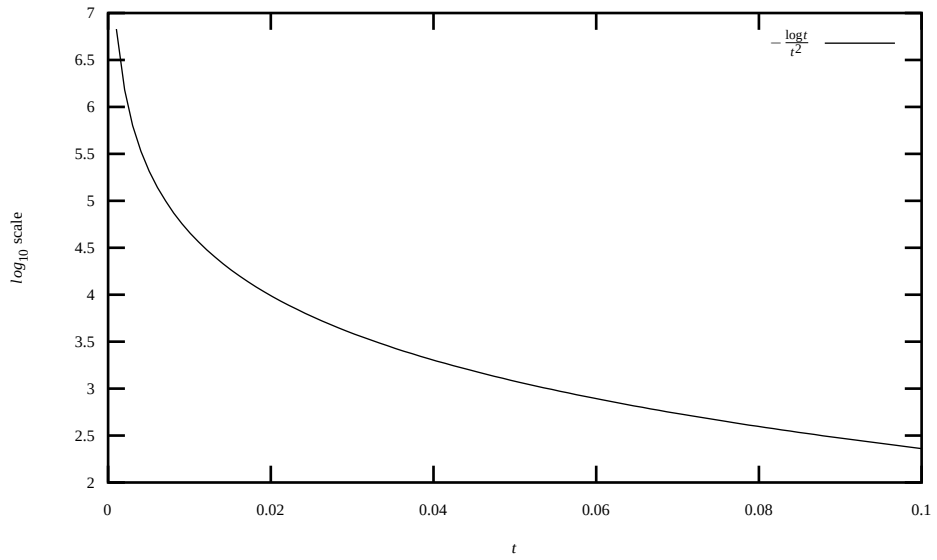


Figure 14.4: Numbers of iterations necessary to achieve a probability of false report on the same order of magnitude as the error margin.

indicates prohibitive computation times for low probability events (Fig. 14.4). *This high cost of computation for low-probability events is not specific to our method; it is true of any Monte-Carlo method*, since it is inherent in the speed of convergence of averages of identically distributed random variables; this relates to the speed of convergence in the central limit theorem [75, ch 1], which says in effect that the average of n identically distributed random variables resembles a Gaussian whose deviate decreases in $1/\sqrt{n}$:

Theorem 14.3.1 (central limit). *Let X_1, X_2,... be independent, identically distributed random variables having mean μ and finite nonzero variance σ^2. Let $S_n = X_1 + \cdots + X_n$. Then*

$$\lim n \to \infty P\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \leq x\right) = \Phi(x) \tag{14.8}$$

Where $\Phi(x)$ is the probability that a standard normal variable is less than x.

It can nevertheless be circumvented by tricks aimed at estimating the desired low probability by computing some other, bigger, probability from which the desired result can be computed.

Fortunately, such an improvement is possible in our method. If we know that $\pi_1([\![c]\!]^{-1}(W)) \subseteq R$, with a measurable R, then we can replace the random variable t_W by its restriction to R: $t_{W|R}$; then $\mathbf{E}t_W = \Pr(R).\mathbf{E}t_{W|R}$. If $\Pr(R)$ and $\mathbf{E}t_W$ are on

the same order of magnitude, this means that $\mathbf{E}t_{W|R}$ will be large and thus that the number of required iterations will be low. *Such a restricting R can be obtained by static analysis, using ordinary backwards abstract interpretation.*

A salient point of our method is that our Monte-Carlo computations are **highly parallelizable**, with linear speed-ups: n iterations on 1 machine can be replaced by n/m iterations on m machines, with very little communication. Our method thus seems especially adapted for clusters of low-cost PC with off-the-shelf communication hardware, or even more distributed forms of computing. Another improvement can be to compute bounds for several W sets simultaneously, doing common computations only once.

14.4 Practical Implementation and Experiments

We have a prototype implementation of our method, implemented on top of an ordinary abstract interpreter doing forward analysis using integer and real intervals. Figures 14.5 to 14.7 show various examples for which the probability could be computed exactly by symbolic integration. Figure 14.8 shows a simple program whose probability of outcome is difficult to figure out by hand. Of course, more complex programs can be handled, but the current lack of support of user-defined functions and mixed use of reals and integers prevents us from supplying real-life examples. We hope to overcome these limitations soon as implementation progresses.

14.5 Equivalence of Semantics

We shall here prove that our semantics based on generator occurences are equivalent to the denotational semantics given in §13.1.2.

Theorem 14.5.1. *For any program P, for any subset W of the range of $[\![P]\!]$,*

$$([\![P]\!]_p.\mu)(W) = (\mu \otimes \mu_R)([\![P]\!]_r^{-1}(W)) \tag{14.9}$$

Proof. Let us proceed by induction on the structure of the program.

Non-probabilistic constructs Let us apply the definitions of $[\![p]\!]_p$ and $[\![p]\!]_r$ where p is a basic, non-probabilistic construct such as an assignment or an arithmetic operation:

$$\begin{aligned}([\![p]\!]_p.\mu)(W) &= \mu([\![p]\!]^{-1}(W)) \\ &= \mu \otimes \mu_R([\![p]\!]^{-1}(W) \times R) = \mu \otimes \mu_R([\![p]\!]_r^{-1}(W))\end{aligned} \tag{14.10}$$

```
int x, i;
know (x>=0 && x<=2);
i=0;
while (i < 5)
{
  x += coin_flip();
  i++;
}
know (x<3);
```

Figure 14.5: **Discrete probabilities.** The analyzer establishes that, with **99% safety**, the probability p of the outcome ($x < 3$) is less than **0.509** given worst-case nondeterministic choices of the precondition ($x \geq 0 \wedge x \leq 2$). The analyzer used $n = 10000$ random trials. Formally, p is $\Pr\big(\texttt{coin\_flip} \in \{0,1\}^5 \mid \exists x \in [0,2] \cap \mathbb{Z}\ [\![P]\!](\texttt{coin\_flip}, x) < 3\big)$. Each `coin_flip` is chosen randomly in $\{0,1\}$ with a uniform distribution. The exact value is **0.5**.

```
double x;
know (x>=0. && x<=1.);
x+=uniform()+uniform()+uniform();
know (x<2.);
```

Figure 14.6: **Continuous probabilities.** The analyzer establishes that, with **99%** safety, the probability p of the outcome ($x < 2$) is less than **0.848** given worst-case nondeterministic choices of the precondition ($x \geq 0 \wedge x \leq 1$). The analyzer used $n = 10000$ random trials. Formally, p is $\Pr\big(\texttt{uniform} \in [0,1]^3 \mid \exists x \in [0,1]\ [\![P]\!](\texttt{uniform}, x) < 2\big)$. Each `uniform` is chosen randomly in $[0,1]$ with the Lebesgue uniform distribution.
The exact value is $5/6 \approx \mathbf{0.833}$.

```
double x, i;
know(x<0.0 && x>0.0-1.0);
i=0.;

while (i < 3.0)
{
  x += uniform();
  i += 1.0;
}
know (x<1.0);
```

Figure 14.7: **Loops.** The analyzer establishes that, with **99% safety**, the probability p of the outcome ($x < 1$) is less than 0.859 given worst-case nondeterministic choices of the precondition ($x < 0 \wedge x > -1$). The analyzer used $n = 10000$ random trials. Formally, p is $\Pr\left(\texttt{uniform} \in [0,1]^3 \mid \exists x \in [0,1]\ [\![P]\!](\texttt{uniform}, x) < 1\right)$. Each `uniform` is chosen randomly in $[0,1]$ with the Lebesgue uniform distribution. The exact value is $5/6 \approx \mathbf{0.833}$.

```
{
  double x, y, z;
  know (x>=0. && x<=0.1);
  z=uniform(); z+=z;
  if (x+z<2.)
  {
    x += uniform();
  } else
  {
    x -= uniform();
  }
  know (x>0.9 && x<1.1);
}
```

Figure 14.8: The analyzer establishes that, with **99% safety**, the probability p of the outcome ($x > 0.9 \wedge x < 1.1$) is less than **0.225** given worst-case nondeterministic choices of the precondition ($x \geq 0 \wedge x \leq 0.1$). Formally, p is $\Pr\left(\texttt{uniform} \in [0,1]^2 \mid \exists x \in [0,0.1]\ [\![P]\!](\texttt{uniform}, x) \in [0.9,1.1]\right)$. Each `uniform` is chosen randomly in $[0,1]$ with the Lebesgue uniform distribution.

Random number generation Let us apply the definitions of $[\![\texttt{random()}]\!]_p$ and $[\![\texttt{random()}]\!]_r$ and the fact that μ_R is a product of independently distributed measures, the one corresponding to this occurrence of `random()` being $\mu_{\texttt{random()}}$:

$$\begin{aligned}([\![\texttt{random()}]\!]_p.\mu)(W) &= \mu \times \mu_{\texttt{random()}}(W) \\ &= \mu \times \mu_R([\![\texttt{random()}]\!]_r^{-1}(W)) \quad (14.11)\end{aligned}$$

Composition Let us first apply the induction hypothesis:

$$\begin{aligned}([\![p_1\texttt{;}\ \ p_2]\!]_p.\mu)(W) &= ([\![p_2]\!]_p \circ [\![p_1]\!]_p.\mu)(W) \\ = ((([\![p_1]\!]_p.\mu) \otimes \mu_R)([\![p_2]\!]_r^{-1}(W)) &= \int ([\![p_1]\!]_p.\mu)(S([\![p_2]\!]_r^{-1}(W),y))\,\mathrm{d}\mu_R(y) \\ &= \int \mu \otimes \mu_R([\![p_1]\!]_r^{-1}(S([\![p_2]\!]_r^{-1}(W),y))\,\mathrm{d}\mu_R(z) \\ &= \iiint \chi_A(x,y,z)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu_R(y)\,\mathrm{d}\mu_R(z) \quad (14.12)\end{aligned}$$

where A is the set of triples (x,y,z) such that

$$\begin{aligned}(x,y) \in [\![p_1]\!]_r^{-1}(S([\![p_2]\!]_r^{-1}(W),z)) &\iff [\![p_1]\!]_r(x,y) \in S([\![p_2]\!]_r^{-1}(W),z) \\ &\iff [\![p_2]\!]_r([\![p_1]\!]_r(x,y),z) \in W. \quad (14.13)\end{aligned}$$

We also have

$$\mu \otimes \mu_R\left([\![p_1\texttt{;}\ \ p_2]\!]_r^{-1}(W)\right) \iint \chi_B(x,y)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu_R(y) \qquad (14.14)$$

Thus the equality $([\![p_1\texttt{;}\ \ p_2]\!]_p.\mu)(W) = (\mu \otimes \mu_R)([\![p_1\texttt{;}\ \ p_2]\!]_r^{-1}(W))$ is equivalent to $\iiint \chi_A(x,y,z)\ \mathrm{d}\mu(x)\ \mathrm{d}\mu_R(y)\ \mathrm{d}\mu_R(z) = \iint \chi_B(x,y)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu_R(y)$.

Let us split the variables y and z into pairs (y_1,y_2) and (z_1,z_2) , the first element of which contains the sequences pertaining to program points in p_1 and the second to program points in p_2 (or in other code fragments). **Since all random generators are independently distributed,** y_1 and z_1 will be distributed according to μ_{R_1} and y_2 and z_2 according to μ_{R_2} such that $\mu_R = \mu_{R_1} \otimes \mu_{R_2}$ up to variable naming. Let us also define C the set of triples (x,y_1,z_2) such that $[\![p_2]\!]_r([\![p_1]\!]_r(x,y_1),z_2) \in W$ (this is possible since **the**

semantics of p_1 only depend on y_1 and the semantics of p_2 only depend on z_2).

Then

$$\begin{aligned}
&\iiint \chi_A(x,y,z)\,\mathrm{d}\mu(x)\,\mathrm{d}\mu_R(y)\,\mathrm{d}\mu_R(z)\\
&= \iiint \chi_A(x,(y_1,y_2),(z_1,z_2))\,\mathrm{d}\mu(x)\,\mathrm{d}\mu_{R_1}(y_1)\,\mathrm{d}\mu_{R_2}(y_2)\,\mathrm{d}\mu_{R_1}(z_1)\,\mathrm{d}\mu_{R_2}(z_2)\\
&\quad = \iiint \chi_C(x,y_1,z_2)\,\mathrm{d}\mu_{R_1}(y_1)\,\mathrm{d}\mu_{R_2}(y_2)\,\mathrm{d}\mu_{R_1}(z_1)\,\mathrm{d}\mu_{R_2}(z_2)\\
&\quad\quad = \iiint \chi_C(x,y_1,z_2)\,\mathrm{d}\mu_{R_1}(y_1)\,\mathrm{d}\mu_{R_2}(z_2)\\
&\quad\quad = \iiint \chi_B(x,(y_1,y_2),(y_1,y_2))\,\mathrm{d}\mu_{R_1}(y_1)\,\mathrm{d}\mu_{R_2}(y_2) \qquad (14.15)
\end{aligned}$$

The equality is thus established.

Tests Let us apply the definition of $[\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p$:

$$\begin{aligned}
&([\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_p.\mu)(W)\\
&\quad = ([\![e_1]\!]_p \circ \phi_{[\![c]\!]}.\mu)(W) + ([\![e_2]\!]_p \circ \phi_{[\![c]\!]^C}.\mu)(W) \qquad (14.16)
\end{aligned}$$

We can apply the induction hypothesis, thus:

$$\begin{aligned}
&= (\phi_{[\![c]\!]}.\mu) \otimes \mu_R([\![e_1]\!]_r^{-1}(W)) + (\phi_{[\![c]\!]^C}.\mu) \otimes \mu_R([\![e_1]\!]_r^{-1}(W))\\
&\quad = \mu \otimes \mu_R([\![e_1]\!]_r^{-1}(W) \cap [\![c]\!]) + \mu \otimes \mu_R([\![e_2]\!]_r^{-1}(W) \cap [\![c]\!]^C)\\
&\quad\quad = \mu \otimes \mu_R([\![\texttt{if } c \texttt{ then } e_1 \texttt{ else } e_2]\!]_r^{-1}(W)) \qquad (14.17)
\end{aligned}$$

Loops Let us define $X_{n,m}$ as follows:

- $X_{n,n} = (W \cap [\![c]\!]^C) \times R$,
- for $0 \leq m < n$,

$$X_{n,m} = \left([(x,y) \mapsto (m.w_r[\![f]\!](x,y),y)]^{-1}(X_{n,m+1})\right) \cap ([\![c]\!] \times R).$$

Let us show by induction on $n-m$ that for all $n \geq 0$, $0 \leq m \leq n$, and all σ-finite measure μ:

$$\mu \otimes \mu_R(X_{n,m}) = \left(\phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^{n-m}.\mu\right)(W) :$$

$m = n$ $\mu \otimes \mu_R(X_{n,n}) = \mu \otimes \mu_R((W \cap [\![c]\!]^C) \times R) = \mu(W \cap [\![c]\!]^C) = (\phi_{[\![c]\!]^C}.\mu)(W)$

$0 \leq m < n$ $\mu \otimes \mu_R(X_{n,m}) = \iint \chi_{A_{n,m}}(x,y)\ \mathrm{d}\mu(x)\ \mathrm{d}\mu_R(y)$ where $(x,y) \in A_{n,m} \iff x \in [\![c]\!] \wedge (m.w_r[\![f]\!](x,y),y) \in X_{n,m+1}$.

y **can be split into two variables** y_1 and y_2 such that $(x,y) \in A_{n,m+1}$ only depends on x and y_1 and $m.w_r[\![f]\!](x,y)$ only depends on x and y_2. By an abuse of notation, we shall write $(x,(y_1,y_2)) \in A_{n,m} \iff x \in [\![c]\!] \wedge (m.w_r[\![f]\!](x,y_2),y_1) \in X_{n,m+1}$. Furthermore, y_1 **and** y_2 **are independently distributed**, according to μ_{R_1} and μ_{R_2} respectively. Therefore

$$\begin{aligned}\mu \otimes \mu_R(X_{n,m}) = \iiint \chi_{A_{n,m}}(x,(y_1,y_2))\ \mathrm{d}\mu(x)\ \mathrm{d}\mu_{R_1}(y_1)\ \mathrm{d}\mu_{R_2}(y_2) = \\ \iint X_{n,m+1}(u,y_1)\ \mathrm{d}([\![f]\!]_p \circ \phi_{[\![c]\!]}.\mu)(x)\ \mathrm{d}\mu_{R_1}(y_1) = \\ \iint X_{n,m+1}(u,y)\ \mathrm{d}([\![f]\!]_p \circ \phi_{[\![c]\!]}.\mu)(x)\ \mathrm{d}\mu_R(y) = \\ ([\![f]\!]_p.\mu) \otimes \mu_R(X_{n,m+1}). \quad (14.18)\end{aligned}$$

Let us apply the induction hypothesis:

$$\mu \otimes \mu_R(X_{n,m}) = \left(\phi_{[\![c]\!]^C} \circ ([\![f]\!]_p \circ \phi_{[\![c]\!]})^{n-(m+1)} \circ [\![f]\!]_p \circ \phi_{[\![c]\!]}.\mu\right)(W)$$

and we know have the property for rank m.

We can therefore apply this equality for the $X_{n,0}$'s:

.□ (14.19)

14.6 Variance reduction techniques

The problem of imprecision inherent in the statistical methods is the variance of the sampled result. We shall see here that it is possible to improve this result given some additionnal knowledge about the behavior of the system on different zones of the input domain [70, §4.1].

The variance V_0 of the system we test is equal to $p(p-1)$. The variance V of a n-sample average is V_0/n. Let us now consider the method, widely used for political and other social polls, where we partition the input domain into m zones

("socio-professional categories" in polling) that allegedly have different behavior with respect to the studied problem; we then run separate polls in each of the categories and take the weighted average of the result:

$$p = \sum_{k=1}^{m} p_k E_k \tag{14.20}$$

where p_k is the probability of zone k and E_k the probability of the test taken on that zone. We aim to approximate this result by taking N_k samples $(s_{k,i})_{1 \leq i \leq N_k}$ in zone k. The approximation is then

$$p = \sum_{k=1}^{m} \frac{p_k}{N_k} \sum_{i=1}^{N_k} s_{k,i} \tag{14.21}$$

and its variance is

$$V = \sum_{k=1}^{m} \frac{p_k^2}{N_k} V_k \tag{14.22}$$

where V_k is the variance of $s_{k,i}$.

An interesting question is how to minimize this variance, keeping $N = \sum_k N_k$ constant — how to get the best result with a fixed number of samples. To simplify the case, we shall analyze it as if the N_k were real numbers x_k. The problem then becomes that of finding a maximum of the function

$$V(x_1, \ldots, x_m) = \sum_{k=1}^{m} \frac{p_k^2}{x_k} V_k \tag{14.23}$$

on the hyperplane $x_1 + \cdots + x_k = N$. Such a local optimum only happens where the gradients of the two functions (V and $x_1 + \cdots + x_k$) are colinear, which means that there exists L such that for all k, $N_k = L.p_k.\sqrt{V_k}$.

In a real implementation, we do not aim for an optimal repartition into sampling zones; for soundness, any repartition will do to construct safety intervals. On the other hand, we want heuristics to minimize the width of the safety intervals. Given the above formula, it is a waste of time to allocate tests to zones of small variance; and as $V_k = p_k(1 - p_k)$ where p_k is the probability of the test on zone k, $V_k \sim p_k$ when p_k is small. But p_k itself can be estimated by sampling! The zones to consider can be generated for instance from the output of a backwards reachability analysis (see §8.3.1).

14.7 Discussion

We have proposed a generic method that combines the well-known techniques of abstract interpretation and Monte-Carlo program testing into an analysis scheme

for probabilistic and nondeterministic programs, including reactive programs whose inputs are modeled by both random and nondeterministic inputs. This method is mathematically proven correct, and uses no assumption apart from the distributions and nondeterminism domains supplied by the user. It yields upper bounds on the probability of outcomes of the program, according to the supplied random distributions, with worse-case behavior according to the nondeterminism; whether or not this bounds are sound is probabilistic, and a lower-bound of the soundness of those bounds is supplied. While our explanations are given using a simple imperative language as an example, the method is by no means restricted to imperative programming.

The number of trials, and thus the complexity of the computation, depends on the desired precision. The method is parallelizable with linear speed-ups. The complexity of the analysis, or at least its part dealing with probabilities, increases if the probability to be evaluated is low. However, static analysis can come to help to reduce this complexity.

his sampling technique has the strong advantage that it can be quite easily implemented onto an existing abstract interpreter. We implemented it into a small prototype interpreter, and we collaborated in getting it implemented into a more "real-life" system. Its sources of over-approximation are twofold:

- the confidence interval, inherent to all sampling techniques;
- the treatment of loops (stopping randomization);
- the underlying abstract interpreter.

We can see two possible disadvantages to this method:

- It overestimates the power of the adversary, since it assumes it knows all the future probabilistic choices. This is not of great consequence, since other analysis methods tend to do the same to a certain extent.
- It only randomizes a prefix of the computation and approximates long-term behavior. On the other hand, applying abstract interpretation and using a widening operator to force convergence amounts to the same.

A third objection was made by H. Wiklicky, which is that with this method, while the approximation introduced by the randomization can be controlled quite strictly, the approximation introduced by abstract interpretation is unknown. Indeed, the quality of a randomized abstract interpreter depends on the quality of the underlying abstract interpreter, and current abstract domains do not give an error amount. We hope that the current work on quantitative probabilistic abstract interpretation [26] will bear some fruit.

Chapter 15

Conclusion

The analysis of nondeterministic and probabilistic infinite state systems is a very challenging enterprise. Indeed, the first difficulty lies in providing a meaningful semantics of those systems, especially since there are different ways to view the same system with respect to the interleaving of probabilistic and nondeterministic choices.

We proposed two classes of methods. The first class of methods involves the representation of symbolic sets of measures (forward analysis) or of symbolic weight functions (backward analysis). As explained in the discussions of those methods, the backward analysis is generally to be preferred for reasons both of ease of implementation and of precision. While symbolic representations using linear combination of symbolic characteristic functions are difficult to handle, they provide "hard" upper bounds on the probabilities of properties. The domain of Gaussian functions has some obvious limitations, especially with respect to the tests, but provides a way to bound the influence of the extreme values of the inputs of the program.

The primary target of our analyses was to provide upper bounds on the probability of reaching certain states. We nevertheless allowed for more elaborate properties to be expressed, including fairness constraints, and extended the traditional value iteration of Markov decision processes so as to apply to those extended properties. As usual in program analysis, termination analysis is harder than reachability analysis, and the greatest-fixpoint approach is not necessarily very effective. On the other hand, probabilistic termination can be simpler than general termination: while general termination proofs involve finding a well-founded ordering and showing that iterations descend in that ordering, proving probabilistic termination is proving that a real-valued decreasing sequence converges to zero. We provided a domain aimed at proving probabilistic termination in the case of the iteration counts following a sub-exponential queue.

The other class of methods is the Monte-Carlo randomization of a conven-

tional abstract interpreter. This method is easy to implement given an existing abstract interpreter, and the inherent slowness of Monte-Carlo computations can be avoided by running preliminary analyses to narrow the scope of the sampling, as well as by massive parallelization. We expect it to be the most applicable method exposed in this thesis.

While the original goal was to provide analysis methods for trace properties of embedded systems, we hope that those methods can be adapted for other uses. For instance, floating-point systems have notoriously been difficult to analyze, and sutble analysis methods have been devised [67]; some of them use a probabilistic model of error [78]. We hope that a method based on probabilistic abstract interpretation may be proposed.

Troisième partie

Annexes

Appendix A

Mathematical technicalities

In this thesis, we make ample use of many mathematical concepts. As some of the definitions and theorems used are largely independent of the context in which they are used, and often are "classical" mathematical tools, I've chosen to group them in this appendix, sorted by broad categories.

Some of the lemmas are trivial and thus given without proof. Some of the definitions and theorems are well-known, and given here only for the sake of coherence of notation and self-containedness; they are given with pointers to textbooks.

A.1 Measure theory and integrals

One of the most essential results of Lebesgue's theory of integration is the following [71]:

Theorem A.1.1 (Lebesgue's monotone convergence theorem). *Let (X, μ) be a measured space. Let f_n be an ascending sequence of functions, whose point-wise limit is f. Then the sequence $\int f_n \mathrm{d}\mu$ is also ascending and $\lim_{n\to\infty} \int f_n \mathrm{d}\mu = \int f \mathrm{d}\mu$.*

A.2 Lattice Theory

A.2.1 Monotone operators and directed sets

Definition A.2.1. An ordered set $(L, \sqsubseteq)$ is call *directed* if for all x and y in L there exists z such that both $x \sqsubseteq z$ and $y \sqsubseteq z$.

Lemma A.2.2. *For any monotone operator $\phi : X \to Y$ and subset K of X, $\phi(\sqcup K) \sqsupseteq \bigsqcup_{f\in K} \phi(f)$.*

Proof. For any $f \in K$, $f \sqsubseteq \sqcup K$ and thus $\phi(f) \sqsubseteq \phi(\sqcup K)$. The result follows. □

Definition A.2.3. Let L_1 and L_2 be two ordered sets. We say that a monotonic function $f : L_1 \to L_2$ is

- ω-upper-continuous if for any ascending sequence $(x_n)_{n\in\mathbb{N}}$ of elements of L such that $\bigsqcup_{n=0}^{\infty} x_n$ exists, $f(\bigsqcup_{n=0}^{\infty} x_n) = \bigsqcup_{n=0}^{\infty} f(x_n)$;
- ω-lower-continuous if for any descending sequence $(x_n)_{n\in\mathbb{N}}$ of elements of L such that $\bigsqcap_{n=0}^{\infty} x_n$ exists, $f(\bigsqcap_{n=0}^{\infty} x_n) = \bigsqcap_{n=0}^{\infty} f(x_n)$;
- ω-continuous if it is both ω-upper-continuous and ω-lower-continuous.

Lemma A.2.4. *The least upper bound of a set of ω-upper-continous functions is ω-upper-continous.*

Definition A.2.5. Let L_1 and L_2 be two ordered sets. We say that a monotonic function $f : L_1 \to L_2$ is

- upper-continuous if for any totally ordered subset K of L_1, $f(\sqcup K) = \sqcup f(K)$;
- lower-continuous if for any totally ordered subset K of L_1, $f(\sqcap K) = \sqcap f(K)$;
- continuous if it is both upper-continuous and lower-continuous.

Lemma A.2.6. *Let Y be an ordered set. Let $\phi : (X \to I) \to Y$ be a monotonic, ω-upper-continuous function. Let K be a directed subset of $X \to I$. Then $\phi(\sqcup K) = \bigsqcup_{f\in K} \phi(f)$.*

Proof. From lemma A.2.2, $\phi(\sqcup K) \sqsupseteq \bigsqcup_{f\in K} \phi(f)$.

K is directed. From lemma 8.1.1, there exists an ascending sequence f_n such that $\sqcup_n f_n = F$. Since ϕ is ω-upper-continuous, $\bigsqcup_n \phi(f_n) = \phi(\sqcup K)$. But $\bigsqcup_n \phi(f_n) \sqsubseteq \bigsqcup_{f\in K} \phi(f)$ since the f_n belong to K. Therefore $\phi(\sqcup K) \sqsubseteq \bigsqcup_{f\in K} \phi(f)$. □

Definition A.2.7. Let F and Y be two ordered sets. Let $O \subseteq F \to Y$ be a set of monotone operators. O is said to be *optimizing* if for all $f \in F$, $\{\phi(f) \mid \phi \in O\}$ is directed.

Lemma A.2.8. *If $K \subseteq X \to I$ is directed and $O \subseteq (X \xrightarrow{mon} I) \xrightarrow{mon} (X \xrightarrow{mon} I)$ is optimizing, then $\{\phi(f) \mid \phi \in O \wedge f \in K\}$ is directed.*

Proof. Let ϕ_1 and ϕ_2 in O. Let f_1 and f_2 in K. K is directed, therefore there exists f_3 in K such that $f_3 \geq f_1$ and $f_3 \geq f_2$. O is optimizing, thus $\{\phi(f_3) \mid \phi \in O\}$ is directed, therefore there exists ϕ_3 in O such that $\phi_3(f_3) \geq \phi_1(f_3)$ and $\phi_3(f_3) \geq \phi_2(f_3)$. Since ϕ_1 and ϕ_2 are monotonic, $\phi_1(f_3) \geq \phi_1(f_1)$ and $\phi_2(f_3) \geq \phi_2(f_2)$. Then $\phi_3(f_3) \geq \phi_1(f_1)$ and $\phi_3(f_3) \geq \phi_2(f_2)$. □

A.2.2 Fixpoints

We are interested in fixpoints over monotone operators over a complete lattice.

Definition A.2.9. A *fixpoint* of a monotone operator $\phi : T \to T$ over a complete lattice is a point $x \in T$ such that $\phi(x) = x$.

Theorem A.2.10 (Tarski). *Let ϕ be a monotone operator over a complete lattice T. The set of fixpoints of ϕ has a least element, noted $\operatorname{lfp}\phi$, and a greatest element, noted $\operatorname{gfp}\phi$. Furthermore,*

$$\operatorname{lfp}\phi = \sqcap\{x \mid \phi(x) \sqsubseteq x\} \operatorname{gfp}\phi = \sqcup\{x \mid x \sqsubseteq \phi(x)\} \tag{A.1}$$

Lemma A.2.11. *Let ψ be an ω-upper-continuous operator over a complete lattice T. Then $\operatorname{lfp}\psi = \bigsqcup_n \psi^n(\bot)$.*

Lemma A.2.12. *Let ψ be an ω-upper-continuous operator over a complete lattice T. Then $\operatorname{lfp}\psi = \bigsqcup_n \psi^n(\bot)$.*

Lemma A.2.13. *Let T_1 and T_2 be two complete lattices. Let $\psi : T_1 \times T_2 \to T_1$ be an ω-upper-continuous operator. Then $y \mapsto \operatorname{lfp}(x \mapsto \psi(x,y))$ is an ω-upper-continuous operator.*

Proof. $y \mapsto \operatorname{lfp}(x \mapsto \psi(x,y))$ is ω-upper-continuous and thus $\operatorname{lfp}(x \mapsto \psi(x,y)) = \bigsqcup_n (y \mapsto \operatorname{lfp}(x \mapsto \psi(x,y)))^n(\bot)$. The result then follows from lemma A.2.4. □

We take the following from Cousot & Cousot [17, definition 2.1]:

Definition A.2.14 (Upper iteration sequence). Let L be a complete lattice, μ the smallest ordinal such that the class $\{\delta : \delta \in \mu\}$ has a cardinality greater than the cardinality $\operatorname{Card}(L)$ of L and $F : L \to L$ a monotone operator. The *μ-termed iteration sequence for F starting with $D \in L$* is the μ-termed sequence $(F^\delta(D))_{\delta\in\mu}$ of elements of L defined by transfinite recursion in the following way:

1. $F^0(D) = D$
2. $F^\delta(D) = F(F^{\delta-1}(D))$ for every successor ordinal $\delta \in \mu$
3. $F^\delta(D) = \bigsqcup_{\beta<\delta} F^\beta(D)$ for every limit ordinal $\delta \in \mu$.

Definition A.2.15 (Limit of a stationary transfinite sequence). We say that the sequence $(X^\delta)_{\delta\in\mu}$ is *stationary* if and only if $\exists\varepsilon \in \mu : \forall\beta \in \mu\ \beta \geq \varepsilon \implies X^\varepsilon = X^\beta$, in which case the *limit* of the sequence is X^ε. We denote by $\lim_\delta X^\delta$ this limit.

Such transfinite sequences enable giving a "constructive" version of Tarski's theorem, without any hypothesis of continuity of the operator whose fixpoint is sought [17, corollary 3.3]:

Theorem A.2.16. *A μ-termed upper iteration sequence $(F^\delta(D))_{\delta\in\mu}$ for F starting with D such that $F \sqsubseteq F(D)$ is a stationary increasing chain, its limit $\lim_\delta F^\delta(D)$ is the least fixpoint of F greater than or equal to D.*

A.2.3 Fixpoint transferts

Lemma A.2.17. *Let T_1 and T_2 be two complete lattices. Let $\alpha : T_1 \to T_2$ be an ω-upper-continuous operator such that $\alpha(\bot) = \bot$. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two ω-upper-continuous operators such that $\psi_2 \circ \alpha = \alpha \circ \psi_1$. Then $\alpha(\mathrm{lfp}\,\psi_1) = \mathrm{lfp}\,\psi_2$.*

Proof. $\alpha(\mathrm{lfp}\,\psi_1) = \alpha(\bigsqcup_n \psi_1^n(\bot)) = \bigsqcup_n \alpha\circ\psi_1^n(\bot) = \bigsqcup_n \psi_2^n(\underbrace{\alpha(\bot)}_{\bot}) = \mathrm{lfp}\,\psi_2$. □

We of course have, dually:

Lemma A.2.18. *Let T_1 and T_2 be two complete lattices. Let $\alpha : T_1 \to T_2$ be an ω-lower-continuous operator such that $\alpha(\top) = \top$. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two ω-lower-continuous operators such that $\psi_2 \circ \alpha = \alpha \circ \psi_1$. Then $\alpha(\mathrm{gfp}\,\psi_1) = \mathrm{gfp}\,\psi_2$.*

Lemma A.2.19. *Let T_1 and T_2 be two lattices. Let $\alpha : T_1 \to T_2$ be a monotone operator. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two monotone operators such that $\psi_2 \circ \alpha = \alpha \circ \psi_1$. If f is a fixpoint of ψ_1 then $\alpha(\psi_1)$ is a fixpoint of ψ_2.*

Proof. $\phi_1(x) = x$ thus $\underbrace{\alpha\circ\phi_1}_{\phi_2\circ\alpha}(x) = \alpha(x)$. □

Lemma A.2.20. *Let T_1 and T_2 be two lattices. Let $\alpha : T_1 \to T_2$ be a monotone, upper-continuous operator. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two monotone operators such that $\alpha \circ \psi_1(x) \sqsubseteq \psi_2 \circ \alpha(x)$ for all $x \in T_1$. Then for all $x \in T_1$, for all $\delta \in \mu$, $\alpha(\psi_1^\delta(x)) \sqsubseteq \psi_2^\delta(\alpha(x))$.*

Proof. Proof by transfinite induction on δ:

- $\delta = 0$, the inequation becomes $\alpha(x) \sqsubseteq \alpha(x)$;
- δ is a successor ordinal; then $\alpha \circ \psi_1^\delta(x) = \alpha \circ \psi_1(\psi_1^{\delta-1}(x)) \sqsubseteq \psi_2 \circ \alpha(\psi_1^{\delta-1}(x))$ by the hypotheses of the lemma; by the induction hypothesis, $\alpha(\psi_1^{\delta-1}(x)) \sqsubseteq \psi_2^{\delta-1}(\alpha(x))$; since ψ_2 is monotonic, $\psi_2 \circ \alpha(\psi_1^{\delta-1}(x)) \sqsubseteq \psi_2(\psi_2^{\delta-1}(\alpha(x))) = \psi_2^\delta(\alpha(x))$;

- δ is a limit ordinal; then $\psi_1^\delta(x) = \bigsqcup_{\beta<\delta} \psi_1^\beta(x)$ and $\psi_2^\delta(\alpha(x)) = \bigsqcup_{\beta<\delta} \psi_2^\beta(\alpha(x))$; $\alpha(\psi_1^\delta(x)) = \bigsqcup_{\beta<\delta} \alpha(\psi_1^\beta(x))$ since α is upper-continuous; by the induction hypothesis, for all $\beta < \delta$, $\alpha(\psi_1^\beta(x)) \sqsubseteq \psi_2^\beta(\alpha(x))$ and thus $\bigsqcup_{\beta<\delta} \alpha(\psi_1^\beta(x)) \sqsubseteq \bigsqcup_{\beta<\delta} \psi_2^\beta(\alpha(x))$; therefore $\alpha(\psi_1^\delta(x)) \sqsubseteq \psi_2^\delta(\alpha(x))$. □

□

Corollary A.2.21. *Let T_1 and T_2 be two lattices. Let $\alpha : T_1 \to T_2$ be a monotone upper-continuous operator such that $\alpha(\bot) = \bot$. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two monotone operators such that $\alpha \circ \psi_1(x) \sqsubseteq \psi_2 \circ \alpha(x)$ for all $x \in T_1$. Then $\alpha(\mathrm{lfp}\, \psi_1) \sqsubseteq \mathrm{lfp}\, \psi_2$.*

Proof. Applying theorem A.2.16, there exist ε_1 and ε_2 such that for all $\beta \geq \varepsilon_1$, $\psi_1^\beta(\bot) = \mathrm{lfp}\, \psi_1$ and for all $\beta \geq \varepsilon_2$, $\psi_2^\beta(\bot) = \mathrm{lfp}\, \psi_2$. Let $\varepsilon = \max(\varepsilon_1, \varepsilon_2)$. The lemma gives us $\alpha(\psi_1^\varepsilon(\bot)) \sqsubseteq \psi_2^\varepsilon(\underbrace{\alpha(\bot)}_{\bot})$, thus the result. □

Lemma A.2.22. *Let T_1 and T_2 be two lattices with a maximal element. Let $\alpha : T_1 \to T_2$ be a monotone operator such that $\alpha(\top) = \top$. Let $\psi_1 : T_1 \to T_1$ and $\psi_2 : T_2 \to T_2$ be two monotone operators such that $\alpha \circ \psi_1(x) \sqsubseteq \psi_2 \circ \alpha(x)$ for all $x \in T_1$. Then $\alpha(\mathrm{gfp}\, \psi_1) \sqsubseteq \mathrm{gfp}\, \psi_2$.*

Proof. Actually, the only property we shall use of $x_0 = \mathrm{gfp}\, \psi_1$ is that it is a fixpoint of ψ_1. $\alpha(x_0) = \alpha \circ \psi_1(x_0) \sqsubseteq \psi_2 \circ \alpha(x_0)$. Since $\mathrm{gfp}\, \psi_2 = \bigsqcup\{y \mid y \sqsubseteq \psi_2(y)\}$ (from Th. A.2.10), the results follows. □

Appendix B

Estimating the probability of a random event by the Monte-Carlo method

We consider a system whose outcome (success or failure) depends on the value of a parameter x, chosen in the set X according to a random distribution μ. The behavior of this system is described by a random variable $V : X \to \{0,1\}$, where 0 means success and 1 failure.

The *law of large numbers* says that if we independently choose inputs x_k, with distribution μ, and compute the experimental average $V^{(n)} = \frac{1}{n}\sum_{k=1}^{n} V(x_k)$, then $\lim_{n\to\infty} V^{(n)} = \mathbf{E}V$ with probability 1, where $\mathbf{E}V$ is the expectation of failure. Intuitively, it is possible to estimate accurately $\mathbf{E}V$ by effectively computing $V^{(n)}$ for a large enough value of n.

Just how far should we go? Unfortunately, a general feature of all Monte-Carlo methods is their slow asymptotic convergence speed. Indeed, the distribution of the experimental average $V^{(n)}$ is a binomial distribution centered around $\mathbf{E}V$. With large enough values of n (say $n \geq 20$), this binomial distribution behaves mostly like a normal (Gaussian) distribution (Fig. B.1) with means $p = \mathbf{E}V$ and standard deviate $\frac{p(1-p)}{\sqrt{n}}$. More generally, the central limit theorem (Th. 14.3.1) predicts that the average of n random variables identically distributed as has the same expectation $\mathbf{E}V$ as V and standard deviate $\frac{\sigma}{\sqrt{n}}$ where σ is the standard deviate of V. The standard deviate measures the error margin on the computed result: samples from a gaussian variable centered on x_0 and with standard deviate σ fall within $[x_0 - 2\sigma, x_0 + 2\sigma]$ about 95% of the time.

We can better evaluate the probability of underestimating the probability by

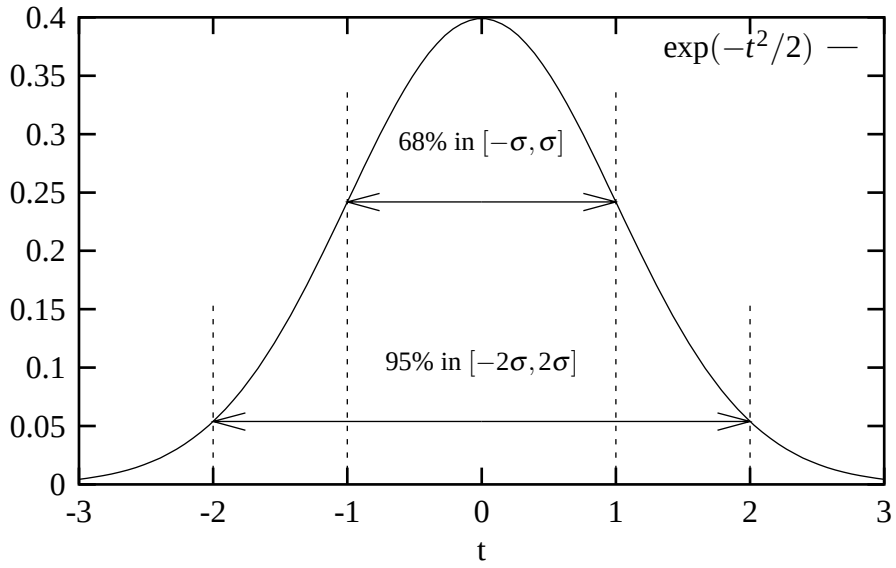


Figure B.1: The Gaussian normal distribution centered on 0, with standard deviate 1.

more than t using the Chernoff-Hoeffding [32] [75, inequality A.4.4] bounds:

$$\Pr\left(\mathbf{E}V \geq V^{(n)} + t\right) \leq e^{-2nt^2} \tag{B.1}$$

This bound, fully mathematically sound, means that the probability of underestimating V using $V^{(n)}$ by more than t is less than e^{-2nt^2}.

Any Monte-Carlo method has an inherent margin of error; this margin of error is probabilistic, in the sense that facts such as "the value we want to compute is in the interval $[a,b]$" are valid up to a certain probability. The size of the interval of safety for a given probability of error varies in $1/\sqrt{n}$.

Bibliographie

[1] Luca de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, Department of Computer Science, June 1998. CS-TR-98-1601.
`ftp://cstr.stanford.edu/pub/cstr/reports/cs/tr/98/1601`

[2] J.M. Arnaudiès and H. Fraysse. *Cours de mathématiques, 4 : Algèbre bilinéaire et géométrie*. Dunod Université, 1990.

[3] Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, and Marta Kwiatkowska. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming (ICALP '97)*, volume 1256 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[4] Christel Baier and Marta Z. Kwiatkowska. Domain equations for probabilistic processes. Technical Report CSR-97-7, University of Birmingham, School of Computer Science, July 1997.
`ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1997/CSR-97-07.ps.gz`

[5] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FST TCS 95 : Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.

[6] François Bourdoncle. *Sémantiques des Langages Impératifs d'Ordre Supérieur et Interprétation Abstraite*. PhD thesis, École Polytechnique, 1992.

[7] C.Baier, M.Kwiatkowska, and G.Norman. Computing probability bounds for linear time formulas over concurrent probabilistic systems. *Electronic Notes in Theoretical Computer Science*, 21, 1999.

[8] Jean-Marie Chesneaux and Jean Vignes. Sur la robustesse de la méthode CESTAC. *C. R. Acad. Sci. Paris Sér. I Math.*, 307(16) :855–860, 1988.

[9] Philippe G. Ciarlet. *Introduction à l'analyse numérique matricielle et à l'optimisation*. Masson, Paris, 1982.

[10] Edmund M. Clarke, Jr, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[11] Rance Cleaveland, Scott A. Smolka, and Amy E. Zwarico. Testing preorders for probabilistic processes. In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium*, volume 623 of *Lecture Notes in Computer Science*, pages 708–719, Vienna, Austria, 13–17 July 1992. Springer-Verlag.

[12] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In *Proc. ICALP'90*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer-Verlag, 1990.

[13] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Electronic Notes in Theoretical Computer Science*, 6, 1997.
`http://www.elsevier.nl/locate/entcs/volume6.html`

[14] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.

[15] P. Cousot and R. Cousot. Abstract intrepretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, January 1977.

[16] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 mars 1978.

[17] Patrick Cousot and Radhia Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.*, 82(1) :43–57, 1979.

[18] Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *J. Logic Prog.*, 2-3(13) :103–179, 1992.

[19] Patrick Cousot and Radhia Cousot. Introduction to abstract interpretation. Notes de cours du DEA, extraites de [18], 1998.
`http://www.dmi.ens.fr/~cousot/cours/DEASP2/CoursDEA_SP2_1998_JLP_92.ps.gz`

[20] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Conference on Principles of Programming Languages*. ACM Press, 1978.

[21] Didier Dacunha-Castelle. *Chemins de l'Aléatoire — Le hasard et le risque dans la société moderne*. Champs. Flammarion, 1999.

[22] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. In *STOC'01, 33rd Annual ACM Symposium on Theory of Computing*. Association for Computer Machinery, 2001.

[23] Luca de Alfaro, Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using MTBDDs and the kronecker representation. In *TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, January 2000.

[24] Alain Deutsch. Interprocedural may-alias analysis for pointers : Beyond *k*-limiting. In *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI)*, pages 230–241, Orlando, Florida, 20–24 June 1994. *SIGPLAN Notices* 29(6), June 1994.

[25] Alain Deutsch. Semantic models and abstract interpretation techniques for inductive data structures and pointers. In *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 226–229, La Jolla, California, June 21–23, 1995.

[26] Alessandra Di Pierro and Herbert Wiklicky. Concurrent constraint programming : Towards probabilistic abstract interpretation. In *2nd International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, 2000.

[27] J.L. Doob. *Measure Theory*, volume 143 of *Graduate Texts in Mathematics*. Springer-Verlag, 1994.

[28] Free Software Foundation. *GNU Octave : A high-level interactive language for numerical computations*.
`http://www.octave.org/doc/octave_toc.html`

[29] Philippe Granger. Improving the results of static analyses programs by local decreasing iteration. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India*, volume 652 of *Lecture Notes in Computer Science*, pages 68–79. Springer-Verlag, December 1992.

[30] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reability. Technical Report R90-13, Swedish Institute of Computer Science, December 1990.
`ftp://ftp.sics.se/pub/SICS-reports/Reports/`
`SICS-R--90-13--SE.ps.Z`

[31] Jifeng He, K. Seidel, and A. McIver. Probabilistic models for the guarded command language. *Science of Computer Programming*, 28(2–3) :171–192, April 1997. Formal specifications : foundations, methods, tools and applications (Konstancin, 1995).

[32] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58(301) :13–30, 1963.

[33] Evert Hoek. Practical rock engineering. Technical report, Evert Hoek Consulting Engineer Inc., 2000. Available from `http://www.rockscience.com/roc/Hoek/Hoeknotes2000.htm`.

[34] Michael Huth and Marta Kwiatkowska. On probabilistic model checking. Technical Report CSR-96-15, University of Birmingham, School of Computer Science, August 1996.
`ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSR-96-15.ps.gz`

[35] Claire Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, 1990.

[36] A. Jung and R. Tix. The troublesome probabilistic powerdomain. In A. Edalat, A. Jung, K. Keimel, and M. Kwiatkowska, editors, *Proceedings of the Third Workshop on Computation and Approximation*, volume 13 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1998. 23 pages.
`http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`

[37] Howard Karloff. *Linear Programming*. Birkhäuser-Verlag, Boston, 1991.

[38] Alexander S. Kechris. *Classical descriptive set theory*. Graduate Texts in Mathematics. Springer-Verlag, New York, 1995.

[39] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1969.

[40] Donald E. Knuth. *The Art of Computer Programming, volume 2 : Seminumerical Algorithms*. Addison-Wesley, 1969.

[41] D. Kozen. Semantics of probabilistic programs. In *20th Annual Symposium on Foundations of Computer Science*, pages 101–114, Long Beach, Ca., USA, October 1979. IEEE Computer Society Press.

[42] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3) :328–350, 1981.

[43] Marta Kwiatkowska and Gethin Norman. Metric denotational semantics for PEPA. Technical Report CSR-96-11, University of Birmingham, School of Computer Science, June 1996.
`ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSR-96-11.ps.gz`

[44] Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of concurrent probabilistic systems using mtbdds and simplex. Technical Report CSR-99-1, University of Birmingham, 1999.

[45] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. Technical Report CSR-00-6, University of Birmingham, School of Computer Science, March 2000. `ftp://ftp.cs.bham.ac.uk/pub/tech-reports/2000/CSR-00-06.ps.gz`

[46] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In C. Palamidessi, editor, *CONCUR 2000 - Concurrency Theory 11th International Conference*, number 1877 in Lecture Notes in Computer Science. Springer-Verlag, 2000.

[47] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1) :1–28, September 1991.

[48] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Efficient dynamic-programming updates in partially observable markov decision processes. Technical Report CS-95-19, Brown University, 1995.

[49] Gavin Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report TR-11-93, Oxford University, 1993.

[50] A. McIver. Reasoning about efficiency within a probabilistic μ-calculus. In *Proc. of PROBMIV*, pages 45–58, 1998. Technical Report CSR-98-4, University of Birmingham, School of Computer Science.

[51] A. K. McIver and Carroll Morgan. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science*, 266(1–2), September 2001.

[52] Antoine Miné. A new numerical abstract domain based on difference-bound matrices. In Olivier Danvy and Andrzej Filinski, editors, *Programs as Data Objects, PADO 2001*, volume 2053 of *Lecture Notes in Computer Science*, pages 155–172. Springer-Verlag, 2001.

[53] John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

[54] David Monniaux. Abstract interpretation of probabilistic semantics. In *Seventh International Static Analysis Symposium (SAS'00)*, number 1824 in Lecture Notes in Computer Science. Springer-Verlag, 2000. Extended version on the author's web site.

[55] David Monniaux. An abstract analysis of the probabilistic termination of programs. In *8th International Static Analysis Symposium (SAS'01)*, number 2126 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[56] David Monniaux. An abstract Monte-Carlo method for the analysis of probabilistic programs (extended abstract). In *28th Symposium on Principles of*

Programming Languages (POPL '01), pages 93–101. Association for Computer Machinery, 2001.

[57] David Monniaux. Backwards abstract interpretation of probabilistic programs. In *European Symposium on Programming Languages and Systems (ESOP '01)*, number 2028 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[58] Carroll Morgan, Annabelle McIver, Karen Seidel, and J. W. Sanders. Probabilistic predicate transformers. Technical Report TR-4-95, Oxford University, February 1995.

[59] Jacques Neveu. *Mathematical Foundations of the Calculus of Probabilities*. Holden-Day, 1965.

[60] Jacques Neveu. *Bases mathématiques du calcul des probabilités*. Masson et Cie, Éditeurs, Paris, 1970. Préface de R. Fortet. Deuxième édition, revue et corrigée.

[61] Simeon Ntafos. On random and partition testing. In Michal Young, editor, *ISSTA 98 : Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 42–48, 1998.
`http://www.acm.org/pubs/articles/proceedings/issta/271771/p42-ntafos/p42-ntafos.pdf`

[62] Panel on Statistical Methods in Software Engineering. *Statistical Software Engineering*. National Academy of Sciences, 1996.
`http://bob.nap.edu/readingroom/books/statsoft/contents.html`

[63] The Open Group. *Single Unix Specification, Version 2*.
`http://www.opengroup.org/onlinepubs/007908799/`

[64] Martin L. Puterman. *Markov decision processes : discrete stochastic dynamic programming*. Wiley series in probability and mathematical statistics. John Wiley & Sons, 1994.

[65] G. Ramalingam. Data flow frequency analysis. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation*, pages 267–277, Philadelphia, Pennsylvania, 21–24 May 1996.

[66] Norm Rasmussen. Reactor safety study — an assessment of accident risks in u. s. commercial nuclear power plants. Technical Report (NUREG-75/014) WASH-1400, U.S. Nuclear Regulatory Commission, Washington D. C. : Government Printing Office, 1975.

[67] Éric Goubault. Static analyses of floating-point operations. In *Static Analysis (SAS '01)*, Lecture Notes in Computer Science. Springer-Verlag, July 2001.

[68] Hartley Rogers, Jr. *Theory of recursive functions and effective computability*. McGraw-Hill, 1967.

[69] Rote. Path problems in graphs. In G. Tinhofer, E. Mayr, H. Noltemeier, and M. M. Syslo, editors, *Computational Graphs Theory*, number 7 in Computing supplement. Springer, 1990.

[70] Reuven Y. Rubinstein. *Simulation and the Monte-Carlo Method*. Wiley series in probabilities and statistics. John Wiley & Sons, 1981.

[71] Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.

[72] Bruce Schneier. *Applied Cryptography*. Wiley, second edition, 1996.

[73] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley Publishing Company, 1996.

[74] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995. Technical report MIT/LCS/TR-676.
`http://cs.unibo.it/~segala/www/pub/phd.tar.gz`

[75] Galen R. Shorack and Jon A. Wellner. *Empirical Processes with Applications to Statistics*. Wiley series in probability and mathematical statistics. John Wiley & Sons, 1986.

[76] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, pages 135–191. Elsevier, 1990.

[77] P. Thévenod-Fosse and H. Waeselynck. Statemate applied to statistical software testing pages 99-109. In *Proceedings of the 1993 international symposium on Software testing and analysis*, pages 99–109. Association for Computer Machinery, June 1993.

[78] J. Vignes. A stochastic arithmetic for reliable scientific computation. *Math. Comput. Simulation*, 35(3) :233–261, 1993.

[79] J. Vignes and R. Alt. An efficient stochastic method for round-off error analysis. In *Accurate scientific computations (Bad Neuenahr, 1985)*, pages 183–205. Springer, Berlin, 1986.

[80] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, Cambridge, Massachusetts, 1993.

Index

Table des figures

L'étude de programmes probabilistes intéresse plusieurs domaines de l'informatique : les réseaux, l'embarqué, ou encore la compilation optimisée. C'est tâche malaisée, en raison de l'indécidabilité des propriétés sur les programmes déterministes à états infinis, en plus des difficultés provenant des aspects probabilistes.

Dans cette thèse, nous proposons un langage de formules permettant de spécifier des propriétés de traces de systèmes de transition probabilistes et non-déterministes, englobant celles spécifiables par des automates de Büchi déterministes. Ces propriétés sont en général indécidables sur des processus infinis.

Ce langage a à la fois une sémantique concrète en termes d'ensembles de traces et une sémantique abstraite en termes de fonctions mesurables. Nous appliquons ensuite des techniques d'interprétation abstraite pour calculer un majorant de la probabilité dans le pire cas de la propriété étudiée et donnons une amélioration de cette technique lorsque l'espace d'états est partitionné, par exemple selon les points de programme. Nous proposons deux domaines abstraits convenant pour cette analyse, l'un paramétré par un domaine abstrait non probabiliste, l'autre modélisant les gaussiennes étendues.

Il est également possible d'obtenir de tels majorants par des calculs propageant les mesures de probabilité en avant. Nous donnons une méthode d'interprétation abstraite pour analyser une classe de formules de cette façon et proposons deux domaines abstraits adaptés à ce type d'analyse, l'un paramétré par un domaine abstrait non probabiliste, l'autre modélisant les queues sous-exponentielles. Ce dernier permet de prouver la terminaison probabiliste de programmes.

Les méthodes décrites ci-dessus sont symboliques et ne tirent pas parti des propriétés statistiques des probabilités. Nous proposons d'autre part une méthode de Monte-Carlo abstrait, utilisant des interpréteurs abstraits randomisés.

Mots clés : probabilités, non-déterminisme, interprétation abstraite, méthode de Monte-Carlo, processus décisionnels de Markov

The study of probabilistic programs is of considerable interest for the validation of networking protocols, embedded systems, or simply for compiling optimizations. It is also a difficult matter, due to the undecidability of properties on infinite-state deterministic programs, as well as the difficulties arising from probabilistic aspects.

In this thesis, we propose a formulaic language for the specification of trace properties of probabilistic, nondeterministic transition systems, encompassing those that can be specified using deterministic Büchi automata. Those properties are in general undecidable on infinite processes.

This language has both a concrete semantics in terms of sets of traces, as well as an abstract semantics in terms of measurable functions. We then apply abstract interpretation-based techniques to give upper bounds on the worst-case probability of the studied property. We propose an enhancement of this technique when the state space is partitioned — for instance along the program points —, allowing the use of faster iteration methods. We propose two abstract domains suitable for this analysis, one parameterized by an abstract domain suitable for nondeterministic (but not probabilistic) abstract interpretation, one modeling extended normal distributions.

An alternative method to get such upper bounds works is to apply forward abstract interpretation on measures. We propose two abstract domains suitable for this analysis, one parameterized by an abstract domain suitable for nondeterministic abstract interpretation, one modeling subexponential queues. This latter domain allows proving probabilistic termination of programs.

The methods described so far are symbolic and do not make use of the statistical properties of probabilities. On the other hand, a well-known way to obtain informations on probabilistic distributions is the Monte-Carlo method. We propose an abstract Monte-Carlo method featuring randomized abstract interpreters.

Keywords: probability, non-determinism, abstract interpretation, Monte-Carlo method, Markov decision processes