

Metodi simbolici per migliorare la precisione di domini astratti numerici

Dino Puller

12 luglio 2006

Sommario

Viene presentata la linearizzazione: ¹ una tecnica per la semplificazione di espressioni, e la propagazione di costanti simboliche per migliorare la precisione di analizzatori statici basati sull'Interpretazione Astratta.

1 Introduzione

Nella teoria dell'interpretazione astratta[3] giocano un ruolo fondamentale i domini astratti. Alcuni di essi sono in grado di dedurre relazioni lineari altri la non correlazione tra variabili di un algoritmo. Va da sé che più sono precisi più l'invariante che possono fornire sarà utile al fine di verificare il corretto funzionamento di un algoritmo. Come si può facilmente intuire, realizzare domini astratti efficaci costa, non solo in termini di complessità matematica del modello stesso, ma pure in fatto di tempo d'esecuzione che un analizzatore impiega per verificare un programma. Bisogna quindi far attenzione a quale gruppo algebrico affidarci, usare ad esempio uno spazio affine, che è in grado di dedurre relazioni lineari, per una espressione quadratica ci fornirà soltanto un allarme.

Le tecniche qui presentate non vogliono definire nuovi domini numerici indipendenti, andranno invece ad affiancarsi a quelli normalmente usati. Vogliono essere uno strumento utile a migliorare la precisione di analizzatori statici basati sull'interpretazione astratta, che facciano uso di domini numerici semplici ma computazionalmente meno onerosi.

2 Il linguaggio

Per prima cosa definiamo il linguaggio che useremo in questa trattazione.

Definizione 2.1. *La grammatica:*

$$\begin{aligned} \text{ASSIGNS} & ::= \text{ASSIGNS ASSIGN} / \text{ASSIGN} / \text{ASSIGNINT} \\ \text{ASSIGN} & ::= \text{VAR} \leftarrow \text{EXP} \\ \text{ASSIGNINT} & ::= \text{VAR} \leftarrow [\text{NUM}, \text{NUM}] \\ \text{EXP} & ::= \text{EXP} + \text{EXP} / \text{EXP} - \text{EXP} / \text{EXP} \times \text{EXP} / \text{EXP} / \text{EXP} \\ & / \text{VAR} \\ & / \text{NUM} \\ \text{VAR} & ::= v \in \mathcal{V} \\ \text{NUM} & ::= n \in \mathbb{D} \text{ con } \mathbb{D} \in \{\mathbb{R}, \mathbb{Z}, \mathbb{Q}\} \cup \{-\infty, +\infty\} \end{aligned}$$

¹L'articolo originale è di Antoine Miné [7]

L'insieme degli stati Σ contiene tutte le funzioni $\sigma : Loc \rightarrow NUM$ dalle locazioni a valori numerici. Usualmente all'interno del dominio sono presenti anche il valori \top , così da soddisfare una condizione di una famiglia di Moore (vedi la definizione ??), che identifica tutti i valori del dominio, e \perp nessuno. Potremmo fare a meno di \perp , dato che si usano catene ascendenti e non quelle discendenti per cercare il punto fisso, ma risulta comodo per identificare quando una variabile non è stata inizializzata.

La semantica denotazione del linguaggio è data per induzione strutturale:

Definizione 2.2. *La semantica:*

$$\begin{aligned}
\mathcal{A}[\top]\sigma &= \top \\
\mathcal{A}[X]\sigma &= \sigma(X) \\
\mathcal{A}[[a, b]]\sigma &= \{x \mid x \in \mathbb{D} \ a \leq x \leq b \ a, b \in \mathbb{Z}\} \\
\mathcal{A}[e_1 \diamond e_2]\sigma &= \{x \diamond y \mid x \in \mathcal{A}[e_1]\sigma \ y \in \mathcal{A}[e_2]\sigma\} \quad \diamond \in \{+, -, \times\} \\
\mathcal{A}[e_1/e_2]\sigma &= \{x/y \mid x \in \mathcal{A}[e_1]\sigma \ y \in \mathcal{A}[e_2]\sigma\} \quad \text{se } \mathbb{D} \neq \mathbb{Z} \\
\mathcal{C}[e_1/e_2]\sigma &= \{\lfloor x/y \rfloor \mid x \in \mathcal{A}[e_1]\sigma, y \in \mathcal{A}[e_2]\sigma\} \quad \text{se } \mathbb{D} \neq \mathbb{Z} \\
\mathcal{C}[X \leftarrow e]\sigma &= \{\sigma[X \mapsto v] \mid v \in \mathcal{A}[e]\sigma\} \\
\mathcal{C}[X \leftarrow [a, b]]\sigma &= \{\sigma[X \mapsto [a, b]] \mid a \leq x \leq b \ a, b \in \mathbb{Z}\}
\end{aligned}$$

La semantica per \top ci servirà solo quando parleremo della propagazioni di costanti simboliche.

2.1 Ordine parziale sulle espressioni

Per definire la correttezza sull'approssimazione di espressioni, prima definiamo l'ordine parziale \sqsubseteq su di esse, e quindi:

$$e_1 \sqsubseteq e_2 \Leftrightarrow \forall \sigma \in (\mathcal{V} \rightarrow \mathbb{D}, \mathcal{A}[e_1]\sigma \subseteq \mathcal{A}[e_2]\sigma$$

Per noi è sufficiente che \sqsubseteq sia definito in un insieme di ambienti $R \in \mathcal{P}(\mathcal{V} \rightarrow \mathbb{D})$ in modo da risultare meno restrittivi:

Definizione 2.3. $e_1 \sqsubseteq_R e_2 \Leftrightarrow \forall \sigma \in R, \mathcal{A}[e_1]\sigma \subseteq \mathcal{A}[e_2]\sigma$

Se $R \subseteq R'$ e $e_1 \sqsubseteq_{R'} e_2 \Rightarrow e_1 \sqsubseteq_R e_2$; con $=_R$ denotiamo la relazione di equivalenza. Con $\sigma : \mathcal{V} \rightarrow \mathbb{D}$ indichiamo lo stato della memoria valutata in un dominio concreto, invece con $\sigma^\# : \mathcal{V} \rightarrow \mathbb{D}^\#$ intendiamo una analoga funzione valutata in un dominio astratto. Ora possiamo passare dall'astratto al concreto e viceversa, tramite l'ausilio degli operatori di Galois, otteniamo così le relazioni: $\sigma^\# = \alpha \circ \sigma$ e $\sigma = \gamma \circ \sigma^\#$.

Teorema 2.1. *Se $e_1 \sqsubseteq_\sigma e_2$ allora il comando $\mathcal{C}[V \leftarrow e_1]\sigma^\#$ può essere sostituito da $\mathcal{C}[V \leftarrow e_2]\sigma^\#$.*

Dimostrazione. Per la definizione di relazione d'ordine parziale: $\mathcal{A}[e_1]\sigma \subseteq \mathcal{A}[e_2]\sigma$ applicando α e per la sua monotonia: $\mathcal{A}[e_1]\sigma^\# \subseteq \mathcal{A}[e_2]\sigma^\#$, ne segue che $\mathcal{C}[V \leftarrow e_2]\sigma^\#$ è una sovra approssimazione del comando $\mathcal{C}[V \leftarrow e_1]\sigma^\#$ valutato nel dominio astratto. \square

Sebbene l'ordine delle espressioni venga valutato nel concreto, possiamo operare delle sostituzioni di comandi con relativi equivalenti, che facciano uso di espressioni valutate in un dominio astratto.

3 Domini Astratti

La linearizzazione delle espressioni fa uso del dominio degli intervalli che è forse uno dei più semplici. Esso sarà la base sulla quale poggia l'aritmetica affine[2] che servirà ad ottenere semplificazioni nelle espressioni.

3.1 Gli intervalli

Definizione 3.1. Definiamo un intervallo come una coppia di valori racchiusi da parentesi quadrate: $[a, b]$ con la relazione $a \leq b$ e $a, b \in \mathbb{Z}$.

Le operazioni sono definite in modo classico ed intuitivo:

$$\begin{aligned} (\perp \diamond x) \vee (x \diamond \perp) &= \perp & \diamond &= \{+, -, \times, /\} \\ [a, b] + [a', b'] &= [a + a', b + b'] \\ [a, b] - [a', b'] &= [a - b', b - a'] \\ [a, b] \times [a', b'] &= [\min\{aa', ab', ba', bb'\}, \max\{aa', ab', ba', bb'\}] \\ [a, b]/[a', b'] &= \begin{cases} \perp & \text{se } 0 \in \{a', b'\} \\ \llbracket \min\{a/a', a/b', b/a', b/b'\} \rrbracket, \llbracket \max\{a/a', a/b', b/a', b/b'\} \rrbracket & \text{altrimenti} \end{cases} \end{aligned}$$

L'operazione di divisione nasconde alcune insidie, infatti se il divisore contiene lo zero, l'analizzatore non segnalerà alcun allarme fintantoché lo zero non sia in uno degli estremi dell'intervallo. Ciò vuol dire che non siamo protetti da una eventuale divisione per zero a tempo d'esecuzione. Tale comportamento è stato preferito a quello più restrittivo: \perp se $0 \in [a', b']$ perché in questo caso, la divisione rischia di annullare l'intera analisi di una espressione.

C'è da fare una piccola nota seppur intuitiva. Si deve tener in considerazione che un computer non è in grado di maneggiare domini matematici ma solo delle loro approssimazioni. Nel nostro caso, dovremo far attenzione ad esempio, che i valori di a e b non vadano in overflow o in underflow, con dei relativi controlli. Purtroppo anche se stiamo realizzando un analizzatore statico del codice, non lo possiamo usare su se stesso, non possiamo quindi usufruire dei suoi allarmi per riscontrare problemi di questo genere.

3.2 La forma Affine

Definizione 3.2. Una forma affine è una combinazione lineare di intervalli per variabili e ha la forma: $i_0 + \sum_k i_k \times V_k$. Se definiamo con \mathbb{I} il dominio degli intervalli e con \mathbb{A} la forma affine:

$$\begin{aligned} + : \mathbb{A} \times \mathbb{A} &\rightarrow \mathbb{A} \\ - : \mathbb{A} \times \mathbb{A} &\rightarrow \mathbb{A} \\ \times : \mathbb{I} \times \mathbb{A} &\rightarrow \mathbb{A} \\ / : \mathbb{A} \times \mathbb{I} &\rightarrow \mathbb{A} \end{aligned}$$

Le operazioni sono definite di seguito:

$$\begin{aligned} (i_0 + \sum_k i_k \times V_k) + (i'_0 + \sum_k i'_k \times V_k) &= ((i_0 + i'_0) + \sum_k (i_k + i'_k) \times V_k) \\ (i_0 + \sum_k i_k \times V_k) - (i'_0 + \sum_k i'_k \times V_k) &= ((i_0 - i'_0) + \sum_k (i_k - i'_k) \times V_k) \\ i \times (i_0 + \sum_k i_k \times V_k) &= ((i \times i_0) + \sum_k (i \times i_k) \times V_k) \\ (i_0 + \sum_k i_k \times V_k)/i &= ((i_0/i) + \sum_k (i_k/i) \times V_k) \end{aligned}$$

La loro correttezza è assicurata dalle classiche operazioni algebriche, dalla associatività e distributività, e dalla correttezza delle relative operazioni definite per gli intervalli, considerando sempre l'algebra classica. Infatti usando gli interi macchina, usati da un qualunque computer, è facile mostrare che $a + b \leq 0$ anche se entrambi gli addendi sono maggiori di zero, basta che $a \leftarrow \text{max_int}$ e $b \leftarrow 1$.

Questa particolare forma matematica tiene traccia delle operazioni che vengono svolte da una o più espressioni, eccezion fatta per le operazioni di moltiplicazione e divisione. A titolo di esempio se abbiamo: $a + b - b + c$ questa tecnica riesce ad accorgersi delle due operazioni opposte riuscendo ad eliminarle ottenendo: $a + c$.

Definiamo due operatori: $\pi : \mathcal{V} \rightarrow \mathbb{I}$ e $\iota : \mathbb{A} \times \Sigma^\# \rightarrow \mathbb{I}$. Il primo ci serve per proiettare una variabile in un intervallo che ne racchiude il suo valore, il secondo invece per intervallizzare una forma affine, ottenendone cioè, dato un ambiente in cui siano definite le variabili, un intervallo. L'ambiente è stato trascritto con $\Sigma^\#$ e non con Σ per indicare che l'insieme degli stati contiene funzioni $\sigma : Loc \rightarrow \mathbb{D}^\#$ e che quindi in memoria abbiamo domini astratti $\mathbb{D}^\#$. Come si può intuire l'intervallo gioca un ruolo fondamentale in questa trattazione ed è proprio l'operatore ι a fare in modo di "aggiustare" gli operandi nella moltiplicazione e divisione, che sono di diverso tipo.

Definizione 3.3 (Per gli intervalli). *L'operatore di proiezione $\pi : \mathcal{V} \rightarrow \mathbb{I}$*

$$\pi(x) = \sigma^\#(x)$$

Banalmente la variabile x deve essere presente in memoria.

Definizione 3.4 (Per gli intervalli). $\iota(i_0 + \sum_k i_k \times V_k)\sigma = i_0 + \sum_k i_k \times \pi(V_k)$ con $\sigma \in \Sigma^\#$.

Teorema 3.1. *La linearizzazione ι è corretta: $\forall exp \forall \sigma exp \sqsubseteq_\sigma \iota(exp)\sigma$.*

Dimostrazione. Questa è una conseguenza della correttezza delle operazioni $+$ e \times fatte su intervalli e della correttezza di π . □

L'assegnazione Le espressioni sono valutate grazie a π , nel dominio degli intervalli (tutte le variabili sono definite in \mathbb{I}) sebbene facciamo uso di un dominio astratto $D^\#$. C'è quindi bisogno dell'operazione inversa, e tale funzione viene svolta dall'assegnamento, che oltre quindi a valutare una espressione, dovrà tradurla in un elemento in $D^\#$.

Per il dominio astratto \mathbb{I} faremo uso di ι .

4 La linearizzazione

L'idea che sta alla base della tecnica della linearizzazione è la seguente:

Esempio 4.1. *Prendiamo i seguenti comandi ed il dominio astratto \mathbb{I} :*

$$\begin{aligned} X &\leftarrow [-10, 10] \\ Y &\leftarrow X - 2 \times X \\ Y &\leftarrow [-10, 10] - [-20, 20] \end{aligned}$$

Otteniamo $\sigma^\# = \{Y \mapsto [-30, 30]\}$. Ma è evidente che il comando può essere riscritto come: $Y \leftarrow -X$, ed in questo caso: $Y \in [-10, 10]$ ottenendo un risultato migliore.

La linearizzazione cerca di riscrivere le espressioni, lavorando solamente sulla loro sintassi, in modo di ricavarne delle nuove più semplici. Ovviamente questo esempio è banale ma quando abbiamo a che fare con software che richiede elaborazioni complesse può essere efficace fare queste semplificazioni in maniera automatica.

La linearizzazione di una espressione e in un ambiente astratto $\sigma^\#$: $(\langle e \rangle)\sigma^\#$ è definita tramite induzione strutturale come segue:

Definizione 4.1.

$$\begin{aligned} (\langle a \rangle)\sigma^\# &= [a, a] \\ (\langle V \rangle)\sigma^\# &= [1, 1] \times V \\ (\langle e_1 + e_2 \rangle)\sigma^\# &= (\langle e_1 \rangle)\sigma^\# + (\langle e_2 \rangle)\sigma^\# \\ (\langle e_1 - e_2 \rangle)\sigma^\# &= (\langle e_1 \rangle)\sigma^\# - (\langle e_2 \rangle)\sigma^\# \\ (\langle e_1 \times e_2 \rangle)\sigma^\# &= \iota((\langle e_1 \rangle)\sigma^\#)\sigma^\# \times (\langle e_2 \rangle)\sigma^\# \\ (\langle e_1 / e_2 \rangle)\sigma^\# &= (\langle e_1 \rangle)\sigma^\# / \iota((\langle e_2 \rangle)\sigma^\#)\sigma^\# \end{aligned}$$

La prima regola è l'unica che traduce una espressione in un intervallo anziché un affine. Ciò nonostante non è un errore, infatti la forma affine è proprio definita a partire da un intervallo base i_0 che appunto questa regola fornisce.

Teorema 4.1. *La linearizzazione di espressioni è corretta: $\forall exp \ exp \sqsubseteq_{\sigma} (\!|exp|\!)_{\sigma^{\sharp}}$. Ipotizzando che tutte le variabili dell'espressione exp siano definite in σ^{\sharp} .*

Dimostrazione. La dimostrazione va fatta induttivamente su exp :

- $exp \equiv a$: $a =_{\sigma} [a, a]$.
- $exp \equiv V$: $V =_{\sigma} [1, 1] \times V$.
- $exp \equiv e_1 + e_2$: Per ipotesi induttiva $e_1 \sqsubseteq_{\sigma} (\!|e_1|\!)_{\sigma^{\sharp}}$ e $e_2 \sqsubseteq_{\sigma} (\!|e_2|\!)_{\sigma^{\sharp}}$, per la monotonia di $+$, $(e_1 + e_2) \sqsubseteq_{\sigma} (\!|e_1|\!)_{\sigma^{\sharp}} + (\!|e_2|\!)_{\sigma^{\sharp}}$ e per la correttezza di $+$ $(\!|e_1|\!)_{\sigma^{\sharp}} + (\!|e_2|\!)_{\sigma^{\sharp}} = (\!|e_1 + e_2|\!)_{\sigma^{\sharp}}$.
- $exp \equiv e_1 - e_2$: Analoga alla precedente.
- $exp \equiv e_1 \times e_2$: Per ipotesi induttiva $e_1 \sqsubseteq_{\sigma} (\!|e_1|\!)_{\sigma^{\sharp}}$ e $e_2 \sqsubseteq_{\sigma} (\!|e_2|\!)_{\sigma^{\sharp}}$, per la monotonia di \times e la correttezza di ι : $(e_1 \times e_2) \sqsubseteq_{\sigma} \iota((\!|e_1|\!)_{\sigma^{\sharp}})_{\sigma^{\sharp}} \times (\!|e_2|\!)_{\sigma^{\sharp}}$ e per la correttezza di \times : $\iota((\!|e_1|\!)_{\sigma^{\sharp}})_{\sigma^{\sharp}} \times (\!|e_2|\!)_{\sigma^{\sharp}} \sqsubseteq_{\sigma} (\!|e_1 \times e_2|\!)_{\sigma^{\sharp}}$
- $exp \equiv e_1/e_2$: Analoga alla precedente.

□

Quando effettuiamo una linearizzazione possiamo solo ottenere risultati approssimati e in generale non c'è un intervallo lineare migliore che astrae una espressione.

Ora possiamo applicare la linearizzazione al comando $\mathcal{C}[V \leftarrow exp]$

Definizione 4.2 (Per gli intervalli). *Data una espressione exp qualunque:*

$$\mathcal{C}[V \leftarrow exp]_{\sigma^{\sharp}} = \mathcal{C}[V \leftarrow \iota((\!|exp|\!)_{\sigma^{\sharp}})_{\sigma^{\sharp}}]_{\sigma^{\sharp}}$$

L'uso di ι ci permette di tradurre l'affine, ottenuto dalla valutazione dell'espressione, in un intervallo. Infatti l'ambiente σ^{\sharp} è composto da soli intervalli, purtroppo effettuando questa operazione perdiamo informazioni sull'espressione appena calcolata.

Sebbene la linearizzazione è in grado di effettuare delle semplificazioni nelle espressioni e quindi è in grado di ottenere una migliore precisione nel valutarle, purtroppo non c'è garanzia che l'intervallo ottenuto sia migliore della valutazione della stessa espressione senza l'applicazione della linearizzazione, e questo a causa dell'operatore ι che fa perdere traccia delle operazioni svolte, condensandole in un unico intervallo risultante.

Esempio 4.2. *Facciamo ora un un esempio completo dell'uso della linearizzazione:*

Sia dato il seguente programma ed il dominio astratto \mathbb{I} :

$$\begin{aligned} X &\leftarrow [-10, 10] \\ Y &\leftarrow X - 2 \times X \end{aligned}$$

Applichiamo la linearizzazione al comando di assegnamento con stato

$$\sigma^{\sharp} = \{X \mapsto [-10, 10]\}:$$

$$\mathcal{C}[Y \leftarrow X - 2 \times X]_{\sigma^{\sharp}} = \mathcal{C}[Y \leftarrow \iota((\!|X - 2 \times X|\!)_{\sigma^{\sharp}})_{\sigma^{\sharp}}]_{\sigma^{\sharp}}$$

Esaminiamo ogni passaggio effettuato sull'espressione:

$$\begin{aligned}
 (X - 2 \times X)\sigma^\# &= ((X)\sigma^\# - (2 \times X)\sigma^\#)\sigma^\# \\
 &= ([0, 0] + [1, 1] \times X - (2))\sigma^\# \times (X)\sigma^\#\sigma^\# \\
 &= ([0, 0] + [1, 1] \times X - [2, 2] \times [0, 0] + [1, 1] \times X)\sigma^\# \\
 &= ([0, 0] + [1, 1] \times X - [0, 0] + [-2, -2] \times X)\sigma^\# \\
 &= [0, 0] + [-1, -1] \times X
 \end{aligned}$$

Ottenendo infine applicando $\iota(\cdot)\sigma^\#$:

$$\sigma^\# = \{X \mapsto [-10, 10], Y \mapsto [-10, 10]\}$$

4.1 Applicazione ad un dominio numerico

Vediamo tramite un esempio come viene applicata la linearizzazione ad un dominio numerico che non sia costituito da un intervallo, diverrà più chiaro il ruolo di π .

Esempio 4.3. Prendiamo come esempio il dominio astratto del segno. In questo caso $\pi(x) = [[\sigma(x)], [\sigma(x)]]$ e consideriamo il seguente programma.

$$\begin{aligned}
 X &\leftarrow 3.14 \\
 Y &\leftarrow X - 2 \times X
 \end{aligned}$$

Vediamo cosa succede nel dominio dei segni:

$$\begin{aligned}
 X &\leftarrow + \\
 Y &\leftarrow + - + \times + \\
 \sigma^\# &= \{Y \mapsto \top\}
 \end{aligned}$$

Come risultato otterremo un allarme. Vediamo invece cosa succede sfruttando la linearizzazione:

$$\begin{aligned}
 X &\leftarrow + \\
 Y &\leftarrow -\pi(X) \quad \text{dove } \pi(X) = [3, 4]
 \end{aligned}$$

La linearizzazione è entrata in funzione ed è riuscita ad operare una semplificazione, ora $Y \mapsto [-4, -3]$ nel dominio degli intervalli. Grazie alla funzione di trasferimento definita per $\mathbb{D}^\#$ il comando $\mathcal{C}[X \leftarrow e]\sigma^\#$ è denotato da:

$$\{\sigma^\#[X \mapsto v'] \mid v \in \mathcal{A}[e]\sigma^\#\} \quad v' = \begin{cases} + & \text{se } v = [a, b] \text{ e } a, b \geq 0 \\ - & \text{se } v = [a, b] \text{ e } a, b < 0 \\ \top & \text{se } v = [a, b] \text{ e } a < 0, b \geq 0 \\ \top & \text{se } v = \top \\ \perp & \text{se } v = \perp \end{cases}$$

Concludendo otteniamo $\sigma^\# = \{Y \mapsto -\}$

4.2 Strategie

Abbiamo visto che l'operatore ι fa perdere traccia delle operazioni svolte, e quindi aumenta l'incertezza del risultato, ma possiamo applicarlo meglio? Certo, esso viene infatti applicato ad uno degli operandi della moltiplicazione, possiamo ad esempio scegliere quello che ci introduce meno incertezza. Vengono di seguito presentate alcune strategie per migliorare la linearizzazione, purtroppo però non esiste una strategia migliore e queste vanno applicate a seconda dei casi.

La moltiplicazione La moltiplicazione è quindi l'operazione su cui agire per non aggiungere troppa incertezza.

4.2.1 Strategie locali

Intervallo minore Una strategia locale consiste nell'intervalizzare l'espressione che porta ad un intervallo più ristretto.

Se indichiamo con \leq la relazione d'ordine tra due intervalli come:

$$[a, b] \leq [a', b'] \equiv b - a \leq b' - a'$$

Allora:

$$(e_1 \times e_2)\sigma^\# = \begin{cases} \iota((e_1)\sigma^\#)\sigma^\# \times (e_2)\sigma^\# & \text{se } \iota((e_1)\sigma^\#)\sigma^\# \leq \iota((e_2)\sigma^\#)\sigma^\# \\ \iota((e_2)\sigma^\#)\sigma^\# \times (e_1)\sigma^\# & \text{altrimenti} \end{cases}$$

Ampiezza relativa Si può migliorare la strategia precedente sfruttando la differenza relativa tra due intervalli anziché quella assoluta. La relazione d'ordine diviene quindi:

$$[a, b] \leq [a', b'] \equiv \frac{b - a}{|b + a|} \leq \frac{b' - a'}{|b' + a'|}$$

4.2.2 Strategie globali

Linearizza tutto Sicuramente la strategia più intuitiva è quella di provare a linearizzare ambedue le espressioni del prodotto. Modificando opportunamente la definizione 4.1 possiamo fare in modo di ottenere, come risultato del prodotto, un insieme di forme affini anziché una soltanto. Quindi si può analizzare separatamente ogni risultato ottenuto e alla fine intersecare \cap in $\mathbb{D}^\#$ gli elementi astratti ottenuti. Purtroppo però questa strategia ha costo esponenziale sul numero di moltiplicazioni.

Tendi a semplificare Dato che sfruttare la forma affine ci permette di ottenere delle semplificazioni, dobbiamo stare attenti che la linearizzazione di un operando non ci faccia perdere occasioni di cancellare qualche elemento. Per esempio se nell'espressione: $X - (Y \times X)$ applichiamo ι su X andremo a perdere una successiva semplificazione.

Omogeneità Prendiamo questo frammento di codice:

$$\begin{aligned} X &\leftarrow [0, 1] \\ Y &\leftarrow [0, 10] \\ Z &\leftarrow [0, 20] \\ T &\leftarrow X \times Y - X \times Z + Z \end{aligned}$$

Se usiamo la strategia che tende a semplificare, non andremo a scegliere di applicare ι ad X , ed otterremo $T \in [-20, 30]$, sebbene $T \leftarrow [0, 1] \times Y - [0, 1] \times Z + Z$ ci permette di dire che $T \in [0, 30]$ e questo intervallo è il migliore. Anche le strategie locali, essendo gli intervalli di Y e Z maggiori di quello di X , tendono a scartare la X , pure usando l'ampiezza relativa al posto di quella assoluta. Per risolvere questo problema possiamo scegliere di applicare la linearizzazione al più piccolo insieme di variabili che rendono l'espressione omogenea, ovvero che tutti i monomi risultanti abbiano lo stesso grado.

5 Propagazioni di costanti simboliche

Lavorando sintatticamente sulle sole espressioni è facile perdere alcune ovvie semplificazioni:

Esempio 5.1.

$$\begin{aligned} X &\leftarrow [0, 5] \\ T &\leftarrow X \\ Y &\leftarrow T - 2 \times X \end{aligned}$$

Con la linearizzazione ottengo:

$$\begin{aligned} T &\leftarrow [1, 1] \times T - [2, 2] \times X \\ \sigma^\# &= \{Y \mapsto [-10, 5]\} \end{aligned}$$

È evidente però che l'ultimo comando poteva essere riscritto come: $Y \leftarrow X - 2 \times X \equiv Y \leftarrow -X$ ottenendo però lo stato: $\sigma^\# = \{Y \mapsto [-5, 0]\}$

Qui entra in gioco la propagazione delle costanti simboliche che riesce ad accorgersi di questo inconveniente. Andremo a definire un dominio di costanti simboliche che dinamicamente ad ogni assegnamento sia in grado di propagare espressioni.

5.1 Dominio di costanti simboliche

Denotiamo con \mathcal{C} l'insieme di tutte le espressioni sintattiche con l'aggiunta dei simboli \top per indicare qualunque espressione e \perp per non indicarne alcuna.

Definizione 5.1. La funzione $occ : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{V})$ restituisce l'insieme delle variabili che occorrono in una espressione. Definita tramite induzione strutturale:

- $exp \equiv \perp: \emptyset$
- $exp \equiv \top: \emptyset$
- $exp \equiv [a, b]: \emptyset$
- $exp \equiv V: \{V\}$
- $exp \equiv e_1 \diamond e_2: occ(e_1) \cup occ(e_2) \quad \diamond \in \{+, -, \times, /\}$

Definizione 5.2. La funzione $subst : \mathcal{C} \times \mathcal{V} \times \mathcal{C} \rightarrow \mathcal{C}$ sostituisce nel primo argomento, ogni occorrenza della variabile V , con l'espressione data dal terzo. Definita tramite induzione strutturale sulla prima espressione:

$subst(exp_1, v, exp_2)$

- $exp_1 \equiv \perp: \perp$
- $exp_1 \equiv \top: \top$
- $exp_1 \equiv [a, b]: [a, b]$
- $exp_1 \equiv V: \{exp_2\}$ se $exp_1 = v \{exp_1\}$ altrimenti
- $exp_1 \equiv e_1 \diamond e_2: subst(e_1, v, exp_2) \cup subst(e_2, v, exp_2) \quad \diamond \in \{+, -, \times, /\}$

Definizione 5.3. Dominio di costanti simboliche

1. Il dominio di costanti simboliche è l'insieme $\mathbb{D}^{\mathcal{C}} = \mathcal{V} \rightarrow \mathcal{C}$ ristretto come segue: non ci devono essere dipendenze cicliche, ovvero non devono esistere coppie distinte di variabili $V_1, \dots, V_n \in \mathcal{V}$ tali che $V_2 \in \text{occ}(\sigma^{\sharp}(V_1)), \dots, V_n \in \text{occ}(\sigma^{\sharp}(V_{n-1}))$ e $V_1 \in \text{occ}(\sigma^{\sharp}(V_n))$.

2. L'ordine parziale \sqsubseteq su $\mathbb{D}^{\mathcal{C}}$ è un ordinamento piatto:

$$\forall X^{\sharp} \in \mathbb{D}^{\mathcal{C}}, \perp \sqsubseteq X^{\sharp} \sqsubseteq \top$$

3. Ogni elemento $S \in \mathbb{D}^{\mathcal{C}}$ rappresenta l'insieme degli ambienti compatibili con l'informazione simbolica:

$$\gamma(S) = \{\sigma \in (\mathcal{V} \rightarrow \mathbb{D}) \mid \forall k, \sigma(V_k) \in \mathcal{A}[\![S(V_k)]\!] \sigma\}$$

Grazie alla condizione di non ciclicità della definizione 5.3 possiamo usare una regola di riscrittura, senza temere la non terminazione del processo. Dato un ambiente astratto $\sigma^{\sharp} \in \mathbb{D}^{\mathcal{C}}$, che per definizione non ha dipendenze cicliche:

$$\mathcal{R}(\sigma^{\sharp}) = \{\text{exp} \rightarrow \text{subst}(\text{exp}, V_i, \sigma^{\sharp}(V_i)) \mid \forall \text{exp} \forall i \text{ tale che } \sigma^{\sharp}(V_i) \neq \top\}$$

Teorema 5.1 (Correttezza della sostituzione). Dato un ambiente astratto $\sigma^{\sharp} \in \mathbb{D}^{\mathcal{C}}$ ed una espressione exp , ogni $\text{subst}(\text{exp}, V, \sigma^{\sharp}(V))$ sovra approssima exp rispetto \sqsubseteq_{σ} :

$$\forall \text{exp}, \sigma^{\sharp}, V, \quad \text{exp} \sqsubseteq_{\sigma} \text{subst}(\text{exp}, V, \sigma^{\sharp}(V))$$

Dimostrazione. Tramite induzione strutturale:

- $\text{exp} \equiv \perp$: $\perp =_{\sigma} \perp$.
- $\text{exp} \equiv \top$: $\top =_{\sigma} \top$.
- $V \notin \text{occ}(\text{exp})$: $\text{exp} =_{\sigma} \text{exp}$.
- $V \in \text{occ}(\text{exp})$: $\sigma^{\sharp}(V)$ contiene una qualsiasi espressione exp tracciata dal dominio $\mathbb{D}^{\mathcal{C}}$ associata alla variabile V .
 - $\text{exp} \equiv V$: $V \sqsubseteq_{\sigma} \sigma^{\sharp}(V) \Leftrightarrow \mathcal{A}[\![V]\!] \sigma \subseteq \mathcal{A}[\![\sigma^{\sharp}(V)]\!] \sigma = \sigma(V) \subseteq \gamma \circ \sigma^{\sharp}(V)$ ma per la definizione 5.3 di γ : $\sigma(V) \in \mathcal{A}[\![\sigma^{\sharp}(V)]\!] \sigma$ e quindi $\mathcal{A}[\![V]\!] \sigma \subseteq \mathcal{A}[\![\sigma^{\sharp}(V)]\!] \sigma$ e così $V \sqsubseteq_{\sigma} \sigma^{\sharp}(V)$.
 - $\text{exp} \equiv e_1 \diamond e_2$ con $\diamond \in \{+, -, \times, /\}$: Per ipotesi induttiva: $e_1 \sqsubseteq_{\sigma} \sigma^{\sharp}(e_1)$ ed $e_2 \sqsubseteq_{\sigma} \sigma^{\sharp}(e_2)$, ora essendo gli operandi $\sigma^{\sharp}(e_1), \sigma^{\sharp}(e_2)$ una sovra approssimazione dei rispettivi $\sigma(e_1), \sigma(e_2)$ non può che essere: $e_1 \diamond e_2 \sqsubseteq_{\sigma} \sigma^{\sharp}(e_1) \diamond \sigma^{\sharp}(e_2)$.

□

La relazione inversa non vale. Prendiamo ad esempio $\text{exp} = X - X$ con $\sigma^{\sharp} = \{X \mapsto [0, 1]\}$. $\text{subst}(\text{exp}, X, [0, 1]) = [-1, 1]$ che è meno preciso di $X - X$. Non c'è perdita di precisione quando $\sigma^{\sharp}(V_i)$ viene valutata in un singoletto, cioè quando $\sigma^{\sharp}(V_i) \neq \top$ e quando $\sigma^{\sharp}(V_i) \neq [a, b]$ con $a \neq b$. Come conseguenza a questo teorema possiamo applicare delle sostituzioni, in ogni ambiente σ^{\sharp} , il valore associato alla variabile V_i da $\text{subst}(\sigma^{\sharp}(V_i), V_j, \sigma^{\sharp}(V_j))$, per ogni V_j :

$$\gamma(\sigma^{\sharp}) \subseteq \gamma([\![V_i \mapsto \text{subst}(\sigma^{\sharp}(V_i), V_j, \sigma^{\sharp}(V_j))]\!] \sigma)$$

Occorre fare attenzione che sebbene il teorema 5.1 ci garantisce di poter far ogni sorta di sostituzione, non ci rende immuni dall'introdurre ciclicità nelle espressioni.

Definizione 5.4. Definiamo le operazioni su \mathbb{D}^c :

$$\begin{aligned} \mathcal{C} \llbracket V \leftarrow e \rrbracket (S^c)(V_k) &= \begin{cases} \text{subst}(e, V, S^c(V)) & \text{se } V = V_k \\ \text{subst}(S^c(V_k), V, S^c(V)) & \text{se } V \neq V_k \end{cases} \\ S^c \cup T^c &= \begin{cases} S^c(V_k) & \text{se } S^c(V_k) = T^c(V_k) \\ \top & \text{altrimenti} \end{cases} \\ S^c \cap T^c &= S^c \end{aligned}$$

L'assegnazione $V \leftarrow e$ prima sostituisce V con $S^c(V)$ in S^c ed e prima di aggiungere l'informazione che mappa V alla nuova espressione di e . Questo è necessario per rimuovere tutte le informazioni su V , non più valide dopo l'assegnamento, e per prevenire dipendenze cicliche. Tutte le operazioni rispettano la condizione di non ciclicità.

Il dominio di costanti simboliche non è in grado di valutare le espressioni e quindi dovrà essere supportato da un altro dominio astratto. Può essere associato quindi alla linearizzazione ma anche ad altri domini che siano in grado di apportare delle semplificazioni che \mathbb{D}^c è in grado di preparare.

Esempio 5.2.

$$\begin{aligned} X \leftarrow [0, 5] & \quad \sigma^\# = \{X \mapsto [0, 5]\} \\ T \leftarrow X & \quad \sigma^\# = \{X \mapsto [0, 5], T \mapsto X\} \\ Y \leftarrow T - 2 \times X & \quad \sigma^\# = \{X \mapsto [0, 5], T \mapsto X\} \\ Y \leftarrow X - 2 \times X & \quad \sigma^\# = \{X \mapsto [0, 5], T \mapsto X\} \end{aligned}$$

A questo punto \mathbb{D}^c si ferma, c'è bisogno di un altro dominio astratto per proseguire la computazione.

Dall'esempio precedente non risulta chiaro quando operare le sostituzioni, vediamo a quali strategie possiamo far ricorso.

5.2 Strategie

Sostituisci sempre Grazie alla non ciclicità degli elementi in \mathbb{D}^c possiamo effettuare tutte le sostituzioni $e \mapsto \text{subst}(e, V, S^c(V))$ per ogni variabile V in qualunque ordine, senza temere che il processo non termini.

Mai espressioni senza variabili La precedente strategia è piuttosto banale, come prima miglioria possiamo pensare di evitare le sostituzioni di espressioni libere da variabili. Tale strategia ci serve per non rischiare di perdere correlazioni tra le occorrenze delle variabili e quindi perdere qualche cancellazione. Nell'esempio precedente X non farà parte del dominio ed eviteremo di sostituirlo con un intervallo, così facendo non perdiamo la semplificazione nell'ultimo comando.

Determinismo Il non determinismo è una delle maggiori cause di imprecisione, quindi dovremmo evitare sostituzioni tali per cui $\llbracket S^c(V) \rrbracket \circ \gamma > 1$. Per non usare l'operatore γ si può più semplicemente richiedere che $S^c(V)$ non sia \top né che contenga un intervallo $[a, b]$ con $a \neq b$. Per contro non conviene evitare troppe sostituzioni, altrimenti il processo di linearizzazione potrebbe risultare inutile.

Ottenere maggior precisione Per ottenere una migliore precisione possiamo provare più sostituzioni diverse nella stessa espressione, computarle separatamente, e alla fine intersecare i risultati ottenuti. Purtroppo però questa strategia ha un costo esponenziale sul numero di sostituzioni.

5.3 Integrazione con un dominio numerico astratto

Un dominio di costanti simboliche deve essere accoppiato ad un dominio numerico astratto, esso infatti non è in grado di valutare le espressioni. \mathbb{D}^C si comporta in modo molto simile ad un ambiente Σ piuttosto che ad un dominio astratto $\mathbb{D}^\#$.

Definizione 5.5. *Il dominio prodotto $\mathbb{D}^\# \times^C$ è ottenuto nel seguente modo:*

- $\mathbb{D}^\# \times^C = \mathbb{D}^\# \times \mathbb{D}^C$
- $\sqsubseteq^\# \times^C = \sqsubseteq^\#$ *La relazione d'ordine è definita dal primo dominio. \mathbb{D}^C è solo un dominio di supporto utile a migliorare la precisione di $\mathbb{D}^\#$ ed ha un ordinamento piatto.*
- $\cup^\# \times^C = \cup^\# \times \cup^C$
- $\cap^\# \times^C = \cap^\# \times \cap^C$
- $\nabla^\# \times^C = \nabla^\# \times \nabla^C$
- $\gamma^\# \times^C(R^\#, S^C) = \gamma(R^\#) \cap \gamma(S^C)$

$\mathbb{D}^\#$ e \mathbb{D}^C sono indipendenti eccetto per ciò che riguarda il comando di assegnamento:

$$\mathcal{C}[[V \leftarrow exp]^\# \times^C (R^\#, S^C)] = (\mathcal{C}[[V \leftarrow subst(exp, V, S^C)]], \mathcal{C}[[V \leftarrow e]](S^C))$$

Sfruttando una qualsiasi strategia di sostituzione per *subst* vista in precedenza.

6 Conclusione

La tecnica della linearizzazione poggia interamente sull'aritmetica affine. Sfrutta la sua caratteristica di non valutare le espressioni immediatamente, quando possibile, quindi si comporta come una lazy-evaluation, in modo così di riuscire a sfruttare la cancellazione tra gli operandi. La propagazione delle costanti non fa altro che riuscire a ricordare la composizione delle espressioni per cercare di semplificarle al momento della loro valutazione. Va da sé che nel caso in cui le espressioni siano scritte a regola d'arte, cioè già semplificate per via analitica, tutte queste tecniche risultano inutili. Certo, spesso nella codifica di un algoritmo si predilige la maggior semplicità di lettura di quest ultimo, piuttosto che la sua ottimizzazione, a maggior ragione ciò avviene nello sviluppo di software critico. Ma purtroppo, a causa della stessa forma affine, siamo in grado di apportare semplificazioni solo tra somme e sottrazioni, nulla riusciamo a fare tra prodotti e divisioni se non in particolari casi.

Nonostante questi problemi, bisogna dire che lo sviluppo di queste tecniche è stato abbastanza semplice, e sono computazionalmente efficienti. La cosa però che risulta maggiormente interessante è l'attenzione che viene posta prima sulla semplificazione delle espressioni, per migliorare la precisione, ma soprattutto sull'idea di base di non creare nuovi domini numerici astratti, ma di tendere a migliorare quelli già presenti tramite tecniche simboliche.

Dove concentrarsi per lavori futuri? A mio avviso tenderei ad abbandonare la forma affine, sebbene sia risultata efficace in questa trattazione. Realizzare un dominio astratto affine permetterebbe sì di migliorare ancora la precisione, infatti l'assegnazione non appiattirebbe più una forma affine ma sarebbe in grado di portarla avanti nella computazione alla ricerca di una cancellazione, ma l'operazione di moltiplicazione è quella che mi dà maggiori pensieri. Purtroppo il prodotto tra due forme affini non è una forma affine: $(i_0 + \sum_k i_k \times V_k) \times (j_0 + \sum_l j_l \times U_l) = (i_0 \times j_0) + (\sum_k j_0 \times i_k \times V_k) + (\sum_l i_0 \times j_l \times U_l) + (\sum_k \sum_l i_k \times j_l \times V_k \times U_l)$ l'ultima sommatoria è quella incriminata. La semplificazione, come mostrato dai risultati ottenuti da Miné, è una buona strada da intraprendere, punterei quindi alla semplificazione delle espressioni tramite l'algebra simbolica[4]. Come effetto collaterale c'è da annoverare la possibilità di riscrittura stessa degli algoritmi: semplificare le espressioni significa, in generale, eseguire in minor numero di operazioni, quindi ottenere algoritmi più veloci e anche più precisi.

Riferimenti bibliografici

- [1] Roberto Giacobazzi Agostino Dovier. Dispense per il corso di fondamenti dell'informatica. In *Linguaggi Formali e Calcolabilità*, 2000.
- [2] Marcus V. A. Andrade, João L. D. Comba, and Jorge Stolfi. Affine arithmetic. In *Abstracts of the International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (INTERVAL/94)*, pages 36–40, St. Petersburg (Russia), March 1994.
- [3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
- [4] Richard J. Fateman. Macsyma's general simplifier philosophy and operation. In *Macsyma Users Conference, Washington, D.C.*, pages 336–343, 1979.
- [5] James Gleick. A bug and a crash. 1996. <http://www.around.com/ariane.html>.
- [6] Bertrand Meyer. *Eiffel: the Language*. Prentice Hall PTR, 1992.
- [7] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, volume 3855 of *LNCS*, pages 348–363. Springer, 2002. <http://www.di.ens.fr/~mine/publi/article-mine-VMCAI06.pdf>.
- [8] G. Winskel. *The formal semantics of programming languages*. The MIT Press, 1993.