Automatische statische Kosten-Analyse für parallele Programme

Martin Desharnais

5. März 2017

Zusammenfassung

Diese Arbeit präsentiert eine typbasierte statische Programmanalyse, die polynomielle Ressourcenschranke für funktionale, parallele Programme automatisch bestimmt. Während der Typableitung werden lineare Gleichungen erzeugt, die den Ressourcenverbraucht der Programmteile darstellt. Das Lösen dieses linearen Systems durch ein LP-Solver liefert die tatsächliche Schranke. Im Folgenden wird den Prozess der Typ- und Gleichungenableitung erklärt.

Inhaltsverzeichnis

1 Einleitung

Eine statische Analyse der Ressourcenkosten eines Programmes ist ein klassisches Thema der Informatik, welches in letzter Zeit immer populärer wurde. Ressourcenkosten können z. B. die benötigte Menge Arbeitsspeicher oder die Anzahl Gleitkommazahlmultiplikationen sein. Formell bewiesene Ressourcenschranken sind in manchen Bereichen wie Echtzeitsysteme, eingebettete Systeme oder Luft- und Raumfahrten sehr wichtig geworden. Die Unfähigkeit eines Systems, seinen Ressourcenverbrauch unter Kontrolle zu behalten, kann in den besten Fällen viel Geld und in den Schlechtesten Leben kosten. Sobald es in Betrieb ist, kann es schwer oder gar unmöglich sein, das unpassende System zu ersetzen.

Eine manuelle Ausarbeitung einer Kostensemantik ist sehr aufwendig und fehleranfällig. Aus diesem Grund wurde und wird noch viel über rechnerunterstützte Analysen geforscht. Obwohl eine automatische Berechnung der Kostenschranken pauschal unentscheidbar ist, können neue Analysen solche Schranken für beschränkte aber nicht-triviale Programme effizient berechnen.

Für sequentielle Programme existieren schon viele automatische und halbautomatische Analysesysteme, die statisch Kostenschranken ableiten können. Typische Strategien sind dimensionierte Typen, Differenzengleichungen und amortisierte Ressourcenanalysen. Das Ziel ist, einen möglichts einfachen arithmetischen Ausdruck über die Größe der Parameter des Programmes automatisch zu berechnen.

Für parallele Programme gab es bis kürzlich keine Analysesysteme und Überlegungen mussten noch manuell bewältigt werden. Diese Situation war unbefriedigend, da parallele Auswertungen eine immer wichtiger werdende Rolle spielen.

2015 veröffentlichten Jan Hoffmann und Zhong Shao eine automatische, statische Ressourcenkostenanalyse für parallele Programme [HS15]. Diese ist eine typbasierte Analyse über funktionale Programme erster Ordnung.

Zwei Begriffe sind bei solchen Analysen zu unterscheiden: Arbeit und Tiefe. Die *Arbeit*[1] drückt die Kosten einer sequentiellen Auswertung und die *Tiefe*[2] drückt die Kosten einer parallelen Auswertung auf einer Maschine mit unendlich vielen Prozessoren aus. Diese letzte Definition könnte unnuztbar aussehen, da es unmöglich ist, eine solche Maschine herzustellen. Es wurde aber bewiesen, dass die Kosten auf einer Maschine mit einer beliebigen Anzahl Prozessoren durch die Tiefe berechnet werden kann.

Das *Theorem des Brents* [Har16, § 37.4] stellt eine asymptotische Oberschranke der Auswertungsschritte auf einem symmetrischen Multiprozessorsystem zur Verfügung. Ein solches SMP-System hat eine fixierte Anzahl Prozessoren und konstante Zugriffszeit auf einen globalen Speicher. Wenn ein Programm mit Arbeit w und Tiefe d auf einem SMP-System mit p Prozessoren ausgewertet wird, dann sind die Kosten $\in \mathcal{O}(\max(w/p, d))$.

Die Arbeit und die Tiefe werden durch unterschiedliche Typableitungen berechnet. Um eine Arbeitschranke abzuleiten, wird die o. g. multivariate amortisierte Ressourcenanalyse für sequentielle Programme verwendet. Um eine Tiefeschranke abzuleiten, wird die neue multivariate amortisierte Ressourcenanalyse für parallele Programme verwendet.

[1] Die Arbeit wird w, für das englische Wort „work“, abgekürzt.
[2] Die Tiefe wird d, für das englische Wort „depth“, abgekürzt.

Die hier betrachtete Programmiersprache wurde absichtlich vereinfacht, um die Essenz der Analyse möglichst kurz und einfach auszurichten. *Resource Aware ML* (RAML), die Implementierung des Analysesystems [HAH12], ist eine Teilmenge von OCaml und bietet mehrere zusätzliche Datentypen und Operationen. RAML wurde benutzt, um die Oberschranken mehrerer typischen funktionalen Programmbeispiele zu berechnen, einschließlich Matrixoperationen, einer Sortierungsfunktion und Mengenoperationen.

2 Eine funktionale Sprache erster Stufe

2.1 Ausdrücke

Die funktionale Sprache, auf der die Analyse ausgeführt wird, ist eine vereinfachte Version einer Sprache der ML-Familie[3]. Ausdrücke bestehen aus ganzen Zahlen n, Variablen x, Funktionsaufrufen $f(x)$, Paaren $\langle x_1, x_2\rangle$, Listen $\text{cons}(x_1, (\text{cons}(x_2, \dots \text{cons}(x_n, \text{nil}) \dots))$[4] und sequentieller bzw. paralleler Auswertung.

$$\begin{aligned} e, e_1, e_2 ::= {} & n \mid x \mid f(x) \mid \langle x_1, x_2\rangle \mid \text{nil} \mid \text{cons}(x_1, x_2) \mid \\ & \text{match } x \text{ with } \langle x_1, x_2\rangle \Rightarrow e \mid \\ & \text{match } x \text{ with nil} \Rightarrow e_1 \mid \text{cons}(x_1, x_2) \Rightarrow e_2 \mid \\ & \text{let } x = e_1 \text{ in } e_2 \mid \\ & \text{par } x_1 = e_1 \text{ and } x_2 = e_2 \text{ in } e \end{aligned}$$

Die einzige Erweiterung des üblichen Kernes ist der „par“-Ausdruck, der die parallele Auswertung zweier Unterausdrücke ausdrückt. In dem Programm „par $x_1 = e_1$ and $x_2 = e_2$ in e“ werden e_1 und e_2 parallel und unabhängig von einander ausgewertet. Dann wird e sequentiell ausgewertet, wobei die Ergebnisse von e_1 bzw. e_2 durch die Namen x_1 bzw. x_2 zur Verfügung stehen. Entgegen der sequentiellen Auswertung, wobei e_2 x_1 benutzten darf, dürfen also e_1 und e_2 erst in e als freie Variablen vorkommen.

Die gewählte Syntax garantiert, dass Ausdrücke immer in „let“-Normalform [SF92] stehen. Also statt `f(g(x), h(y))`, schreibt man folgendenmaßen:

```
let a = g(x) in
let b = h(y) in
f(a, b)
```

2.2 Einfaches Typsystem

Das Typsystem besteht aus einem einzigen atomischen Typ, dem der ganzen Zahlen, und aus drei zusammengesetzten Typen, nämlich Paaren, Listen und Funktionen.

$$\begin{aligned} A, B ::= {} & \text{int} \mid L(A) \mid A * B \\ F ::= {} & A \to B \end{aligned}$$

[3] So werden funktionale Programmiersprachen bezeichnet, welche die Haupteigenschaften von ML haben: statisches Typsystem, Typableitung, parametrisierte Polymorphie, Funktionen erster Klasse, automatische Speicherbereinigung, strikte Auswertung, algebraische Datentypen und Musterabgleich.

[4] Eine alternative Notation ist $[1, 2, \dots, x_n]$.

Seien $\mathcal{A}$ die Menge aller Datentypen und $\mathcal{F}$ die Menge aller Funktionstypen. Eine Signatur $\Sigma : \text{FID} \to \mathcal{F}$ ist eine partiale Abbildung von Funktionsbezeichnern zu Funktionstypen. Ein Kontext $\Gamma : \text{Var} \to \mathcal{A}$ ist eine partiale Abbildung von Variablenbezeichnern zu Variablentypen. Ein einfaches Typurteil $\Sigma; \Gamma \vdash e : A$ gibt an, dass der Ausdruck e unter der Signatur Σ und dem Kontext Γ den Typ A hat.

Ein wohltypisiertes Programm besteht aus einer Signatur Σ und einer Familie $(e_f, y_f)_{f \in \text{dom}(\Sigma)}$ von Ausdrücken e_f und Variablen y_f, sodass $\Sigma; \{y_f : A\} \vdash e_f : B$, wenn $\Sigma(f) = A \to B$.

2.3 Begleitendes Beispiel

Im Folgenden wird die Arbeit bzw. die Tiefe von Programmen analysiert, die Listen multiplizieren. Dafür wird eine Funktion `mult`[5] mit Typ $\text{int} * L(\text{int}) \to L(\text{int})$ benötigt, die alle Elemente einer Liste mit einem Argument multipliziert.

```
mult(k, xs) = match xs with
  | nil -> nil
  | cons(y, ys) -> let z = n * y in
                   let zs = mult(n, ys) in
                   cons(z, zs)
```

3 Semantik

Die Analyse wird in Bezug auf eine großschrittige operationelle Semantik (im Folgenden großschrittige Semantik abgekürzt) ausgeführt, die jedem Auswertungsschritt Kosten zuordnet. Eine operationelle Semantik fasst auf, wie Programme als eine Sequenze von Berechnungsschritten von einem Programmzustand zum Folgenden betrachtet wird. Dabei wird zwischen der kleinschrittigen Semantik, die jeden atomischen Schritt beschreibt, und der großschrittigen Semantik, die nur den Entstehungsprozess des Endzustands beschreibt, unterschieden.

Im Gegensatz zur üblichen großschrittige Semantik kann die hier benutzte Variante sowohl komplette als auch partielle Auswertungen ausdrücken. Um dies zu erreichen, sind Semantikurteile induktiv durch endliche Unterbäume von womöglich unendlichen Auswertungsbäumen definiert. Laut den Autoren der Analyse kombiniere diese Erweiterung die Vorteile von klein- und großschrittiger Semantik.

Dank dieser Erweiterung können Ressourcen für sowohl terminierende als auch nicht-terminierende Programme berechnet werden. In der Tat gibt es manche Kostenmetriken, bei denen nicht-terminierende Berechnungen endliche Kosten haben. Ein Beispiel einer solchen Kostenmetrik ist diejenige, welche die Anzahl der Paar- und Listliterale zählt. Dies könnte benutzt werden, um zu beweisen, dass ein Programm eine begrenzte Menge Arbeitsspeicher verwendet. Unter dieser Kostenmetrik hat die nicht-terminierende Funktion `f(x) = f(x)` mit Typ $\text{int} \to \text{int}$ sowohl eine Arbeit als auch eine Tiefe von Null.

[5] Als Variablenbezeichnungsmuster gilt, dass eine Variable genau dann mit einem „s“ endet, wenn jene eine Liste bezeichnet.

Ein besonderer Fall der Analyse tritt ein, wenn in einer Kostenmetrik alle nicht-terminierende Programme unendliche Kosten haben, da eine Schranke Termination impliziert. Ein Beispiel einer solchen Kostenmetrik ist diejenige, welche die Anzahl der Funktionsaufrufe zählt.

3.1 Semantikurteile

Semantikurteile werden bzgl. Halden- und Stapelspeicher und einer Kostenmetrik formuliert. Sei Loc eine unendliche Menge von Speicheradressen in dem Haldenspeicher darstellenden Lokalisierungen und Val die Menge von Werten

$$\{n, \langle \ell_1, \ell_2 \rangle, \mathrm{nil}, \mathrm{cons}(\ell_1, \ell_2)\}$$

bei der $n \in \mathbb{N}$ und $\ell_1, \ell_2 \in Loc$.

Ein Haldenspeicher ist eine partielle Abbildung $H : Loc \to Val$, die Lokalisierungen zu Werten abbildet. Die Notation $H, \ell \mapsto x$ wird als Synonym für $H \cup \{\ell \mapsto x\}$ verwendet. Die Einfügung eines neuen Elementes in den Haldenspeicher wird mit „Allokation“ bennant. Ein Beispiel eines Haldenspeichers ist $H_{[1,2,3]}$, der die Liste $[1,2,3] = \mathrm{cons}(1, \mathrm{cons}(2, \mathrm{cons}(3, \mathrm{nil})))$ enthält.

$$H_{[1,2,3]} = \left\{ \begin{array}{ll} \ell_1 \mapsto \mathrm{nil}, & \ell_2 \mapsto 3 \\ \ell_3 \mapsto \mathrm{cons}(\ell_2, \ell_1), & \ell_4 \mapsto 2, \\ \ell_5 \mapsto \mathrm{cons}(\ell_4, \ell_3), & \ell_6 \mapsto 1, \\ \ell_7 \mapsto \mathrm{cons}(\ell_6, \ell_5) & \end{array} \right\}$$

Ein Stapelspeicher ist eine partielle Abbildung $V : Var \to Loc$, die Variablenbezeichnern zu Lokalisierungen abbildet. Die Notation $V, x \mapsto \ell$ wird ebenso als Synonym für $V \cup \{x \mapsto \ell\}$ verwendet.

Eine Kostenmetrik $M : K \to \mathbb{Q}_0^+$ gibt die Kosten jedes Auswertungsschrittes an, bei der K die Menge der Schritte ist.

$$K = \{\mathrm{const}, \mathrm{pair}, \mathrm{nil}, \mathrm{cons}, \mathrm{var}, \mathrm{app}, \mathrm{matP}, \mathrm{matN}, \mathrm{matC}, \mathrm{let}, \mathrm{par}\}$$

Die Notation M^{k} wird als Synonym für $M(k)$ verwendet.

Ein erster besonderer Fall ist die Kostenfreimetrik $M^{\mathrm{cf}} = \{x \mapsto 0 \,|\, x \in K\}$, die jeden Schritt auf Null abbildet. Die Auswertung aller Ausdrücke bzgl. der Kostenfreimetrik haben Null Arbeit und Null Tiefe.

Ein zweiter besonderer Fall ist die einfache Kostenmetrik $M = \{x \mapsto 1 \,|\, x \in K\}$, die jeden Schritt auf Eins abbildet. Die Arbeit bzgl. der einfachen Kostenmetrik ist genau die Anzahl der Auswertungsschritte.

Ein Semantikurteil hat die Form

$$V, H \overset{M}{\vdash} e \Downarrow \rho \,|\, \langle w, d \rangle$$

bei der ρ entweder ein Paar $\langle \ell, H' \rangle$, bei dem ℓ eine Lokalisierung und H' ein neuer Haldenspeicher ist, oder eine nicht-terminierte Auswertung $\circ$ ist.

Die Bedeutung ist Folgende: unter gegebenen Stapelspeicher V und Haldenspeicher H wird der Ausdruck e zu ρ ausgewertet. Außerdem ist bzgl. der Kostenmetrik M die Arbeit w und die Tiefe d.

3.2 Semantikregeln

Die Semantikregeln E:Const, E:Pair, E:Nil und E:Cons sagen, dass das Literal einer Zahl bzw. eines Paares bzw. einer Liste konstante Kosten für sowohl die Arbeit als auch die Tiefe haben. Diese Semantikregeln sind die einzigen, die Allokationen durchführen. In allen kommenden Regeln bleibt der Haldenspeicher ungeändert.

$$\frac{n \in \mathbb{Z} \qquad H' = H, \ell \mapsto n}{V, H \mid\!\!\overset{M}{\vdash} n \Downarrow \langle \ell, H' \rangle \mid \langle M^{\text{const}}, M^{\text{const}} \rangle} \text{E:Const}$$

$$\frac{H' = H, \ell \mapsto \langle V(x_1), V(x_2) \rangle}{V, H \mid\!\!\overset{M}{\vdash} \langle x_1, x_2 \rangle \Downarrow \langle \ell, H' \rangle \mid \langle M^{\text{pair}}, M^{\text{pair}} \rangle} \text{E:Pair}$$

$$\frac{H' = H, \ell \mapsto \text{nil}}{V, H \mid\!\!\overset{M}{\vdash} \text{nil} \Downarrow \langle \ell, H' \rangle \mid \langle M^{\text{nil}}, M^{\text{nil}} \rangle} \text{E:Nil}$$

$$\frac{H' = H, \ell \mapsto \text{cons}(x_1, x_2)}{V, H \mid\!\!\overset{M}{\vdash} \text{cons}(x_1, x_2) \Downarrow \langle \ell, H' \rangle \mid \langle M^{\text{cons}}, M^{\text{cons}} \rangle} \text{E:Cons}$$

Die Semantikregel E:Var beschreibt, dass eine Variable ebenfalls konstante Kosten für sowohl die Arbeit als auch die Tiefe hat.

$$\frac{V(x) = \ell}{V, H \mid\!\!\overset{M}{\vdash} x \Downarrow \langle \ell, H \rangle \mid \langle M^{\text{var}}, M^{\text{var}} \rangle} \text{E:Var}$$

Die Semantikregel E:App sagt, dass die Kosten eines Funktionsaufrufs die Summe der Auswertungskosten des Funktionsrumpfes und der konstanten Kosten M^{app}. Für die Auswertung des Funktionsrumpfes e_f wird ein neuer Stapelspeicher erzeugt, welcher als einziges Element die Abbildung des Funktionsargumentes y_f auf die Lokalisierung $V(x)$ hat.

$$\frac{\{y_f \mapsto V(x)\}, H \mid\!\!\overset{M}{\vdash} e_f \Downarrow \rho \mid \langle w, d \rangle}{V, H \mid\!\!\overset{M}{\vdash} f(x) \Downarrow \rho \mid \langle M^{\text{app}} + w, M^{\text{app}} + d \rangle} \text{E:App}$$

Die Semantikregeln E:matP, E:matN und E:matL beschreibt, dass die Kosten der Fallunterscheidung eines Paares bzw. einer Liste die Summe der Auswertungskosten des Unterausdrucks und konstanter Kosten sind. Für die Auswertung des Unterausdrucks werden die Stapelspeicher erweitert, wenn neue Variablen zur Verfügung stehen.

$$\frac{H(V(x)) = \langle \ell_1, \ell_2 \rangle \qquad V, x_1 \mapsto \ell_1, x_2 \mapsto \ell_2, H \mid\!\!\overset{M}{\vdash} e \Downarrow \rho \mid \langle w, d \rangle}{V, H \mid\!\!\overset{M}{\vdash} \text{match } x \text{ with } \langle x_1, x_2 \rangle \Rightarrow e \Downarrow \rho \mid \langle M^{\text{matP}} + w, M^{\text{matP}} + d \rangle} \text{E:matP}$$

$$\frac{H(V(x)) = \text{nill} \qquad V, H \mid\!\!\overset{M}{\vdash} e_1 \Downarrow \rho \mid \langle w, d \rangle}{V, H \mid\!\!\overset{M}{\vdash} \text{match } x \text{ with nil} \Rightarrow e_1 \mid \text{cons}(x_1, x_2) \Rightarrow e_2 \Downarrow \rho \mid \langle M^{\text{matN}} + w, M^{\text{matN}} + d \rangle} \text{E:matN}$$

$$\frac{H(V(x)) = \mathrm{cons}(\ell_1, \ell_2) \qquad V, x_1 \mapsto \ell_1, x_2 \mapsto \ell_2, H \overset{M}{\vdash} e_2 \Downarrow \rho \,|\, \langle w, d\rangle}{V, H \overset{M}{\vdash} \text{match } x \text{ with nil} \Rightarrow e_1 \,|\, \mathrm{cons}(x_1, x_2) \Rightarrow e_2 \Downarrow \rho \,|\, \langle M^{\mathrm{matN}} + w, M^{\mathrm{matN}} + d\rangle}\ \text{E:MATL}$$

Die Semantikregel E:LET2 sagt die Kosten der sequentiellen Auswertung zweier Ausdrücke, wobei der Erste terminiert. Zuerst wird der erste Ausdruck bis zu einer Lokalisierung ℓ und einem neuen Haldenspeicher ausgewertet. Dieser wird dann mit der neuen Variable x erweitert. Schließlich wird der zweite Ausdruck ausgewertet. Die totale Arbeit und Tiefe sind nun die Summe der Auswertungskosten und konstanter Kosten.

$$\frac{V, H \overset{M}{\vdash} e_1 \Downarrow \langle \ell, H'\rangle \,|\, \langle w_1, d_1\rangle \qquad V, x \mapsto \ell, H' \overset{M}{\vdash} e_2 \Downarrow \rho \,|\, \langle w_2, d_1\rangle}{V, H \overset{M}{\vdash} \text{let } x = e_1 \text{ in } e_2 \Downarrow \rho \,|\, \langle M^{\mathrm{let}} + w_1 + w_2, M^{\mathrm{let}} + d_1 + d_2\rangle}\ \text{E:LET2}$$

Die Semantikregel E:LET1 beschreibt die Kosten der sequentiellen Auswertung zweier Ausdrücke, wobei der Erste nicht terminiert. Der erste Ausdruck wird bis zu $\circ$ ausgewertet. Die totale Arbeit und Tiefe sind nur die Summe der Auswertungskosten dieses ersten Ausdrucks und konstanter Kosten.

$$\frac{V, H \overset{M}{\vdash} e_1 \Downarrow \circ \,|\, \langle w, d\rangle}{V, H \overset{M}{\vdash} \text{let } x = e_1 \text{ in } e_2 \Downarrow \circ \,|\, \langle M^{\mathrm{let}} + w, M^{\mathrm{let}} + d\rangle}\ \text{E:LET1}$$

Die Semantikregel E:PAR2 sagt die Kosten der parallelen Auswertung zweier Ausdrücke, wobei beide terminieren. Zuerst werden sie bis zu Lokalisierungen und neuen Haldenspeichern ausgewertet. Dann wird der Stapelspeicher mit den Abbildungen von x_1 bzw. x_2 erweitert. Außerdem wird ein neuer Haldenspeicher $H_1 \uplus H_2$ erzeugt, der die Vereinigung der zwei gelieferten Haldenspeicher H_1 und H_2 ist. Die Notation $H_1 \uplus H_2$ definiert einen neuen Haldenspeicher H, sodass $\forall \ell \in H_1 \cap H_2.\ H_1(\ell) = H_2(\ell)$ und $\mathrm{dom}(H) = \mathrm{dom}(H_1) \cup \mathrm{dom}(H_2)$ und

$$H(x) = \begin{cases} H_1(x) & \text{wenn } x \in \mathrm{dom}(H_1) \cap \mathrm{dom}(H_2) \wedge H_1(x) = H_2(x) \\ H_1(x) & \text{wenn } x \in \mathrm{dom}(H_1) \backslash \mathrm{dom}(H_2) \\ H_2(x) & \text{wenn } x \in \mathrm{dom}(H_2) \backslash \mathrm{dom}(H_1) \end{cases}$$

Schließlich wird der lezte Ausdruck e ausgewertet. Die totale Arbeit und Tiefe sind nun $M^{\mathrm{par}} + w_1 + w_2 + w$ bzw. $M^{\mathrm{par}} + \max(d_1, d_2) + d$.

$$\frac{\begin{array}{c} V, H \overset{M}{\vdash} e_1 \Downarrow \langle \ell_1, H_1\rangle \,|\, \langle w_1, d_1\rangle \qquad V, H \overset{M}{\vdash} e_2 \Downarrow \langle \ell_2, H_2\rangle \,|\, \langle w_2, d_2\rangle \\ V, x_1 \mapsto \ell_1, x_2 \mapsto \ell_2, H_1 \uplus H_2 \overset{M}{\vdash} e \Downarrow \langle \ell, H'\rangle \,|\, \langle w, d\rangle \end{array}}{V, H \overset{M}{\vdash} \text{par } x_1 = e_1 \text{ and } x_2 = e_2 \text{ in } e \Downarrow \langle \ell, H'\rangle \,|\, \langle M^{\mathrm{par}} + w_1 + w_2 + w, M^{\mathrm{par}} + \max(d_1, d_2) + d\rangle}\ \text{E:PAR2}$$

Die Semantikregel E:PAR1 beschreibt die Kosten der nicht-terminierenden parallelen Auswertung zweier Ausdrücke. Zuerst werden die zwei ersten Ausdrücke e_1 und e_2 mit Arbeit w_1 bzw. w_2 und Tiefe d_1 bzw. d_2 ausgewertet. Die Auswertung von mindestens einem führt bis zu $\circ$. Die totale Arbeit und Tiefe sind nun $M^{\mathrm{par}} + w_1 + w_2$ bzw. $M^{\mathrm{par}} + \max(d_1, d_2)$.

$$\frac{V, H \overset{M}{\vdash} e_1 \Downarrow \rho_1 \mid \langle w_1, d_1 \rangle \qquad V, H \overset{M}{\vdash} e_2 \Downarrow \rho_2 \mid \langle w_2, d_2 \rangle \qquad \rho_1 = \circ \vee \rho_2 = \circ}{V, H \overset{M}{\vdash} \text{par } x_1 = e_1 \text{ and } x_2 = e_2 \text{ in } e \Downarrow \circ \mid \langle M^{\text{par}} + w_1 + w_2, M^{\text{par}} + \max(d_1, d_2) \rangle} \text{E:PAR1}$$

Die Semantikregel E:ABORT sagt, dass die Auswertung eines Ausdrucks jederzeit ohne Kosten für sowohl die Arbeit als auch die Tiefe gestoppt werden kann.

$$\frac{}{V, H \overset{M}{\vdash} e \Downarrow \circ \mid \langle 0, 0 \rangle} \text{E:ABORT}$$

3.3 Beispiele

Wir benutzen jetzt die oben definierte Semantik, um die Arbeit und Tiefe der Funktion `mult` manuell herauszufinden.

Seien V ein Stapelspeicher, H ein Haldenspeicher und M^* eine die Anzahl der Multiplikationsoperatoren zählende Kostenmetrik. Die Funktion `mult` hat für einige konkrete Argumente die folgenden Semantikurteile:

$$\begin{aligned}
V, H &\overset{M}{\vdash} \text{mult}(2, []) &&\Downarrow \langle \ell, H' \rangle \mid \langle 0, 0 \rangle \\
V, H &\overset{M}{\vdash} \text{mult}(2, [1]) &&\Downarrow \langle \ell, H' \rangle \mid \langle 1, 1 \rangle \\
V, H &\overset{M}{\vdash} \text{mult}(2, [1, 2]) &&\Downarrow \langle \ell, H' \rangle \mid \langle 2, 2 \rangle \\
V, H &\overset{M}{\vdash} \text{mult}(2, [1, 2, 3]) &&\Downarrow \langle \ell, H' \rangle \mid \langle 3, 3 \rangle
\end{aligned}$$

Im Allgemeinfall ist sowohl die Arbeit als auch die Tiefe von `mult(k, xs)` die Länge n, bei der $n = |xs|$.

$$V, H \overset{M}{\vdash} \text{mult}(k, xs) \Downarrow \langle \ell, H' \rangle \mid \langle n, n \rangle$$

Als weiteres Beispiel werden ein paar Ableitungen unter der Kostenmetrik M^{app} von $w(x)$ abgewägt, bei der w ein nicht-terminierender Ausdruck ist. Ohne Beschränkung der Allgemeinheit wird w als die selbstrekursive Funktion `w(x) = w(x)` und x als Null definiert.

$$\frac{}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle 0, 0 \rangle} \text{E:ABORD}$$

$$\frac{\dfrac{}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle 0, 0 \rangle} \text{E:ABORD}}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle M^{\text{app}}, M^{\text{app}} \rangle} \text{E:APP}$$

$$\frac{\dfrac{\dfrac{}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle 0, 0 \rangle} \text{E:ABORD}}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle M^{\text{app}}, M^{\text{app}} \rangle} \text{E:APP}}{\{x \mapsto \ell\}, \{\ell \mapsto 0\} \overset{M^{\text{app}}}{\vdash} w(x) \Downarrow \circ \mid \langle 2 \cdot M^{\text{app}}, 2 \cdot M^{\text{app}} \rangle} \text{E:APP}$$

Es ist möglich, $V, H \mid\!\!\frac{M}{} w(x) \Downarrow \langle \ell, H' \rangle \mid \langle n \cdot M^{\text{app}}, n \cdot M^{\text{app}} \rangle$ für alle $n \in \mathbb{N}$ abzuleiten. Der Grund dafür ist, dass willkürlich gewählt wird, wann die Semantikregel E:Abord verwendet wird, um die Ableitung zu enden.

3.4 Theoreme

Das erste Theorem sagt, dass Ressourcenkosten einer partiellen Auswertung kleiner oder gleich sind wie die einer terminierenden Auswertung.

Theorem 1. *Wenn* $V, H \mid\!\!\frac{M}{} e \Downarrow \langle \ell, H' \rangle \mid \langle w, d \rangle$ *und* $V, H \mid\!\!\frac{M}{} e \Downarrow \circ \mid \langle w', d' \rangle$ *, dann* $w' \leq w$ *und* $d' \leq d$*.*

Das Erhaltung-Theorem sagt, dass ein wohltypisierter Ausdruck in einer wohlgeformten Umgebung zu einem wohltypisierten Ausdruck des selben Typs in einer wohlgeformten Umgebung ausgewertet wird.

Theorem 2 (Erhaltung)**.** *Wenn* $\Sigma ; \Gamma \vdash e : B$ *und* $H \vDash V : \Gamma$ *und* $V, H \mid\!\!\frac{M}{} e \Downarrow \langle \ell, H' \rangle \mid \langle w, d \rangle$ *, dann* $H' \vDash V : \Gamma$ *und* $H' \vDash \ell : B$*.*

Das Fortschritt-Theorem sagt, dass ein wohltypisierter Ausdruck in einer wohlgeformten Umgebung entweder zu einem Wert ausgewertet wird oder dass sowohl die Arbeit als auch die Tiefe unbegrenzt sind.

Theorem 3 (Fortschritt)**.** *Wenn* M *eine einfache Kostenmetrik ist und* $\Sigma ; \Gamma \vdash e : B$ *und* $H \vDash V : \Gamma$*, dann gibt es entweder einen Wert* ℓ*, sodass* $V, H \mid\!\!\frac{M}{} e \Downarrow \langle \ell, H' \rangle \mid \langle w, d \rangle$*, oder* $x, y \in \mathbb{N}$*, sodass* $V, H \mid\!\!\frac{M}{} e \Downarrow \circ \mid \langle x, n \rangle$ *und* $V, H \mid\!\!\frac{M}{} e \Downarrow \circ \mid \langle n, y \rangle$ *für alle* $n \in \mathbb{N}$*.*

Korollarium 1 (des Fortschritt-Theorems)**.** *Die Schranke der Tiefe einer einfachen Kostenmetrik impliziert Terminierung.*

4 Die Analyse

Eine gedämpfte Ressourcenanalyse ist eine typbasierte Technik, um Obergrenzen für Ressourcenkosten eines Programmes abzuleiten. Ein Vorteil dieser Technik ist, dass die notwendige Typableitung mit Hilfe der linearen Programmierung effizient implementiert werden kann.

Die Grundidee der Analyse ist, dass Typen mit Potential beschreibende Annotationen dekoriert werden. Dieses Potential Φ ist eine Funktion, welche die Größe einer Datenstruktur auf eine positive rationale Zahl abbildet. Die Annotationen sind Tupel, die multivariate Polynome darstellen, also Polynome mit mehreren Variablen. Das Potential eines Kontexts ist eine Kombination der Potentiale aller Elemente.

Das Typsystem gewährleistet, dass das Potential eines Kontexts reicht, um die Kosten einer Auswertung zu decken. Das übrige Potential wird benutzt, um die folgenden Ausdrücke auszuwerten.

4.1 Analyse der Funktion `mult`

Es wurde schon in Sektion 3.3 informell gezeigt, dass $V, H \mid\!\!\frac{M^*}{} \text{mult}(k, xs) \Downarrow \langle \ell, H' \rangle \mid \langle n, n \rangle$, bei dem $n = |ys|$. Ein Mensch kommt schnell zu diesem Schluss,

indem er für eine gewisse Menge von Argumenten Semantikbäume ableitet und eine Tendenz bei der Entwicklung der Arbeit bzw. der Tiefe bemerkt. Zu so einem kreativen Prozess ist ein Rechner aber unfähig.

Um zu einer solchen Schranke zu kommen, wird das Folgende annotierte Typurteil abgeleitet:

$$k : \text{int}, xs : L(\text{int})\,;\, Q \mid\!\!\!\overset{M^*}{\vdash} \text{mult}(k, xs) : \langle L(\text{int}), Q' \rangle$$

Hier sind Q und Q' die durch Tupel kodierte Polynome, die das Potential Φ_Q bzw. $\Phi_{Q'}$ darstellen.

Die Typisierungsregeln des Typsystems zeigen, wie lineare Formeln gebaut werden, welche den Zusammenhang zwischen der Arbeit und dem Anfangs- und Endpotential darstellen. In diesem Fall, wird die folgende Formel gebaut, bei der $V, H \mid\!\!\!\overset{M^*}{\vdash} \text{mult}(k, xs) \Downarrow \langle \ell, H' \rangle \mid \langle w, d \rangle$:

$$\Phi_Q(k, xs) \geq w + \Phi_{Q'}(H'(\ell))$$

In Worten gefasst muss das Anfangspotential größer oder gleich sein, wie die Summe der Arbeit und des Endpotentials. Eine dieser Formel entsprechende Belegung ist Folgende:

$$\Phi_Q(x, ys) = |ys| \qquad \text{und} \qquad \Phi_{Q'}(H'(\ell)) = 0$$

Die intuitive Bedeutung ist, dass das Potential $|ys|$ zur Verfügung stehen muss, wenn `mult(k, xs)` ausgewertet wird. Bei der Auswertung wird das ganze Potential benutzt, um die Arbeit w der Auswertung zu decken. Deswegen gibt es danach kein Potential mehr. Weil das Korrektheit-Theorem (cf. § 4) gewährleistet, dass $w \leq \Phi_Q(k, xs) - \Phi_{Q'}(H'(\ell))$, ist $|ys|$ eine Obergrenze der Arbeit von `mult`.

Eine Obergrenze darf aber deutlich höher als die tatsächlichen Kosten sein. Die folgende Belegung ist also ebenso gültig:

$$\Phi_Q(k, xs) = 5 \cdot |xs|^2 + 1\,000 \qquad \text{und} \qquad \Phi_{Q'}(H'(\ell)) = 0$$

4.2 Sequentielle Auswertung

Wenn ein Programm aus mehreren Ausdrücken besteht, werden die Kosten jeder Auswertung mit demselben Potential gedeckt. Das Anfangspotential wird benutzt, um den ersten Ausdruck zu analysieren. Das Endpotential wird dann als Anfangspotential des folgenden Ausdrucks benutzt. Dieser Prozess wiederholt sich bis zum letzten Ausdruck, dessen Endpotential auch das Endpotential des Programmes ist. Im Folgenden ein konkretes Beispiel.

```
mult2(xs) = let ys = mult(2, xs) in
            let zs = mult(3, xs) in
            (ys, zs)
```

Die folgenden Typurteile werden abgeleitet:

$$xs : L(\text{int})\,;\, Q_1 \mid\!\!\overset{M^*}{\longrightarrow} \text{mult}(2, xs) : \langle L(\text{int}), Q_2\rangle$$
$$xs : L(\text{int}), ys : L(\text{int})\,;\, Q_2 \mid\!\!\overset{M^*}{\longrightarrow} \text{mult}(3, xs) : \langle L(\text{int}), Q_3\rangle$$
$$xs : L(\text{int}), ys : L(\text{int}), zs : L(\text{int})\,;\, Q_3 \mid\!\!\overset{M^*}{\longrightarrow} \langle ys, zs\rangle : \langle L(\text{int}), Q_4\rangle$$

Entsprechende lineare Formeln werden ebenso gebaut:

$$\Phi_{Q_1}(xs) \geq w_1 + \Phi_{Q_2}(ys)$$
$$\Phi_{Q_2}(ys) \geq w_2 + \Phi_{Q_3}(zs)$$
$$\Phi_{Q_3}(zs) \geq w_3 + \Phi_{Q_4}(H'(\ell))$$
$$w_1 = w_2$$
$$w_3 = 0$$

Hier ist $w_1 = w_2$, weil die zwei Aufrufe $\text{mult}(2, xs)$ und $\text{mult}(3, xs)$ die gleiche Arbeit w haben. Außerdem ist $w_3 = 0$, weil keine Multiplkation im Paarliteral vorkommt. Nach Vereinfachung bleibt eine einzige Formel übrig.

$$\Phi_{Q_1}(xs) \geq w + w + \Phi_{Q_4}(H'(\ell))$$

Eine Belegung dieser Formel ist Folgende:

$$\Phi_{Q_1}(xs) = 2 \cdot |xs| \qquad \text{und} \qquad \Phi_{Q_4}(H'(\ell)) = 0$$

4.3 Parallele Auswertung

Bei der parallelen Auswertung werden Arbeit und Tiefe unterschiedlich berechnet. Die Arbeit wird genau wie bei der sequentiellen Auswertung analysiert, während die Tiefe eine neue Strategie benötigt.

Die Grundidee zur Analyse der Tiefe einer parallelen Auswertung ist, jedem Ast zu erlauben, das ganze Potential zu benutzen. Beispielsweise wird das folgende Programm betrachtet:

```
mult2par(xs) = par ys = mult(2, xs)
               and zs = mult(3, xs) in
               (ys, zs)
```

Die folgenden Typurteile werden abgeleitet:

$$xs : L(\text{int})\,;\, Q_1 \mid\!\!\overset{M^*}{\longrightarrow} \text{mult}(2, xs) : \langle L(\text{int}), Q_2\rangle$$
$$xs : L(\text{int})\,;\, Q_1 \mid\!\!\overset{M^*}{\longrightarrow} \text{mult}(3, xs) : \langle L(\text{int}), Q_3\rangle$$
$$xs : L(\text{int}), ys : L(\text{int}), zs : L(\text{int})\,;\, Q_? \mid\!\!\overset{M^*}{\longrightarrow} \langle ys, zs\rangle : \langle L(\text{int}), Q_4\rangle$$

Entsprechende lineare Formeln werden ebenso gebaut:

$$\Phi_{Q_1}(xs) \geq d_1 + \Phi_{Q_2}(ys)$$
$$\Phi_{Q_1}(xs) \geq d_2 + \Phi_{Q_3}(zs)$$
$$\Phi_{Q_?}(?) \geq d_3 + \Phi_{Q_4}(H'(\ell))$$
$$d_1 = d_2$$
$$d_3 = 0$$

Die Frage ist, welches Potential $Q_?$ entspricht. Es kann nicht Q_1 sein, denn sonst wäre die Analyse der zwei parallelen Äste umsonst durchgeführt worden. Zudem können es weder Q_2 noch Q_3 sein, sonst wäre eine der zwei Analysen umsonst durchgeführt worden.

Eine verlockende Antwort wäre, $Q_?$ als die Summe von Q_1 und Q_2 zu definieren. Es lohnt sich, ein paar Belegungen auszuprobieren:

$\Phi_{Q_1}(xs) = \lvert xs\rvert$	$\Phi_{Q_2}(ys) = 0$	$\Phi_{Q_3}(xs) = 0$	$\Phi_{Q_4}(H'(\ell)) = 0$
$\Phi_{Q_1}(xs) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_2}(ys) = \lvert xs\rvert$	$\Phi_{Q_3}(xs) = \lvert xs\rvert$	$\Phi_{Q_4}(H'(\ell)) = 2 \cdot \lvert xs\rvert$
$\Phi_{Q_1}(xs) = 3 \cdot \lvert xs\rvert$	$\Phi_{Q_2}(ys) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_3}(xs) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_4}(H'(\ell)) = 4 \cdot \lvert xs\rvert$

Es ist nun möglich eine negative Tiefe zu haben: $\Phi_{Q_1}(xs) - \Phi_{Q_4}(H'(\ell)) = 3 \cdot |xs| - 4 \cdot |xs| = -|xs|$. Diese Funktion wird also in einer negativen Zeit ausgewertet: wir haben eine Zeitmaschine erfunden!

Ein Potential ist allerdings eine positive rationale Zahl und deshalb kann diese Strategie keine Lösung sein.

Es gibt nur drei mögliche Verhaltensweisen bzgl. der Kosten, wie zwei parallele Ausdrücke ausgewertet werden können: entweder sind die Kosten des Ersten höher, die Kosten des Zweiten oder sie sind gleich. Da nur eine Obergrenze der Kosten gesucht wird, kann o. B. d. A angenommen werden, sodass die niedrigsten Kosten Null sind. Wenn die Kosten der zwei Äste gleich sind, kann willkürlich einer ausgewählt werden, dessen Kosten Null werden sollen.

Die Analyse wird also zweimal durchgeführt: jede Runde wird die Kostenfreimetrik benutzt, sodass nur einer der parallelen Ausdrücke Kosten hat. Sei ein Ausdruck „par $x_1 = e_1$ and $x_2 = e_2$ in e“. In der ersten Runde werden e_1, e_2 und e sequentiell analysiert, wobei die Kostenfreimetrik M^{cf} für die Analyse von e_1 verwendet wird. In der zweiten Runde werden e_1, e_2 und e nochmal sequentiell analysiert und dieses Mal wird e_2 bzgl. der Kostenfreimetrik M^{cf} analysiert.

Die folgenden Typurteile werden bei der ersten Runde abgeleitet:

$$\emptyset \,;\, Q_1 \left|\overline{}^{M^{\text{cf}}}\right. e_1 : \langle A, Q_1\rangle$$
$$\emptyset \,;\, Q_1 \left|\overline{}^{M^{*}}\right. e_2 : \langle B, Q_2\rangle$$
$$x_1 : A, x_2 : B \,;\, Q_2 \left|\overline{}^{M^{*}}\right. e : \langle C, Q_4\rangle$$

Die folgenden Typurteile werden bei der zweiten Runde abgeleitet:

$$\emptyset\,;\, Q_1 \mid\!\!\overset{M^*}{\vdash} e_1 : \langle A, Q_3\rangle$$
$$\emptyset\,;\, Q_3 \mid\!\!\overset{M^{\mathrm{cf}}}{\vdash} e_2 : \langle B, Q_3\rangle$$
$$x_1 : A, x_2 : B\,;\, Q_3 \mid\!\!\overset{M^*}{\vdash} e : \langle C, Q_4\rangle$$

Die linearen Regeln sind Folgende:

$$\Phi_{Q_1} \geq d_2 + \Phi_{Q_2}$$
$$\Phi_{Q_2} \geq d_3 + \Phi_{Q_4}$$
$$\Phi_{Q_1} \geq d_1 + \Phi_{Q_3}$$
$$\Phi_{Q_3} \geq d_3 + \Phi_{Q_4}$$

Nach Vereinfachung:

$$\Phi_{Q_1} \geq \max(d_1, d_2) + d_3 + \Phi_{Q_4}$$

Es ist nun möglich, eine Belegung für das Potential von `mult2par` zu geben:

$\Phi_{Q_1}(xs) = \lvert xs\rvert$	$\Phi_{Q_2}(ys) = 0$	$\Phi_{Q_3}(xs) = 0$	$\Phi_{Q_4}(H'(\ell)) = 0$
$\Phi_{Q_1}(xs) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_2}(ys) = \lvert xs\rvert$	$\Phi_{Q_3}(xs) = \lvert xs\rvert$	$\Phi_{Q_4}(H'(\ell)) = \lvert xs\rvert$
$\Phi_{Q_1}(xs) = 3 \cdot \lvert xs\rvert$	$\Phi_{Q_2}(ys) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_3}(xs) = 2 \cdot \lvert xs\rvert$	$\Phi_{Q_4}(H'(\ell)) = 2 \cdot \lvert xs\rvert$

5 Ressourcenpolynome

5.1 Binomialkoeffizienten

Der Bynomialkoeffizient $\binom{n}{k}$ wird „n über k“ gelesen und hat die folgende Definition:

$$\frac{n!}{k!\,(n-k)!} \tag{1}$$

Er wird als Grundbaustein der Polynome benutzt. Intuitiv kann angenommen werden, dass $\binom{n}{k} \in \mathcal{O}(n^k)$. Um sich davon zu überzeugen, kann die Defi-

nition 1 für einige k instanziiert werden.

$$\begin{aligned}
\binom{n}{0} &= 1 && \in \mathcal{O}(1) \\
\binom{n}{1} &= n && \in \mathcal{O}(n) \\
\binom{n}{2} &= \frac{1}{2}\,n^2 - \frac{1}{2}\,n && \in \mathcal{O}(n^2) \\
\binom{n}{3} &= \frac{1}{6}\,n^3 - \frac{1}{2}\,n^2 + \frac{1}{3}\,n && \in \mathcal{O}(n^3) \\
\binom{n}{4} &= \frac{1}{24}\,n^4 - \frac{1}{4}\,n^3 + \frac{11}{24}\,n^2 - \frac{1}{4}\,n && \in \mathcal{O}(n^4)
\end{aligned}$$

5.2 Grundpolynome

Für jeden Datentyp gibt es eine Menge P von Grundpolynomen $p : [\![A]\!] \to \mathbb{N}$.

$$\begin{aligned}
P(\text{int}) &= \{\, a \mapsto 1 \,\} \\
P(A_1 * A_2) &= \{\, \langle a_1, a_2 \rangle \mapsto p_1(a_1) \cdot p_2(a_2) \mid p_1 \in P(A_1),\, p_2 \in P(A_2) \,\} \\
P(L(A)) &= \{\, \Sigma\Pi[p_1, \ldots, p_k] \mid k \in \mathbb{N},\, p_i \in P(A) \,\}
\end{aligned}$$

Der letzte Teilsatzt enthält darüber hinaus

$$\Sigma\Pi[p_1, \ldots, p_k]([a_1, \ldots, a_n]) = \sum_{1 \le j_1 < \cdots < j_k \le n} \prod_{1 \le i \le k} p_i(a_{j_i})$$

Hierbei ist es wichtig zu merken, dass $\ell \mapsto \binom{|\ell|}{k}$ für $k \in \mathbb{N}$ in allen $P(L(A))$ ist.

5.3 Ressourcenpolynome

Ein Ressourcenpolynom $p : [\![A]\!] \to \mathbb{Q}_0^+$ ist eine Summe von mit beliebigen Koeffizienten $q_i \in \mathbb{Q}_0^+$ multiplizierten Grundpolynomen $p_i \in P(A)$ des Datentyps:

$$p(a) = \sum_{i \in \mathbb{N}} q_i \cdot p_i(a)$$

5.4 Kommentierte Typen

5.4.1 Indexe

Um jedem Grundpolynom einen Namen zu geben, wird zuerst die Index-Menge $\mathcal{I}(\mathcal{A})$ definiert, welche die Struktur eines Typs beschreibt. Die Grundidee ist, jeden atomischen Typ (in dieser vereinfachten Sprache gibt es nur Zahlen) durch den Einheit-Index $\circ$ zu ersetzen.

$$\begin{aligned}
\mathcal{I}(\text{int}) &= \{\circ\} \\
\mathcal{I}(A_1 * A_2) &= \{\langle i_1, i_2 \rangle \mid i_1 \in \mathcal{I}(A_1),\, i_2 \in \mathcal{I}(A_2)\} \\
\mathcal{I}(L(A)) &= \{[i_1, \ldots, i_k] \mid k \in \mathbb{N},\, i_j \in \mathcal{I}(A)\}
\end{aligned} \tag{2}$$

Als Instanziierungsbeispiel der Definition 2 kann Folgendes gelten:

$$\begin{aligned}
\mathcal{I}(\text{int}) &= \{\circ\} \\
\mathcal{I}(L(\text{int})) &= \{[], [\circ], [\circ, \circ], [\circ, \circ, \circ], \dots\} \\
\mathcal{I}(\text{int} * \text{int}) &= \{\langle \circ, \circ \rangle\} \\
\mathcal{I}(\text{int} * L(\text{int})) &= \{\langle \circ, [] \rangle, \langle \circ, [\circ] \rangle, \langle \circ, [\circ, \circ] \rangle, \langle \circ, [\circ, \circ, \circ] \rangle, \dots\} \\
\mathcal{I}(L(\text{int}) * L(\text{int})) &= \{\langle [], [] \rangle, \langle [], [\circ] \rangle, \langle [\circ], [] \rangle, \langle [\circ], [\circ] \rangle, \langle [], [\circ, \circ] \rangle, \langle [\circ], [\circ, \circ] \rangle, \dots\} \\
\mathcal{I}(L(L(\text{int}))) &= \{\{[]\} \cup \{[i] \mid i \in \mathcal{I}(L(\text{int})\} \cup \{[i_1, i_2] \mid i_1, i_2 \in \mathcal{I}(L(\text{int})\} \cup \cdots\}
\end{aligned}$$

Für jeden Typ A wird dann jedem Index $i \in \mathcal{I}(\mathcal{A})$ ein Grundpolynom $p_i \in P(A)$ zugeordnet.

Wenn $A = \text{int}$, dann

$$p_\circ(v) = 1$$

Wenn $A = A_1 * A_2$, dann

$$p_{\langle i_1, i_2 \rangle}(\langle v_1, v_2 \rangle) = p_{i_1}(v_1) \cdot p_{i_2}(v_2)$$

Wenn $A = L(A)$, dann

$$p_{[i_1, \dots, i_m]}(v) = \Sigma\Pi[p_{i_1}, \dots, p_{i_m}](v)$$

Mit Hilfe der Indexe ist es jetzt möglich, jedes Grundpolynom präzise zu nennen. Formell ausgedrückt:

$$\forall A. \forall p \in P(A). \exists i \in \mathcal{I}(A). p_i = p$$

Da die Analyse nur mit beliebig großen aber endlichen Polynomen arbeiten kann, wird ein Weg benötigt, die Grundpolynome zu beschränken. Dafür wird der Grad $\deg(i)$ eines Indexes $i \in \mathcal{I}(A)$ wie folgt definiert:

$$\begin{aligned}
\deg(\circ) &= 0 \\
\deg(\langle i_1 * i_2 \rangle) &= \deg(i_1) + \deg(i_2) \\
\deg([i_1, \dots, i_n]) &= k + \deg(i_1) + \cdots + \deg(i_k)
\end{aligned} \tag{3}$$

Es ist nun möglich, mit einer endlichen Menge von Grundpolynomen $p_i \in P(A)$ zu arbeiten, wenn die Indexe i aus der Menge $\mathcal{I}_k(A) = \{i \in \mathcal{I}(A) \mid \deg(i) \leq k\}$ stammen.

5.4.2 Potential eines Typs

Eine Typannotation Q_A eines Datentyps A ist ein Tupel von Koeffizienten $q_i \in \mathbb{Q}_0^+$, bei denen $i \in \mathcal{I}_k(A)$:

$$Q_A = \langle q_{i_1}, q_{i_2}, \dots, q_{i_k} \rangle \tag{4}$$

Seien $k = 3$ und $A = L(\text{int})$, dann ist $Q_{L(\text{int})} = \langle q_{[]}, q_{[\circ]}, q_{[\circ,\circ]}, q_{[\circ,\circ,\circ]} \rangle$.

Ein annotierter Datentyp ist ein Paar $\langle A, Q_A \rangle$.

Wenn H ein Haldenspeicher ist, dann wird das Potential Φ_H des Wertes $H(\ell)$ eines Datentyps A wird folgendenmaßen definiert:

$$\Phi_H(\ell : \langle A, Q_A \rangle) = \sum_{i \in \mathcal{I}_k(A)} q_i \cdot p_i(H(\ell)) \tag{5}$$

Sei zum Beispiel $k = 3$, $A = L(\text{int})$ und $Q_A = \langle 2, \frac{1}{2}, 0, 8 \rangle$, dann ist das Potential $\Phi_H(\ell : \langle A, Q_A \rangle) = 2 + \frac{1}{2}n + 8\binom{n}{3}$, bei dem $n = |a|$.

5.4.3 Potential eines Kontexts

Im Typsystem werden nicht nur Datentypen, sondern auch Kontexte typisiert. Dafür müssen die Definitionen 2 der Indexmenge, 3 des Grades, 4 der Typannotation und 5 des Potentials erweitert werden.

Sei der Kontext $\Gamma = \{x_1 : A_1, \ldots, x_n : A_n\}$, so wird die Indexmenge $\mathcal{I}(\Gamma)$ wie wie folgt definiert:

$$\mathcal{I}(\Gamma) = \{\langle i_1, \ldots, i_n \rangle \mid i_1 \in \mathcal{I}(A_1), \ldots, i_2 \in \mathcal{I}(A_n)\}$$

Der Grad $\deg(i)$ eines Indexes $i \in \mathcal{I}(\Gamma)$ wird wie folgt definiert:

$$\deg(\langle i_1, \ldots, i_n \rangle) = \deg(i_1) + \cdots + \deg(i_n)$$

Die Definition $\mathcal{I}_k(\Gamma) = \{i \in \mathcal{I}(\Gamma) \mid \deg(i) \leq k\}$ ist gleichartig wie die von $\mathcal{I}_k(A)$.

Eine Kontextannotation Q eines Kontexts Γ ist ein Tupel von Koeffizienten $q_i \in \mathbb{Q}_0^+$, bei denen $i \in \mathcal{I}_k(\Gamma)$:

$$Q = \langle q_{i_1}, q_{i_2}, \ldots, q_{i_n} \rangle$$

Ein annotierter Kontext wird $\Gamma; Q$ geschrieben.

Seien V ein Stapelspeicher und H ein Haldenspeicher, dann wird das Potential $\Phi_{V,H}$ eines Kontexts Γ wie folgt definiert:

$$\Phi_{V,H}(\Gamma, Q) = \sum_{\langle i_1, \ldots, i_n \rangle \in \mathcal{I}_k(\Gamma)} q_{\vec{i}} \prod_{j=1}^{n} p_{i_j}(x_j)$$

6 Annotiertes Typsystem

6.1 Typurteile

Ein Typurteil hat die Form

$$\Sigma\,;\,\Gamma\,;\{Q_1, \ldots, Q_n\} \mid\!\!\!\overset{M}{-}\, e : \langle A, Q' \rangle,$$

bei der Σ eine Menge von annotierten Funktionstypen ist, $n \in \{1, 2\}$, M eine Kostenmetrik ist, $\Gamma; Q_i$ ein annotierter Kontext für alle $i = \{1, \ldots, n\}$ ist, e ein Ausdruck ist und $\langle A, Q' \rangle$ ein annotierter Datentyp ist.

Die Bedeutung ist die Folgende: wenn für alle $i \in \{1, \ldots, n\}$ mehr als $\Phi(\Gamma; Q_i)$ Ressourcen zur Verfügung stehen, dann reichen sie, um die Kosten der Auswertung von e in Bezug auf die Kostenmetrik M zu decken. Außerdem bleiben $\Phi(v : \langle A, Q' \rangle)$ Ressourcen übrig.

6.2 Notation

Seien Q eine Kontextannotation für Γ_1, Γ_2 und $j \in \mathcal{I}(\Gamma_2)$. Die Projektion $\pi_j^{\Gamma_1}(Q)$ von Q auf Γ_1 ist die Kontextannotation Q', bei der $q'_i = q_{\langle i,j \rangle}$.

Ein wichtiges Konzept im Typsystem ist die *addierende Verschiebung*, die benutzt wird, um die Verteilung des Potentials eines Kontexts bei einer Fallunterscheidung oder bei der Nutzung eines Konstruktors zu ändern. Sei $\Gamma, y : L(A)$ ein Kontext und $\langle q_{i_1}, \ldots, q_{i_k} \rangle$ eine Kontextannotation für $i_j \in \mathcal{I}_k(\Gamma, y : L(A))$. Die *addierende Verschiebung für Listen* $\lhd_L(Q)$ ist eine Kontextannotation $\langle q'_{i_1}, \ldots, q'_{i_k} \rangle$ für $i_j \in \mathcal{I}_k(\Gamma, x : A, xs : L(A))$ und für einen Kontext $\Gamma, x : A, xs : L(A))$, welche so definiert ist:

$$q'_{\langle i,j,\ell \rangle} = \begin{cases} q_{\langle i,j::\ell \rangle} + q_{\langle i,\ell \rangle} & j = 0 \\ q_{\langle i,j::\ell \rangle} & j \neq 0 \end{cases}$$

6.3 Typisierungsregeln

Die Typisierungsregeln T:Const, T:Pair, T:Nil und T:Cons sagen, dass die Auswertung einer Zahl, eines Paares bzw. einer Liste konstante Kosten hat.

$$\frac{n \in \mathbb{Z} \qquad Q = Q' + M^{\text{const}}}{\Sigma\,;\,\cdot\,;\{Q\} \overset{M}{\vdash} n : \langle \text{int}, Q' \rangle}\,\text{T:Const}$$

$$\frac{Q = Q' + M^{\text{pair}}}{\Sigma\,;\,x_1 : A_1, x_2 : A_2\,;\{Q\} \overset{M}{\vdash} \langle x_1, x_2 \rangle : \langle A, Q' \rangle}\,\text{T:Pair}$$

$$\frac{Q = Q' + M^{\text{nil}}}{\Sigma\,;\,\cdot\,;\{Q\} \overset{M}{\vdash} \text{nil} : \langle L(A), Q' \rangle}\,\text{T:Nil}$$

$$\frac{Q = \lhd_L(Q') + M^{\text{cons}}}{\Sigma\,;\,x : A, xs : L(A)\,;\{Q\} \overset{M}{\vdash} \text{cons}(x, xs) : \langle L(A), Q' \rangle}\,\text{T:Cons}$$

Die Typisierungsregel T:Var sagt, dass eine Variable in dem Kontext vorkommen muss und dass ihre Auswertung die konstanten Kosten M^{var} hat.

$$\frac{Q = Q' + M^{\text{var}}}{\Sigma\,;\,x : A\,;\{Q\} \overset{M}{\vdash} x : \langle A, Q' \rangle}\,\text{T:Var}$$

Die Typisierungsregel T:App beschreibt die Kosten eines Funktionsaufrufs, nämlich die Summe der Auswertungskosten und der konstanten Kosten M^{app} ist.

$$\frac{Q = P + M^{\text{app}} \qquad \langle A, P \rangle \to \langle A', Q' \rangle \in \Sigma(f)}{\Sigma\,;\,\cdot\,;\{Q\} \overset{M}{\vdash} n : \langle \text{int}, Q' \rangle}\,\text{T:App}$$

Die Typisierungsregel T:MatP besagt die Auswertung der Fallunterscheidung eines Paares, bei der die Kosten die Summe der Kosten der Auswertung des Unterausdrucks und der konstanten Kosten M^{matP} sind.

$$\frac{Q = P + M^{\mathrm{matP}} \qquad \Sigma\,;\,\Gamma, x_1 : A_1, x_2 : A_2\,;\,\{P\} \mid\!\!\overset{M}{-} e : \langle B, Q'\rangle}{\Sigma\,;\,p : A_1 * A_2\,;\,\{Q\} \mid\!\!\overset{M}{-} \text{match } x \text{ with } \langle x_1, x_2\rangle \Rightarrow e : \langle B, Q'\rangle}\ \text{T:MatP}$$

Die Typisierungsregel T:MatL beschreibt die Auswertung der Fallunterscheidung einer Liste, wobei das Potential Q die Kosten des Falls sowohl der leeren Liste als auch der nicht-leeren Liste decken kann. In beiden Fällen bestehen die Kosten aus der Summe der Auswertungskosten des Unterausdrucks und der konstanten Kosten. Die konstanten Kosten sind M^{matN} für die leere Liste und M^{matL} für die nicht-leere Liste.

$$\frac{\begin{array}{ll}\pi_0^\Gamma(Q) = R + M^{\mathrm{matN}} & \Sigma\,;\,\Gamma\,;\,\{R\} \mid\!\!\overset{M}{-} e_1 : \langle B, Q'\rangle \\ \lhd_L(Q) = P + M^{\mathrm{matL}} & \Sigma\,;\,\Gamma, x_1 : A, x_2 : L(A)\,;\,\{P\} \mid\!\!\overset{M}{-} e_2 : \langle B, Q'\rangle\end{array}}{\Sigma\,;\,x : L(A)\,;\,\{Q\} \mid\!\!\overset{M}{-} \text{match } x \text{ with nil} \Rightarrow e_1 \mid \mathrm{cons}(x_1, x_2) \Rightarrow e_2 : \langle B, Q'\rangle}\ \text{T:MatL}$$

Die Typisierungsregel T:Let stellt die sequentielle Auswertung zweier Ausdrücke dar, bei denen die Kosten die Summe der Auswertung der Ausdrücke und der konstanten Kosten M^{let} sind.

$$\frac{Q = P + M^{\mathrm{let}} \qquad \Sigma\,;\,\Gamma_1, \Gamma_2\,;\,\{P\} \mid\!\!\overset{M}{-} e_1 \rightsquigarrow \Gamma_2, x : A\,;\,P' \qquad \Sigma\,;\,\Gamma_2, x : A\,;\,\{P'\} \mid\!\!\overset{M}{-} e_2 : \langle B, Q'\rangle}{\Sigma\,;\,\Gamma_1, \Gamma_2\,;\,\{Q\} \mid\!\!\overset{M}{-} \text{let } x = e_1 \text{ in } e_2 : \langle B, Q'\rangle}\ \text{T:Let}$$

Die Typisierungsregel T:Par besagt die parallele Auswertung zweier Ausdrücke, wobei die zwei Ergebnisse in der Auswertung eines dritten Ausdrucks zur Verfügung stehen.

$$\frac{\begin{array}{cc}Q = Q' + M^{\mathrm{par}} & P = P' + M^{\mathrm{par}} \\ \Sigma\,;\,\Delta, \Gamma_1, \Gamma_2\,;\,\{Q'\} \mid\!\!\overset{M}{-} e_1 \rightsquigarrow \Delta, \Gamma_2, x_1 : A_1\,;\,Q'' & \Sigma\,;\,\Delta, \Gamma_1, \Gamma_2\,;\,\{P'\} \mid\!\!\overset{\mathrm{cf}}{-} e_1 \rightsquigarrow \Delta, \Gamma_2, x_1 : A_1\,;\,P'' \\ \Sigma\,;\,\Delta, \Gamma_2, x_1 : A_1\,;\,\{Q''\} \mid\!\!\overset{\mathrm{cf}}{-} e_2 \rightsquigarrow \Delta, x_1 : A_1, x_2 : A_2\,;\,R & \Sigma\,;\,\Delta, \Gamma_2, x_1 : A_1\,;\,\{P''\} \mid\!\!\overset{M}{-} e_2 \rightsquigarrow \Delta, x_1 : A_1, x_2 : A_2\,;\,R \\ \multicolumn{2}{c}{\Sigma\,;\,\Delta, x_1 : A_1, x_2 : A_2\,;\,\{R\} \mid\!\!\overset{M}{-} e : \langle B, R'\rangle}\end{array}}{\Sigma\,;\,\Delta, \Gamma_1, \Gamma_2, \Delta\,;\,\{Q, P\} \mid\!\!\overset{M}{-} \text{par } x_1 = e_1 \text{ and } x_2 = e_2 \text{ in } e : \langle B, R'\rangle}\ \text{T:Par}$$

Eine Variable in einem Kontext kann nur einmal benutzt werden. Wenn ein Programm die selbe Variable mehrere Mal zugreifen will, muss diese kopiert werden. Die Typisierungsregel T:Share beschreibt die Teilung des Potentials einer Variable in Zwei. Sei $\Gamma, x_1 : A, x_2 : A; Q$ ein annotierter Kontext, bei dem x_1 und x_2 zwei Auftritte von der selben Variable sind, dann ist $\curlyvee\, Q$ eine neue Kontextannotation, sodass $\Phi(\Gamma, x_1 : A, x_2 : A; Q) = \Phi(\Gamma, x : A; \curlyvee\, Q)$.

$$\frac{\Sigma\,;\,\Gamma, x_1 : A, x_2 : A\,;\,\{P_1, \ldots, P_m\} \mid\!\!\overset{M}{-} e : \langle B, Q'\rangle \qquad \forall i\, \exists j.\, Q_j = \curlyvee\ P_i}{\Sigma\,;\,\Gamma, x : A\,;\,\{Q_1, \ldots, Q_n\} \mid\!\!\overset{M}{-} e[x/x_1, x/x_2] : \langle B, Q'\rangle}\ \text{T:Share}$$

Die Typisierungsregeln T:Weak-A, T:Weak-C und B:Bind werden hier nicht erklärt. Mehr Informationen darüber können im Quellenartikel gefunden werden.

$$\frac{\Sigma\,;\,\Gamma\,;\,\{P\} \mid\!\!\frac{M}{}\, e : \langle B, Q'\rangle \qquad Q \geq P + c \qquad Q' \leq P' + c}{\Sigma\,;\,\Gamma\,;\,\{Q\} \mid\!\!\frac{M}{}\, e : \langle B, Q'\rangle}\ \text{T:Weak-A}$$

$$\frac{\Sigma\,;\,\Gamma\,;\,\{P\} \mid\!\!\frac{M}{}\, e : \langle B, P'\rangle \qquad \forall i \in \mathcal{I}(\Gamma).\ p_i = q_{(i,0)}}{\Sigma\,;\,\Gamma, x : A\,;\,\{Q\} \mid\!\!\frac{M}{}\, e : \langle B, Q'\rangle}\ \text{T:Weak-C}$$

$$\frac{\begin{array}{c} i = \vec{0} \implies \Sigma\,;\,\Gamma\,;\,\{\pi_i^\Gamma(Q)\} \mid\!\!\frac{M}{}\, e : \langle A, \pi_i^{x:A}(Q')\rangle \\ \forall i \in \mathcal{I}(\Delta) \qquad i \neq \vec{0} \implies \Sigma_i\,;\,\Gamma\,;\,\{\pi_i^\Gamma(Q)\} \mid\!\!\frac{\text{cf}}{}\, e : \langle A, \pi_i^{x:A}(Q')\rangle \end{array}}{\Sigma\,;\,\Gamma, \Delta\,;\,\{Q\} \mid\!\!\frac{M}{}\, e \rightsquigarrow \Delta, x : A\,;\,Q'}\ \text{B:Bind}$$

6.4 Theoreme

Das wichtigste Theorem über das Typsystem ist das der Korrektheit. Dieser besagt, dass ein annotiertes Typurteil eines Ausdrucks e eine Schranke der Tiefe d aller sowohl terminierenden als auch nich-terminierenden Auswertungen von e in einer wohlgeformten Umgebung definiert.

Zudem ist die Eigenschaft einer terminierenden Auswertung noch stärker: die Differenz zwischen dem Anfangs- und Endpotential ist eine Obergrenze der Tiefe d der Auswertung.

Theorem 4 (Korrektheit)**.** *Wenn $H \vDash V : \Gamma$ und $\Sigma;\Gamma;\mathcal{Q} \vdash e : \langle B, Q'\rangle$, dann gibt es einen $Q \in \mathcal{Q}$, sodass Folgendes stimmt:*

1. $V, H \mid\!\!\frac{M}{}\, e \Downarrow (\ell, H') \mid (w, d) \implies d \leq \Phi_{V,H}(\Gamma; Q) - \Phi_{H'}(\ell : \langle B, Q'\rangle)$
2. $V, H \mid\!\!\frac{M}{}\, e \Downarrow \rho \mid (w, d) \implies d \leq \Phi_{V,H}(\Gamma; Q)$

Wenn die Kostenmetrik einfach ist, d. h. alle Konstante Eins sind, dann folgt aus dem Korrektheit-Theorem, dass eine Obergrenze der Tiefe die Terminierung des Auswertung beweist.

Korollarium 2. *Sei M eine einfache Kostenmetrik. Wenn $H \vDash V : \Gamma$ und $\Sigma;\Gamma;Q \vdash e : \langle A, Q'\rangle$, dann gibt es $w \in \mathbb{N}$ und $d \leq \Phi_{V,H}(\Gamma; Q)$, sodass $V, H \mid\!\!\frac{M}{}\, e \Downarrow (\ell, H') \mid (w, d)$ für manche ℓ und H'.*

7 Schluss

Die hier dargestellte Analyse benutzt ein annotiertes Typsystem, um Oberschranken an den Ressourcenverbrauch von parallelen, funktionalen Programmen zu bestimmen. Die Schranken sind multivariate Polynome, welche die Klasse der sogenanten effizienten Algorithmen enthält.

Literatur

[HAH12] Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Resource Aware ML. In *24rd International Conference on Computer Aided Verification (CAV'12)*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer, 2012.

[Har16] Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2016.

[HS15] Jan Hoffmann and Zhong Shao. Automatic static cost analysis for parallel programs. In *Proceedings of the 24th European Symposium on Programming on Programming Languages and Systems - Volume 9032*, pages 132–157, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[SF92] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, LFP '92, pages 288–298, New York, NY, USA, 1992. ACM.