technische universität
dortmund

Proseminarausarbeitung

Operationelle Semantik

Abdulrahman Alrefai

26. August 2019

im Rahmen des Seminars

Software Verification

von Prof. Dr. Falk Howar

Sommersemester 2019

Basierend auf:
Glynn Winskel. The formal semantics of programming languages: an introduction, *MIT press*

Fakultät für Informatik
Lehrstuhl 14 – Software Engineering
TU Dortmund

Inhaltsverzeichnis

Alle `todo`-Befehle entfernen

1 Einführung

Die Bedeutung der präzisen Semantik für Programmier- und Spezifikationssprachen wurde seit den sechziger Jahren mit der Entwicklung der ersten höheren Programmiersprachen anerkannt. Die Verwendung der operativen Semantik, d. h. einer Semantik, die explizit beschreibt, wie Programme schrittweise durchgeführt werden, und die möglichen Zustandstransformationen durchführen. Beim Erlernen einer neuen Programmiersprache wird normalerweise nur die Syntax angeschaut, um zu wissen, wie bestimmte Sprachkonstrukte geschrieben werden. Syntax behandelt jedoch nur korrekt gebildete Sätze. Dies bedeutet, dass der Programmierer verloren geht, wenn es notwendig wird, zu überprüfen, ob das angegebene Programm tatsächlich den beabsichtigten Vorgang abläuft. Es ist jedem Programmierer passiert, trotz eines syntaktisch korrekten Programm falsche Ergebnisse zurückgeliefert werden. Der Grund dafür ist, dass das Programm nicht unter allen Umständen semantisch korrekt ist. Programme sind normalerweise zu groß und zu komplex, um sie zu verstehen und zu überprüfen, daher ist es wichtig, dass man ein tiefes Verständnis über operationale Semantik hat. Die Semantik für eine Programmiersprache stellt abstrakte Entitäten bereit, die nur die relevanten Merkmale aller möglichen Ausführungen darstellen, und ignoriert Details, die für die Korrektheit von Implementierungen nicht relevant sind. Operationale Semantik erzeugt ein gekennzeichnetes Übergangssystem, dessen Zustände die geschlossenen Begriffe über einer algebraischen Signatur sind und deren Übergänge zwischen Zuständen induktiv aus einer Sammlung Übergangsregeln der Form erhalten werden: $\frac{permises}{Conclusion}$. (Wenn alle Prämissen stimmen, dann tun die Schlussfolgerungen)

2 Zustände

Beschreibung der Bedeutung einer Programmierung Sprache, indem spezifiziert wird, wie sie auf einem abstrakten Computer ausgeführt wird. Sie wird im Speicher durchgeführt und die verschiedenen verfügbaren Ressourcen verwenden. Der Inhält der speicher ändert sich :

- Variablen werden aktualisiert .
- neue Datenbank .

Außerdem kann der Inhalt einer variablen im Moment des Programmstarts einen großen Einfluss auf das Endergebnis haben. SO müssen wir den Status des Speichers während der Ausführung berücksichtigen. wir interessieren uns nur einen Teil vom Speicher(Werte der Variablen). Jetzt können wir Status so definieren: Es ist eine Funktion , die Variablen

Werte gibt. Die Menge der Zustände Σ besteht aus Funktionen σ : Loc$\rightarrow$ N .Somit ist σ (X) der Wert oder Inhalt der Stelle X im Zustand σ .

2.1 Operationelle Semantik

Bedeutung eines Programms von einem bestimmten Zustand (Startzustand) und Beobachtung, wie Status im Ende der Ausführung des Programms geändert wird (Endzustand).

3 Operationelle Semantik für Einfache Imperative Sparache

syntaktische Menge mit IMP

- n $\in$ Num (positive und negative ganze Zahlen mit Null)
- b $\in$ BExp (Boolean Expression)
- a $\in$ AExp (Arithmetic Expression)
- c $\in$ commands
- X $\in$ Loc (Location).

Formationsregeln: abstrakte Syntax , indem neue Expressions erstellt werden . Hier interessieren wir nur die Bedeutung von Programm ,also nicht wie können das aufschreiben.

- a::= n | a_0 + a_1 | a_0 - a_1 | a_0 * a_1.
- b::= true | false | a_0 == a_1 | $a_0 \geq a_1$ | $a_0 \leq a_1$.
- c ::= skip | X := a | c_0, c_1 | if b then c_0 else c_1 | while b do c .

Hinweise :

- wenn a_0 und a_1 arithmetische Ausdrücke sind, ist es auch die Resultat von a_0 - a_1 etc.
- (**::**=) sollte gelesen werden als "kann sein ".
- (|) sollte gelesen werden als (oder) .

Beispiel : Die Boolean Expression b kann als True sein , wenn die Bedienung erfüllt ist , sonst False .

3.1 Syntaktische Menge

Angenommen , dass wir zwei Elemente haben a_0 , $a_1 \in$ gleiche syntaktische Menge . wenn wir so $a_0 == a_1$ schreiben, heißt das a_0 und a_1 gleich Werte haben oder identisch sind . Die Arithmetic Expression 12 - 2 wird von ganze zahlen aufgebaut. aber sie sind nicht Syntaktisch identisch mit 10 , obwohl wir wissen , dass die Ergebnis identisch ist . so 12-2 ist nicht Syntaktisch identisch mit 10.

4 Auswertung

Arithmetic Expression wird in Integer ausgewertet und Boolean Expression wird in True oder False value ausgewertet. wie ein Programm verhaltet sich ?

- <1-> Status definieren
- <2-> Auswertung der Boolean Expression und Arithmetic Expression.
- <3-> Kommando führt aus

4.1 Auswertung von Arithmetic Expression

Die Auswertung eines arithmetischen Ausdrucks a in einem Zustand σ to n.

$$(\mathrm{a},\sigma) \rightarrow \mathrm{n}.$$

Auswertung eines Arithmetic Expression (a_0 * a_1):

- wir werten a_0 in n_0 aus
- wir werten a_1 in n_1 aus
- Jetzt können wir n_0 * n_1 und n als Ergebnis von a_0 * a_1 erhalten.

Also brauchen wir Regeln für die Auswertung.
Regeln :

1. Auswertung von Nummern : Alle Nummern sind schon ausgewertet.

$$(\mathrm{n},\sigma) \rightarrow \mathrm{n}.$$

....no premises braucht $\frac{}{(\mathrm{n},\sigma) \rightarrow \mathrm{n}}$

2. Auswertung von Location : wird in seiner Inhalt ausgewertet .

$$(\mathrm{X},\sigma) \rightarrow \sigma(X).$$

....no premises braucht $\frac{}{(\mathrm{X},\sigma) \rightarrow \sigma(X)}$

3. Auswertung von Sum : n ist die Eregebnis von n_0 + n_1

$$\frac{(a_0\ ,\ \sigma) \to n_0\ (a_1\ ,\ \sigma) \to n_1}{(a_0{+}a_1{,}\sigma) \to \mathrm{n}}$$

4. Auswertung von Subs:n ist die Eregebnis von n_0 - n_1

$$\frac{(a_0\ ,\ \sigma) \to n_0\ (a_1\ ,\ \sigma) \to n_1}{(a_0{-}a_1{,}\sigma) \to \mathrm{n}}$$

5. Auswertung von Prod: n ist die Eregebnis von n_0 * n_1

$$\frac{(a_0\ ,\ \sigma) \to n_0\ (a_1\ ,\ \sigma) \to n_1}{(a_0{*}a_1{,}\sigma) \to \mathrm{n}}$$

Hinweis: Regeln ohne premises : heißt Axiome.
Beispiel : ((Init+30)-(7+9),σ)→ 14

$$\frac{\dfrac{\overline{(\mathrm{Init},\sigma)\to 0\ 2} \quad \overline{(30,\sigma)\to 30\ 1}}{(\mathrm{Init}{+}30,\sigma)\to 30\ 3} \qquad \dfrac{\overline{(7,\sigma)\to 7\ 2} \quad \overline{(9,\sigma)\to 9\ 2}}{(7{+}9,\sigma)\to 16\ 3}}{((\mathrm{Init}{+}30){-}(7{+}9),\sigma)\to 14\ 4}$$

*Hinweis:Init ist eine location mit $\sigma(\mathrm{Init}) = 0$.

4.2 Theorem-Umgebung

Definition 1 (Gerade Zahlen)**.** *Eine ganze Zahl $z \in \mathbb{Z}$ heißt* gerade*, wenn sie durch 2 teilbar ist.*

Theorem 1 (Optionaler Titel)**.** *Es gibt voll viele gerade Zahlen.*

Beweis. 2 ist zum Beispiel eine. Oder 18, 0, -100 oder 42. □

Literatur

A. E. Brouwer und W. H. Haemers. *Spectra of graphs.* Springer, 2011.

A. Rajaraman und J. D. Ullman. *Mining of massive datasets.* Cambridge University Press, 2011.