Einführung in ASM: Grundlagen

Anpassungsfähige Spezifikationstechnik.

Modellierung in der richtigen Abstraktionsebene.

Natürliche und leicht verständliche Semantik.

Material: Siehe http://www.di.unipi.it/AsmBook/

Theoretische Fundierung: ASM Thesen

Abstrakte Zustandsmaschinen als Berechnungsmodelle

Turing Maschinen (RAM, part.rek. Fkt,..) dienen als Berechnungsmodell, d.h. präzisierung der berechenbaren Funktionen. Prinzipiell lässt sich jede algorithmische Lösung durch eine geeignete TM simulieren.

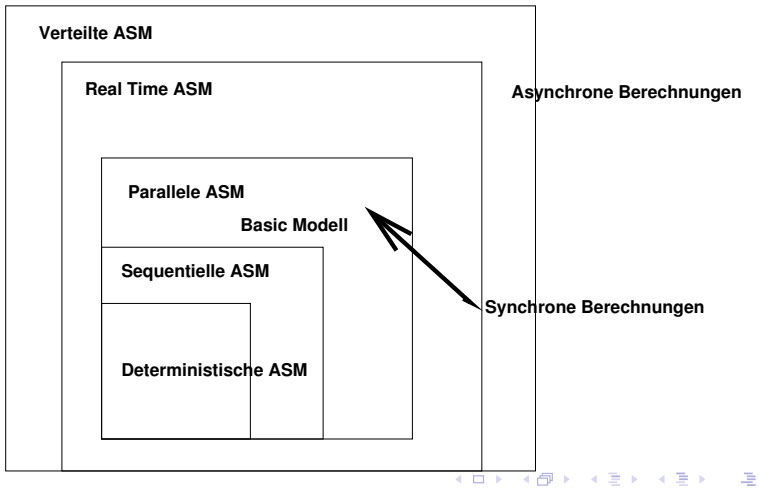
Problem: Simulation nicht einfach, da oft verschiedene Abstraktionsstufen und unterschiedliche Granularität.

Frage: Kann man TM so verallgemeinern, dass jeder Algorithmus, unabhängig von seiner Abstraktionsstufe, von einer verallgemeinerten Maschine natürlich und treu simuliert werden kann.
Wie sehen die Zustände und Instruktionen solcher Maschinen aus?

Einfach: If Bedingung Then Aktion

ASM These

ASM Thesis The concept of abstract state machine provides a universal computation model with the ability to simulate arbitrary algorithms on their natural levels of abstraction. Yuri Gurevich



Sequentielle ASM These

- Das Modell der sequentiellen ASM's ist universell für alle sequentiellen Algorithmen.
- Jeder sequentielle Algorithmus kann, unabhängig von seiner Abstraktionsstufe, schrittweise von einer sequentiellen ASM simuliert werden.

Um diese These zu untermauern benötigt man eine Präzisierung der sequentiellen Algorithmen und der sequentiellen ASM's.

$\rightsquigarrow$ Postulate für Sequentialität

Sequentialitäts-Postulate

- Sequentielle Zeit:
  Berechnungen sind linear geordnet.
- Abstrakte Zustände:
  Jede Art von statischer mathematischer Realität kann durch eine Struktur der Logik erster Stufe dargestellt werden. (Tarski)
- Bounded Exploration:
  Jeder Berechnungsschritt hängt nur von einer endlichen (vom Algorithmus abhängigen) beschränkten Zustandsinformation ab.

Y. Gurevich Sequential Abstract State Machines Capture Sequential Algorithms, ACM Transactions on Computational Logic, 1, 2000, 77-111.

Sequentielle Zeit

Sei A ein sequentieller Algorithmus. Zu A gehören:

- Eine Menge (Zustandsmenge) $S(A)$ von Zuständen von A.
- Eine Teilmenge $I(A)$ von $S(A)$ deren Elemente initiale Zustände von A heißen.
- Eine Abbildung $\tau_A : S(A) \to S(A)$, die Einschrittfunktion von A.

Ein Lauf (oder eine Berechnung) von A ist eine endliche oder unendliche Folge von Zustäden von A

$$X_0, X_1, X_2, \ldots$$

wobei X_0 ein initialer Zustand ist und $\tau_A(X_i) = X_{i+1}$ für alle i gilt.

Logische Zeit und nicht physikalische Zeit.

Abstrakte Zustände

Definition 3.1 *Algorithmen A and B sind äquivalent falls $S(A) = S(B)$, $I(A) = I(B)$ und $\tau_A = \tau_B$. Insbesondere haben äquivalente Algorithmen die gleichen Läufe.*

Sei A ein sequentieller Algorithmus:

- Zustände von A sind Strukturen erster Stufe.
- Alle Zustände von A haben gleiches Vokabular (Signatur).
- Die Einschrittfunktion verändert nicht die Basismenge $B(X)$ eines Zustands.
- $S(A)$ und $I(A)$ sind abgeschlossen gegen Isomorphismen und jeder Isomorphismus von Zustand X auf Zustand Y ist auch ein Isomorphismus von Zustand $\tau_A(X)$ auf $\tau_A(Y)$.

Übungen

Zustände: Signaturen, Interpretationen, Terme, Grundterme, Wert ...
Signaturen (Vokabular): Funktionen- und Relationennamen, Stelligkeit ($n \geq 0$)
Annahme: *true*, *false*, *undef* (Konstanten), *Boole* (einstellig) und $=$ sind in jeder Signatur enthalten. Die Interpretation von *true* ist stets verschieden von der von *false*, *undef*. Relationen werden als Funktionen mit Wert in den Interpretationen von *true*, *false* betrachtet. Einstellige Relationen werden als Teilmengen der Basismenge der Interpretation gesehen. Für ein Grundterm t im Vokabular sei $Val(t, X)$ sein Wert im Zustand X. Funktionen werden in dynamischen und statischen aufgeteilt, je nachdem ihre Interpretation sich bei Zustandsübergängen ändern oder nicht ändern können.
Übung: Modelliere Zustände von TM als abstrakten Zustand.
Modelliere Zustand des standard Euklidischen Algorithmus.

Bounded Exploration

- Parallelität Betrachte den folgenden Graph- Erreichbarkeits-Algorithmus der folgenden Schritt iteriert. Es wird angenommen, dass am Anfang nur ein Knoten die unäre Relation R erfüllt.

  do for all x, y with $Edge(x, y) \wedge R(x) \wedge \neg R(y)$ $\quad R(y) := true$

  In jedem Berechnungsschritt wird eine unbeschränkte Anzahl von lokalen Änderungen eines globalen Zustands durchgeführt.
- Unbeschränkte Schrittinformation
  Test auf isolierte Knoten in Graphen:

  if $\forall x \exists y\ Edge(x, y)$ then Output := false else Output := true

  In einem Schritt werden nur beschränkt viele lokale Veränderungen gemacht, es wird jedoch ein unbeschränkter Teil des Zustands in einem Schritt berücksichtigt.
  Wie erfasst man diese Eigenschaften ?$\rightsquigarrow$ Atomare Aktionen

Update Mengen

Betrachte die Struktur X als Speicher: Ist f ein j-stelliger Funktionsname und $\bar{a}$ ein j-Tupel von Basiselementen von X, dann heißt das Paar $(f, \bar{a})$ eine Lokation und $Content_X(f, \bar{a})$ ist der Wert der Interpretation von f für $\bar{a}$ in X.
Ist $(f, \bar{a})$ eine Lokation von X und b ein Element von X, so heißt $(f, \bar{a}, b)$ ein Update von X. Das Update ist trivial falls $b = Content_X(f, \bar{a})$. Um ein Update auszuführen wird der aktuelle Inhalt der Lokation durch b ersetzt. Eine Menge von Updates von X ist Konsistent falls es kein Paar von Updates mit gleicher Lokation und ungleiche Werte in der Menge gibt. Eine Menge Δ von Updates wird ausgeführt in dem man alle Updates in der Menge simultan ausführt falls die Menge konsistent ist, ansonsten wird nichts gemacht. Das Ergebnis wird mit $X + \Delta$ bezeichnet.

Lemma 3.1 *Sind X, Y Strukturen über die gleiche Signatur und mit gleicher Basismenge, dann gibt es eine eindeutige konsistente Menge Δ von nichttrivialen Updates von X mit $Y = X + \Delta$. Sei $\Delta \leftrightharpoons Y - X$.*

Update Mengen von Algorithmen, Erreichbare Elemente

Definition 3.2 *Sei X ein Zustand von Algorithmus A. Nach Definition haben X und $\tau_A(X)$ gleiche Signatur und Basismenge. Setze:*

$$\Delta(A, X) \leftrightharpoons \tau_A(X) - X \quad d.h.\ \tau_A(X) = X + \Delta(A, X)$$

Wie kann man Elemente der Basismenge eines Zustands in der Beschreibung des Algorithmus überhaupt ansprechen?
$\rightsquigarrow$ Über Grundterme der Signatur.

Definition 3.3 *Ein Element a einer Struktur X ist erreichbar wenn $a = Val(t, X)$ für ein Grundterm t im Vokabular von X.*
Eine Lokation $(f, \bar{a})$ von X ist erreichbar wenn jedes Element im Tupel $\bar{a}$ erreichbar ist.
Ein Update $(f, \bar{a}, b)$ von X ist erreichbar wenn $(f, \bar{a})$ und b erreichbar sind.

Bounded Exploration Postulat

Zwei Strukturen X und Y mit gleichem Vokabular Sig stimmen auf einer Menge T von Sig- Termen überein, falls $Val(t, X) = Val(t, Y)$ für alle $t \in T$. Das Vokabular (Signatur) eines Algorithmus ist das Vokabular seiner Zustände.

Sei A ein sequentieller Algorithmus.

- Es gibt eine endliche Menge T von Termen im Vokabular von A, so dass gilt $\Delta(A, X) = \Delta(A, Y)$ für Zustände X, Y von A die auf T übereinstimmen.

Intuition: Algorithmus A untersucht nur den Teil eines Zustands der erreichbar ist über die Termmenge T. Stimmen zwei Zustände auf dieser Termmenge überein, so sind die Update Mengen des Algorithmus für beide Zustände gleich.

Die Menge T ist ein bounded-exploration Zeuge für A.

Beispiel

Betrachte Algorithmus A:

if P(f) then f := S(f)

Zustände mit Interpretationen mit Basismenge $\mathbb{N}$, von P Teilmenge der natürlichen Zahlen, für S die Nachfolgerfunktion und f eine Konstante. Offenbar erfüllt A die Postulate Sequentielle Zeit und Abstrakte Zustände. Man könnte meinen, dass $T_0 = \{f, P(f), S(f)\}$ ein bounded-exploration Zeuge für A ist.
Sei X der kanonischer Zustand von A mit f = 0 und P(0) gelte.
Setze $a \leftrightharpoons Val(true, X)$ und $b \leftrightharpoons Val(false, X)$, so dass $Val(P(0), X) = Val(true, X) = a$.
Sei Y der Zustand den man aus X erhält durch Reinterpretation von *true* als b und *false* als a, d.h. $Val(true, Y) = b$ und $Val(false, Y) = a$. Der Wert von $P(0)$ wurde nicht verändert:
$Val(P(0), Y) = a$, also $P(0)$ gilt nicht in Y . Somit stimmen X, Y über T_0 überein aber $\Delta(A, X) \neq \emptyset = \Delta(A, Y)$.
Die Menge $T = T_0 \cup \{true\}$ ist ein bounded-exploration Zeuge für A.

Sequentielle Algorithmen

Definition 3.4 *Ein sequentieller Algorithmus ist ein Objekt A welches die drei Postulate erfüllt.*
Insbesondere hat A ein Vokabular und einen bounded-exploration Zeugen T. O.B.d.A ist T Subterm-Abgeschlossen und enthält true, false, undef. Die Terme aus T heißen kritisch und ihre Interpretationen in einem Zustand X heißen kritische Werte in X.

Lemma 3.2 *Ist $(f, a_1, ..., a_j, a_0)$ ein Update in $\Delta(A, X)$, dann sind alle Elemente $a_0, a_1, ..., a_j$ kritische Werte in X.*

Beweis: Übung.
Die Menge der kritischen Terme hängt nicht von X ab, somit gibt es eine feste obere Schranke für die Größe von $\Delta(A, X)$ und A verändert in jeden Schritt somit eine beschränkte Anzahl von Lokationen. Jedes einzelne Update in $\Delta(A, X)$ ist eine atomare Aktion von A. D.h. $\Delta(A, X)$ besteht aus einer beschränkten Menge atomarer Aktionen von A.

Update Regeln

Definition 3.5 *Eine Update Regel über die Signatur Sig hat die Form*

$$f(t_1, ..., t_j) := t_0$$

wobei f Funktion und die t_i Terme in Sig sind. Um die Regel in der Sig-Struktur X zu feuern berechne die Werte $a_i = Val(t_i, X)$ und führe Update $((f, a_1, ..., a_j), a_0)$ über X aus.
Parallele Update Regel über Sig: Seien R_i Update Regeln über Sig, dann

par
  R_1
  R_2
  . *Bezeichnung: Block (falls leer skip)*
  .
  .
  R_k
endpar *feuert durch gleichzeitiges feuern der R_i.*

Sequentielle ASM-Programme

Definition 3.6 ***Semantik von Updateregeln.*** *Ist R eine Update Regel $f(t_1, ..., t_j) := t_0$ und $a_i = Val(t_i, X)$ dann setze*

$$\Delta(R, X) \leftrightharpoons \{(f, (a_1, ..., a_j), a_0)\}$$

Ist R eine par-Update Regel mit Bestandteilen $R_1, ...R_k$ so setze

$$\Delta(R, X) \leftrightharpoons \Delta(R1, X) \cup \cdots \cup \Delta(Rk, X).$$

Folgerung 3.1 *Insbesondere gibt es für jeden Zustand X eine Regel R^X die nur kritische Terme verwendet mit* $\Delta(R^X, X) = \Delta(A, X)$.

Beachte: Stimmen X, Y auf den kritischen Termen überein, so gilt $\Delta(R^X, Y) = \Delta(A, Y)$. Sind X, Y Zustände und $\Delta(R^X, Z) = \Delta(A, Z)$ für Zustand Z der isomorph zu Y ist, dann auch $\Delta(R^X, Y) = \Delta(A, Y)$.
Betrachte Äquivalenzrelation $E_X(t1, t2) \leftrightharpoons Val(t1, X) = Val(t2, X)$ auf T. X, Y sind T-ähnlich, falls $E_X = E_Y \rightsquigarrow \Delta(R^X, Y) = \Delta(A, Y)$.Übung

Sequentielle ASM-Programme

Definition 3.7 *Ist φ ein Boolescher Term über Sig und R_1, R_2 Regeln über Sig dann ist*

if φ then R_1
else $R2$
endif *eine Regel*

Zur Feuerung der Regel im Zustand X werte φ in X aus. Ist das Ergebnis true, so $\Delta(R, X) = \Delta(R_1, X)$ sonst $\Delta(R, X) = \Delta(R_2, X)$.

Definition 3.8 *Ein sequentielles ASM Programm Π über die Signatur Sig ist eine Regel über Sig. Dementsprechend ist $\Delta(\Pi, X)$ wohldefiniert für jede Sig-Struktur X.*

Lemma 3.3 ***Hauptergebnis:*** *Für jeden sequentiellen Algorithmus A über Sig gibt es ein sequentielles ASM-programm Π über Sig mit $\Delta(\Pi, X) = \Delta(A, X)$ für alle Zustände X von A.*