Spesifikasie van bewerkings

Z spesifikasies is veral nuttig om *datastrukture* te spesifiseer, maar dis ook dikwels nodig om ingewikkelde *bewerkings* in meer detail te spesifiseer.

Nuttige notasies:

1. Grammatikas (of EBNF)
2. Gebruik van proposisie logika vir spesifikasie
3. Gebruik van eindige outomate vir spesifikasie
4. Harel-diagramme—'n uitbreiding van eindige outomate wat opgeneem is in die spesifikasietaal UML

(1) Grammatikas

Nuttig vir spesifikasie van patroon herkenners:

- Sintaksontleder is 'n patroon herkenner—spesifiseer met EBNF.
- Verbetering van masjienkode ("peep hole optimisation")—spesifiseer patrone wat vervang moet word met 'n grammatika.
- Teks verwerking (spesifiseer verplasing van sekere tekspatrone met ander patrone.

(2) Spesifikasie met proposisie logika

- Nuttig om ingewikkelde beslissing-strukture in 'n stelsel te spesifiseer en te ontleed om logiese foute uit te skakel.
- Beskou alle moontlike kondisies individueel en spesifiseer formeel.
- Vereenvoudig spesifikasie (indien moontlik) en lei kode meganies af vanaf spesifikasie.

Eenvoudige voorbeeld

Spesifikasie in terme van alle moontlike kondisies en aksies uitgevoer:

$a \wedge b \rightarrow action0$
$a \wedge \neg b \rightarrow action1$
$\neg a \wedge b \rightarrow action2$
$\neg a \wedge \neg b \rightarrow action1$

Omdat *action*1 onder twee verskillende omstandighede uitgevoer word, kan ons vereenvoudig:

$$\begin{aligned} & (a \wedge \neg b) \vee (\neg a \wedge \neg b) \\ \equiv\ & (a \vee \neg a) \wedge \neg b \text{ [distr]} \\ \equiv\ & \neg b \text{ [exmid]} \end{aligned}$$

Vereenvoudigde (maar ekwivalente) spesifikasie:

$a \wedge b \rightarrow action0$
$\neg b \rightarrow action1$
$\neg a \wedge b \rightarrow action2$

- Volgorde van toetsing is belangrik vir effektiwiteit.
- Teken beslissingsboom om optimale oplossing te vind.
- Wenk: toets veranderlikes wat in die meeste kondisies voorkom eerste.

In die voorbeeld kom b of $\neg b$ in alle kondisies voor. Die volgende effektiewe kode is afleibaar vanaf die vereenvoudigde spesifikasie:

```
if b then
  if a then action0 else action2 fi
else action1
fi
```

As a eerste getoets word, is die kode meer kompleks en minder effektief, wat soms 'n beduidende verskil kan maak.

Voordele

- Volledigheid: dis moontlik om te verseker dat alle moontlike kondisies ingesluit word.
- Betroubaarheid van implementering: maklik om te versker dat die spesifikasie bevredig word.
- Effektiwiteit: vereenvoudiging van die spesifikasie kan lei tot vinniger en eenvoudiger kode.

(3) Niedeterministiese eindige outomate

$$M = (Q, \Sigma, \delta, q_0, F)$$

1. Q is 'n eindige versameling toestande ("states")
2. Σ is 'n eindige versameling toevoersimbole
3. δ is 'n funksie van $Q \times \Sigma$ na $\mathbb{P}(Q)$ (dit word die oorgangsfunksie genoem)
4. $q_0 \in Q$ is die aanvangstoestand
5. $F \subseteq Q$ is die versameling van eindtoestande

Bewegings

- 'n Outomaat maak 'n reeks bewegings om 'n sekere toevoerstring te herken
- Elke beweging word bepaal deur
  - die huidige toestand
  - die huidige toevoersimbool
- Elke beweging doen die volgende
  - die huidige toestand verander soos aangedui deur die oorgangsfunksie
  - die leeskop skuif aan na die volgende simbool in die toevoerstring

Konfigurasies van 'n eindige outomaat

- Die konfigurasie van die outomaat word volledig bepaal deur
  - die huidige toestand $q \in Q$
  - die string simbole $w \in \Sigma^*$ in die toevoer
- 'n Konfigurasie is 'n paar $(q, w) \in Q \times \Sigma^*$
- 'n Beginkonfigurasie is 'n paar (q_0, w)
- 'n Eindkonfigurasie is 'n paar (q, e), waar e die leë string is en $q \in F$

Herkenning van toevoerstringe

- 'n Beweging van outomaat M is 'n relasie $\vdash$ op konfigurasies $(C_i \vdash C_j)$
- As $\delta(q, a)$ 'n toestand q' bevat, dan is $(q, aw) \vdash (q', w)$ 'n geldige beweging vir alle $w \in \Sigma^*$ (a is 'n enkelsimbool)
- Die nul-beweging verander nie die konfigurasie van outomaat M nie ($C \vdash^0 C'$ en $C = C'$)
- 'n k-beweging $C_0 \vdash^k C_k$ vir $k \geq 1$ impliseer 'n reeks konfigurasies $C_0, \ldots, C_{k-1}$ sodat $C_i \vdash C_{i+1}$ moontlik is vir alle i, $0 \leq i < k$.
- 'n String w word aanvaar deur enige outomaat M as $(q_0, w) \vdash^* (q, e)$ vir enige $q \in F$. ($\vdash^*$ is die refleksiewe en transitiewe sluiting van $\vdash$)

Voorbeeld

$M = (\{p, q, r\}, \{0, 1\}, \delta, p, \{r\})$

$\delta(p, 0) = \{q\}$
$\delta(p, 1) = \{p\}$
$\delta(q, 0) = \{r\}$
$\delta(q, 1) = \{p\}$
$\delta(r, 0) = \{r\}$
$\delta(r, 1) = \{r\}$

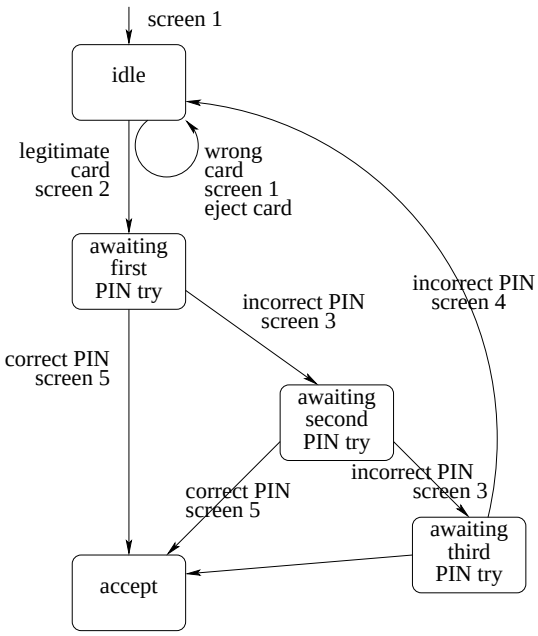
Outomaat M aanvaar alle stringe van 0'e en 1'e met twee opeenvolgende 0'e.

Voorbeeld:

$$\begin{aligned}(p, 01001) &\vdash (q, 1001)\\ &\vdash (p, 001)\\ &\vdash (q, 01)\\ &\vdash (r, 1)\\ &\vdash (r, e)\end{aligned}$$

Oefeninge

1. Ontwerp 'n eindige outomaat wat stringe aanvaar met simbole uit die versameling $\{a, b, 0, 1\}$. Die eerste simbool in 'n string moet 'n letter wees. (moontlike toepassing: herkenning van veranderlike name)
2. Ontwerp 'n eindige outomaat wat stringe aanvaar wat 'n ewe aantal 0'e en 'n ewe aantal 1'e bevat. (moontlike toepassing: pariteitberekenings in datakommunikasie)



Finite Automaton for PIN Tries

Struktuur van PIN-program

```
CONST
  (* state names and corresponding numbers *)
  idle = 0; try1 = 1; try2 = 2; try3 = 3;
  acc = 4;
VAR
  s: INTEGER; (* current state *)
  event: INTEGER; (* event number *)
  old_s: INTEGER;
  (* data structure for transition function *)
BEGIN
  Init;
  s := idle;
  REPEAT
    old_s := s;
    s := NewState(s, event);
    Action(old_s, event)
  UNTIL s = acc
END
```


Voordele van outomate vir spesifikasie

- Outomaat-interpreteerder is algemeen bruikbaar (effektief genoeg indien in saamsteltaal geskryf)
- Spesifikasie en implementering is dieselfde— geen probleem dus om te waarborg dat spesifikasie bevredig word nie
- Ingewikkelde kontrolevloei kan beskryf word
- Maklik om kontrolvloei te verander (maklik onderhoubaar)
- Outomatiese ontledings vir korrektheid is moontlik

Oefening: kommunikasie tussen rekenaars

Gebruik outomate om 'n stelsel te spesifiseer wat twee rekenaars S en R met mekaar laat kommunikeer via 'n onbetroubare kanaal. Elke boodskap wat gestuur word deur rekenaar S bevat ekstra inligting wat rekenaar R instaat stel om te bepaal of die boodskap korrek is of nie. Indien 'n boodskap korrek is, moet R 'n erkenningsboodskap stuur aan S. Foutiewe boodskappe moet herhaaldelik versend word totdat 'n korrekte weergawe ontvang word. Aangesien enige boodskap kan wegraak (of verander kan word tydens versending), moet 'n klok gebruik word om te verseker dat S nie byvoorbeeld vir ewig sal wag vir 'n erkenningsboodskap nie. Die stelsel moet verseker dat geen boodskap meer as een keer deur R aanvaar sal word nie. (Wenk: nommer die boodskappe op een of ander manier.)