Random bit generator in TEX

Abstract
Een in TEX gecodeerde randomgenerator maakt het mogelijk om willekeurige getallen en beslissingen te verwerken in de productie van documenten.

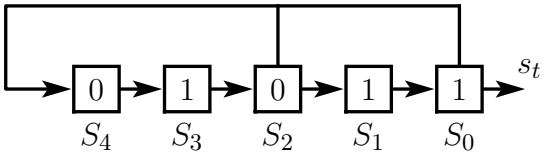
Keywords
random, pseudorandom, schuifregister, lfsr

Inleiding

De *hvdm-rng* module maakt bitreeksen voor willekeurige getallen en willekeurige beslissingen. Een voorbeeld waar dit gebruikt kan worden is het zetten van meerkeuzevragen. Door de volgorde van de items te veranderen kunnen dezelfde vragen er van keer tot keer verschillend uitzien. In feite is dit de oorspronkelijke motivatie voor het implementeren van de generator.
Deze module is een op ConTEXt aangepaste en opgewaardeerde versie van mijn vroegere LaTEX *random.sty* package; de laatste versie 2.4 daarvan is gedateerd 1994/10/21 en gepubliceerd in TUGboat.[1] In de hier gepresenteerde versie is de periode van de rij aanzienlijk langer geworden, de macros zijn aangepast aan de manier waarop in ConTEXt parameters worden gebruikt en een paar macros die het gebruik vergemakkelijken zijn toegevoegd. De beschrijving van het generatormechanisme volgt in grote lijnen het TUGboat–artikel. Het wordt hier in het Nederlands gepubliceerd in de hoop dat het voor de lezers van de MAPS nuttig en interessant zal blijken.

Schuifregisters

Omdat een deel van mijn interesses ligt in het lesgeven over geheimschrift in het vak Cryptografie, is het niet verwonderlijk dat daarin de inspiratie voor de macros is gezocht. Willekeurige bitrijen worden in de cryptografie gebruikt om gegevensstromen te vercijferen. Leek dat in 1994 nog een wat exotische toepassing, sindsdien draagt iedereen in z'n mobiele telefoon zoiets bij zich. Van de mechanismen die in aanmerking komen is het schuifregister erg geschikt door zijn eenvoud. En hoewel de eenvoudigste variant, het lineaire schuifregister, uit cryptografisch oogpunt onvoldoende veiligheid te bieden heeft, is het voor toepassing in TEX voldoende zolang beveiliging niet het doel is.
De primaire referentie voor schuifregisters is het befaamde boek van Golomb.[2] Maar er zijn ook meer recente bronnen, waaronder mijn eigen syllabus voor het onderwijs aan de Universiteit van Amsterdam. Voor het begrijpen van de TEX–code is het nuttig een idee te hebben van de werking. De figuur toont een klein lineair schuifregister. Het bestaat uit vijf zogenaamde *registertrappen* die ieder een enkel bit kunnen bevatten. Deze registertrappen zijn van links naar rechts S_4, ..., S_0 genoemd. In de figuur is als voorbeeld in elke trap een bitwaarde ingetekend. Tezamen vormen deze waarden de *toestand* van het register op dat moment; in de figuur is de toestand dus 01011.



Lineair schuifregister met vijf trappen

Het schuifregister werkt als volgt. In elke cyclus wordt het bit in S_0 als resultaat uitgevoerd; op tijdstip t wordt dat het bit s_t. Een nieuw bit wordt gemaakt door optelling van de bits in S_2 en S_0; dat zijn de registertrappen van waaruit een verticale lijn naar boven loopt. Deze optelling geschiedt onder verwaarlozing van de overloop en de mogelijke uitkomsten zijn dus: $0{+}0 = 1{+}1 = 0$ of $0{+}1 = 1{+}0 = 1$. Het nieuwe bit neemt plaats in S_4. Op hetzelfde moment schuiven alle bits een plaats op naar rechts: $S_4 \rightarrow S_3$, $S_3 \rightarrow S_2$, enz. In het voorbeeld zal $s_t = 1$ en de nieuwe toestand wordt 10101.
Men kan zo vaak een nieuw bit produceren als nodig. Uiteraard zal de reeks zich op een gegeven moment gaan herhalen. Als het register om te beginnen 00000 bevat is dat meteen al het geval. Gemakkelijk valt in te zien dat het nieuwe bit dan ook weer een 0 wordt en dat blijft zo. Hiervan kan gebruik gemaakt worden om de randomproductie effectief uit te schakelen zonder iets aan het programma te behoeven te veranderen. Indien de constructie van het schuifregister handig gekozen is leidt elke andere begintoestand tot een nieuwe, totdat alle mogelijkheden zijn uitgeput. Voor een n–traps register zullen dit er $2^n - 1$ zijn; 31 in het voorbeeld.
Het schuifregister kan worden beschreven met een polynomiale functie in de bit variable $x \in \{0, 1\}$. Deze functie, de zogenaamde *karakteristieke functie* luidt voor het voorbeeld

$$f(x) = 1 + x^2 + x^5$$

Wanneer de karakteristieke functie aan bepaalde eisen voldoet is dat een garantie voor de maximaal mogelijke periode. Een tweede eis die aan het schuifregister werd opgelegd is, dat het eenvoudig te implementeren moet zijn. Dit houdt onder meer in dat het register hooguit 31 trappen mag tellen, omdat anders de toestand niet meer in een countregister past en overflow geeft. Hoe minder aftappunten (S_2 en S_0 in het voorbeeld), hoe eenvoudiger de implementatie; twee stuks geeft een karakteristieke functie met drie termen en is het theoretisch minimum. Een geschikte polynoom die aan al deze eisen voldoet is die welke in het aangehaalde TUGboat–artikel gebruikt is

$$f(x) = 1 + x^{21} + x^{22}$$

Deze levert een random rij met een periode van 4.194.303 bits en is eenvoudig te implementeren zonder dat een hulpregister nodig is. Een andere goede mogelijkheid is het schuifregister gebaseerd op

$$f(x) = 1 + x^2 + x^{29}$$

Dit register heeft een flink langere periode van 536.870.911 maar gebruikt een hulpregister bij het berekenen van het volgend bit.[3] In tegenstelling tot de versie van 1994 is nu gekozen voor de langere reeks.

Implementatie

De toestand van het schuifregister wordt opgeslagen in countregister `\SRstate`, de constante `\SRconst` heeft de waarde 2^{28} en wordt gebruikt om door bijtelling het hoogste bit een 1 te kunnen maken, `\SRlast` dient voor het bewaren van het laatst gemaakte getal en `\SRtemp` is een hulpregister.

```
\newcount\SRstate
\def\SRconst{268435456}
\newcount\SRlast
\newcount\SRtemp
```

Het effect van een stap op het schuifregister wordt bereikt door het register te delen door twee, als gevolg waarvan alle bits een plaats naar rechts opschuiven. Het oorspronkelijk minst significante bit wordt vergeleken met dat twee plaatsen links ervan. Indien deze twee van elkaar verschillen moet een 1 aan de linkerkant van het statusregister worden toegevoegd. Macro `\SRadvance` bewerkstelligt dit. De veranderingen in het statusregister geschieden globaal, zodat het schuifregister onder alle omstandigheden blijft doorlopen en niet bij het verlaten van een lokale groep terugspringt naar een vorig punt in de cyclus. Dit toch al noodzakelijk globale karakter van de update maakt het aantrekkelijk om met een `\begingroup`..`\endgroup` paar de verandering van het hulpregister lokaal te houden.

```
\def\SRadvance{\begingroup
    \SRtemp\SRstate
    \divide\SRtemp\plusfour
    \ifodd\SRstate
        \global\divide\SRstate\plustwo
        \ifodd\SRtemp
        \else
            \global\advance\SRstate\SRconst\relax
        \fi
    \else
        \global\divide\SRstate\plustwo
        \ifodd\SRtemp
            \global\advance\SRstate\SRconst\relax
        \fi
    \fi\endgroup}
```

Deze macro is de kern van de generator. De andere macros zijn voor het gebruik ervan.

`\SRset, \SRset[n].` Het schuifregister blijft stationair in de eerste vorm of als expliciet n = 0 en levert dan alleen nullen af. Voor alle andere input wordt de begintoestand ingesteld op een waarde beneden het maximum van 2^{29}. Extra code is toegevoegd om rekening te houden met de mogelijkheid dat voor de initialisatie een klein getal is opgegeven. Bij initialiseren met een klein getal worden in het begin lange reeksen nullen geproduceerd. Dit komt omdat in de 29 trappen van het register dan veel nullen op een rij zitten. Op zich is dat niet erg en theoretisch moet zo'n reeks altijd een keer voorkomen. Maar het maakt dat het begin van de randomrij er een beetje weinig random uitziet. Vooral indien maar weinig randomgetallen nodig zijn is dat een nadeel. Daarom doet het register eerst 1000 stappen om op gang te komen.[4] Met de huidige snelle computers zal het amper merkbaar zijn.

```
\def\SRset{\dosingleargument\doSRset}
\def\doSRset[#1]{%
    \doifemptyelse{#1}%
        {\global\SRstate\zeropoint}%
        {\global\SRstate#1\relax}%
    \ifnum\SRstate<\zeropoint \global\SRstate-\SRstate \fi
    \SRtemp\SRconst\relax
    \advance\SRtemp\SRtemp
    \advance\SRtemp\minusone
    \loop \ifnum\SRstate>\SRtemp
        \global\divide\SRstate\plustwo
    \repeat
    \SRtemp\plusthousand
    \loop \ifnum\SRtemp>\zeropoint
```

```
        \advance\SRtemp\minusone
        \SRadvance
    \repeat
    \ignorespaces}
```

`\SRbit.` Genereert het eerstvolgende randombit.

```
\def\SRbit{\SRadvance\ifodd\SRstate 1\else 0\fi}
```

`\ifSR` *<true>* `\else` *<false>* `\fi.` Neemt de true–tak als het schuifregister een 1 afgeeft en de false–tak voor een 0.

```
\def\ififtrue{\iftrue}
\def\ififfalse{\iffalse}
\def\ifSR{\SRadvance
    \ifodd\SRstate  \expandafter\ififtrue
    \else           \expandafter\ififfalse
    \fi}
```

`\SRnumber[n], \SRnext[n].` Macro `\SRnumber[n]` maakt een getal van n bits. Dit getal wordt geproduceerd door `\SRnext[n]`, dat het in register `\SRlast` plaatst. Deze splitsing over twee macros is gedaan omdat men niet eenvoudig het resultaat van `\SRnumber` aan een countregister kan toekennen. Eerst produceren in `\SRlast` en vandaaruit toekennen aan het gewenste register lukt wel.

```
\def\SRnext[#1]{\SRtemp#1\relax
    \global\SRlast\zeropoint
    \loop \ifnum\SRtemp>\zeropoint
        \advance\SRtemp\minusone
        \global\advance\SRlast\SRlast
        \SRadvance
        \ifodd\SRstate
            \global\advance\SRlast\plusone
        \fi
    \repeat
    \ignorespaces}
\def\SRnumber[#1]{\SRnext[#1]\the\SRlast}
```

1. H. van der Meer, *Random Bit Generator in TEX*, TUGboat vol.15 (1994) nr.1, p. 57–58.
2. S.W. Golomb, *Shift Register Sequences*, 1967, Holden Day, San Francisco.
3. Strikt genomen is deze bewering onjuist. Het is mogelijk om de berekening zo uit te splitsen dat een hulpregister niet nodig is. Maar waarom gekunsteld programmeren als dat niet echt nodig is?
4. Dit is analoog aan hetgeen in een mobiele telefoon gebeurt bij het instellen van de schuifregisters voor elk spraakframe; deze doen 100 stappen voordat de uitvoer gebruikt wordt.

Hans van der Meer
hansm@science.uva.nl