Formale Grundlagen der Informatik 3

Einführung in
PROMELA und SPIN

Prof. Stefan Katzenbeisser
Security Engineering Group
Technische Universität Darmstadt

skatzenbeisser@acm.org
http://www.seceng.informatik.tu-darmstadt.de

PROMELA
Einleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

PROMELA = Process Meta-Language

- Modellierungssprache für nebenläufige (verteilte) Systeme/Protokolle
  - Netzwerkprotokolle
  - Multi-Threaded Programme, die über gemeinsame Variablen oder Nachrichten interagieren
- Simulation und Verifikation mittels SPIN
- Starke Ähnlichkeiten mit C
- Ähnlichkeit von Kontrollstrukturen mit ALGOL, Bash

SECENG SECURITY ENGINEERING

PROMELA
Sprache

TECHNISCHE UNIVERSITÄT DARMSTADT

- Besteht aus:
  - Prozessen
  - Nachrichten-Kanälen
  - Variablen

- Synchronisation und Nachrichten zwischen Prozessen möglich

- keine Programmiersprache(!):
  - Keine Methoden, Libraries oder GUI
  - Nicht-deterministisch (bis auf sehr einfache Modelle)

SECENG SECURITY ENGINEERING

PROMELA
Example (Dekker's Mutex Algorithm [1962])

TECHNISCHE UNIVERSITÄT DARMSTADT

```
bit turn;
bool flag[2];
byte cnt;

active [2] proctype mutex(){
  pid i, j;
  i = _pid;
  j = 1 - _pid;
again:
  flag[i] = true;
  do
  :: flag[j] ->
        if
        :: turn == j -> flag[i] =
  false;
                  (turn != j) ->
  flag[j] = true
        :: else -> skip
        fi
  :: else -> break
  od;
  cnt++;
  assert(cnt == 1);
  cnt--;

  turn = j;
  flag[i] = false;
  goto again
}
```

Variablen

Array

Prozess

Label

Schleife

Guard

Selection

Assertion

Sprung

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Basisdatentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Name	Größe (bit)	Signed?	Wertbereich
bit	1	unsigned	0 … 1
bool	1	unsigned	0 … 1
byte	8	unsigned	0 … 255
mtype	8	unsigned	0 … 255
short	16	signed	-2^{15} … $2^{15} - 1$
int	32	signed	-2^{15} … $2^{15} - 1$

- Arithmetische Operatoren: +, -, *, /, %
- Semantik entspricht C
- Nicht-initialisierte Variablen werden per default `0` gesetzt
- Keine String Variablen
- Compiler verschiebt Deklarationen an den Beginn des Prozesses (< 6.0)

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Prozesse

TECHNISCHE UNIVERSITÄT DARMSTADT

Zentrales Element zur Modellierung

```
proctype Proc1(){
  do
  :: a < 10 -> a = a + 1
  :: printf(a)
  od
}
```

- Kann durch Keyword `active` direkt gestartet werden
- Alternativ durch `run Proc1();` aus anderem Prozess startbar
- Nebenläufige Programmierung modellierbar

SECENG SECURITY ENGINEERING

PROMELA
Simulation eines Modells

TECHNISCHE UNIVERSITÄT DARMSTADT

Hello-World Beispiel:

```
active proctype P(){
  printf("Hello world!\n")
}
```

Ausführen in der Kommandozeile

```
spin hello.pml
Hello world!
1 process created
```


SECENG SECURITY ENGINEERING

PROMELA
Sprache – komplexe Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

Array

```
int x [5];
x[1] = 42;
```

Eigene Datentypen

```
typedef State{
  short id;
  int value;
}
State.id = 1; State.value = 42
```

- Können aus anderen Datentypen bestehen, aber keine Selbstreferenz (!)

SECENG SECURITY ENGINEERING

PROMELA
Sprache – komplexe Datentypen (2)

Enumerations über `mtype` (message type) möglich

```
mtype = {idle, busy, error}; /* set mtype states */
mtype state = error;
printf("Current system state: %e\n", state)
```

- Maximal 255 mtypes möglich
- Labels werden intern auf Zahlen übersetzt

PROMELA
Sprache – erste Befehle

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausgabe von Text und Werten, Funktionsweise wie in C

```
printf(x [, args])
```

Assignment

```
byte cnt = 250
```

Vergleiche

```
cnt == 1
cnt != flag
cnt < 12
```

Operatoren C-artig

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Kontrollstrukturen

TECHNISCHE UNIVERSITÄT DARMSTADT

In PROMELA keine Unterscheidung zwischen `if` und `switch - case`

```
if
:: (a < 5) -> commandA; commandA2
:: (a == 5) -> commandB
:: else -> commandC
fi
```

Semikolon in diesem Kontext zur Sequenzierung von mehreren Befehlen für einen Fall

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Kontrollstrukturen (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Was passiert in folgendem Beispiel?

```
if
:: (a < 5) -> commandA; commandA2
:: (a < 10) -> commandB
fi
```

- Guards können "überlappen"!
  - es wird zufällig aus den erfüllten Guards gewählt
- Statements können "blocken"
  - wenn kein Guard erfüllt wird, hält der Prozess bis ein Guard erfüllt wird

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Kontrollstrukturen (3)

TECHNISCHE UNIVERSITÄT DARMSTADT

```
active proctype P(){
  bool p = … ;
  if
    :: p -> …
    :: true -> …
  fi
}
```

Guard 2 kann unabhängig von p jederzeit gewählt werden

```
active proctype P(){
  bool p = … ;
  if
    :: p -> …
    :: else -> …
  fi
}
```

Guard 2 kann nur gewählt werden, wenn p `false` ist

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Schleifen (do .. od)

TECHNISCHE UNIVERSITÄT DARMSTADT

Example gcd computation

```
int a = 15, b = 20;
do
  :: a > b -> a = a - b
  :: b > a -> b = b - a
  :: a == b -> break
od
```

- Beenden der Schleife durch `break` oder `goto` – Statement
- Guards werden wie bei `if` zufällig gewählt und können blocken

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Guard Statement Syntax

TECHNISCHE UNIVERSITÄT DARMSTADT

```
:: guard-statement      ->    command
```

Zu beachten:

- Symbol -> ist in PROMELA überladen
  - auch bedingte Ausdrücke möglich: `(guard -> then : otherwise)`
  - bei bedingten Ausdrücken sind Klammern erforderlich
- erstes Statement nach `::` wird als guard evaluiert

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Schleifen (for)

TECHNISCHE UNIVERSITÄT DARMSTADT

Auch for-Schleifen sind modellierbar

```
#define N 10
…
  int i;
 int sum = 0;
 for(i : 1 .. N){
   sum = sum + 1
 }
…
```

- For-Schleifen können geschachtelt werden
- Intern wandelt der Compiler die Schleife in eine do-Schleife um

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Sprünge

TECHNISCHE UNIVERSITÄT DARMSTADT

Umwandlung von einer for-Schleife in do-Schleife:

```
#define N 10
…
 int i;
 int sum = 0;
 for(i : 1 .. N){
  sum = sum + 1
 }
…
```

```
#define N 10
…
 int i = 1;
 int sum = 0;
 do
 :: i > N -> goto exitloop
 :: else -> sum = sum + 1; i++
 od;
 exitloop:
   print("Out of loop")
…
```


SECENG SECURITY ENGINEERING

PROMELA
Sprache – Sprünge (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Sprünge können verwendet werden um zu einem bestimmten Anweisungsblock zu "springen"
- Erfordern:
  - Label (Punkt zu dem gesprungen wird)
  - Sprunganweisung

```
goto command1
…
command1:
```

- Sprünge nur innerhalb eines Prozesses
- Label müssen eindeutig sein (innerhalb von einem Prozess)
- Vorsicht vor unübersichtlichem Code!

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Arrays und Schleifen

TECHNISCHE UNIVERSITÄT DARMSTADT

Iteration über Arrays ist möglich:

```
byte a[N];
byte i; byte sum = 0;
a[0] = 0; a[1] = 1; a[2] = 2; … ; a[40] = 48;
for(i in a){
  sum = sum + a[i]
}
```

- Nur eindimensionale Arrays möglich
- Arrays haben immer konstante Größen

SECENG SECURITY ENGINEERING

PROMELA
Sprache – Messages

TECHNISCHE UNIVERSITÄT DARMSTADT

Nachrichten Kanäle können genutzt werden um den Transport von Nachrichten von einem Prozess zu einem anderen zu modellieren

Initialisierung eines Channels

```
chan cname = [16] of {byte}
```

Senden eines Wertes über den Channel

```
cname ! expr;
```

Lesen eines Wertes

```
cname ? expr;
```

Zur Synchronisation (Sender/Receiver wartet bei senden/empfangen)

```
chan port = [0] of {byte]
```


SECENG SECURITY ENGINEERING

PROMELA
Sprache – Inline Code

TECHNISCHE UNIVERSITÄT DARMSTADT

- keine Methoden oder Funktionen in PROMELA
- aber Makro-artige Abkürzungen sind möglich

```
inline setDate(D, DD, MM, YY){
  D.day = DD; D.month = MM; D.year = YY
}
active proctype P(){
  Date d;
  setDate(d, 1, 7, 62)
}
```

Beachte:

- kein neuer Scope; Definition von Variablen sollte vermieden werden

SECENG SECURITY ENGINEERING

PROMELA
Nicht deterministische Modelle

TECHNISCHE UNIVERSITÄT DARMSTADT

Deterministische Modelle sind trivial:

- einzelner Prozess ohne überlappende Guards
    - alle Variablen sind initialisiert
    - keine Nutzereingaben
    - jeder Zustand ist entweder blockend oder hat einen Nachfolger
- es gibt genau einen Programmablauf

Nicht triviale Programme sind nicht-deterministisch

- Gründe für Nicht-Determinismus:
    - nicht deterministische Wahl von Alternativen bei überlappenden Guards
    - Scheduling von nebenläufigen Prozessen

SECENG SECURITY ENGINEERING

PROMELA
Nicht deterministische Modelle – Überlappende Guards

Beispiel:

```
byte range;
if
  :: range = 1
  :: range = 3
  :: range = 5
fi
```

- `range` erhält nicht-deterministisch Werte aus {1,3,5}

PROMELA
Nicht deterministische Modelle – Prozess Scheduling

TECHNISCHE UNIVERSITÄT DARMSTADT

```
active proctype P(){
  printf("Proc P, Statement 1\n")
  printf("Proc P, Statement 2\n")
}

active proctype Q(){
  printf("Proc Q, Statement 1\n")
  printf("Proc Q, Statement 2\n")
}
```


SECENG SECURITY ENGINEERING

PROMELA
Nicht deterministische Modelle – Prozess Scheduling

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bis zu 255 Prozesse möglich, diese arbeiten nebenläufig
- Realisierung von Nebenläufigkeit auf nur einem Prozessor
  - Scheduler wählt zufällig, aus welchem Prozess ein weiteres Statement ausgeführt werden soll
  - Dadurch sind viele verschiedene Abläufe möglich
  - => nicht deterministische Ausführung

- Ausführungen von Modellen sind entweder: **unendlich**, **terminierend** oder **blockend**

PROMELA
Nicht deterministische Modelle – Atomicity

TECHNISCHE UNIVERSITÄT DARMSTADT

Definition Atomicity:

Ein Ausdruck oder Statement eines Prozesses welcher komplett ohne die Möglichkeit von Interleaving ausgeführt wird, heißt: **atomar**.

In PROMELA:

- Zuweisungen, Sprünge, skip und Anweisungen sind atomar
  - Conditional expressions `(p -> q : r)` sind atomar
- Guarded commands sind nicht atomar!
- Erzwingbar durch `atomic`-Block

SECENG SECURITY ENGINEERING

PROMELA
Anwendung

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Modellierung der relevanten Features eines Systems mit PROMELA
   - Abstraktion von komplexen Berechnungen
   - Ersetzung von unbeschränkten Datenstrukturen durch endliche

2. Wähle Eigenschaften, die das PROMELA Modell erfüllen soll
   - Allgemeine Eigenschaften
     - Mutex
     - keine Deadlocks, keine Starvation
   - System-spezifische Eigenschaften
     - Abwesenheit von Zuständen
     - Sequenz von Events

SECENG SECURITY ENGINEERING

PROMELA
Anwendung

TECHNISCHE UNIVERSITÄT DARMSTADT

3. Verifiziere, dass alle Ausführungen des Modells die Eigenschaften erfüllen
   - Meist sind mehrere Iterationen nötig um Modell und Eigenschaften korrekt zu formulieren
   - Fehlgeschlagene Verifikation gibt Feedback in Form eines Gegenbeispiels

SECENG SECURITY ENGINEERING

Wie können wir ein Modell verifizieren?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wir brauchen einen Model Checker (MC)
  - MC ist so entworfen, dass er versucht den Nutzer zu widerlegen
  - versucht nicht die Korrektheit der Eigenschaften zu prüfen, sondern ein Gegenbeispiel zu finden

Wie kann man auf die Weise die Korrektheit beweisen?

- Findet der MC kein Gegenbeispiel, so ist die Korrektheit der Eigenschaft bewiesen
- Die Suche des MC nach einem Gegenbeispiel ist "erschöpfend" (exhaustive)

SECENG SECURITY ENGINEERING

Erschöpfende Suche – Exhaustive Search

TECHNISCHE UNIVERSITÄT DARMSTADT

Exhaustive Search = zur Auflösung von Nicht-Determinismus (ND) testet der MC jeden möglichen Weg der Ausführung

Im Fall von PROMELA gibt es zwei Arten von ND:

- Explizit, lokal:
  Überlappende Guards

```
:: x < 5 -> …
:: x < 10 -> …
:: x < 15 -> …
```

- Implizit, global:
  Scheduling von Prozessen

SECENG SECURITY ENGINEERING

SPIN
Einleitung

TECHNISCHE UNIVERSITÄT DARMSTADT

SPIN = **S**imple **P**ROMELA **In**terpreter

Mittlerweile allerdings:

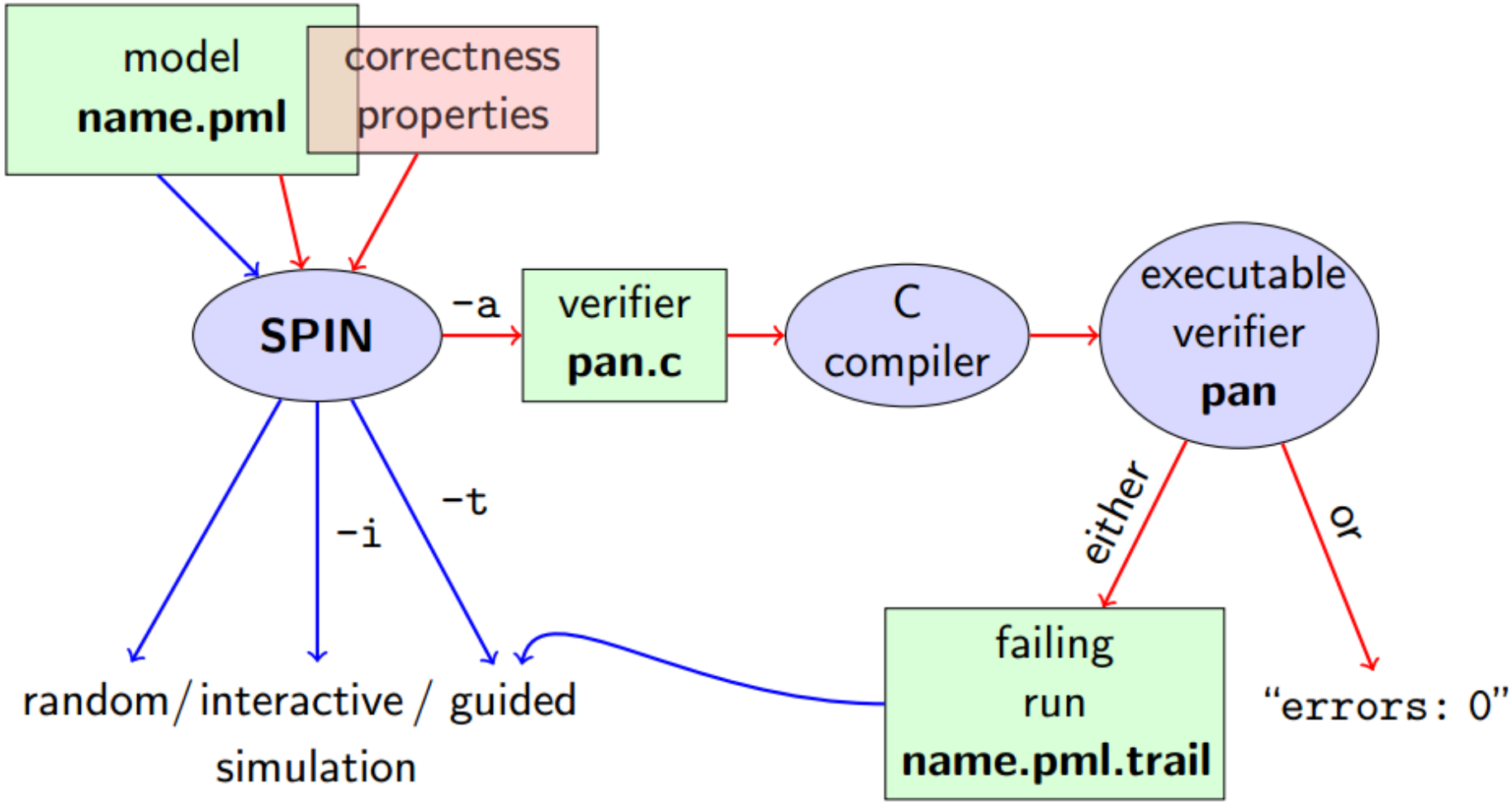
- Nicht mehr „Simple"
- Nicht mehr nur „Interpreter"

Anwendung:

- Werkzeug zur Simualtion/Verifikation von PROMELA Modellen auf Fehler im Design
  - Deadlocks, Race Conditions, Verletzung von Aussagen (Assertions)
  - Safety Eigenschaften
  - Liveness Eigenschaften
- Prüft Modelle erschöpfend gegen Korrektheitseigenschaften

SECENG SECURITY ENGINEERING

SPIN
Workflow: Übersicht



SPIN
Korrektheitseigenschaften

Korrektheitseigenschaften können innerhalb oder außerhalb des PROMELA Modells angegeben werden

Innerhalb des Modells:

- Assertion Statements
- Meta Labels
  - end labels
  - accept labels
  - progress labels

Außerhalb des Modells:

- Never Claims
- Temporal Logik Formeln

SPIN
Assertion Statements

TECHNISCHE UNIVERSITÄT DARMSTADT

Zur Formulierung von Korrektheitseigenschaften

- haben in PROMELA die Form `assert(expr)`
- `expr` ist ein beliebiger PROMELA Ausdruck

- `expr` ist üblicherweise vom Typ bool
- `assert(expr)` kann überall vorkommen, wo ein Ausdruck erwartet wird

```
if
  :: b1 -> statement1;
        assert(x < y)
…
fi
```


SECENG SECURITY ENGINEERING

SPIN
Assertion Statements - Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```
byte a, b, max;
select(a: 1 .. 3);
select(b: 1 .. 3);
if
  :: a >= b -> max = a
  :: a <= b -> max = b
fi
assert(max == (a > b -> a : b))
```

- Erstes Beispiel für eine Korrektheitseigenschaft
- ab jetzt ist model checking möglich!

SECENG SECURITY ENGINEERING

SPIN
Anwendung - Simulation

TECHNISCHE UNIVERSITÄT DARMSTADT

Zufällige Simulation

```
spin funnyCode.pml
```

komplette Liste der Parameter unter:
http://spinroot.com/spin/Man/Spin.html

Interaktive Simulation

```
spin -i funnyCode.pml
```

Guided Simulation (anhand eines vorherigen Trails)

```
spin -t funnyCode.pml
```

Zufällige Simulation mit Ausgabe des jew. Zustandes

```
spin -p funnyCode.pml
```

Zufällige Simulation über N Schritte

```
spin -uN funnyCode.pml
```


SECENG SECURITY ENGINEERING

SPIN
Anwendung – Erzeugen eines Verifizierers

- SPIN kann aus dem Modell und den Korrektheitseigenschaften einen Verifizierer erzeugen
  - Verifizierer besteht aus C Code
  - Dateiname "pan.c" (es können zusätzliche Dateien erzeugt werden)

```
spin –a funnyCode.pml
```

- Anschließend zu ausführbarem Verifizierer kompilieren

```
gcc –o pan pan.c
```

pan: historisch von "protocol analyzer", jetzt "process analyzer"

SPIN
Anwendung – Model Checking

TECHNISCHE UNIVERSITÄT DARMSTADT

- Nach dem Kompilieren kann man den Verifizierer ausführen

```
./pan
```

- Es wird eine Vielzahl an Informationen ausgegeben
- Angabe zum Ergebniss der Verifikation:
  - "errors: 0" => Korrektheitseigenschaften verifiziert
  - "errors: n" (n > 0) => Gegenbeispiel gefunden; Beispiel in .trail gespeichert

SECENG SECURITY ENGINEERING