Softwaretechnik

Lecture 08: Verification of Parallel Programs

Peter Thiemann

Universität Freiburg, Germany

SS 2009

Appetizer

- Provide a glimpse of the problems
- Defer in-depth treatment to special lectures
- See: Apt, Olderog. Verification of Sequential and Concurrent Programs. Springer-Verlag, New York, 1991.

Example: Searching for Zeros

Let $f : \mathbf{Z} \to \mathbf{Z}$ be a function with at least one zero.
Develop a program *ZERO* that finds one such zero, *i.e.*, it finds an $z \in \mathbf{Z}$ such that $f(z) = 0$.

Example: Searching for Zeros

Let $f : \mathbf{Z} \to \mathbf{Z}$ be a function with at least one zero.
Develop a program *ZERO* that finds one such zero, *i.e.*, it finds an $z \in \mathbf{Z}$ such that $f(z) = 0$.

Approach

- Divide zeros of f in subproblems
  1. find positive zeros, $z > 0 \wedge f(z) = 0$ and
  2. find non-positive zeros, $z \leq 0 \wedge f(z) = 0$.
- Let S_1 and S_2 be programs that solve the respective subproblems
- The program $[S_1 \parallel S_2]$ solves the full problem

Example: Searching for Zeros

Let $f : \mathbf{Z} \to \mathbf{Z}$ be a function with at least one zero.
Develop a program *ZERO* that finds one such zero, *i.e.*, it finds an $z \in \mathbf{Z}$ such that $f(z) = 0$.

Approach

- Divide zeros of f in subproblems
  1. find positive zeros, $z > 0 \wedge f(z) = 0$ and
  2. find non-positive zeros, $z \leq 0 \wedge f(z) = 0$.
- Let S_1 and S_2 be programs that solve the respective subproblems
- The program $[S_1 \parallel S_2]$ solves the full problem

$[S_1 \parallel S_2]$ executes S_1 and S_2 in parallel. It terminates when both S_1 and S_2 terminate.

Solution #1

- Define S_1 by

```
found = false; x = 0;
while (!found) {
  x = x+1;
  found = f(x) == 0;
}
```

- Define S_2 by

```
found = false; y = 1;
while (!found) {
  y = y−1;
  found = f(y) == 0;
}
```

- Consider *ZERO-1* $\equiv [S_1 \parallel S_2]$ with found shared between S_1 and S_2

Problem with Solution #1

Solution #2

- Define S_1 by

```
x = 0;
while (!found) {
  x = x+1;
  found = f(x) == 0;
}
```

- Define S_2 by

```
y = 1;
while (!found) {
  y = y−1;
  found = f(y) == 0;
}
```

- Consider *ZERO-2*≡found = false; $[S_1 \parallel S_2]$

Problem with Solution #2

Solution #3

- Define S_1 by

```
x = 0;
while (!found) {
  x = x+1;
  if (f(x) == 0) found = true;
}
```

- Define S_2 by

```
y = 1;
while (!found) {
  y = y−1;
  if (f(y) == 0) found = true;
}
```

- Consider *ZERO-3*$\equiv$found = false; $[S_1 \parallel S_2]$

Problem with Solution #3

Solution #4

- Define S_1 by

```
x = 0;
while (!found) {
  await (turn == 1) turn = 2;
  x = x+1;
  if (f(x) == 0) found = true;
}
```

- Define S_2 by

```
y = 1;
while (!found) {
  await (turn == 2) turn = 1;
  y = y−1;
  if (f(y) == 0) found = true;
}
```

- Let *ZERO-4*≡turn = 1; found = false; $[S_1 \parallel S_2]$

Problem with Solution #4

Solution #5

- Define S_1 by

```
x = 0;
while (!found) {
  await (turn == 1) turn = 2;
  x = x+1;
  if (f(x) == 0) found = true;
}
turn = 2;
```

- Define S_2 by

```
y = 1;
while (!found) {
  await (turn == 2) turn = 1;
  y = y−1;
  if (f(y) == 0) found = true;
}
turn = 1;
```

- Let *ZERO-5*≡turn = 1; found = false; $[S_1 \parallel S_2]$

Solution #6

- Define S_1 by

```
x = 0;
while (!found) {
  wait (turn == 1);
  turn = 2;
  x = x+1;
  if (f(x) == 0) found = true;
}
turn = 2;
```

- Define S_2 by

```
y = 1;
while (!found) {
  wait (turn == 2);
  turn = 1;
  y = y−1;
  if (f(y) == 0) found = true;
}
turn = 1;
```

- Let *ZERO-6*≡turn = 1; found = false; $[S_1 \parallel S_2]$