Recap

• What is the difference between a model and a program?
• Syntax of while-programs?
• Semantics of while-programs?
• SOS: How?
• SOS-rule for while?
• Do nondeterministic programs exist?
• Sources of nondeterminism?
Parallelism

• Parallelism is the primary source of nondeterminism in programs
• Thus: hard to understand, error-prone
• But: of increasing importance (multicores)
• in Java, parallelism by multithreading
• key issue: synchronization
  ▪ via shared variables or message passing
  ▪ test-and-set, lock / semaphores, monitors
Parallelism Concept Language

• we add the following new constructs to the language of while-programs
  ▪ \{ \Pi_1 \parallel \Pi_2 \} or, more generally, \{ \Pi_1 \parallel \ldots \parallel \Pi_n \}
  ▪ await (b) \Pi;

• semantics
  ▪ parallel (interleaved) execution of the \Pi_i
  ▪ blocking wait until condition is satisfied;
    program fragment within await is noninterruptable

• for simplicity, assignments are atomic actions
  ▪ (unlike Java; cf. “synchronized” and “volatile” attributes)
Interleaving Semantics

• A state of the program consists of
  - an assignment of values to variables
  - a set of program counters (depending on the number of parallel components), and

• SOS-rules for parallel programs
  - if \((\mathbf{u}, \mathbf{i}, \mathbf{v}) \models b\) and \((\Pi, \mathbf{v}) \rightarrow^* (\text{skip}, \mathbf{v'})\), then \((\text{await} (b) \Pi, \mathbf{v}) \rightarrow (\text{skip}, \mathbf{v'})\)

  - if \((\Pi_1, \mathbf{v}) \rightarrow (\Pi_1', \mathbf{v'})\), then \((\{\Pi_1 \parallel \Pi_2\}, \mathbf{v}) \rightarrow (\{\Pi_1' \parallel \Pi_2\}, \mathbf{v'})\)

  - if \((\Pi_2, \mathbf{v}) \rightarrow (\Pi_2', \mathbf{v'})\), then \((\{\Pi_1 \parallel \Pi_2\}, \mathbf{v}) \rightarrow (\{\Pi_1 \parallel \Pi_2'\}, \mathbf{v'})\)

  - \((\{\text{skip} \parallel \text{skip}\}, \mathbf{v}) \rightarrow (\text{skip}, \mathbf{v})\)

• In general, several possible executions! (tree of possibilities)
Examples

• int n=0;
  {
    for (int i = 0; i<100; i++) n++;
  ||
    for (int i = 0; i<100; i++) n--;
  }

• int n=0; int l, r;
  {for (int i = 0; i<100; i++) {l=n; l++; n=l;}}
  || {for (int i = 0; i<100; i++) {r=n; r--; n=r;}}

• int n=0;
  {for (int i = 0; i<100; i++) await (1) {l=n; l++; n=l;}}
  || {for (int i = 0; i<100; i++) await (1) {r=n; r--; n=r;}}
More Examples

- $a=0; \ b=0; \ \{a\times=a; \ a-=5; \ |\ | \ b=2*b+3; \ b=1-b; \}$
- $a=0; \ \{a\times=a; \ a-=5; \ |\ | \ a=2*a+3; \ a=1-a; \}$
- $a=0; \ \{a++; \ |\ | \ a--; \}$
- $\{a=0; \ a++; \ |\ | \ a=0; \ a-- \}$
- $a=0; \ \{\text{await } (a>=0); \ a++; \ |\ | \ \text{await } (a<=0); \ a--; \}$
- $a=0; \ \{\text{await } (a>=0) \ a++; \ |\ | \ \text{await } (a<=0) \ a-- \}$

Executions of second example:
Deadlocks

- $a=0; \ b=0;
  \{\text{await } (a!=0) || \text{await } (b!=0)\}$

- $a=0; \ b=0;
  \{\text{await } (a==1) \ b=1 || \text{await } (b==1) \ a=1\}$

- $\text{prt}=T; \ \text{dsk}=T;
  \{\text{await } (\text{prt}) \ \text{prt}=F; \ \text{await} (\text{dsk}) \ \text{dsk}=F; \ \text{foo}; \ \text{prt}=T; \ \text{dsk}=T;
  || \ \text{await } (\text{dsk}) \ \text{dsk}=F; \ \text{await} (\text{prt}) \ \text{prt}=F; \ \text{bar}; \ \text{prt}=T; \ \text{dsk}=T; \}$

Checking for deadlock freedom?
A more practical example

Calculate binomial coefficient

\[
\binom{n}{k} = \frac{n \times (n-1) \times \ldots \times (n-k+1)}{1 \times 2 \times 3 \times \ldots \times k}
\]

in parallel by divisor and dividend:

```c
a=n; b=0; c=1;
{ while (a!=n-k) {c=c*a; a--;}  
  ||  
  while (b!=k) {b++; c=c/b;}  
}
```

Problem with this solution?
Correction

a=n; b=0; c=1;
{
    while (a ≠ n-k) {c=c*a; a--;}
||
    while (b ≠ k) {b++; await (a+b ≤ n); c=c/b;}
}

Correctness of this solution?
a=n; b=0; c=1;

\[ a = n; \quad b = 0; \quad c = 1; \]

\[ \alpha: \{ \]
\[ \quad \beta_1: \text{while } (a \neq n-k) \{ \]
\[ \quad \quad \beta_2: c = c \times a; \]
\[ \quad \quad \beta_3: a--; \]
\[ \quad \} \quad \beta_4: \]
\[ \]||
\[ \gamma_1: \text{while } (b \neq k) \{ \]
\[ \quad \gamma_2: b++; \]
\[ \quad \gamma_3: \text{await } (a+b \leq n); \]
\[ \quad \gamma_4: c = c / b; \]
\[ \} \quad \gamma_5: \]
\[ \]

\[ \begin{pmatrix} n \\ k \end{pmatrix} = \frac{n \times (n-1) \times \ldots \times (n-k+1)}{1 \times 2 \times 3 \times \ldots \times k} \]

- Proof idea: at the await (\( \gamma_3 \)) it holds that \( c = \frac{n \times (n-1) \times \ldots \times (n-i+1)}{1 \times 2 \times \ldots \times (j-1)} \) \( a = n-i \), \( b = j \)

- If \( a+b \leq n \), then \( n-i+j \leq n \), thus \( j \leq i \).

In this case, \( c \) is divisible by \( i \):
- \( n \) is divisible by 1
- \( n \times (n-1) \) is divisible by 2
- \( n \times (n-1) \times (n-2) \) is divisible by 2 and 3
- \( n \times (n-1) \times (n-2) \times (n-3) \) is divisible by \( 1 \times 2 \times 3 \times 4 \)

- What must a model checker know to do such a proof?
SMT

- Satisfiability modulo theories: Checking the satisfiability of a formula, where certain functions/relations are interpreted according to a particular mathematical theory
- Theory: set of axioms / consequences
- Theory of linear orders, natural numbers, reals, rings, fields, ...
- E.g., is \((x+y=2)\) satisfiable?
- E.g., is \((a^n+b^n=c^n \land n>2)\) satisfiable?
SMT

- Satisfiability of quantifier-free formulas is decidable for many theories of interest in model checking
  - Equality with “Uninterpreted Function Symbols”
  - Linear Arithmetic (Real and Integer)
  - Arrays (i.e., updatable maps)
  - Finite sets and multisets
  - Inductive data types (enumerations, lists, trees, . . . )
  - . . .

- Thanks to advances in SAT and in decision procedures, this can be done very efficiently by current SMT solvers
- SMT solvers differ from traditional theorem provers by having built-in theories, and using specialized methods to reason about them
Model Checking by SMT

- SMT-based model checking techniques blur the line between traditional model checking and deductive verification
- Different approaches, e.g., predicate abstraction, bounded model checking, backward reachability, temporal induction, ...