Disjoint Parallel Programs

Shangping Ren

Department of Computer Science
Illinois Institute of Technology

March 24, 2014
Outline

1. Basics
2. Syntax
3. Semantics
4. Verification
Execution of Parallel Programs

- We run sequential programs in parallel by *interleaving* their execution — execute $S_1$ for a while, then execute $S_2$ for a while, etc.
- Execution can end in different states depending on how the interleaving is done.
Execution of Parallel Programs

Example

Evaluate \([x := x + 1 || x := x \times 2]\) by

1. Evaluating \(x := x + 1\) and then \(x := x \times 2\), or
2. Evaluating \(x := x \times 2\) and then \(x := x + 1\)

We have \(\models \{x = 5\} x := x + 1 \{x = 6\} \text{ and } \models \{x = 5\} x := x \times 2 \{x = 10\}\), but
\(\models \{x = 5\} [x := x + 1 || x := x \times 2] \{x = 11 \lor x = 12\}\)?
Execution of Parallel Programs

Example

Evaluate \([x := x + 1 || y := y * 2]\) by

1. Evaluating \(x := x + 1\) and then \(y := y * 2\), or
2. Evaluating \(y := y * 2\) and then \(x := x + 1\)

We have \(|= \{x = 5\} x := x + 1 \{x = 6\}\) and
\(|= \{y = 5\} y := y * 2 \{y = 10\}\), but
\(|= \{x = 5 \land y = 5\} [x := x + 1 || y := y * 2] \{x = 6 \land y = 10\}\)
Disjoint Components

Definition (Disjoint Programs)
Let $S_1$ and $S_2$ be two while programs, $S_1$ and $S_2$ are called disjoint programs if:
- $\text{change}(S_1) \cap \text{var}(S_2) = \phi$
- $\text{var}(S_1) \cap \text{change}(S_2) = \phi$

Question: why not require
\[
\text{change}(S_1) \cap \text{change}(S_2) = \phi
\]
\[
\text{var}(S_1) \cap \text{var}(S_2) = \phi
\]
Disjoint Components Examples

Example

Given $S_1 \equiv x := z$ and $S_2 \equiv y := z$, are $S_1$ and $S_2$ disjoint?

$$\text{change}(S_1) = \{x\}, \text{change}(S_2) = \{y\}$$

$$\text{var}(S_1) = \{x, z\}, \text{var}(S_2) = \{y, z\}$$

$$\text{change}(S_1) \cap \text{var}(S_2) = \text{var}(S_1) \cap \text{change}(S_2) = \emptyset,$$ hence $S_1$ and $S_2$ are disjoint.

What about $S_1 \equiv x := z$ and $S_2 \equiv y := x$, are $S_1$ and $S_2$ disjoint?

What about $S_1 \equiv a[1] := z$ and $S_2 \equiv y := a[2]$, are $S_1$ and $S_2$ disjoint?
Disjoint Components Examples

Disjoint

\[ x := y \quad \text{and} \quad q := y \]

Disjoint

\[ x := q \quad \text{and} \quad y := q \]

Not Disjoint

\[ x := q \quad \text{and} \quad q := y \]
In the previous definition, if $S_2$ is an expression $t$, or an assertion $p$, program $S$ and expression $t$ (assertion $p$) are disjointed if $\text{change}(S) \cap \text{var}(t) = \phi$ or $\text{change}(S) \cap \text{var}(p) = \phi$. 
Statements and Programs

Parallel Statement : $[S_1||, \ldots, ||S_n]$ where $n > 1$ and $S_1, \ldots, S_n$ are pairwise disjoint.

Disjoint Parallel Programs :

$$S ::= \text{skip} \mid \quad u := t \mid \quad S_1;S_2 \mid \quad \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid \quad \text{while } B \text{ do } S_1 \text{ od} \mid \quad [S_1||, \ldots, ||S_n]$$
Interleaving Semantics

Transition Rule

$$< S_i, \sigma > \rightarrow < T_i, \tau >$$

$$< [S_1 \parallel \ldots \parallel S_i \parallel \ldots \parallel S_n], \sigma > \rightarrow < [S_1 \parallel \ldots \parallel T_i \parallel \ldots \parallel S_n], \tau >$$

where $i \in \{1, \ldots, n\}$. 
Transition Rule Examples

\[
\begin{align*}
< [x := 1||y := 2||z := 3], \sigma > & \\
\rightarrow < [E||y := 2||z := 3], \sigma[x := 1] > & \\
\rightarrow < [E||E||z := 3], \sigma[x := 1][y := 2] > & \\
\rightarrow < [E||E||E], \sigma[x := 1][y := 2][z := 3] > & \\
\rightarrow < E, \sigma[x := 1][y := 2][z := 3] > &
\end{align*}
\]
Lemma (Absence of Blocking)

Every configuration \(< S, \sigma >\) with \(S \not\equiv E\) and a proper state \(\sigma\) has a successor configuration in the transition relation \(\rightarrow\).
Correctness Semantics

- **Partial Correctness Semantics**
  \[ M[S] : \Sigma \rightarrow \mathcal{P}(\Sigma) \]
  with \( M[S](\sigma) = \{ \tau \mid < S, \sigma > \rightarrow^* < E, \tau > \} \).

- **Total Correctness Semantics**
  \[ M_{tot}[S] : \Sigma \rightarrow \mathcal{P}(\Sigma \cup \{ \bot \}) \]
  with \( M_{tot}[S](\sigma) = M[S](\sigma) \cup \{ \bot \mid S \text{ can diverge from } \sigma \} \).
Determinism

Reconsider the example:

\[
\langle [x := 1 || y := 2 || z := 3], \sigma \rangle
\]

Is the following transition correct?

\[
\langle [x := 1 || y := 2 || z := 3], \sigma \rangle \\
\rightarrow \langle [x := 1 || E || z := 3], \sigma[y := 2] \rangle \\
\rightarrow \langle [E || E || z := 3], \sigma[y := 2][x := 1] \rangle \\
\rightarrow \langle [E || E || E], \sigma[y := 2][x := 1][z := 3] \rangle \\
\rightarrow \langle E, \sigma[y := 2][x := 1][z := 3] \rangle
\]

- How many different ways to reach \( \langle E, \tau \rangle \)?
- What about the determinism lemma given in Chapter 3?
Recall

Lemma (Determinism (Lemma 3.1))
For any `while` program $S$ and a proper state $\sigma$, there is exactly one computation of $S$ starting in state $\sigma$.

Definition (Computation)
A computation of $S$ starting in $\sigma$ is a transition sequence of $S$ starting in $\sigma$ that cannot be extended.
Need “Weaker” Determinism

For

\[
< [x := 1||y := 2||z := 3], \sigma >
\]

we can have more than one transition sequence to reach \( < E, \tau > \).

Consider

\[
x = 2 + 3 + 5
\]

\[
x = 5 + 5 \quad \quad x = 2 + 8
\]

\[
x = 10
\]

Lemma 3.1 does not hold for parallel programs!

\[\implies\] we need “weaker” determinism.
Definition (Reduction System)

Reduction system is a pair $(A, \rightarrow)$ where $A$ is a set and $\rightarrow$ is a binary relation on $A$, i.e., $\rightarrow \subseteq A \times A$. If $a \rightarrow b$ holds, we say that $a$ can be replaced by $b$. The notation $\rightarrow^\ast$ denotes the transitive reflexive closure of $\rightarrow$. 
Semantics

Diamond Property

Definition (Diamond Property)

Given a reduction system \((A, \rightarrow)\), \(\forall a, b, c \in A\), if \(a \rightarrow b \land a \rightarrow c \land b \neq c\), then \(\exists d \in A\), such that \(b \rightarrow d \land c \rightarrow d\), i.e., for \(a, b, c \in A\) and \(b \neq c\)

\[
\begin{align*}
\text{there is some } d \in A \text{ such that} \\
\begin{array}{c}
\text{a} \\
\downarrow \\
\begin{array}{c}
\text{b} \\
\downarrow \\
\text{d} \\
\end{array} \\
\downarrow \\
\begin{array}{c}
\text{c} \\
\end{array}
\end{array}
\end{align*}
\]

Shangping Ren
Disjoint Parallel Programs
March 24, 2014 19 / 55
The Diamond Lemma

For disjoint parallel program $S$, whenever

\[
< S, \sigma > < T, \tau > < S_1, \sigma_1 > < S_2, \sigma_2 >
\]

for $< S_1, \sigma_1 > \neq < S_2, \sigma_2 >$, there is some $< T, \tau >$ such that

\[
< S_1, \sigma_1 > < S_2, \sigma_2 > < T, \tau >
\]
Confluent

Definition (Confluent Property)

Given a reduction system \((A, \rightarrow)\), \(\forall a, b, c \in A\), if 
\(a \rightarrow^* b \land a \rightarrow^* c \land b \neq c\), then \(\exists d \in A\), such that \(b \rightarrow^* d \land c \rightarrow^* d\), i.e., for \(a, b, c \in A\) and \(b \neq c\)

\[ \begin{array}{c}
  a \\
  \downarrow^* \\
  b \\
  \downarrow^* \\
  d \\
  \downarrow^* \\
  c \\
  \end{array} \]

there is some \(d \in A\) such that

\[ \begin{array}{c}
  b \\
  \downarrow^* \\
  d \\
  \downarrow^* \\
  c \\
  \end{array} \]
Lemma (Confluent)

For a reduction system \((A, \rightarrow)\), if \(\rightarrow\) satisfies the diamond property, then it is confluent.

The converse of the above lemma is not true.
Infinity Lemma

Lemma (Infinity)

Given a reduction system \((A, \rightarrow)\) where \(\rightarrow\) satisfies the diamond property and elements \(a, b, c \in A\) with \(a \rightarrow b\), \(a \rightarrow c\) and \(b \neq c\). If there exists an infinite sequence \(a \rightarrow b \rightarrow \cdots\) passing through \(b\), then there exists also an infinite sequence \(a \rightarrow c \rightarrow \cdots\) passing through \(c\).
→-Maximal

**Definition (→-maximal)**

For a given reduction system \((A, \rightarrow)\), and \(b \in A\), if there is \(\nexists c \in A\) with \(b \rightarrow c\), then \(b\) is \(\rightarrow\)-maximal.

**Definition \((yield(a))\)**

Let \((A, \rightarrow)\) be a reduction system and \(a \in A\), \(yield(a)\) is defined as

\[
yield(a) = \{ b | a \rightarrow^* b \text{ and } b \text{ is } \rightarrow\text{-maximal} \} \\
\cup \{ \bot | \text{there exists an infinite sequence } a \rightarrow a_1 \rightarrow \ldots \}
\]
Lemma (Yield)

Let \((A, \rightarrow)\) be a reduction system where \(\rightarrow\) satisfies the diamond property. Then \(\forall a \in A, \text{yield}(a)\) has exactly one element.

Finally, we have

Lemma (Determinism)

For every disjoint parallel program \(S\) and proper state \(\sigma\), \(\mathcal{M}_{\text{tot}}[S](\sigma)\) has exactly one element.
Sequentialization

Definition (i/o Equivalent)
Two computations are i/o equivalent if they start in the same state and are either both infinite or both finite and then yield the same final state.

Lemma (Sequentialization Lemma)
Let $S_1, \ldots, S_n$ be pairwise disjoint while programs. The we have

$$M[[S_1 \mid \ldots \mid S_n]] = M[S_1; \ldots; S_n]$$

and

$$M_{tot}[[S_1 \mid \ldots \mid S_n]] = M_{tot}[S_1; \ldots; S_n]$$
Sequentialization Rule (Rule 23)

\[
\begin{align*}
\{p\} S_1; \cdots ; S_n \{q\} \\
\{p\}[S_1 \parallel \cdots \parallel S_n] \{q\}
\end{align*}
\]

if \( S_i \) are disjoint.

Example

Prove

\[
\models_{\text{tot}} \{x = y\}[x := x + 1 \parallel y := y + 1]\{x = y\}
\]
Disjoint Parallelism Rule (Rule 24)

where \( S_1, \ldots, S_n \) are pairwise disjoint, and
\[
\text{free}(q_i, p_i) \cap \text{change}(S_j) = \emptyset \text{ for all } i \neq j.
\]
Disjoint Parallelism Rule (Rule 24) — Where Went Wrong?

We have

- \{y = 1\}x := 0\{y = 1\}
- \{\text{true}\}\ y := 0\{\text{true}\}

then by the disjoint parallelism rule we have

\[\{y = 1\}[x := 0|\ y := 0]\{y = 1\}\]

Disjointness of the pre- and postconditions and the component programs is necessary for the disjoint parallelism rule!
Disjoint Parallelism Rule (Rule 24) — Where Went Wrong?

We have

- \( \{ y = 1 \} x := 0 \{ y = 1 \} \), and
- \( \{ \text{true} \} y := 0 \{ \text{true} \} \)

then by the disjoint parallelism rule we have

\[
\{ y = 1 \}[x := 0 || y := 0]\{ y = 1 \}
\]

Disjointness of the pre- and postconditions and the component programs is necessary for the disjoint parallelism rule!
What Does the Disjoint Parallelism Rule Imply?

Disjoint Parallelism Rule

\[ \{ p_i \} S_i \{ q_i \}, \ i \in \{ 1, \ldots, n \} \]
\[ \{ \bigwedge_{i=1}^{n} p_i \}{S_1 \parallel \cdots \parallel S_n}\{ \bigwedge_{i=1}^{n} q_i \} \]

where \( S_1, \ldots, S_n \) are pairwise disjoint, and \( free(q_i, p_i) \cap change(S_j) = \phi \) for all \( i \neq j \).

From individual component’s pre/post conditions, we can deduce system’s pre/post conditions.
Example

Can we use the previous disjoint parallelism rule (Rule 24) to prove the following Hoare Triple?

\[
\{ x = y \} [x := x + 1 || y := y + 1] \{ x = y \}
\]

Answer

The answer is no as the disjoint constraint does not hold. Any way to get around? We could introduce a new variable:

\[
\{ x = z \} x := x + 1 \{ x = z + 1 \}
\]

and

\[
\{ y = z \} y := y + 1 \{ y = z + 1 \}
\]

Then what?
Example

Can we use the previous disjoint parallelism rule (Rule 24) to prove the following Hoare Triple?

\[ \{ x = y \}[x := x + 1 \parallel y := y + 1]\{ x = y \} \]

Answer

The answer is no as the disjoint constraint does not hold. Any way to get around? We could introduce a new variable:

\[ \{ x = z \}x := x + 1\{ x = z + 1 \} \]

and

\[ \{ y = z \}y := y + 1\{ y = z + 1 \} \]

Then what?
Disjoint Parallelism Rule Examples — Cont.

The we have

\[ \{x = z \land y = z\}| [x := x + 1||y := y + 1]| \{x = z + 1 \land y = z + 1\} \]

By consequence rule: \( x = z + 1 \land y = z + 1 \rightarrow x = y \), we have

\[ \{x = z \land y = z\}| [x := x + 1||y := y + 1]| \{x = y\} \]

Do we have \( x = y \rightarrow x = z \land y = z \)?

What about

\[ \{x = y\}; z := x\{x = z \land y = z\} \]

if so, then we can have

\[ \{x = y\}; z := x; [x := x + 1||y := y + 1]| \{x = y\} \]
Auxiliary Variables

**Question**
If we have

\[
\{ x = y \} z := x ; [ x := x + 1 \| y := y + 1 ] \{ x = y \}
\]

can we throw away \( z := x \) and have

\[
\{ x = y \} [ x := x + 1 \| y := y + 1 ] \{ x = y \}
\]

**Answer**
If \( z := x \) does not have any impact on the values of any variables we are interested in, then we can ’throw it away’, and these variables are called *auxiliary variables.*
Auxiliary Variables

Question
If we have

\[
\{x = y\} z := x; [x := x + 1||y := y + 1]\{x = y\}
\]

can we throw away \(z := x\) and have

\[
\{x = y\}[x := x + 1||y := y + 1]\{x = y\}
\]

Answer
If \(z := x\) does not have any impact on the values of any variables we are interested in, then we can ’throw it away’, and these variables are called auxiliary variables.
Intuitively, auxiliary variables are those which do not have impact on variables we are interested. In other words, they should not change the program’s control flow neither the data flow:

**Purely logical variables:** are variables that do not appear in the program text at all, just in pre/post conditions (yes)

**Write-only variables:** are variables that appear only on the left-hand side of assignments, nowhere else (yes)

**Assigned with auxiliary variable:** if $a$ and $b$ are auxiliary variables, and $z := a + b$, then $z$ is auxiliary variable (why?)
What Makes a Variable Auxiliary?

Intuitively, auxiliary variables are those which do not have impact on variables we are interested. In other words, they should not change the program’s control flow neither the data flow:

**Purely logical variables:** are variables that do not appear in the program text at all, just in pre/post conditions (yes)

**Write-only variables:** are variables that appear only on the left-hand side of assignments, nowhere else (yes)

**Assigned with auxiliary variable:** if $a$ and $b$ are auxiliary variables, and $z := a + b$, then $z$ is auxiliary variable (why?)
What Makes a Variable Auxiliary?

Intuitively, auxiliary variables are those which do not have impact on variables we are interested. In other words, they should not change the program’s control flow neither the data flow:

**Purely logical variables:** are variables that do not appear in the program text at all, just in pre/post conditions (yes)

**Write-only variables:** are variables that appear only on the left-hand side of assignments, nowhere else (yes)

**Assigned with auxiliary variable:** if $a$ and $b$ are auxiliary variables, and $z := a + b$, then $z$ is auxiliary variable (why?)
What Makes a Variable Auxiliary?

Intuitively, auxiliary variables are those which do not have impact on variables we are interested. In other words, they should not change the program’s control flow neither the data flow:

Purely logical variables: are variables that do not appear in the program text at all, just in pre/post conditions (yes)

Write-only variables: are variables that appear only on the left-hand side of assignments, nowhere else (yes)

Assigned with auxiliary variable: if $a$ and $b$ are auxiliary variables, and $z := a + b$, then $z$ is auxiliary variable (why?)
Auxiliary Variables

**Definition (Auxiliary Variables)**

Let $A$ be a set of simple variables in a program $S$. We call $A$ a set of *auxiliary variables* of $S$ if each variable from $A$ occurs in $S$ only in assignments of the form $z := t$ with $z \in A$.

- No variable in $A$ can appear in an `if` or `while` test.
- For every assignment $u := t$ in the program, if $t$ uses any variable from $A$, then $u$ must be in $A$, i.e., $\text{var}(t) \cap A \neq \emptyset \rightarrow u \in A$.
  - program variable := program variable;
  - auxiliary variable := program variable + auxiliary variable;
  - program variable := auxiliary variable (wrong! Why?)
List all the possible auxiliary variable set $A$ for the following programs:

$$S_1 \equiv z := x; \ [x := x + 1 \| y := y + 1]$$

$$S_2 \equiv x := y; \ y := a+b; \ \text{if } b \geq 0 \ \text{then} \ x := x+1 \ \text{else} \ y := y+1 \ \text{fi}$$

$$S_3 \equiv x := y; \ \text{while} \ a > 1 \ \text{do} \ x := x \ast a; \ a := a - k \ \text{od}$$
We have

\[
\{x = y\} z := x; [x := x + 1\|y := y + 1]\{x = y\}
\]

Auxiliary variable set: \(A = \{z\}\). If we are only interested in the value \(x\) and \(y\), we can remove the assignment for \(z\) as it is auxiliary variable and does not influence the program. Hence, we can throw away \(z := x\) and have

\[
\{x = y\}[x := x + 1\|y := y + 1]\{x = y\}
\]
Auxiliary Variable Rule (Rule 25)

Rule 25

\[
\begin{align*}
\{p\} S \{Q\} \\
\{p\} S_0 \{q\}
\end{align*}
\]

where for some set of auxiliary variables \( A \) of \( S \) with \( \text{free}(q) \cap A = \emptyset \), and \( S_0 \) is the program resulted from \( S \) by deleting all assignments to variables in \( A \).
Deleting Assignments to Auxiliary Variables

1. Replace each assignment to an auxiliary variable by a `skip` statement.

2. Remove unnecessary `skip` statements:
   - Replace `skip ; S` or `S ; skip` with `S`;
   - Replace `if B then skip else skip fi` with `skip ;`
   - Repeat the removal steps until we cannot replace any further.
An Example of Disjoint Parallelism

Find the summation of the array values, \( SUM = \sum_{i=0}^{N-1} a[i] \), and \( N \) is an even number.

Thread \( S_1 \): Summation from \( a[0] \) to \( a[m-1] \):

\[
S_1 \equiv \text{while } i \neq m \text{do} \\
\quad x := x + a[i]; i := i + 1 \\
\text{od}
\]

Thread \( S_2 \): Summation from \( a[m] \) to \( a[N-1] \):

\[
S_2 \equiv \text{while } j \neq N \text{do} \\
\quad y := y + a[j]; j := j + 1 \\
\text{od}
\]

\[
SUM \equiv m := N/2; i := 0; j := m; x := 0; y := 0; \\
[S_1 \parallel S_2]; \\
sum := x + y
\]
An Example of Disjoint Parallelism

We want to prove:

\[
\{\text{true} \} \text{SUM}\{ \text{sum} = \sum_{k=0}^{N-1} a[k] \}
\]

Need to prove

\[
\{ i = 0 \land m = N/2 \} S_1\{ q_1 \}
\]

where \( q_1 \equiv 0 \leq i \leq m \land x = \sum_{k=0}^{i-1} a[k] \land i = m \)

and

\[
\{ m = N/2 \land j = m \} S_2\{ q_2 \}
\]

where \( q_2 \equiv m \leq j \leq N \land y = \sum_{k=0}^{j-1} a[k] \land j = N \)
An Example of Disjoint Parallelism

Then

\[ \{ p_1 \land p_2 \}[S_1 \parallel S_2]\{ q_1 \land q_2 \} \]

To complete the proof

\[
\{ \text{true} \}
\]

\[
m := N/2; \ i := 0; \ j := m; \ x := 0; \ y := 0;
\]

\[
\{ p_1 \land p_2 \}
\]

and

\[
\{ p_1 \land p_2 \} \ sum := x + y \{ \text{sum} = \sum_{k=0}^{N-1} a[k] \} \]
Example 1

Is $[x := x \times 3 || y := x + 3]$ a disjoint parallel program? Show the evaluation graph for $< [x := x \times 3 || y := x + 3], \sigma[x := 3] >$.

Solution

$$change(S_1) = \{x\} \quad var(S_1) = \{x\}$$

$$change(S_2) = \{y\} \quad var(S_2) = \{x, y\}$$

Not disjoint parallel programs.
Example 1

Is \([x := x \ast 3 || y := x + 3]\) a disjoint parallel program? Show the evaluation graph for \(\langle [x := x \ast 3 || y := x + 3], \sigma[x := 3] \rangle\).

Solution

\[
\begin{align*}
\text{change}(S_1) &= \{x\} \quad \text{var}(S_1) = \{x\} \\
\text{change}(S_2) &= \{y\} \quad \text{var}(S_2) = \{x, y\}
\end{align*}
\]

Not disjoint parallel programs.
Examples — Disjointness

Solution — cont.

\[
< [x := x \times 3 || y := x + 3], \sigma[x := 3] >
\]

\[
< [E || y := x + 3], \sigma[x := 9] > \quad < [x := x \times 3 || E], \sigma[x := 3][y := 6] >
\]

\[
< [E || E], \sigma[x := 9][y := 12] > \quad < [E || E], \sigma[x := 9][y := 6] >
\]
Example 2

Is \[ \text{while } x > 0 \text{ do skip od } || x := 0 \] a disjoint parallel program? Show the evaluation graph for 
\(< \text{while } x > 0 \text{ do skip od } || x := 0, \sigma[x := 3] >. \]

Solution

\[
\begin{align*}
\text{change}(S_1) &= \phi & \text{var}(S_1) &= \{x\}; \\
\text{change}(S_2) &= \{x\} & \text{var}(S_2) &= \{x\}
\end{align*}
\]

Not disjoint parallel programs.
Example 2

Is \( \text{while } x > 0 \text{ do skip od } \| x := 0 \) a disjoint parallel program? Show the evaluation graph for \\
\(< \text{while } x > 0 \text{ do skip od } \| x := 0 \>, \sigma[x := 3] > \).

Solution

\[
\begin{align*}
\text{change}(S_1) &= \phi \\
\text{var}(S_1) &= \{ x \} \\
\text{change}(S_2) &= \{ x \} \\
\text{var}(S_2) &= \{ x \}
\end{align*}
\]

Not disjoint parallel programs.
Examples — Disjointness

Solution — cont.

\[
\begin{align*}
\langle \text{while } x > 0 \text{ do skip od} \parallel x := 0 \rangle, \sigma[x := 3] > & \\
\langle \text{skip} ; \text{while } x > 0 \text{ do skip od} \parallel x := 0 \rangle, \sigma[x := 3] > & \\
\downarrow & \\
\langle \text{skip} ; \text{while } x > 0 \text{ do skip od} \parallel E \rangle, \sigma[x := 0] > & \\
\downarrow & \\
\langle \text{while } x > 0 \text{ do skip od} \parallel E \rangle, \sigma[x := 0] > & \\
\downarrow & \\
\langle E \parallel E \rangle, \sigma[x := 0] > & 
\end{align*}
\]
Example 3

Is \([a := \text{false}; \text{if } a \text{ then } x := 3 \text{ else } y := 4 \text{ fi} \ || x := 0]\) a disjoint parallel program? Show the evaluation graph for the program in a state \(\sigma[a := \text{true}]\)

Solution

\[
\begin{align*}
\text{change}(S_1) &= \{a, x, y\} & \text{var}(S_1) &= \{a, x, y\}; \\
\text{change}(S_2) &= \{x\} & \text{var}(S_2) &= \{x\}
\end{align*}
\]

Not disjoint parallel programs.
Examples — Disjointness

Example 3

Is \([a := \text{false}; \text{if } a \text{ then } x := 3 \text{ else } y := 4 \text{ fi} || x := 0]\) a disjoint parallel program? Show the evaluation graph for the program in a state \(\sigma[a := \text{true}]\)

Solution

\[
\begin{align*}
\text{change}(S_1) &= \{a, x, y\} & \text{var}(S_1) &= \{a, x, y\}; \\
\text{change}(S_2) &= \{x\} & \text{var}(S_2) &= \{x\}
\end{align*}
\]

Not disjoint parallel programs.
Example 4: Which of the following collections of triples are parallel disjoint?

\[
\begin{align*}
\{\text{true}\} & \ x := y \ast 3 \{\text{true}\} \\
\{y > 0\} & \ x := y + 3 \{x > 0\} \\
\{\text{true}\} & \ x := x \ast 2 \{x \text{ is even}\} \\
\{x \text{ is even}\} & \ y := x + 3 \{y \text{ is odd}\}
\end{align*}
\]
Examples — Disjointness

Example 4: Which of the following collections of triples are parallel disjoint?

\[
\begin{align*}
\{ \text{true} \} & \ x := 3 \{ x > 0 \} \\
\{ y \neq 0 \} & \text{if } y < a \text{ then } x := 2 \text{ else } x := 5 \text{ fi } \{ x > 0 \}
\end{align*}
\]

\[
\begin{align*}
\{ \forall x > 0 : g(x, y) > y \} & \ u := v; \ v := g(6, y) \{ v > u \}, \\
\{ x > 0 \} & \ x := x + 3 \{ x > 0 \}, \\
\{ y > 0 \} & \ z := z + 3; \ z := z \ast 2 \{ y > 0 \},
\end{align*}
\]

Shangping Ren

Disjoint Parallel Programs

March 24, 2014

49 / 55
Example 5: Prove the Correctness of Hoare Triples

Prove \( \{ a > 0 \} [x := a || y := a + 1 || z := a + 2] \{0 < x < y < z \} \) using the disjoint parallelism rule and auxiliary variable rule (if needed).

Proof:

\[
\begin{align*}
\{ a > 0 \land a = a \} x &:= a \{ a > 0 \land x = a \} \\
\{ a + 1 = a + 1 \} y &:= a + 1 \{ y = a + 1 \} \\
\{ a + 2 = a + 2 \} z &:= a + 2 \{ z = a + 2 \}
\end{align*}
\]

As \( S_j \) does not change \( p_i \) or \( q_i \), and \( S_i \) and \( S_j \) are pairwise disjoint for \( i \neq j \), hence, the disjoint parallelism rule can be directly applied. We have

\[
\begin{align*}
\{ a > 0 \} [x := a || y := a + 1 || z := a + z] \{ x = a \land y = a + 1 \land z = a + 2 \} \\
i.e., \{ a > 0 \} [x := a || y := a + 1 || z := a + z] \{ 0 < a = x < y < z \}.
\end{align*}
\]
Example 5: Prove the Correctness of Hoare Triples

Prove $\{ a > 0 \}[x := a||y := a + 1||z := a + 2]\{0 < x < y < z\}$ using the disjoint parallelism rule and auxiliary variable rule (if needed).

Proof:

\[
\begin{align*}
\{ a > 0 \land a = a \} x &:= a \{ a > 0 \land x = a \} \\
\{ a + 1 = a + 1 \} y &:= a + 1 \{ y = a + 1 \} \\
\{ a + 2 = a + 2 \} z &:= a + 2 \{ z = a + 2 \}
\end{align*}
\]

As $S_j$ does not change $p_i$ or $q_i$, and $S_i$ and $S_j$ are pairwise disjoint for $i \neq j$, hence, the disjoint parallelism rule can be directly applied. We have

$\{ a > 0 \}[x := a||y := a + 1||z := a + z]\{x = a \land y = a + 1 \land z = a + 2\}$

i.e.,

$\{ a > 0 \}[x := a||y := a + 1||z := a + z]\{0 < a = x < y < z\}$. 
Example 5: Prove the Correctness of Hoare Triples

\[
\{ a = 1 \land x = y \} \\
[ \text{if } a > 0 \text{ then } x := x + 1 \text{ else } z := x + 1 \text{ fi } || y := y + 1] \\
\{ x = y \}
\]

using the disjoint parallelism rule and auxiliary variable rule (if needed).
Proof: We cannot directly use disjoint parallelism rule. Add auxiliary variable \( w \), and if we can prove the following, by the auxiliary variable rule, we prove the original problem.

\[
\{ a = 1 \land x = y \} \\
w := x; [\text{if } a > 0 \text{ then } x := x + 1 \text{ else } z := x + 1 \text{ fi } || y := y + 1] \\
\{x = y\}
\]

In order to use disjoint parallelism rule, we need to come up with individual Hoare triples:

\[
\{ y = w \} \\
y := y + 1 \\
\{y = w + 1\}
\]

and
Examples — Proof Outline — cont.

\[
\{ a = 1 \land x = w \}\]  
if \( a > 0 \) then \( x := x + 1 \) else \( z := x + 1 \) fi 
\{ x = w + 1 \}

To prove the above Hoare Triple, we need to prove

\[
\{ a = 1 \land x = w \land a > 0 \}\]  
\( x := x + 1 \)  
\{ x = w + 1 \}

and

\[
\{ a = 1 \land x = w \land a \leq 0 \}\]  
\( z := x + 1 \)  
\{ x = w + 1 \}
Hence we have

\[
\{ a = 1 \land x = w \land y = w \} \\
[\text{if } a > 0 \text{ then } x := x + 1 \text{ else } z := x + 1 \text{ fi } || y := y + 1] \\
\{ x = w + 1 \land y = w + 1 \}
\]

i.e.,

\[
\{ a = 1 \land x = w \land y = w \} \\
[\text{if } a > 0 \text{ then } x := x + 1 \text{ else } z := x + 1 \text{ fi } || y := y + 1] \\
\{ x = y \}
\]
Last, we need to prove

\[ \{ a = 1 \land x = y \} w := x \{ a = 1 \land x = w \land y = w \} \]