Disjoint Programs

CS 536: Science of Programming, Spring 2022

A. Why

- Parallel programs are harder to reason about because parts of a parallel program can interfere with other parts.
- Reducing the amount of interference between threads lets us reason about parallel programs by combining the proofs of the individual threads.
- Disjoint parallel programs ensure that no thread can interfere with the execution of another thread.
- The sequentialization rule (though imperfect) gives us a way to prove the correctness of disjoint parallel programs.

B. Objectives

At the end of this work you should be able to

- Draw evaluation graphs for parallel programs.
- Recognize disjoint parallel programs and correctness triples
- Use the rule for sequentialization of parallel programs

C. Questions

1. What are *vars(S)*, *change(S)*, and how are they used in the definition of "a pair of disjoint programs"?
2. What is a disjoint parallel program? What kind of parallel computation does it model?
3. What are the diamond and confluence/Church-Rosser properties and what do they imply about the evaluation graph for a disjoint parallel program?
4. What is the sequentialization proof rule for disjoint parallel programs?
5. Consider the two programs below:
   - *if x then { a := y+1 } else { b := y*2 }*
   - *if ¬x then { a := y² } else { b := y² }*

   a. Using the syntactic test with *change(...)* and *vars(...)*, are the two programs disjoint?

j	*k*	*Change j*	*Vars k*	*j Interfere With k*
1	2			
2	1			

b. Now, studying the programs from a semantic standpoint, can the two programs interfere if they're run in parallel? (Here that would mean both programs change *a* or change *b*.)

6. If we run $\langle [s_1 \parallel s_2], \sigma \rangle$ where thread 1 interferes with thread 2, what is the inevitability of semantic interference? Is it guaranteed to occur for any σ? Some σ? Some execution path? Every execution path?

Solution to Practice 23

CS 536: Science of Programming, Fall 2021

Class 23: Disjoint Parallel Programs

1. *vars(S)* and *change(S)* = are the sets of variables that appear in *S* or in (left-hand sides of) assignments in *S* respectively. Threads *S* and *S′* are disjoint if neither can change the variables used by the other: $change(S) \cap vars(S') = change(S') \cap vars(S) = \emptyset$.

2. A disjoint parallel program has pairwise disjoint threads. It models computations on *n* different processors that can share readable memory but not writeable memory.

3. The diamond property says that if $\langle S, \sigma \rangle \rightarrow$ both $\langle T_1, \sigma_1 \rangle$ and $\langle T_2, \sigma_2 \rangle$, then $\langle T_1, \sigma_1 \rangle$ and $\langle T_2, \sigma_2 \rangle$ both $\rightarrow$ some $\langle T, \tau \rangle$. Confluence/Church-Rosser replaces $\rightarrow$ by $\rightarrow^*$. Disjoint parallel programs have the diamond property (so are confluent) and have a unique result state; the evaluation graph has a unique sink node.

4. The Sequentialization rule says that if $S_1, \ldots, S_n$ are pairwise disjoint, then the truth of $\{p\}\ S_1; \ldots; S_n\ \{q\}$ implies the truth of $\{p\}\ [S_1 \| \ldots \| S_n]\ \{q\}$. ("Truth" here means satisfaction or validity under partial or total correctness.)

5. (Possibly disjoint programs)
   - *if x then a := y+1 else b := y*2 fi*
   - *if ¬x then a := y² else b := y² fi*

   a. No, the programs are not disjoint: Both have change sets with *a,* and *b* and variables used sets with *a,* b, and *y*.

j	*k*	*Change j*	*Vars k*	Interfere?
1	2	*a b*	*a b y*	Yes
2	1	*a b*	*a b y*	Yes

b. From a semantic standpoint, the programs don't interfere. If x is true, thread 1 sets *a* and thread 2 sets *b*; if x is false, thread 1 sets *b* and thread 2 sets *a*.

6. If all we know about two programs is that they aren't parallel disjoint, we can't say very much about whether or not interference will occur at runtime. It's not hard to create examples that never interfere (like the one in problem 5) or always interfere or only sometimes interfere (depending on the start state).

