Universitetet i Oslo
Institutt for Informatikk
PMA
Einar Broch Johnsen, Martin Steffen

INF 4140: Models of Concurrency

Høst 2014 **Handout 4** 19. 11. 2014

Handout 4 PL

Issued: 19. 11. 2014

In this handout, we simply collect all the rules we had during the lecture (in various parts), for easy reference. They had partly appeared already in earlier handouts.

1 Sequential reasoning

Table 1 covers the rules for the sequential core language. Assign and Skip are rules without premise (or a premise assumed "true"), i.e., those two rules are thus also called *axioms.* There's one rule per construct of the language plus one rule, which is *independent* from the language at hand: that's the rule Consequence (also known as weakening or adaptation rule). In the rules as given, only the one-armed condition (without an else-case) is covered (by rule Cond′). For the two-armed if-then-else, there exists and corresponding rule Cond, similar to Cond, which is not shown here.

$$\{\, P_{x\leftarrow e} \,\}\ x := e\ \{\, P \,\}\quad \textsc{Assign} \qquad\qquad \{\, P \,\}\ \texttt{skip}\ \{\, P \,\}\quad \textsc{Skip}$$

$$\frac{\{\, P \,\}\ S_1\ \{\, R \,\} \qquad \{\, R \,\}\ S_2\ \{\, Q \,\}}{\{\, P \,\}\ S_1; S_2\ \{\, Q \,\}}\ \textsc{Seq}$$

$$\frac{\{\, P \wedge B \,\}\ S\ \{\, Q \,\} \qquad P \wedge \neg B \Rightarrow Q}{\{\, P \,\}\ \texttt{if}\ B\ \texttt{then}\ S\ \{\, Q \,\}}\ \textsc{Cond}'$$

$$\frac{\{\, I \wedge B \,\}\ S\ \{\, I \,\}}{\{\, I \,\}\ \texttt{while}\ B\ \texttt{do}\ S\ \{\, I \wedge \neg B \,\}}\ \textsc{While}$$

$$\frac{\{\, P \,\}\ S\ \{\, Q \,\} \qquad P' \Rightarrow P \qquad Q \Rightarrow Q'}{\{\, P' \,\}\ S\ \{\, Q' \,\}}\ \textsc{Consequence}$$

Table 1: Rules for the core sequential language

2 Concurrency

The rules dealing with the basic concurrect constructs (for shared-variable concurrency) are shown in Table 2. The treatment of await-statement is straightforward (cf. rule AWAIT). The reason why it's rather simple is that its body is executed *atomically.* So, since S is protected, programming with with the await-statement is very simple, inside the conditional critical section, in that one can ignore concurrency. As a consequence, also the corresponding proof rule works without needing to take care of concurrency. Concurrency becomes problematic, of course, in the rule of the fork and join construct (using `co` and `oc`), see rule COOC. The rule itself looks simple and easy to understand. The complexity likes in the premises where *interference freedom* needs to be established. In practice, to *find* proof-conditions annotating the given program such that interference freedom hold can be complex. Even being given a proof-ouline which actually is interference-free, establishing that fact requires a certain amout of combinatorial checks (due to the possible interleavings).

$$\frac{\{\ P \wedge B\ \}\ S\ \{\ Q\ \}}{\{\ P\ \}\ \langle \texttt{await}(B)\ S\rangle\ \{\ Q\ \}}\ \textsc{Await}$$

$$\frac{\{\ P_i\ \}\ S_i\ \{\ Q_i\ \}\quad \text{are } \textit{interference free}}{\{\ P_1 \wedge \ldots \wedge P_n\ \}\ \texttt{co}\, S_1 \parallel \ldots \parallel S_n\, \texttt{oc}\ \{\ Q_1 \wedge \ldots \wedge Q_n\ \}}\ \textsc{Cooc}$$

Table 2: Rules for synchronization and concurrency

3 Monitors

In the case of monitors, we were largely interested in reasoning internally to the monitor, in particular, specifying the intended behavior of the monitor, at least the *safety* aspects of the behavior, in the form of an internal invariant, which we called *monitor invariant.* The proof rules/axioms for the monitor synchronization statements are given in Table 3. Note in particular: rule WAIT specifically talks about the said monitor invariant I.

$$\{\ I_{\#cv \leftarrow (\#cv+1)}\ \}\ \texttt{wait}(cv)\ \{\ I\ \}\quad \textsc{wait}$$

$$\{\ ((\#cv = 0) \Rightarrow P) \wedge ((\#cv \neq 0) \Rightarrow P_{\#cv \leftarrow (\#cv-1)})\ \}\ \texttt{signal}(cv)\ \{\ P\ \}\quad \textsc{Signal}$$

$$\{\ P_{\#cv \leftarrow 0}\ \}\ \texttt{signal\_all}(cv)\ \{\ P\ \}\quad \textsc{SignalAll}$$

Table 3: Rules for monitor constructs

4 Asynchronous communication

In the course of the lecture, asynchronous communication and reasoing about it was looked at in the context of "agent" communcation.[1]

Finding an argument for that an asynchronously communicating progam does what it's indended to do stressed a *compositinal* approach, i.e., reasoning on a local level concentrating on one component in isolation on the one hand, and, on the other hand, combinging the local behavior of the components into a global behavior and its properties. It's crucial, when arguing about the global behavior that the internals of the communicating agents are treated as *black boxes* i.e., no details of internal variables etc. must enter the picture in the argument there. That reflects of course the fact that also when programming one component, the programmer can rely on this component, only, and it's interactions via sending and receiving with the environment.

Technically, we introduced local and gloval histories resp. local and global history variables. The local histories capture the interaction behavior of a component.

The *overall* structure of a argument for correctness involves *3* ingredients:

1. connecting the local histories to the global history.
2. *local* reasoning, in the standard manner
3. induction (also locally) to prove that the *local history invariant* is preserved by communciation.

The first part, connecting the local and the global levels (via their resp. histories) is achieve by using *projections*:

$$\mathit{legal}(H) \wedge_i \; (h_{A_i} = H/\alpha_{A_i}) \tag{1}$$

$$I(H) = \mathit{legal}(H) \wedge_i \; I_{A_i}(H/\alpha_{A_i}) \tag{2}$$

The second point, the local reasoning is covered by the rules Table 4.

$$\frac{}{\{\, Q_{h\leftarrow h;A\uparrow B:m} \,\}\ \textbf{send}\ B : m\ \{\, Q \,\}}\ \textsc{Send}$$

$$\frac{}{\{\, \forall \vec{x}\ .\ Q_{h\leftarrow h;B\downarrow A:m(\vec{x})} \,\}\ \textbf{await}\ B : m(\vec{x})\ \{\, Q \,\}}\ \textsc{Receive}_1$$

$$\frac{}{\{\, \forall \vec{x}, X\ .\ Q_{h\leftarrow h;X\downarrow A:m(\vec{x})} \,\}\ \textbf{await}\ X\,?\,m(\vec{x})\ \{\, Q \,\}}\ \textsc{Receive}_2$$

$$\frac{\{\, P_1 \,\}\ S_1\ \{\, Q \,\} \qquad \{\, P_2 \,\}\ S_2\ \{\, Q \,\}}{\{\, P_1 \wedge P_2 \,\}\ (S_1[\,]S_2)\ \{\, Q \,\}}\ \textsc{Nondet}$$

Table 4: Rules for async. communication (local reasoning)

Finally, to complete the picture, the internal behavior of the component must be glued to what is specified about the local history (in the form of the local history invariant). The base case is to establish that the invariant holds 3.[2] The other three implications cover the induction cases.

[1] There's of course also asynchronous channel communcation or other forms, only that we did not treat that using formal reasoning.

[2] That a property of histories is actually a history invariant *requires* that the property or predicate is

$$I_A(\epsilon) \tag{3}$$

$$(\ h = (h'; A{\uparrow}B : m(e)) \wedge I_A(h') \wedge Q(\vec{x}, h)\ ) \Rightarrow I_A(h) \tag{4}$$

$$(\ h = (h'; B{\downarrow}A : m(\vec{y})) \wedge I_A(h') \wedge Q(\vec{x}, h)\ ) \Rightarrow I_A(h) \tag{5}$$

$$(\ h = (h'; X{\downarrow}A : m(\vec{y})) \wedge I_A(h') \wedge Q(\vec{x}, h)\ ) \Rightarrow I_A(h) \tag{6}$$

Table 5: Induction for the local history invariant

prefix-closed. Therefore, if the base case does not hold, one has formulated an invariant that worse than wrong: it's not even an invariant at all.