Verifikation nebenläufiger Programme
Wintersemester 2004/05

Ulrich Hannemann Jan Bredereke

---

18 Assumption-Commitment-based Reasoning

Existing compositional methods can be classified as follows:

- Either a process or system is characterised regardless of any assumption about the behaviour of its parallel environment.
- Or a process or system is characterised only in so far as its environment satisfies certain assumptions (i.e., when these assumptions are violated nothing about the behaviour of that process is claimed).

The first class can be seen as a special instance of the latter by choosing the predicate *true* as the assumption about the environment (thus expressing that no assumptions are made about the environment). Now for synchronous distributed message passing the resulting compositional proof method, which is based on assumptions about the environment, is the assumption-commitment paradigm. It was discovered by Jayadev Misra and Mani Chandy in 1981 [MC81].

Formally, an assumption-commitment correctness formula (or A-C formula for short) has the form:

$$\langle A, C\rangle : \{\varphi\}\ P\ \{\psi\},$$

where P denotes a program and A, φ, ψ, C denote predicates. For an A-C formula we require that A and C are predicates whose values do not depend on the values of any program variables.

Informally, a valid A-C formula has the following meaning:

If φ holds in the initial state, including the communication history, in which P starts its execution, then

(i) C holds initially, and C holds after every communication provided A holds after all preceding communications, and

(ii) if P terminates and A holds after all previous communications (including the last one) then ψ holds in the final state including the final communication history.

Here A expresses an *assumption* describing the expected behaviour of the environment of P, C expresses a *commitment* which is guaranteed by process P itself as long as the environment does not violate assumption A, and φ and ψ express pre- and postconditions upon the state of P. In general, assumption and commitment reflect the communication interface between parallel components and do *not* refer to the local program variables of a process, whereas pre- and postconditions facilitate reasoning about sequential composition and iteration as in Hoare logic, and *do* refer to these variables. All predicates can refer to logical variables.

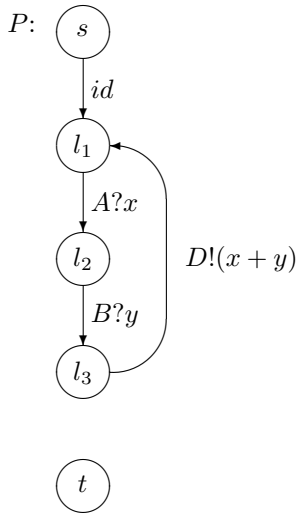

Figure 1: Structure of adder P.

Example 18.1 As an illustration of A-C-based reasoning we give a formal specification of an adder module P using distributed communication.

We have the following A-C correctness formula for P in an arbitrary environment

$$\langle true,\, \#D = \#A = \#B \geq 1 \rightarrow last(D) = last(A) + last(B)\rangle : \\ \{\#D = \#A = \#B = 0\}\ P\ \{false\}.$$

(Here, for $chan \in \mathrm{CHAN}$, $\#chan$ denotes the number of communications via channel $chan$, and $last(chan)$ refers to the latest value sent via channel $chan$.)

Note that the antecedent $\#D = \#A = \#B \geq 1$ of the implication in the commitment expresses that control of P is at l_1 (see Fig. 1). Now, in order for x and y to be properly initialised, and the value of $x + y$ to have been sent along channel D, the loop (l_1, l_2, l_3, l_1) should have been executed at least once, under the assumption that, initially, the number of recorded communications is zero, as expressed by the precondition $\#D = \#A = \#B = 0$. Since P does not terminate, its postcondition is *false*.

We want to use program P as the even number generator, but obviously we cannot unconditionally guarantee that all values sent via channel D are even. By introducing as a restriction on the environment that it always provides odd numbers along channels A and B, we obtain for P the following A-C correctness formula:

$$\langle Ass_1,\, (\#A \geq 1) \wedge (\#B \geq 1) \wedge (\#D \geq 1) \rightarrow even(last(D))\rangle : \\ \{\#D = \#A = \#B = 0\}\ P\ \{false\},$$

where

$$Ass_1 \stackrel{\text{def}}{=} (\#A > 0 \rightarrow odd(last(A))) \wedge (\#B > 0 \rightarrow odd(last(B)))$$

($odd(v)$ or even (v) states that v denotes an odd or even value), respectively. □

The new element in the A-C method is that it allows for the specification (and verification) of the ongoing interaction between a process and its environment, and therefore also applies to infinite computations and open systems. Consequently, one can also specify *reactive systems* [HP85] using this method. The A-C method captures the behaviour of these systems on the level of correctness formulae because assumption and commitment are required to hold for both finished (i.e., terminated) and unfinished (nonterminated) computations. That is, we shall introduce a prefix-closed semantics and require the assumption-commitment relationship to hold for all prefixes of a computation as described in the semantics. More precisely, this requirement not only involves the initial-final state semantics of processes, but also checking correctness of the assumption-commitment relationship after exchange of every intermediate message. In order to capture all intermediate stages of a computation, we define the semantics of a process in terms of a set of four-tuples consisting of the initial state, the current state, a termination flag, and a communication history. This representation of the semantics of a process by such four-tuples satisfies the property that for all prefixes of a communication history, whose termination flag indicates that the computation is finished, there exist four-tuples in the semantics with the same initial state and a termination flag indicating that the computation in question is unfinished.

Technically, we introduce the new concept of *A-C-inductive assertion networks* for reasoning about the sequential parts, i.e., the transition diagrams, of a concurrent system. By means of compositional proof rules such assertion networks can be used for deducing properties of the whole system. Our main technical instrument for obtaining a compositional method is again the introduction of a single *logical history* variable which records the sequence of communications generated by each component.

We slightly modify the definition of our basic program components, the sequential synchronous transition diagrams.

Definition 18.2 A sequential synchronous transition diagram is a quadruple (L, T, s, t), where L is a finite set of locations l, T is a finite set of transitions (l, a, l') with a an instruction, and s and t are the entry and exit locations, respectively, with exit location t having no outgoing edge, and entry location s *having no incoming edge.* □

Note that requiring s to have no incoming edge is no substantial restriction since we can always introduce a new entry location s' and a new transition (s', id, s). This condition simplifies the formulation of our proof method. *Composite systems* are either basic sequential synchronous transition diagrams B, sequentially composed composite systems $P_1; P_2$, or the parallel composition $P_1 \| P_2$ of two composite systems. We call a composite transition diagram also a *program* or (if it is a component of a parallel composition) a *process.*

18.1 Semantics

The definition of a compositional semantics for composite transition diagrams reflects the fact that we consider both finished and unfinished computations in the A-C formalism in contrast to, e.g., the previous section. Yet the semantics $\mathcal{O}[\![P]\!]$ for basic transition diagrams is based on the same labelled transition

relation as in Definition 15.1 which is used to define the semantics $\mathcal{O}_l(P)$ as in Session 15. For basic transition diagrams B we extend these definitions by adding a *termination flag* $\tau \in \{\top, \bot\}$, where termination for a basic transition diagram – this is indicated by setting τ to $\top$ – is characterised by the fact that its exit location t is reached.

A *computation* of a process is characterised by its initial state σ, its end state σ', its communication sequence θ and a termination flag τ where $\tau = \top$ indicates a terminated computation and $\tau = \bot$ indicates a nonterminated (or "unfinished") one. We compose two computations sequentially if the first computation is a terminated one. Parallel composition of computations is defined by requiring that the projection of the resulting communication sequences on the channels which belong to their associated processes are local communication sequences of these processes.

Definition 18.3 The compositional semantics $\mathcal{O}[\![P]\!]$ of a program P is defined as follows. For $B \equiv (L, T, s, t)$,

- $\mathcal{O}[\![B]\!] \stackrel{\text{def}}{=} \bigcup_{l \in L}\{(\sigma, \sigma', \theta, \bot) | (\sigma, \sigma', \theta) \in \mathcal{O}_l(B)\}$
  $\cup \{(\sigma, \sigma', \theta, \top) | (\sigma, \sigma', \theta) \in \mathcal{O}_t(B)\}$,

- $\mathcal{O}[\![P_1; P_2]\!] \stackrel{\text{def}}{=} \{(\sigma, \sigma_1, \theta, \bot) | (\sigma, \sigma_1, \theta, \bot) \in \mathcal{O}[\![P_1]\!]\}$
  $\cup \{(\sigma, \sigma_2, \theta, \tau) | \exists \sigma_1, \theta_1, \theta_2 . (\sigma, \sigma_1, \theta_1, \top) \in \mathcal{O}[\![P_1]\!]$
  $\wedge\, (\sigma_1, \sigma_2, \theta_2, \tau) \in \mathcal{O}[\![P_2]\!] \wedge \theta = \theta_1 \cdot \theta_2\}$,

- $\mathcal{O}[\![P_1 \| P_2]\!] \stackrel{\text{def}}{=} \{(\sigma, \sigma', \theta, \tau) | \text{for } i = 1, 2,\ (\sigma, \sigma'_i, \theta{\downarrow}P_i, \tau_i) \in \mathcal{O}[\![P_i]\!]$
  $\wedge\, \theta = \theta{\downarrow}Chan(P_1 \| P_2)$
  $\wedge\, (\tau = \top \leftrightarrow (\tau_1 = \top \wedge \tau_2 = \top))$,
  where
  $$\sigma'(x) = \begin{cases} \sigma'_1(x), & \text{if } x \in var(P_1), \\ \sigma'_2(x), & \text{if } x \in var(P_2), \\ \sigma(x), & \text{otherwise}\}. \end{cases}$$

Note that, due to the condition $\theta = \theta{\downarrow}Chan(P_1 \| P_2)$ in the definition of $\mathcal{O}[\![P_1 \| P_2]\!]$, θ does not contain communications along channels not occurring in $P_1 \| P_2$.

It is easy to see that the semantic operation of parallel composition defined above is commutative and associative.

Observe that in the above definition the requirement that a local history of a basic transition diagram can be obtained as the projection of one global history θ guarantees that an input on a channel is indeed synchronised with the corresponding output. Observe also that for basic transition diagrams only terminated computations are marked with $\top$. Finally observe that only when *both* P_1 and P_2 have reached their exit locations, is the resulting computation of $P_1 \| P_2$ marked as terminated.

Note that the $\mathcal{O}[\![P]\!]$ semantics also contains nonterminated computations. These are needed because the validity (or truth) of an A-C correctness formula for a transition diagram P requires $\mathcal{O}[\![P]\!]$ to contain all prefixes of communication sequences of P. A process P satisfies (A, C) provided that P's environment must violate A *before* P can violate C, i.e., *at any stage of an on-going computation* P's actions should satisfy C *as long as* A *has been satisfied before by*

P's environment. This is mathematically expressed by requiring (A, C) to be satisfied by all prefixes of a computation (precisely, by the prefixes of communication sequences) of P. This we obtain for a basic transition diagram P by defining $\mathcal{O}[\![P]\!]$ as the union of all unfinished and terminated computations, and for a composed transition diagram P by observing that the definitions of ";" and "$\|$" given above preserve prefix closure. Note that we consider $(\sigma, \sigma', \theta, \bot)$ as the prefix of $(\sigma, \sigma', \theta, \top)$.

18.2 Validity

Recall that an A-C correctness formula has the form

$$\langle A, C\rangle : \{\varphi\}\ P\ \{\psi\},$$

where A and C are trace predicates and φ and ψ ordinary predicates. A trace predicate A is a predicate which involves no program variables $\bar{x} \subseteq Pvar$; its satisfaction depends only on the communication sequence which is recorded in the value of h and on its logical variables. For a trace predicate A we have that for all σ and σ',

$$\sigma \models A \ \Leftrightarrow\ \sigma' \models A \text{ iff } \sigma(x) = \sigma'(x),\ \text{for } x \in Lvar \cup \{h\}. \qquad \square$$

Formally, validity of A-C formulae is defined as below.

Definition 18.4 (Validity)

$$\models \langle A, C\rangle : \{\varphi\}\ P\ \{\psi\} \ \text{ if}$$
$$\forall(\sigma, \sigma', \theta, \tau) \in \mathcal{O}[\![P]\!]\,.$$
$$\sigma \models \varphi \Rightarrow$$
$$((\forall\theta' \prec \theta.(\sigma : h \mapsto \sigma(h) \cdot \theta') \models A) \Rightarrow (\sigma : h \mapsto \sigma(h) \cdot \theta) \models C) \wedge$$
$$((\tau = \top \wedge (\forall\theta' \preceq \theta.(\sigma : h \mapsto \sigma(h) \cdot \theta') \models A)) \Rightarrow (\sigma' : h \mapsto \sigma(h) \cdot \theta) \models \psi). \qquad \square$$

We need to explain why this formal definition covers the informal one given above.

The sub-formula

$$((\forall\theta' \prec \theta.(\sigma : h \mapsto \sigma(h) \cdot \theta') \models A) \Rightarrow (\sigma : h \mapsto \sigma(h) \cdot \theta) \models C)$$

for $\theta = \langle\rangle$ implies that $\sigma \models C$. This means that $\models \langle A, C\rangle : \{\varphi\}\ P\ \{\psi\}$ implies that if the precondition φ holds also the commitment C must hold initially.

Secondly, Definition 18.4 expresses that if φ holds in the initial state σ (with communication history $\sigma(h)$) in which P starts its execution then C must hold after every communication, say, resulting in local communication history θ, provided A holds in $\sigma(h) \cdot \theta'$ for all prefixes θ' of θ. This differs from clause (i) of the intuitive meaning of validity of an A-C formula in that A should also hold for $\sigma(h)$ (obtained by taking the empty sequence $\langle\rangle$ as prefix of θ).

In case $\sigma(h) = \langle\rangle$ one can without loss of generality require that $\sigma \models A$ holds, since this choice is left open in clause (i).

The case $\sigma(h) \neq \langle\rangle$ corresponds to sequential composition with a previous composite transition diagram P_1. By soundness and completeness of our proof method, one can safely assume that $\sigma \models A$ holds).

Clause (ii) of the intuitive meaning of $\models \langle A, C\rangle : \{\varphi\}\ P\ \{\psi\}$ is covered by Definition 18.4 on similar grounds.

References

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO Advanced Science Institutes Series F: Computer and System Sciences*, pages 477–498. Springer-Verlag, 1985.

[MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(7):417–426, 1981.