Abstract

We describe a composition rule for hierarchically composed components that may involve circular reasoning between the components. It is similar to previous work by McMillan, specialized to component level reasoning. In contrast to McMillan's work, our composition rule can be used in provers that only support safety properties (e.g. k-induction model checkers) as long as the system and component contracts consist of state invariants. The composition rule still holds for richer contracts, but the resulting verification conditions then require a general purpose model checker.

Hierarchical Circular Compositional Reasoning

Andrew Gacek, Andreas Katis, Michael W. Whalen, and Darren Cofer

12/18/2011
(revised 3/10/2014)

1 Compositional Reasoning

We partition the formal analysis of a complex system architecture into a series of verification tasks that correspond to the decomposition of the architecture. By partitioning the verification effort into proofs about each subsystem within the architecture, the analysis will scale to handle large system designs. Additionally, the approach naturally supports an architecture-based notion of requirements refinement: the properties of components necessary to prove a system-level property in effect define the requirements for those components. We call the tool that we have created for managing these proof obligations AGREE: Assume Guarantee Reasoning Environment.

The goal is to enable distributed development by establishing the formal requirements of subcomponents that are used to assemble a system architecture. If we are able to establish a system property of interest using the contracts of its components, then we have a means for performing virtual integration of components. We can use the contract of each of the components as a specification for suppliers and have a great deal of confidence that if all the suppliers meet the specifications, the integrated system will work properly.

In our composition formulation, we use past-time LTL [1]. This logic supports a uniform formulation of composition obligations that can be used for both liveness properties and safety properties. For the reasoning framework, we use the LTL operators G and U supplemented by the past time operators H (historically) and Z (in the previous instant) [4]. They are defined formally over paths σ and time instants t as follows:

$$\sigma, t \vdash G(a) \quad \text{iff} \quad \forall u, u \geq t : \sigma, u \vdash a \tag{1.1}$$
$$\sigma, t \vdash a\ U\ b \quad \text{iff} \quad \exists u, u \geq t : ((\sigma, u \vdash b) \wedge (\forall v, t \leq v < u : \sigma, v \vdash a)) \tag{1.2}$$
$$\sigma, t \vdash H(a) \quad \text{iff} \quad \forall u, u \leq t : \sigma, u \vdash a \tag{1.3}$$
$$\sigma, t \vdash Z(a) \quad \text{iff} \quad (t = 0) \vee (\sigma, t - 1 \vdash a) \tag{1.4}$$
$$\sigma, t \vdash a \vee b \quad \text{iff} \quad (\sigma, t \vdash a) \vee (\sigma, t \vdash b) \tag{1.5}$$
$$\sigma, t \vdash a \wedge b \quad \text{iff} \quad (\sigma, t \vdash a) \wedge (\sigma, t \vdash b) \tag{1.6}$$
$$\sigma, t \vdash \neg a \quad \text{iff} \quad \neg(\sigma, t \vdash a) \tag{1.7}$$
$$\sigma, t \vdash a \implies b \quad \text{iff} \quad \sigma, t \vdash \neg a \vee b \tag{1.8}$$

1.1 Verification Conditions

Formally, in our framework a component contract (hereafter just a contract) is an assume-guarantee pair (A, P), where each element of the pair is a PSL formula. Informally, we would like to say something along the lines of

> "it is always the case that if the assumption is true, then the component will ensure that the guarantee is true."

However, it may be the case that if an assumption is violated, then it is no longer possible for a component to satisfy a guarantee in the future. Instead, what should be stated is that

> "It is always the case that if the assumptions have held at all instants up to and including the current time, then the guarantee holds at the current time."

which can be formalized: $G(H(A) \implies P)$.

Components are organized hierarchically into systems as shown in the example of Figure 1. We want to be able to compose proofs starting from the leaf contracts (those whose implementation is specified outside of the architecture model) through several layers of the architecture. Each layer of the architecture is considered to be a system with inputs and outputs and containing a collection of components. A system S can be described by its own contract (A_s, P_s) plus the contracts of its components C_s, so we have $S = (A_s, P_s, C_s)$. Components "communicate" in the sense that their formulas may refer to the same variables. For a given layer, the proof

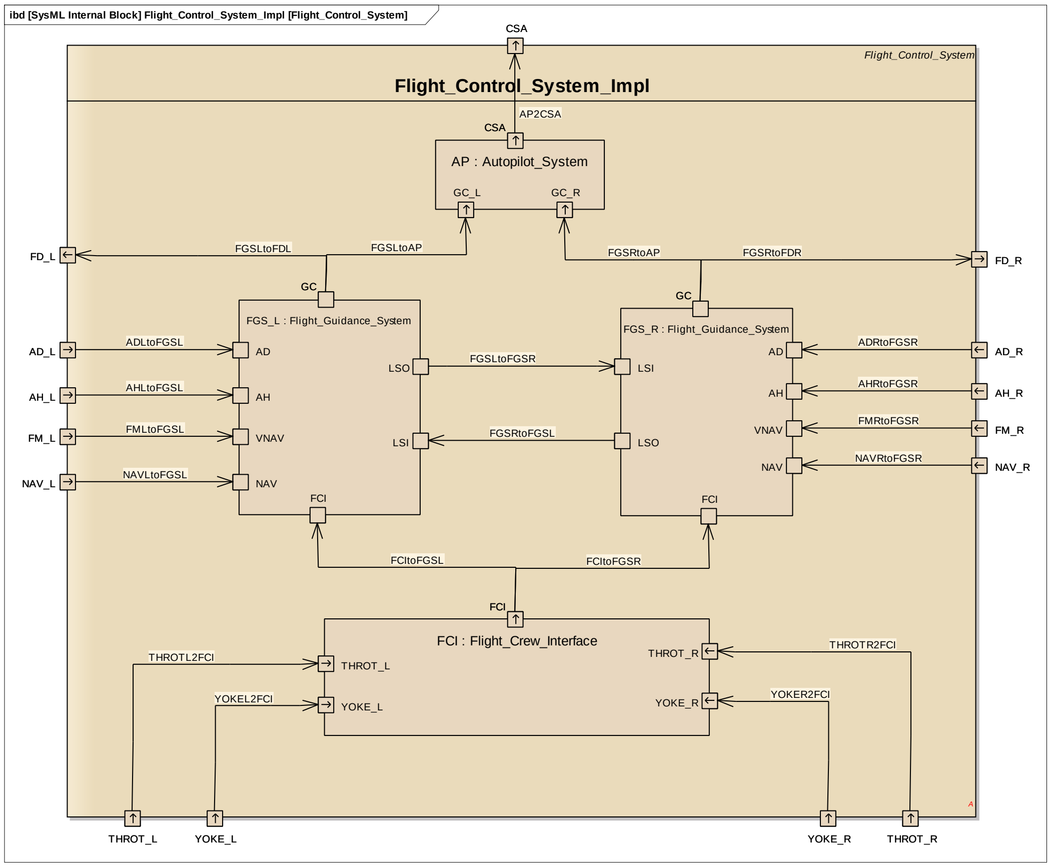

Figure 1: Example layer of an architectural model

obligation is to demonstrate that the system guarantee P_s is provable given the behavior of its subcomponents C_s and the system assumption A_s.

Our goal is to prove the formula $G(H(A_s) \implies P_s)$ given the contracted behavior $G(H(A_c) \implies P_c)$ for each component c within the system. It is conceivable that for a given system instance a sufficiently powerful model checker could prove this goal directly from the system and component assumptions. However, we take a more general approach: we establish generic verification conditions which together are sufficient to establish the goal formula. Moreover, we provide verification conditions which have the form of safety properties whenever all the assumptions and guarantees are state invariants. In such cases, this allows the verification conditions to be proved even by model checkers which are limited to safety properties, such as k-induction model checkers.

1.2 Handling Cycles in the Model

It is often the case that architectural models contain cyclic communication paths among components (e.g., the FGS_L and FGS_R in Figure 1), and that these components mutually depend on the correctness of one another. Therefore, we need to consider circular reasoning among contracts. To accomplish this reasoning, we use a framework similar to the one from Ken McMillan in [3], and break these cycles using induction over time.

We establish the correctness of the contracts of cyclic components assuming the correctness of those contracts at previous time instants. We strengthen this by establishing an order among components so that the correctness of some contracts may even be used at the current time step. Therefore, there is no circularity in our reasoning. Formally, we define a well-founded order $<$ between contracts. If $A < B$, then B can refer to A's assumptions and guarantees at the current instant, while A can refer to B's assumptions and guarantees only at previous instants.

Following McMillan, for a contract c, we define Δ_c to be the set of contracts whose assumptions and guarantees are true up until (but not including) the current time and Θ_c to be the set of contracts whose assumptions and guarantees are true up to and including the current time. We also define $c^\wedge$ for a contract c to be $(A_c \wedge P_c)$ and $C^\wedge$ to be $\{c^\wedge \mid c \in C\}$. Every element in Θ_c must be less than c according to the order $<$. However, any element $c \in C$ can be an element of Δ_c, so $\Theta_c \subseteq \Delta_c \subseteq C^\wedge$.

Theorem 1.1. *Let the following be given:*

- $S = (A_s, P_s, C_s)$ *with assumption* A_s*, guarantee* P_s *and component contracts* C_s*,*
- *a well-founded order* $<$ *on* C
- *Sets* $\Theta_c \subseteq \Delta_c \subseteq C^\wedge$*, such that* $q \in \Theta_c$ *implies* $q < c$
- *For all* $c \in C$*,* $\vdash G(H(A_c) \implies P_c)$

Then if for all $c \in C$

$\vdash G((H(A_s) \wedge Z(H(\Delta_c)) \wedge \Theta_c) \implies A_c)$

and

$\vdash G((H(A_s) \wedge H(C^\wedge)) \implies P_s)$

then

$\vdash G(H(A_s) \implies P_s)$

In other words, for a system with n components there are $n + 1$ verification conditions: one for each component and one for the system as a whole. The component verification conditions establish that the assumptions of each component are implied by the system level assumptions and the properties of its sibling components. These verification conditions are naturally cyclic, but the cycle is broken using the well-founded ordering $<$ and the one-step delay operator Z. The system level verification condition shows that the system guarantees follow from the system assumptions and the properties of each subcomponent. This is essentially an expansion of the original goal, $\vdash G(H(A_s) \implies P_s)$, with additional information obtained from each component.

Proof Sketch: It is possible to prove Theorem 1.1 directly using induction over time. The idea is, at each step, to go through each component c (from smallest to largest based on the $<$ ordering) and show that its assumptions hold in the current step. Then we can use the assumption $\vdash G(H(A_c) \implies P_c)$ to show that P_c also holds in the current step. Once we have done this for each component we can use the system level verification condition to show that the system level guarantees hold in the current step. Formally, the proof is by induction over time using the strengthened goal formula

$$\vdash G(H(A_s) \implies (H(P_s) \wedge H(C^\wedge)))$$

The desired goal formula then follows directly.

1.3 Equivalence with McMillan's Formulation

Another approach to proving Theorem 1.1 is encoding it using McMillan's circular reasoning framework. This is fairly straightforward to do. In fact, the two approaches show many similarities which provides a strong argument for the quality of approach. Since our approach is equivalent to an encoding of the problem in McMillan's framework, a reasonable question to ask would be: why bother creating a new framework? As mentioned to earlier, the answer has to do with the structure of the formalization. If all of the contracts consist of state invariants, then the verification conditions that we generate are safety properties. In such cases, we can use tools that are limited to proving safety properties.

We first introduce McMillan's formalization and adapt it to our notions of component contracts. We then show that our formalization is sound with respect to McMillan's approach. Furthermore, we show that if we assume that the components C satisfy their contracts, then the formalizations are equivalent.

McMillan's framework focuses on circular reasoning involving a set of LTL formulas Q and a set of non-circular formulas Γ that are "givens" for the circular reasoning problem. McMillan's formulation is as follows:

Theorem 1.2 (McMillan). *Let the following be given:*

- *Sets of formulas* Γ *and* Q
- *A well-founded order* $\prec$ *on* Q
- *Sets* $\Theta_q \subseteq \Delta_q \subseteq Q$*, such that* $r \in \Theta_q$ *implies* $r \prec q$

Then if for all $q \in Q$
$\vdash \Gamma \implies \neg(\Delta_q \; U \; (\Theta_q \wedge \neg q))$
then
$\vdash G(q)$
for all $q \in Q$*.*

Proof: in [3].

1.3.1 Encoding in McMillan

Our component contracts can be easily encoded in McMillan's formulation. We choose Γ to contain the component-level contracts:

$$\Gamma = \{G(H(A_c) \implies P_c) \mid c \in C\}$$

and choose Q to contain the formulas that establish the system guarantee and the component level assumptions, given the system assumptions:

$$Q = \{H(A_s) \implies P_s\} \cup \{H(A_s) \implies A_c \mid c \in C\}$$

We name the obligations q_c for the formula corresponding to $c \in C$ and q_s for the formula corresponding to the system-level property.

We create $\prec$ as follows:

$$\forall c1, c2 \in C : q_{c1} \prec q_{c2} \text{ iff } c1 < c2$$
$$\forall c \in C : q_c \prec q_s$$

If $<$ is well founded, then it is trivial to show that $\prec$ is also well founded. Finally, we create Θ_q and Δ_q as follows:

$$\begin{aligned}\forall\ q_c \in Q : \Theta_{qc} &= \{q_{c1} \mid c1 \in \Theta_c\} \\ \forall\ q_c \in Q : \Delta_{qc} &= \{q_{c1} \mid c1 \in \Delta_c\} \\ \Theta_{qs} = \Delta_{qs} &= \{q_c \mid c \in C\}\end{aligned}$$

Recall from Theorem 1.2 that we can establish $G(q)$ for each $q \in Q$, so this formulation will establish $G(H(A_s) \implies P_s)$, which is the formula that we actually care about; the other elements of Q are necessary to establish this formula.

The first step towards showing an equivalence between the McMillan formulation and Theorem 1.1 is to demonstrate an LTL equivalence between the 'until' based formulation and the past-time LTL formulation. **Note:** We use $\sigma \vdash F$ as a shorthand for $\sigma, 0 \vdash F$.

Lemma 1.3 (Until and Past-Time Equivalence)**.** *For any formulas a, b, and c and paths σ, the following are equivalent:*

$$\sigma \vdash \neg(a\ U\ (b \wedge \neg c)) \tag{1.9}$$

$$\sigma \vdash G((Z(H(a)) \wedge b) \implies c) \tag{1.10}$$

Proof. We appeal to the definitions of LTL operators described in Section 1 and derive the following:

$$\begin{aligned}(1.9)\ &\neg(\exists j, j \geq 0 : (\sigma, j \vdash b \wedge \neg c) \wedge (\forall k, 0 \leq k < j : \sigma, k \vdash a)) \\ (1.10)\ &\forall l, l \geq 0 : (\sigma, l \vdash (Z(H(a)) \wedge b) \implies c)\end{aligned}$$

Simplifying (1.9) and (1.10) we derive:

$$\forall j, j \geq 0 : ((\sigma, j \vdash \neg b \vee c) \vee (\exists k, 0 \leq k < j : \sigma, k \vdash \neg a)) \tag{1.11}$$

$$\forall l, l \geq 0 : (\sigma, l \vdash \neg(Z(H(a)) \wedge b) \vee c) \tag{1.12}$$

($\implies$). Suppose (1.11) is true. We fix l to arbitrary constant l' and choose j to match. We now can expand the LTL operator definitions (1.11) and (1.12) to the following:

$$\neg(\sigma, l' \vdash b) \vee (\sigma, l' \vdash c) \vee (\exists k, 0 \leq k < l' : \sigma, k \vdash \neg a) \tag{1.13}$$

$$\begin{aligned} &\neg(\sigma, l' \vdash b) \vee (\sigma, l' \vdash c) \vee \\ &\quad (\neg(l' = 0) \wedge (\exists m, 0 \leq m < (l' - 1) : \sigma, m \vdash \neg a)) \end{aligned} \tag{1.14}$$

We then note that the (1.14) term can be further simplified. If $l' = 0$, then the second conjunct will be trivially false, so the first conjunct is unnecessary:

$$\neg(\sigma, l' \vdash b) \vee (\sigma, l' \vdash c) \ \vee (\exists m, 0 \leq m < (l' - 1) : \sigma, m \vdash \neg a)) \tag{1.15}$$

At this point, both terms are equivalent modulo alpha conversion.

($\Longleftarrow$) The proof is exactly the same except that we fix j to arbitrary constant l' then choose l to match.

$\square$

Corollary 1.4. *For any path σ, the following verification conditions are equivalent:*

$$\sigma \vdash \Gamma \implies \neg(\Delta_q \ U \ (\Theta_q \wedge \neg q)) \tag{1.16}$$

$$\sigma \vdash \Gamma \implies (G((Z(H(\Delta_q)) \wedge \Theta_q) \implies q)) \tag{1.17}$$

$\square$

We now have a formulation quite similar to Theorem 1.1, but it is still not a safety property, due to the use of the G operator on terms in Γ on the left side of the implication. We show that this intermediate formulation is equivalent to the formulation in Theorem 1.1, which does not contain Γ on the left.

Theorem 1.5. *Let the following be given:*

- *System $S = (A_s, P_s, C_s)$ with assumption A_s, guarantee P_s and component contracts C_s,*
- *a well-founded order $<$ on C*
- *Sets $\Theta_c \subseteq \Delta_c \subseteq C^\wedge$, such that $q \in \Theta_c$ implies $q < c$*
- *For all $c \in C$, $\vdash G(H(A_c) \implies P_c)$*
- *Sets of formulas Γ and Q, defined as follows:*

$$\Gamma = \{G(H(A_c) \implies P_c) \mid c \in C\}$$
$$Q = \{H(A_s) \implies P_s\} \cup \{H(A_s) \implies A_c \mid c \in C\}$$

- *A well-founded order $\prec$ on Q, defined as follows:*

$$\forall c1, c2 \in C : q_{c1} \prec q_{c2} \;\; \textit{iff} \;\; c1 < c2$$
$$\forall c \in C : q_c \prec q_s$$

- *Sets $\Theta_q \subseteq \Delta_q \subseteq Q$, such that $r \in \Theta_q$ implies $r \prec q$, defined as follows:*

$$\forall\ q_c \in Q : \Theta_{qc} = \{q_{c1} \mid c1 \in \Theta_c\}$$
$$\forall\ q_c \in Q : \Delta_{qc} = \{q_{c1} \mid c1 \in \Delta_c\}$$
$$\Theta_{qs} = \Delta_{qs} = \{q_c \mid c \in C\}$$

- *Formulas:*

$$\begin{aligned}&((\forall c \in C : \;(\sigma \vdash G((H(A_s) \wedge Z(H(\Delta_c)) \wedge \Theta_c) \implies A_c))) \wedge \\ &(\sigma \vdash (G((H(A_s) \wedge H(C^\wedge)) \implies P_s))))\end{aligned} \tag{1.18}$$

$$(\forall q \in Q : (\sigma \vdash \Gamma \implies (G((Z(H(\Delta_q)) \wedge \Theta_q) \implies q)))) \tag{1.19}$$

Then for any path σ, (1.19) $\implies$ (1.18) (Soundness). Furthermore, if $(\sigma \vdash \Gamma)$, then (1.18) $\implies$ (1.19) (Relative Completeness).

This theorem has a long statement, but the pieces are not too difficult. The givens of the proof describe the mapping of hierarchical processes into McMillan's framework and the correlation between $<$ and $\prec$. In the conclusion, the forms of (1.18) and (1.19) just reflect the implicit quantification in the earlier definitions (Theorems 1.1 and 1.2) into explicitly quantified statements: (1.18) consists of the component obligations conjuncted with the system guarantee, and (1.19) simply quantifies over the formulas in Q.

Before we present the proof, we comment on the inequivalence between (1.19) and (1.18). Our formulation is strictly stronger than McMillan's, which is captured in the requirement that $\sigma \vdash \Gamma$ holds in the (1.18) $\implies$ (1.19) direction of the proof. Informally, the difference corresponds to how subcomponents are allowed to fail: in our statement, along a path σ, the system requirement must hold until the first time instant in which one of the subcomponents does not meet its guarantee, given its assumptions. In McMillan, if a subcomponent does not meet its guarantee at any point in time, then the system requirement need not hold; i.e., suppose that $(\sigma \vdash \Gamma)$ in formula (1.19) is false. If that is the case, there exists some component $x \in C$ such that $(\sigma, 0 \vdash G(H(A_x) \implies P_x))$ is false, so there is some time instant (possibly arbitrarily far in the future) in which A_x is historically true and P_x is false.

Fortunately, we are only interested in paths in which the components meet their guarantees (it is assumed that they do), so this inequivalence does not lead to problems in reasoning. Therefore, in this direction, we assume that $(\sigma, 0 \vdash \Gamma)$ is always true.

Proof. First, by the definition of Q, we can pair off the obligations in Q with the obligations in C and the system guarantee. We also note that we can rewrite the system guarantee portion of (1.18) equivalently as follows:

$$\sigma \vdash (G((H(A_s) \wedge Z(H(C^\wedge)) \wedge C^\wedge) \implies P_s)))$$

so as to derive a uniform structure for the mapping one-to-one between obligations in (1.18) and (1.19).

Suppose that we start from a given obligation q and its corresponding contract c. We need to show the equivalence of the following obligations:

$$\sigma \vdash \Gamma \implies (G((Z(H(\Delta_q)) \wedge \Theta_q) \implies q)) \tag{1.20}$$
$$\sigma \vdash G((H(A_s) \wedge Z(H(\Delta_c)) \wedge \Theta_c) \implies A_c) \tag{1.21}$$

We begin by expand the definitions of q, $\implies$, and the LTL operators of (1.20) and (1.21), yielding:

$$\begin{aligned}&\neg(\sigma \vdash \Gamma) \vee \\ &\quad (\forall l, l \geq 0 : (\sigma, l \vdash \neg((Z(H(\Delta_q)) \wedge \Theta_q) \vee (\neg H(A_s) \vee A_c))))\end{aligned} \tag{1.22}$$
$$\forall n, n \geq 0 : \sigma, n \vdash (H(A_s) \wedge Z(H(\Delta_c)) \wedge \Theta_c) \implies A_c) \tag{1.23}$$

We now show that (1.22) and (1.23) are equivalent by establishing implications between the formulas in either direction.

($\Longleftarrow$) Suppose that (1.23) is true. We instantiate l to a fresh constant l', then choose l' for n. Replacing LTL operators with their definitions, we derive:

$$\begin{aligned}&\neg(\sigma \vdash \Gamma) \vee \neg(\sigma, l' \vdash Z(H(\Delta_q))) \vee \neg(\sigma, l' \vdash \Theta_q) \vee \\ &\quad \neg(\sigma, l' \vdash H(A_s)) \vee (\sigma, l' \vdash A_c)\end{aligned} \tag{1.24}$$
$$\neg(\sigma, l' \vdash H(A_s)) \vee \neg(\sigma, l' \vdash Z(H(\Delta_c))) \vee \neg(\sigma, l' \vdash \Theta_c) \vee (\sigma, l' \vdash A_c) \tag{1.25}$$

If $(\sigma, l'|-H(A_s))$ is false or $(\sigma, l' \vdash A_c)$ is true, then both (1.24) and (1.25) reduce to true. Thus, we assume that they are true and false, respectively, for the remainder of the proof.

Suppose that $\sigma, l' \vdash Z(H(\Delta_c))$ is false. In this case, there exists some component $x \in \Delta_c$ such that $\neg(\sigma, l' \vdash Z(H(A_x \wedge P_x)))$. Examining the definitions of Z and H, there exists some previous state $m < l'$ in which A_x is false or P_x is false. Without loss of generality, we can choose the earliest such state for m. Suppose A_x is false at m; then $\sigma, l' \vdash Z(H(H(A_s) \implies A_x))$ is false, so $\sigma, l' \vdash Z(H(q_x))$ is also false. This means that $\sigma, l' \vdash Z(H(\Delta_q))$ is false and (1.24) is true. Alternately, suppose P_x is false at m. From our assumption of m being the first state in which $(A_x \wedge P_x)$ is false, A_x must be true in all earlier states. But then $\sigma \vdash G(H(A_x) \implies P_x)$ is false, so Γ is false and (1.24) is true. Since both (1.24) and (1.25) reduce to true if $\sigma, l' \vdash Z(H(\Delta_c))$ is false, we assume it is true for the rest of the proof.

Suppose finally that $\sigma, l' \vdash \Theta_c$ is false. Examining Θ_c, there is at least one component x such that $(\sigma, l' \vdash A_x \wedge P_x)$ is false. If A_x is false, then $H(A_s) \implies A_x$ is false, so q_x false and Θ_q is false and (1.24) is true. Suppose instead that P_x is false and A_x is true. Since we have assumed $\sigma, l' \vdash Z(H(\Delta_c))$ is true, then $(P_x \wedge A_x)$ is true in all previous steps. But then $\sigma \vdash G(H(A_x) \implies P_x)$ is false, so Γ is false and (1.24) is true.

$(\implies)$ Suppose that (1.22) is true. We instantiate n to a fresh constant n', then choose n' for l. Replacing LTL operators with their definitions, we derive:

$$\begin{aligned}\neg(\sigma \vdash \Gamma) \vee \neg(\sigma, n' \vdash Z(H(\Delta_q))) \vee \neg(\sigma, n' \vdash \Theta_q) \vee \\ \neg(\sigma, n' \vdash H(A_s)) \vee (\sigma, n' \vdash A_c)\end{aligned} \tag{1.26}$$

$$\neg(\sigma, n' \vdash H(A_s)) \vee \neg(\sigma, n' \vdash Z(H(\Delta_c))) \vee \neg(\sigma, n' \vdash \Theta_c) \vee (\sigma, n' \vdash A_c) \tag{1.27}$$

From the proof statement, we assume $(\sigma \vdash \Gamma)$, so $\neg(\sigma \vdash \Gamma)$ is false and the truth of (1.22) is due to one of the other clauses. If $(\sigma, n' \vdash H(A_s))$ is false or $(\sigma, n' \vdash A_c)$ is true, then both (1.26) and (1.27) reduce to true. Thus, we assume that they are true and false, respectively, for the remainder of the proof.

Suppose that $(\sigma, n' \vdash Z(H(\Delta_q)))$ is false. In this case, there exists some component $x \in \Delta_q$ such that $\neg(\sigma, n' \vdash Z(H(q_x)))$, so q_x is false, meaning that $H(A_s) \implies A_x$ is false for a state $0 < k < n'$, and A_x is false at k.

$Z(H(A_x \wedge P_x))$ is therefore false at k, so $\sigma, n' \vdash Z(H(\Delta_c))$ is false and (1.27) reduces to true.

Suppose finally that $\sigma, n' \vdash \Theta_q$ is false. This means that there exists a component x such that q_x is false, so $(H(A_s) \implies A_x)$ is false, meaning that A_x is false at time n'. Therefore, $(\sigma, n' \vdash A_x \wedge P_x)$ is false, and (1.27) is true. $\square$

Using Corollary 1.4 and Theorem 1.5, we can immediately connect the component embedding in McMillan's original formulation with our AGREE formulation.

1.4 Encoding Verification Conditions as Invariants

In this section, we describe how the past-time LTL operators can be encoded into a state transition system. In this section, we use the synchronous language Lustre [2], for the encoding, but equivalent encodings could be created in a variety of modeling formalisms such as NuSMV [4] and SAL [5].

Synchronous dataflow languages model computation as a sequence of computations over paths (usually called *flows* in the literature). All of these languages allow reference to both the current value of a variable and the immediately previous value of a variable through a *delay* expression. In the initial step, the initial value of the delay operator must be provided.

Lustre generalizes these concepts by allowing lookback of values of *expressions* rather than just *variables*. In addition, it allows examining positions earlier than the immediately preceding instant. These generalizations add no additional expressive power to the language, but often allow a more concise formulation of computations. In Lustre, the expression performing the lookback is called `pre` and the initialization expression is called *arrow* (`->`). We can define these expressions similarly to LTL operators over paths as follows:

$$\sigma, t \vdash \texttt{pre}(e) = \begin{cases} \bot & \text{if } (t = 0), \\ \sigma, (t-1) \vdash e & \text{if } (t \neq 0) \end{cases} \tag{1.28}$$

$$\sigma, t \vdash a \texttt{ -> } b = \begin{cases} \sigma, t \vdash a & \text{if } (t = 0) \\ \sigma, t \vdash b & \text{if } (t \neq 0) \end{cases} \tag{1.29}$$

Using these operators, we can define variable x that counts up from zero as follows:

```
x = 0 -> (pre(x) + 1);
```

To encode our verification conditions, we must be able to encode the operators H and Z. Here are the definitions of those operators, copied from Section 1:

$$\begin{array}{lll} \sigma, t \vdash H(a) & \text{iff} & \forall u, u \leq t : \sigma, u \vdash a \\ \sigma, t \vdash Z(a) & \text{iff} & (t = 0) \vee (\sigma, t-1 \vdash a) \end{array}$$

The encodings are quite straightforward. We define a new variable for each instance of the H and Z operators. Suppose we had formulas $H(a)$ and $Z(b)$ where a and b are encoded in Lustre as `enc_a` and `enc_b`. We define fresh Boolean variables `H_a` and `Z_b` in Lustre. Then the encoding is as follows:

```
H_a = enc_a and (true -> pre(H_a)) ;
Z_b = true -> pre(enc_b) ;
```

It is straightforward to see that these encodings behave equivalently to the LTL operators. Following the H operator, the `H_a` equation accumulates the truth of the a expression starting from the initial step. Following the Z operator, the `Z_b` equation is initially true and thereafter is true if b is true in the previous computational step.

1.5 Mapping The Formal Framework to AADL Diagrams

TBD. Sketch: Add additional "processes" that describe the communications channels between components in the AADL model.

References

[1] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[2] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language Lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.

[3] K. L. McMillan. Circular compositional reasoning about liveness. Technical Report 1999-02, Cadence Berkeley Labs, Berkeley, CA 94704, 1999.

[4] The NuSMV Toolset: Users Manual, 2005. Available at http://nusmv.irst.itc.it/.

[5] SAL product web site. via the world-wide-web: http://www.csl.sri.com/projects/sal/.