

Contract-Based Architecture Exploration of Cyber-Physical Systems via Satisfiability Modulo Convex Programming

Yifeng Xiao and Pierluigi Nuzzo

Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, USA

{yifeng_xiao, pierluigi.nuzzo}@berkeley.edu

Abstract—Exploring system architectures that must satisfy a set of heterogeneous requirements while minimizing a cost metric is a computationally challenging task, due to the exponential growth of the design space with the number of architecture parameters and the interdependencies between architecture choices and system properties. We present a compositional architecture exploration methodology that encodes these interdependencies as assume-guarantee contracts expressed by satisfiability modulo convex programming (SMC) formulas. Inspired by the SMC paradigm, our approach coordinates integer programming for architecture selection with convex programming for requirement verification. It then introduces a two-level pruning scheme that leverages irreducible infeasible sets of convex constraints and verification counterexamples to generate infeasibility certificates that can effectively eliminate infeasible design configurations from the search space. Evaluations on representative benchmarks, including aircraft power distribution networks and reconfigurable production lines, show an order-of-magnitude speedup on large problem instances and the solution of problems on which prior methods time out, enabling tractable and scalable compositional design of complex system architectures.

I. INTRODUCTION

A system architecture can be seen as an interconnection of, possibly heterogeneous, components assembled to perform a certain function while satisfying a set of requirements. Examples are cyber-physical system (CPS) architectures for applications such as autonomous vehicles, smart grids, and industrial automation. The design of these architectures is challenging due to the vast and heterogeneous design space and the lack of systematic methodologies for ensuring the correctness and optimality of the design. Effective frameworks are needed to address multi-dimensional design spaces and account for critical system requirements like functionality, timing, safety, and energy consumption. As domain-specific formalisms, languages, and tools to address the various design aspects continue to proliferate [1], [2], achieving interoperability remains challenging.

Several approaches have been proposed to support the design of CPSs [3]. Among these, assume-guarantee (A/G) contracts have often been adopted as a robust theoretical and methodological framework for compositional reasoning, enabling effective and rigorous mechanisms for tractable requirement analysis, system verification, and design space exploration via abstractions and decompositions across heterogeneous domains [4]–[10]. In the context of design space exploration, a system architecture is often represented as a

This work was supported in part by the National Science Foundation under Grants 2514683, 2514748, and 2448886.

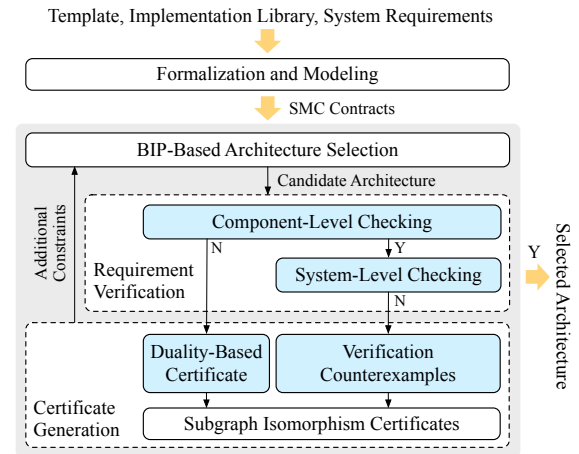


Fig. 1 Architecture exploration with SMC contracts.

parameterized graph and the design problem is formulated as a mixed integer linear program, for which increasingly more efficient encodings and solution strategies are sought for [11]–[15]. However, the computational cost of mixed integer linear programming (MILP) increases exponentially with the size of the system. Recent work has improved scalability by decomposing the architecture exploration problem into smaller sub-problems and by iteratively eliminating infeasible architecture configurations using a method based on subgraph isomorphism to reduce the search space [15]. However, this approach remains somewhat limited to contracts modeling only a pre-defined set of design aspects (e.g., timing and flow conservation) and heuristic methods to provide feedback and guide the exploration process when the contracts are violated.

This paper addresses these limitations by introducing a novel encoding where contracts are expressed using *satisfiability modulo convex programming (SMC) formulas* that can capture a broader set of design aspects and effectively model the interdependencies between architectural configurations and functional properties. SMC was showed to provide an effective decision procedure for the satisfiability of formulas over Boolean variables and convex constraints, arising in the context of many hybrid system verification and synthesis problems [16]–[19]. This paper aims to build upon the effectiveness of the SMC paradigm and extend it to also solve optimization problems, in addition to satisfiability problems, by coordinating binary integer programming (BIP) and convex programming (CP) to support CPS architecture exploration.

As shown in Fig. 1, given an architecture template (i.e., a set of components and possible connections between their ports), an implementation library, and the system requirements, we encode the requirements as SMC contracts and cast the architecture exploration problem as a BIP problem. A candidate design is then selected to minimize a cost function, and verified via a CP-based decision procedure against design requirements formulated both at the component level (i.e., consistency and compatibility among contracts) and the system level (i.e., refinement between contracts). If verification fails, infeasibility certificates derived from the identification of irreducible infeasible sets (IISs) of constraints based on duality theory or verification counterexamples are extended via subgraph isomorphism and iteratively added to prune the search space. Our contributions can be summarized as follows:

- We introduce SMC contracts for compositional architecture exploration, capturing dependencies between architectural choices and system properties.
- We extend the SMC framework to address optimization problems with respect to objective functions.
- We devise a systematic way to generate infeasibility certificates for architecture exploration using IISs and verification counterexamples.

We demonstrate the methodology on two CPS benchmarks, achieving one-order-of-magnitude speedup and the solution of problem instances on which prior methods time out.

II. PRELIMINARIES

A. Assume-Guarantee (A/G) Contracts

Let M denote a component, i.e., an element of a system, characterized by a set of variables V_M and a set of behaviors $[[M]]$ over V_M . A contract $C = (V_M, A, G)$ specifies A and G as sets of behaviors over V_M . A is the set of assumptions on the environment of M , while G is the set of guarantees provided by M , given that the assumptions are satisfied. We say that M is a valid implementation of C , i.e., $M \models C$, if all the behaviors of M are included in the guarantees given the assumptions of C , i.e., $[[M]] \subseteq G \cup A$. Meanwhile, a component E_c is a valid environment of C if all the behaviors of E_c are contained in the assumptions of C . A contract is *consistent* if and only if there exists a valid implementation, i.e., $G \cup \bar{A} \neq \emptyset$, and it is *compatible* if there exists a valid environment E_c , i.e., $A \neq \emptyset$.

The *refinement* relation supports reasoning about contract replaceability. A contract C' can be replaced by a contract C when C refines C' , written $C \preceq C'$, i.e., C has weaker assumptions and stronger guarantees. *Composition* (\otimes) of contracts can be used to build a system-level contract out of multiple component-level contracts. We use contract refinement to evaluate whether a composition of component-level contracts meets the requirements at the system level. We refer to the literature [4] for further details on these operations.

B. Satisfiability Modulo Convex Programming

SMC has been shown to be effective in tackling challenging combinatorial problems over discrete and continuous variables in the context of CPS verification and synthesis [16]–[18]. It adopts a coordination between a Boolean satisfiability

(SAT) solver and a convex solver, inspired by the “lazy” satisfiability modulo theory (SMT) paradigm [20]. The SAT solver efficiently reasons about combinations of Boolean constraints and suggests possible assignments for the convex constraints. The convex solver checks the feasibility of the given assignments and provides reasons for conflicts, i.e., infeasibility certificates, whenever inconsistencies are found. In this paper, we solve the architecture synthesis problem by decomposing it into a BIP problem and two CP problems, followed by a procedure to generate infeasibility certificates.

C. Architecture Exploration

In our framework, a *component* M_i is characterized by a set of variables $V_{M_i} = I_i \cup O_i \cup U_i \cup X_i$, where I_i and O_i are sets of input and output variables, or *ports*, respectively, U_i is a set of *attribute* variables, such as cost or performance, and X_i is a set of *internal* variables. We use \mathbf{u}_i and \mathbf{x}_i to denote the vectors collecting all the assignments to the variables in U_i and X_i , respectively. We use \mathbf{C}_i , a set of contracts, to specify the behavior of the component with respect to different design aspects, or *viewpoints*, over V_{M_i} . Similarly, the *target system* M_s can be characterized by the variables $V_{M_s} = I_s \cup O_s \cup U_s \cup X_s$, where I_s and O_s are the input and output ports of the system, respectively. \mathbf{C}_s is a set of contracts capturing the system-level requirements.

Definition 1 (System Architecture [12]). A system architecture is a directed graph $\mathcal{G} = (V, E)$, where V denotes the set of components and $E \subseteq V \times V$ is the set of interconnections between the components.

A *template* $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$ is a system architecture, where $E_{\mathcal{T}}$ is the set of all the possible interconnections between components in $V_{\mathcal{T}}$. A *library* \mathcal{L} is a set of *implementations*, i.e., concrete system components that can instantiate the components in $V_{\mathcal{T}}$. An implementation L_x is concrete, in that its attribute variables have constant values, denoted by $\bar{\mathbf{u}}_x$.

Definition 2 (Port Connection). Given a template \mathcal{T} and a target system M_s , the set of all possible port connections is defined as $\mathcal{P} \subseteq \bigcup_{M_i \in V_{\mathcal{T}}} O_i \cup I_s \times \bigcup_{M_j \in V_{\mathcal{T}}} I_j \cup O_s$, where each element $(a, b) \in \mathcal{P}$ represents a possible interconnection from an output port a to an input port b .

A component interconnection (edge) $(M_i, M_j) \in E_{\mathcal{T}}$ per Definition 1 exists if and only if there is at least one port connection (a, b) such that $a \in O_i$ and $b \in I_j$ as in Definition 2. Thus, the admissible port connections must be consistent with the admissible component interconnections in a template.

Definition 3 (Mapping). Given a template \mathcal{T} and a library \mathcal{L} , the set of mappings is defined as $\mathcal{M} \subseteq V_{\mathcal{T}} \times \mathcal{L}$, where each element $(i, x) \in \mathcal{M}$ associates a component $M_i \in V_{\mathcal{T}}$ with an implementation $L_x \in \mathcal{L}$ that instantiates the component.

Definition 4 (Component Type [12]). A *partition* $\Pi = \{\Pi_1, \Pi_2, \dots\}$ of an architecture \mathcal{G} is a set of nonempty subsets of V such that V is the disjoint union of these subsets. We say that component M_i has type k_i , e.g., it is a node of type “source” or “sink”, if and only if $M_i \in \Pi_{k_i}$, where $\Pi_{k_i} \in \Pi$.

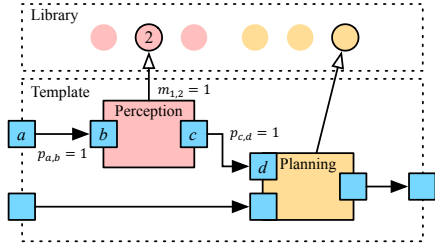


Fig. 2 System architecture example.

Then, M_i can only be mapped to an implementation of the same type, i.e., an implementation in the partition \mathcal{L}_{k_i} of \mathcal{L} .

We frame the exploration problem by introducing binary variables $p_{a,b}$ for each $(a,b) \in \mathcal{P}$ to denote the presence or absence of a port connection between ports a and b , and binary variables $m_{i,x}$ for each $(i,x) \in \mathcal{M}$ to indicate whether component M_i is mapped to implementation L_x of the same type. We denote by \mathbf{p} and \mathbf{m} the vectors of values assigned to all such variables, whose cardinalities satisfy $|\mathbf{p}| = |\mathcal{P}|$ and $|\mathbf{m}| = |\mathcal{M}|$, respectively.

Given assignments over all port connections \mathbf{p}^* and mappings \mathbf{m}^* , a *mapping architecture* is defined as a tuple $\mathcal{A} = (\mathcal{T}_{\mathcal{A}}, \mathcal{L}_{\mathcal{A}})$, where $\mathcal{T}_{\mathcal{A}}$ is a subgraph of the template \mathcal{T} and $\mathcal{L}_{\mathcal{A}} \subseteq \mathcal{L}$ is a set of implementations induced by \mathbf{p}^* and \mathbf{m}^* , respectively.

An illustrative example is provided in Fig. 2, where blue squares represent ports, circles denote implementations, filled arrows indicate a set of instantiated port connections (according to \mathbf{p}^*), empty arrows show the instantiated mappings (according to \mathbf{m}^*), and different colors correspond to different types of components.

Problem 1 (Optimized Architecture Selection). *Given a template \mathcal{T} , an implementation library \mathcal{L} , the target system M_s , the component-level contracts \mathbf{C}_i for each $M_i \in V_{\mathcal{T}}$, and the system-level contracts \mathbf{C}_s , find a mapping architecture \mathcal{A} that minimizes a cost function $f : \{0, 1\}^{|\mathcal{P}|} \times \{0, 1\}^{|\mathcal{M}|} \rightarrow \mathbb{R}$, while satisfying (refining) the contracts \mathbf{C}_s , i.e.,*

$$\underset{\mathbf{p}, \mathbf{m}}{\text{minimize}} f(\mathbf{p}, \mathbf{m}) \quad \text{s.t.} \quad \bigotimes_{M_i \in V_{\mathcal{T}, \mathcal{A}}} C_i^v \preceq C_s^v, \quad \forall v,$$

where $C_s^v \in \mathbf{C}_s$ is the v -th viewpoint contract for M_s .

In this paper, we propose a strategy to solve Problem 1 by iterating over three steps: (1) candidate architecture selection; (2) component-level analysis; and (3) system-level analysis. Each analysis consists of a verification step possibly followed by the generation of an infeasibility certificate.

III. SMC CONTRACT-BASED FORMALIZATION

We encode both the component-level and system-level requirements in terms of contracts captured by SMC formulas as follows. We recall that a *propositional logic formula* ψ is combination of atomic propositions (Boolean variables) via logical operators such as conjunction (\wedge), disjunction (\vee), and negation (\neg). A *convex constraint* c is instead an inequality of the form $c(\mathbf{x}) \leq 0$, where $c(\mathbf{x})$ is a convex function of the real variable $\mathbf{x} \in \mathcal{D} \subseteq \mathbb{R}^n$ and \mathcal{D} is a convex set. Formally, constraint c describes the set $\{\mathbf{x} \in \mathcal{D} : c(\mathbf{x}) \leq 0\}$.

Definition 5 (SMC contracts). *We define an SMC contract for component M_i as $C_i = (B_i, X_i, \mu_{A_i}, \mu_{G_i})$, where:*

- 1) B_i is the subset of binary variables in \mathbf{p} and \mathbf{m} associated with M_i .
- 2) X_i is the set of internal variables of M_i .
- 3) μ_{A_i} and μ_{G_i} are the assumptions and guarantees, respectively, defined as the sets of variable assignments satisfying the following formulas:

$$\mu_{A_i} := \bigwedge_k (\psi_{A_i,k} \rightarrow c_{A_i,k}(\mathbf{x}_i) \leq 0), \quad (1)$$

$$\mu_{G_i} := \bigwedge_k (\psi_{G_i,k} \rightarrow c_{G_i,k}(\mathbf{x}_i) \leq 0), \quad (2)$$

where $\psi_{A_i,k}$ and $\psi_{G_i,k}$ are propositional formulas over B_i , and $c_{A_i,k}$ and $c_{G_i,k}$ are convex functions over X_i .

SMC contracts enable the conditional activation of quantitative constraints based on architectural decisions. The Boolean structure captures discrete choices such as component selection and interconnection, while the convex constraints encode system properties such as timing, energy, or cost.

Example 1 (Timing Contract). *We consider a timing contract C_1 for the perception component in Fig. 2. Each port is associated with internal variables τ and t representing, respectively, the nominal time and the actual time of occurrence of an event. C_1 assumes that, if the component is implemented by L_x , the timing jitter at the input is bounded. It then requires that the timing jitter at the output be also bounded and that the end-to-end latency never exceed a maximum bound:*

$$\begin{aligned} X_1 &= \{t_b, \tau_b, t_c, \tau_c\}, \quad B_1 = \{p_{a,b}, p_{c,d}, m_{1,x}, \forall L_x \in \mathcal{L}_{k_1}\} \\ \mu_{A_1} &= \bigwedge_{L_x \in \mathcal{L}_{k_1}} ((p_{a,b} \wedge m_{1,x}) \rightarrow |t_b - \tau_b| \leq \tilde{j}_{x,b}) \\ \mu_{G_1} &= \bigwedge_{L_x \in \mathcal{L}_{k_1}} \left(((p_{c,d} \wedge m_{1,x}) \rightarrow |t_c - \tau_c| \leq \tilde{j}_{x,c}) \right. \\ &\quad \left. \wedge ((p_{a,b} \wedge p_{c,d} \wedge m_{1,x}) \rightarrow \tau_c - t_b \leq \tilde{l}_{x,(b,c)}) \right), \end{aligned}$$

where $\tilde{j}_{x,b}$ is the jitter at port b for implementation L_x , and $\tilde{l}_{x,(b,c)}$ is the maximum allowable latency for L_x to process its input signal from port b and produce an output to port c .

SMC contracts immediately capture, via implications, the relationship between certain template and mapping configurations and the associated requirements. In particular, given a mapping architecture determined by \mathbf{p}^* and \mathbf{m}^* , the formulas that form the antecedents of the implications in (1) and (2) can be immediately evaluated. The assumption and guarantee formulas in the SMC contracts can then be *simplified* as conjunctions of convex constraints, denoted by $\hat{\mu}_{A_i}$ and $\hat{\mu}_{G_i}$, depending only on the internal variables \mathbf{x}_i , leading to a simplified contract $\hat{C}_i = (X_i, \hat{\mu}_{A_i}, \hat{\mu}_{G_i})$.

Example 2 (Simplified Timing Contract). *For the perception component in Fig. 2, based on the selected mapping, the contract C_1 can be simplified as \hat{C}_1 , where:*

$$\begin{aligned} \hat{\mu}_{A_1} &= |t_b - \tau_b| \leq \tilde{j}_{2,b}, \\ \hat{\mu}_{G_1} &= (|t_c - \tau_c| \leq \tilde{j}_{2,c}) \wedge (\tau_c - t_b \leq \tilde{l}_{2,(b,c)}). \end{aligned}$$

By this simplification, compositional system analysis is facilitated by generating simplified subsystem contracts from a

system-level contract C_s , based on the connection assignments of the subsystem architectures. This decomposition enables parallel verification or synthesis of subsystems, thereby accelerating the overall design exploration process.

IV. SMC-BASED ARCHITECTURE EXPLORATION

We present our architecture exploration methodology, which tightly coordinates architecture selection over discrete variables with continuous constraint verification to efficiently prune infeasible architectures. While SMC contracts can capture general convex constraints, in this paper, we restrict our solution approach to *affine constraints*, which remain convex upon negation or inversion [16].

A. BIP-Based Architecture Selection

We build up on the SMC approach [16], which uses a SAT solver to determine whether a subset of constraints should be active, and extend it to support optimization by incorporating an objective function and casting the architecture selection problem as a BIP problem. The goal is to select mappings and port connections to minimize the overall system cost:

$$\underset{\mathbf{p}, \mathbf{m}}{\text{minimize}} f(\mathbf{p}, \mathbf{m}) \quad \text{s.t.} \quad \bigwedge_{\varphi \in \mathbb{C}_{\text{Cer}}} \varphi(\mathbf{p}, \mathbf{m}), \quad (3)$$

where \mathbb{C}_{Cer} is a set of logical constraints over the architectural variables (\mathbf{p}, \mathbf{m}) , obtained from the infeasibility certificates possibly generated during the solution process. \mathbb{C}_{Cer} is initialized as empty and updated at each iteration. Solving (3) yields optimal connections \mathbf{p}^* and mappings \mathbf{m}^* .

B. Component-Level Analysis

Given the assignments \mathbf{p}^* and \mathbf{m}^* from the solution of Problem (3), we simplify the contracts for the components in $V_{\mathcal{T}, \mathcal{A}}$ as detailed in Section III and cast feasibility problems over all the internal variables $\mathbf{x}_s, \mathbf{x}_i$, for all $M_i \in V_{\mathcal{T}, \mathcal{A}}$, to verify the component-level requirements. Let \mathbf{x} be the vector aggregating all these internal variables.

1) *Consistency and Compatibility Checking*: At the component level, we check whether the assumptions and guarantees of the system contract and of all the components' contracts can be satisfied simultaneously, ensuring that the architecture contract is both consistent (admits an implementation) and compatible (admits an environment). This verification problem is formulated as

$$\hat{\mu}_{A_s} \wedge \hat{\mu}_{G_s} \wedge \bigwedge_{M_i \in V_{\mathcal{T}, \mathcal{A}}} (\hat{\mu}_{A_i} \wedge \hat{\mu}_{G_i}). \quad (4)$$

If this formula is satisfiable, the algorithm proceeds to the system-level analysis.

2) *Duality-Based Certificate Generation*: If (4) is unsatisfiable, we aim to generate an *infeasibility certificate*, i.e., a smaller set of infeasible constraints over the architecture variables that makes the entire problem (4) infeasible. A certificate identifies a class of architectures that violates the component-level contracts and should be excluded from the search space. The smaller and more compact the certificate, the larger the class of infeasible architectures pruned out of the search space will be. We therefore seek for compact certificates by identifying an IIS out of the constraint set, defined as follows.

Definition 6 (Irreducibly Infeasible Set [21]). *A constraint set $\mathbb{C} = \{c_1, \dots, c_n\}$ is said to be an irreducibly infeasible set if it is infeasible, and removing any constraint from \mathbb{C} leads to the remaining set of constraints being feasible.*

We check the feasibility of problem (4) and look for an IIS by solving a Sum of Slacks (SSF) feasibility problem and analyzing its dual variables at optimum [22]. Given a set of constraints $\{c_1, \dots, c_n\}$, the SSF problem is formulated as

$$\underset{s_1, \dots, s_n \geq 0}{\text{minimize}} \sum_{i=1}^n s_i, \quad \text{s.t.} \quad c_1(\mathbf{x}) \leq s_1, \dots, c_n(\mathbf{x}) \leq s_n. \quad (5)$$

Assume that the set of optimal dual variables associated with the constraints in (5) is λ^* , and λ_+^* is the set of (strictly) positive optimal dual variables. We use the fact that positive optimal dual variables identify a set of constraints that contains at least one IIS. These constraints will then provide our infeasibility certificate. Because each constraint appears in the definition of a contract within a clause of the form $\psi_i \rightarrow c_i(\mathbf{x}) \leq 0$, we can exclude the infeasible architectures associated with the certificate by requiring that all the binary variables ψ_i corresponding to the constraints in the certificate must not be simultaneously true, i.e., the following clause must be true:

$$\varphi_{\text{dua}} = \bigvee_{i \in \mathcal{J}_{\text{iis}}} \neg \psi_i, \quad \mathcal{J}_{\text{iis}} = \{i : \lambda_i \in \lambda_+^*\}. \quad (6)$$

C. System-Level Analysis

For the candidate design \mathcal{A} , if all contracts are consistent and compatible, we proceed to verify whether the composite contract refines the system-level contract C_s .

1) *Refinement Checking*: We verify whether the simplified composite contract $\hat{C}_c = \hat{C}_1 \otimes \dots \otimes \hat{C}_{|V_{\mathcal{T}, \mathcal{A}}|}$ refines the simplified system-level contract \hat{C}_s by solving a pair of CP problems for the assumptions and the guarantees. Refinement holds if and only if the following problems are both unfeasible [15]:

$$\neg \hat{\mu}_{A_c} \wedge \hat{\mu}_{A_s}, \quad (7)$$

$$\hat{\mu}_{G_c} \wedge \neg \hat{\mu}_{G_s}, \quad (8)$$

where

$$\hat{\mu}_{A_c} = \bigwedge_{M_i \in V_{\mathcal{T}, \mathcal{A}}} \hat{\mu}_{A_i} \vee \bigvee_{M_i \in V_{\mathcal{T}, \mathcal{A}}} (\hat{\mu}_{A_i} \wedge \neg \hat{\mu}_{G_i}), \quad (9)$$

$$\hat{\mu}_{G_c} = \bigwedge_{M_i \in V_{\mathcal{T}, \mathcal{A}}} (\hat{\mu}_{G_i} \vee \neg \hat{\mu}_{A_i}). \quad (10)$$

Specifically, the problem in (7) can be rewritten as

$$\bigvee_{M_i \in V_{\mathcal{T}, \mathcal{A}}} \left(\neg \hat{\mu}_{A_i} \wedge \hat{\mu}_{A_s} \wedge \bigwedge_{M_i \in V_{\mathcal{T}, \mathcal{A}}} (\neg \hat{\mu}_{A_i} \vee \hat{\mu}_{G_i}) \right), \quad (11)$$

where we check whether we can find any $M_i \in V_{\mathcal{T}, \mathcal{A}}$ such that (11) is satisfiable. If not, the assumption refinement holds. A similar formulation can be used for the guarantees. If both (7) and (8) are unsatisfiable, the candidate design \mathcal{A} is returned as the optimal solution; otherwise, we generate an infeasibility certificate.

2) *Counterexample-Guided Certificate Generation*: If either (7) or (8) is feasible—indicating that the composite contract does not refine the system-level contract—we construct

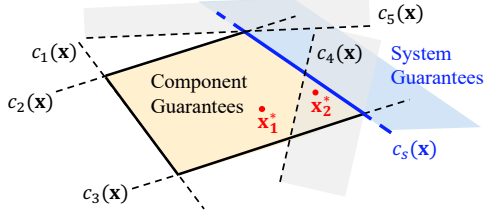


Fig. 3 Illustration of counterexample-guided certificate generation: The yellow region represents the component-level guarantees defined by $\{c_1, c_2, c_3\}$, while the blue region indicates the system-level guarantees.

certificates to eliminate assignments of \mathbf{p} and \mathbf{m} that lead to such counterexamples.

a) *Assumption Refinement Violation*: Based on (11), we formulate a CP problem for each $M_i \in V_{\mathcal{T}_A}$ to check whether

$$\neg \hat{\mu}_{A_i} \wedge \hat{\mu}_{A_s} \wedge \bigwedge_{M_i \in V_{\mathcal{T}_A}} (\neg \hat{\mu}_{A_i} \vee \hat{\mu}_{G_i})$$

is feasible. Assuming that the set \mathbf{M}_A contains all the M_i for which the problem is feasible and $\hat{\mu}_{A_i} = \bigwedge_{k \in K_{A_i}} (c_{A_i, k}(\mathbf{x}_i) \leq 0)$, we can define the index set \mathcal{J}_A as follows:

$$\mathcal{J}_A = \{(i, k) \mid M_i \in \mathbf{M}_A, k \in K_{A_i}\}. \quad (12)$$

Then, the certificate leads to the following clause:

$$\varphi_A = \bigwedge_{(i, k) \in \mathcal{J}_A} \neg \psi_{A_i, k}, \quad (13)$$

which excludes all the combinations of port connections and mappings that make (11) feasible.

b) *Guarantee Refinement Violation*: When the refinement checking problem in (8) is feasible, it produces a solution \mathbf{x}^* that serves as a counterexample for guarantee refinement. We then determine the set of component-level constraints that are violated at \mathbf{x}^* :

$$\mathbb{C}_{\text{UNSAT}} := \{c_{G_i, k} \mid c_{G_i, k}(\mathbf{x}^*) > 0, M_i \in V_{\mathcal{T}}\}. \quad (14)$$

Recalling that each constraint appears in the definition of a contract within a clause of the form $\psi_i \rightarrow c_i(\mathbf{x}) \leq 0$, we extract the Boolean conditions ψ_i associated with the violated constraints and construct the following clause,

$$\varphi_G = \bigvee_{c_i \in \mathbb{C}_{\text{UNSAT}}} \psi_i, \quad (15)$$

which selects at least one constraint violated for \mathbf{x}^* , preventing the generation of the same counterexample in future iterations.

Example 3. Consider the example shown in Fig. 3, where we solve (8) with component-level constraints $\{c_1, c_2, c_3\}$ and system-level constraint c_s and obtain a counterexample \mathbf{x}_1^* . Then, the additional set of component-level constraints violated at \mathbf{x}_1^* is $\{c_4, c_5\}$. The corresponding clause is $\varphi_G = \psi_4 \vee \psi_5$. If we instead obtain a counterexample \mathbf{x}_2^* , then we can provide a smaller $\mathbb{C}_{\text{UNSAT}} = \{c_5\}$, leading to $\varphi_G = \psi_5$.

To prune the search space efficiently, we prefer a smaller $\mathbb{C}_{\text{UNSAT}}$, i.e., a counterexample \mathbf{x}^* that violates fewer component-level constraints. A counterexample \mathbf{x}^* to (8) must satisfy $c_{G_s, k}(\mathbf{x}^*) > 0$ for all k , where $c_{G_s, k}(\mathbf{x}^*)$ quantifies the magnitude of the violation of the system-level guarantees. Smaller system-level violations may suggest that

Algorithm 1 SMC-Based Architecture Exploration

Require: Template \mathcal{T} , implementation library \mathcal{L} , target system M_s
Ensure: Selected mapping architecture \mathcal{A}

```

1:  $\mathbb{C}_{\text{Cer}} \leftarrow \{\}$ 
2: while True do
3:    $(\mathbf{p}^*, \mathbf{m}^*) \leftarrow \text{BIP\_SOLVE}(\mathcal{T}, \mathcal{L}, \mathbb{C}_{\text{Cer}})$   $\triangleright$  Solve Problem (3)
4:   if  $(\mathbf{p}^*, \mathbf{m}^*)$  is feasible then
5:      $\hat{C}_c \leftarrow \text{COMPOSE\_CONTRACT}(\mathcal{T}, \mathcal{L}, \mathbf{p}^*, \mathbf{m}^*)$ 
6:      $\hat{C}_s \leftarrow \text{SYSTEM\_CONTRACT}(M_s, \mathbf{p}^*, \mathbf{m}^*)$ 
7:      $\mathbf{s}^* \leftarrow \text{SSF\_SOLVE}(\hat{C}_c, \hat{C}_s)$   $\triangleright$  Solve Problem (5)
8:     if  $\mathbf{s}^* \neq \mathbf{0}$  then
9:       Generate  $\varphi_{\text{dua}}$  using (6)
10:       $\mathbb{C}_{\text{Cer}} \leftarrow \mathbb{C}_{\text{Cer}} \cup \{\varphi_{\text{dua}}\} \cup \text{ISO\_CERT}(\varphi_{\text{dua}}, \mathbf{p}^*, \mathbf{m}^*)$   $\triangleright$ 
          Add additional constraints based on subgraph isomorphism [15]
11:      continue
12:     else
13:        $\mathbf{x}^* \leftarrow \text{CP\_SOLVE}(\hat{\mu}_{A_c}, \hat{\mu}_{A_s})$   $\triangleright$  Solve Problem (11)
14:       if  $\mathbf{x}^*$  is feasible then
15:         Generate  $\varphi_A$  using (12) and (13)
16:          $\mathbb{C}_{\text{Cer}} \leftarrow \mathbb{C}_{\text{Cer}} \cup \{\varphi_A\} \cup \text{ISO\_CERT}(\varphi_A, \mathbf{p}^*, \mathbf{m}^*)$ 
17:         continue
18:        $\mathbf{x}^* \leftarrow \text{CP\_SOLVE}(\hat{\mu}_{G_c}, \hat{\mu}_{G_s})$   $\triangleright$  Solve Problem (16)
19:       if  $\mathbf{x}^*$  is feasible then
20:         Generate  $\varphi_G$  using (14) and (15)
21:          $\mathbb{C}_{\text{Cer}} \leftarrow \mathbb{C}_{\text{Cer}} \cup \{\varphi_G\} \cup \text{ISO\_CERT}(\varphi_G, \mathbf{p}^*, \mathbf{m}^*)$ 
22:         continue
23:       Generate  $\mathcal{A}$  from  $(\mathbf{p}^*, \mathbf{m}^*)$ 
24:       return  $\mathcal{A}$ 
25:     else
26:       break

```

more component-level constraints are closer to being satisfied, which may yield a smaller $\mathbb{C}_{\text{UNSAT}}$. We therefore compute \mathbf{x}^* by solving:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \sum_k c_{G_s, k}(\mathbf{x}) \quad \text{s.t.} \quad \hat{\mu}_{G_c} \wedge \neg \hat{\mu}_{G_s}. \quad (16)$$

φ_{dua} , φ_A , and φ_G are formulated over the binary variables \mathbf{p} and \mathbf{m} . To further reduce the search space of the BIP problem (3), we incorporate graph pruning techniques based on subgraph isomorphism [15]. The generated clauses are then added to the architectural constraint set \mathbb{C}_{Cer} . The overall procedure is summarized in Algorithm 1. The methodology is applied iteratively until a feasible mapping architecture \mathcal{A} is identified or the BIP problem becomes infeasible, indicating that no valid \mathcal{A} exists.

V. EVALUATION RESULTS

We implemented our methodology using the Python interface provided by CHASE [6] for contract formalization and modeling. We interfaced our libraries with Gurobi [23] to solve the BIP and CP problems for architecture exploration, verification, and certificate generation.

A. Aircraft Electric Power Network

We applied our method to the exploration of an aircraft electric power distribution network (EPN), which has often been used as a benchmark in the related literature [24]. An EPN delivers power from generators (GEN) to a set of loads via AC and DC buses. Rectifier units (RU) are used to convert AC power to DC power. An example of a selected EPN architecture is shown in Fig. 4(a). Components are grouped based on their locations, i.e., on the left (L) or right (R) side,

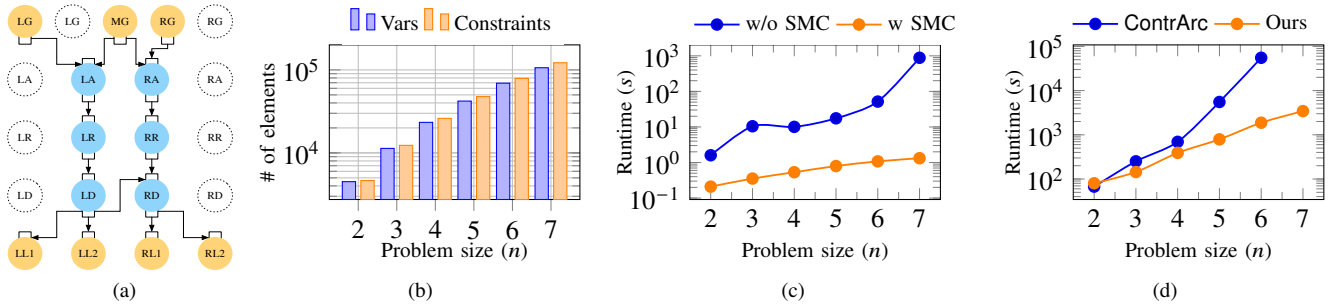


Fig. 4 EPN architecture and exploration results: (a) Example generated EPN architecture ($n = 2$); (b) Growth in the number of variables and constraints with problem size; (c) Runtime improvement due to SMC; (d) Runtime comparison with ContrArc [15].

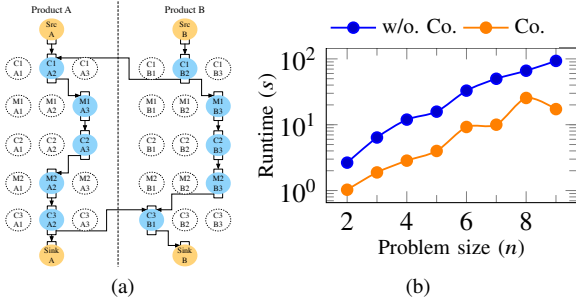


Fig. 5 RPL architecture and compositional synthesis.

with the following types: generators ($LG/RG/MG$), where MG is used to denote the auxiliary power unit (APU), AC buses (LB/RB), RUs (LR/RR), DC buses (LD/RD), and loads (LL/RL). APUs can be connected to both sides of the system. We aim to generate an architecture that satisfies both timing and power balance requirements while minimizing the overall cost. The problem size n denotes the number of components per type in the provided template. Figure 4(b) illustrates how the number of variables and constraints in the exploration problem increases with n .

We first evaluate the impact of using the SMC approach on the efficiency of component-level analysis. As shown in Figure 4(c), the approach labeled “w/o SMC” directly solves a monolithic MILP formulation to find a feasible architecture that satisfies the component-level requirements. In contrast, “w. SMC” leverages an iterative approach solving a sequence of BIPs guided by the generation of duality-based certificates to eliminate infeasible configurations. Incorporating an SMC approach significantly reduces runtime—especially as the number of components and constraints increases—highlighting the scalability of our approach.

We compare the runtime of our approach with ContrArc [15], as shown in Figure 4(d). The results demonstrate that our methodology significantly improves scalability, achieving up to $29\times$ speedup at $n = 6$ and maintaining tractability for larger problem sizes where ContrArc fails to complete within the 18-hour timeout. For smaller instances (e.g., $n = 2$ or 3), both methods show similar runtime due to the limited design space. The runtime gap between the two methods increases notably with system size, highlighting our approach’s efficiency.

B. Reconfigurable Production Line

As a second benchmark, we consider the design of a reconfigurable production line (RPL) system [15], [25]. An RPL includes a source (Src) that provides elements of a product, machines (M) that process these elements, and a sink ($Sink$). Machines are connected by conveyors (C). Each implementation in the system is labeled with a cost c and a subtype s . Nodes Src and M are characterized by a flow rate F^S and throughput F^P , respectively, and M also has a processing rate F^C . We assemble two products A and B on the left and right production lines, respectively. Each line has two machines along the path from the source to the sink. Let n be the number of machines and conveyors in the template. Figure 5(a) shows the generated architecture of the RPL system with $n = 3$.

We evaluate the effectiveness of compositional synthesis by synthesizing each production line independently under assumptions about the part processing rates of the other production line. The system must meet a throughput requirement of at least 16 parts/min (PPR), with each subsystem required to achieve at least 8 parts/min. Figure 5(b) illustrates the impact of compositional synthesis on runtime. “w/o Co.” denotes the result of a monolithic synthesis approach applied to the entire architecture, while “Co.” denotes the result of an approach that synthesizes each production line independently. Compositional synthesis achieves lower runtime as problem size n increases, highlighting its scalability benefits.

VI. CONCLUSION

We presented a scalable cyber-physical system architecture exploration methodology that encodes the architecture requirements using assume-guarantee contracts expressed as satisfiability-modulo-convex-programming (SMC) formulas. SMC contracts allow directly capturing dependencies between certain architectural choices and the associated system properties. Moreover, they facilitate the analysis of the architecture both at the component and the system level, and the generation of infeasibility certificates that can efficiently prune the search space. Evaluation on benchmarks from the related literature, show that our methodology enables tractable exploration beyond the limitations shown by prior methods. Future work includes exploiting parallelization and investigating the use of AI-assisted techniques to accelerate the architecture exploration process.

REFERENCES

- [1] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. IEEE Int. Symp. Object Orient. Real Time Distrib. Comput.*, May 2008.
- [2] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyber-physical system integration," *Proc. IEEE*, vol. 100, 2011.
- [3] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *Proc. IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. (TCAD)*, vol. 36, 2016.
- [4] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.
- [5] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proc. IEEE*, vol. 103, 2015.
- [6] P. Nuzzo, M. Lora, Y. A. Feldman, and A. L. Sangiovanni-Vincentelli, "CHASE: Contract-based requirement engineering for cyber-physical system design," in *Proc. IEEE/ACM Design Autom. Test Europe (DATE)*, 2018.
- [7] C. Leet, C. Oh, M. Lora, S. Koenig, and P. Nuzzo, "Co-design of topology, scheduling, and path planning in automated warehouses," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
- [8] —, "Task assignment, scheduling, and motion planning for automated warehouses for million product workloads," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7362–7369.
- [9] N. V. Naik, A. Pinto, and P. Nuzzo, "Contract-based hierarchical modeling and traceability of heterogeneous requirements," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4298–4309, 2024.
- [10] —, "Contract embeddings for layered control architectures," *Proc. ACM Transactions on Embedded Computing Systems (TECS)*, 2025.
- [11] D. Kirov, P. Nuzzo, R. Passerone, and A. Sangiovanni-Vincentelli, "ArchEx: An extensible framework for the exploration of cyber-physical system architectures," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, 2017.
- [12] P. Nuzzo, N. Bajaj, M. Masin, D. Kirov, R. Passerone, and A. L. Sangiovanni-Vincentelli, "Optimized selection of reliable and cost-effective safety-critical system architectures," *Proc. IEEE Trans. on Comput.-Aided Design of Integr. Circuits and Syst. (TCAD)*, 2019.
- [13] D. Kirov, P. Nuzzo, R. Passerone, and A. Sangiovanni-Vincentelli, "Optimized selection of wireless network topologies and components via efficient pruning of feasible paths," in *Proc. IEEE/ACM Design Autom. Conf. (DAC)*, 2018.
- [14] D. Kirov, P. Nuzzo, A. Sangiovanni-Vincentelli, and R. Passerone, "Efficient encodings for scalable exploration of cyber-physical system architectures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 1, pp. 30–43, 2024.
- [15] Y. Xiao, C. Oh, M. Lora, and P. Nuzzo, "Efficient exploration of cyber-physical system architectures using contracts and subgraph isomorphism," in *Proc. IEEE/ACM Design Autom. Test Europe (DATE)*, 2024, pp. 1–6.
- [16] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "SMC: Satisfiability modulo convex programming," *Proc. IEEE*, vol. 106, no. 9, pp. 1655–1679, 2018.
- [17] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. Sangiovanni-Vincentelli, "CalCS: SMT solving for non-linear convex constraints," in *Proc. Formal Methods in Computer Aided Design (FMCAD)*, 2010, pp. 71–79.
- [18] Y. Shoukry, P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "SMC: Satisfiability modulo convex optimization," in *Proc. Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*, 2017, pp. 19–28.
- [19] A. Fayyazi, Y. Xiao, P. Nuzzo, and M. Pedram, "Efficient counterexample-guided fairness verification and repair of neural networks using satisfiability modulo convex programming," in *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2025, pp. 367–375.
- [20] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of satisfiability*. IOS Press, 2021, pp. 1267–1329.
- [21] J. W. Chinneck and E. W. Dravnieks, "Locating minimal infeasible constraint sets in linear programs," *Proc. ORSA Journal on Computing*, vol. 3, no. 2, pp. 157–168, 1991.
- [22] N. Naik and P. Nuzzo, "Robustness contracts for scalable verification of neural network-enabled cyber-physical systems," in *Proc. ACM/IEEE Int. Conf. on Formal Methods and Models for System Design (MEMOCODE)*, 2020, pp. 1–12.
- [23] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: <https://www.gurobi.com>
- [24] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia, "A contract-based methodology for aircraft electric power system design," *Proc. IEEE Access*, vol. 2, pp. 1–25, 2013.
- [25] Y. Zhang, B. Xu, D. Wang, and X. Dang, "Architecture design of flexible production line control system for reconfigurable operations and maintenance," in *Proc. IEEE Int. Conf. on Cloud Computing, Big Data Application and Software Engineering (CBASE)*, 2024, pp. 880–883.