ABSTRACT

Contract models underlying architecture-level verification methods must suit a range of different accuracy vs analytical complexity tradeoffs depending on domain. For example, trustworthiness in safety-critical systems is enabled by representational simplicity leading to comprehensible proofs while real-time systems require precise characterisation of execution time. A family of mutually-compatible parameterised contract models enabling such tradeoffs is needed, supporting reasoning about consistency and conformance (replaceability) which is bidirectional (from requirements to provisions and vice versa) and parametric (context-sensitive). This paper proposes a framework for such a family. The framework extends a previous formalisation of parameterised contracts. It provides more general notions of conformance, bidirectional reasoning and parameterisation, suitable for compositional architectural analyses of software products and product lines, for which software architects do not only need checking but scope for restricting or enriching service and interface contracts in predictable and compositional ways. The family of mechanisms presented here covers a range of levels of expressiveness, spanning the established four levels of component contracts, and is worked out in detail with examples for two common existing representations—tables and finite automata.

Categories and Subject Descriptors
D.2.11 [Software Engineering]: Software Architectures; D.2.13 [Software Engineering]: Reusable Software—Reuse Models; D.2.4 [Software Engineering]: Software Program Verification—Formal Methods, Assertion Checkers, Contracts; C.4 [Performance of Systems]: Performance Attributes, Modeling Techniques

Keywords
Software Components, Software Architectures, Design by Contract, Parameterised Contracts, Architectural Dependence

1. INTRODUCTION

The need for design-time (over and above run-time) verification of component-based systems motivates contracts. Meyer’s celebrated design-by-contract method [18] defines a contract as a logical invariant on all objects of a type, which is checkable at run time, but also possibly checkable at compile time. In architectural design as supported in modern programming languages, architecture definition languages or design methods, these formulae are typically distributed to the entities at hand: classes with the class invariants, methods with pre- and post-conditions and so forth. A more general notion of contracts has also been proposed for components [5] as a kind of union of four distinct levels: syntactic, behavioural, synchronisation (in distributed/concurrent contexts) and extra-functional. A major concern is the notion of replaceability or conformance, whether a given component meets its contract, or more generally, whether one contract subsumes another.

It is now emerging that to be practical, models for component contracts must be suitable for a range of domains, enabling key simplicity vs precision (and thus analytical efficiency) tradeoffs. For example, trustworthiness in safety-critical systems is enabled by representational simplicity leading to comprehensible proofs, dynamic architectures require abstract representations enabling efficient analyses, while real-time systems require precise characterisation of execution time.

We therefore propose a formal architectural description framework providing a useful, general notion of parametric contractual conformance across a number of possible static contractual representations.

Our motivating example is the well-known production cell (cell) case study [15]. A VRML model of the cell is shown in Figure 1. As in the work we are extending [22, 23], the overall process is as follows. It is assumed the environment is responsible for placing metal blanks on a feed belt (fore-
In our work in particular we are interested in the “wicked” design problems where real parallelism, concurrency and extra-functional properties are intertwined, and derive from requirements such as functionality, timing and reliability, and can be dealt with more or less satisfactorily at the ar-

---

**Figure 1: Production Cell**

---

ground), which are in turn placed on a table which can rotate and raise to enable robot arm 1 to take it with a magnet gripper. Robot arm 1 is used to convey metal to the movable plate of a press. Arm 1 and arm 2 are both attached to a rotating base, which turns anti-clockwise to enable arm 1 to unload the metal onto the movable plate of a press. The press presses blanks by raising the plate. Robot arm 2 is used to convey metal from the press to the deposit belt. Since arm 1 and arm 2 are at different heights, the press plate must be lowered to the bottom position before arm 2 can load it. From the press the base rotates anti-clockwise to allow arm 2 to unload to the deposit belt. Finally the base rotates clockwise so that arm 1 points to the table.

We are concerned with the design and verification of the supervisory controller for the cell considered as an off-the-shelf assembly of mechatronic components. Components are assumed packaged with software which provides abstractions hiding low level details related to precise movement via more complex sensors and actuators. Integration is typically achieved via interconnection of controller components connected via controller area network buses, with co-ordination software implemented on one or more programmable logic controllers. Such a system is therefore distributed, with significant complexity arising from the intrinsic concurrency of the various mechatronic elements and their controllers and the need for synchronisation.

Previous papers have motivated a need to support design verification, in particular of consistency and conformance, with respect to safety, synchronisation and timing properties, in a context involving possibly dynamic adaptation by component or interface contract replacement [24]. For example: for safety, the base must not be over-rotated, metal blanks must not be dropped in specific locations, and for efficiency the cycle time of the cell should be minimised. For synchronisation, there are several alternatives already outlined above. For dynamic adaptation, a correct, safe notion of component and contract conformance is needed, which can if necessary be invoked on the fly, necessitating very efficient re-verification of consistency and conformance.

Significantly, what the cell can provide, in terms of cycle time and safety properties, depends fundamentally on the properties of its elements, its configuration, and the way it is used, as follows.

Although both arms can be extended and retracted, loading and unloading must be partly restricted with respect to movement of the table and press by synchronisation. However there are at least three alternative synchronisations, with different properties. Selection of these could be through either design/deployment configuration or usage (runtime configuration): Safe: The safety property that arms do not rotate extended in the presence of the press can be ensured by requiring that arm load operations be interleaved with base rotations. Fast: The cell’s worst-case cycle time (WCCT) improves under weaker synchronisation, at the cost of a more complex proof of safety properties. Single Arm: Finally, if the press has no lower position, the cell can still function in a configuration where only arm 1 is used (and e.g. the deposit belt is physically raised), albeit with a very long cycle time.

In design and composition, it is necessary to reason about a component C both forwards, from what may be available to C to what C can thus provide, as well as backwards, from what is needed of C to what C thus requires. For example, one might wish to reason (forwards) from a known-safe synchronisation to a time budget or from a restriction on provided properties (such as availability of arm 2) to feasible behaviours; or, (backwards) from a WCCT time budget to a set of restrictions on usage. Such issues motivated work on parameterised (component) contracts, introduced by Reussner [20]. In parameterised contracts, each individual component is parameterised with respect to its environment: what it can provide, and how its own requirements are met, are related. Parameterised contracts have been formalised and subsequently used as the basis for accurate characterisation of extra-functional characteristics for components, and realized as stateful models with specific characterisations of relevant properties, e.g. probability density functions with probabilistic state machines.

In the context of performance modeling, Koziolek [14] proposed that a variety of models may be needed to provide suitable tradeoffs between analytical cost and accuracy for a given domain. We envisage a spectrum of representations corresponding to such tradeoffs, which are nevertheless mutually compatible at some minimal level of abstraction.

The need for a simple unified contractual framework beyond just performance is signalled first by the diversity of quality aspects under consideration at requirements and architecture level. Typically different domains require individual specialists using different notations. Depending on requirements one may choose an individual domain-specific language based on one or more of: finite automata, concurrent/weighted automata, Markov chains, process algebras, data structures or subtyping hierarchies or (temporal) logics. Additionally in architecture (especially product line architecture), by definition, such requirements co-exist and interact with functionality (features) and reasoning about them must be co-ordinated. Also, architecture is increasingly concerned with reuse and integration of existing off the shelf components, each possibly with their own “contract” in a specific notation. Finally, as requirements propagate through an architecture dependency structure there is a ripple effect where requirements are translated by major components across domains. For example functional features may be translated into timing, or safety into reliability.

In our work in particular we are interested in the “wicked” design problems where real parallelism, concurrency and extra-functional properties are intertwined, and derive from requirements such as functionality, timing and reliability, and can be dealt with more or less satisfactorily at the ar-
chitectural level. We need a way of tracing dependencies in a unified way without wanting to combine the domain specific languages into a universal domain specific language. Our recent efforts have focused on an ADL integrating component behaviour specifications from a variety of such languages. Faced with the prospect of developing a complex unified contract language capturing a large variety of notions, we turned to an alternative approach. We seek to enable local/partial simplicity, using appropriate domain-specific languages for contracts at individual components, combined with global accuracy, enabling adequately accurate evaluation along lines of architectural dependence of compatibility and estimation of properties at the system level. Our approach in this paper is to generalize parameterised contracts for this purpose.

Prior work, despite its utility and impact, in particular on the design of the Palladio component model [4], has not led to a formal and fully generic framework for parameterised contracts, suitable for extension to a range of different models. The formalisation of parameterised contracts so far characterised them as bijections over powersets of trace languages [22], which is problematic since (i) provided/required type domains must be of equivalent size, providing insufficient scope for formalising adaptation where some differences in properties are seen as irrelevant, and (ii) although regular languages and rational trace languages are powerful and increasingly used, there are alternatives and extensions to these for which the formal essence of parameterised contracts could potentially be applied.

In this paper we describe a mathematical framework for a family of parameterised contract models, Galois contracts (GCs), which extends the prior formalisation of parameterised contracts from a bijection to a Galois connection, and from powersets of trace automata to semilattices. We define a generic notion of parameterised contract conformance/generalization, and a richer model of component adaptation. The family thus covers a range of levels of expressiveness, in particular touching on all four levels of component contracts as proposed by Beugnard. We show how this extension enables a range of precision-simplicity tradeoffs. In particular we show that the framework generalizes two important existing formal representations already in use for expressing contracts—tables and finite automata. We also show that Galois contracts may be used heterogeneously, that is, different specific notions of contract can be intermingled based on their common generic semantic foundation.

We have constructed Galois contracts so as to work (we hypothesise) across a large range of specifications for many different architectural design contexts. In this paper we focus on two specific languages, but we have also considered others. Here we give examples of GCs for tables (a simple notation) and trace automata, which subsume classical finite automata but also incorporate a richer notion of concurrency.

In this paper we explicitly exclude issues of composition from the scope. Composition has been discussed in prior work and we emphasise that our work is expected to be closely compatible with composition methods already discussed (see note in related work).

The paper is organised as follows. We first show how parameterised contracts can be expressed simply as (contract) tables and consistently interpreted. We then formalise generic Galois contracts, and show how contract tables are a special case of these. We then show how GCs can be modelled using finite automata as trace GCs and used to model concurrent systems and their properties.

2. CONTRACT TABLES

With suitable interpretation, tables can be a powerful semi-formal notation for modeling parameterised contracts.

Consider the worst-case time of the cell in various configurations and operating environments. Given a scenario involving some specific synchronisation, what is the worst-case cycle time? Given a cycle time budget, what scenarios are allowed?

Table 1 shows an (abbreviated) contract table for the cell giving the WCCT (max(CT)) associated with various scenarios (also known as features, see below). The table is abbreviated by ellipses (...) hiding other features not of interest.

Given a set of features, to find a conservative upper bound for cycle time, simply read off the largest value of max(CT) for all given features. Equivalently, given a time budget \( t \), the maximal set of features available within the time budget are all those whose values for WCCT are less than \( t \).

Following tradition in worst-case execution time analysis, the value of WCCT is treated as a conservative upper bound: it is typically critical that no observed value ever exceed the bound, but not so critical that the bound be tight. However, if only a single context-free bound for the component is available, reasoning about component properties typically cannot be accurate. On the other hand, as here, guaranteeing a lower execution time bound when restricted to a specific feature may enable more accurate reasoning over assemblies of multiple components [9].

<table>
<thead>
<tr>
<th>Feature</th>
<th>Time Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>Safe</td>
<td>( t_2 )</td>
</tr>
<tr>
<td>...</td>
<td>( t_{\text{max}} )</td>
</tr>
</tbody>
</table>

Table 1: Contract Table for Cell

The benefit of contract tables lies in their simplicity and versatility. As shown, the rules for reading them are reasonably simple. With proper interpretation, they provide a powerful tool for expressing parameterised contracts and reasoning accurately about context-dependent properties.

Features are any relevant aspects of configuration, deployment or performance (thus more general than a "scenario"). Features are divided into two categories—provided (rows) and required (columns). Features are not to be confused with modes, which are mutually exclusive abstract states of the runtime system. In systems with only static design-time configuration, features determine system behaviour for the entire deployment lifetime. Here the set of provided features is \( P = \{ \text{safe, fast, ...} \} \) and the set of required features is \( R = \{ WCCT \leq t_1, ..., \} \). The different treatment of required features is explained by the fact that Table 1 is actually an abbreviation of Table 2, a feature dependence relation (FDR) (represented as a cross table) where required features are spelled out in more detail, and consequently less readable. There exist standard techniques for mapping between these representations for any (lattice-)ordered domain which we
of different size. Certain values of $U$ (and $V$) are unchanged when passed through $s \circ w$ (or $w \circ s$). Call these values canonical (with respect to the Galois connection). Call the set of canonical values in $U$ (or $V$) the kernel of $U$ (or $V$). In addition, all non-canonical values are mapped by $w$ and $s$ to canonical ones immediately.

**Theorem 3.2 (Composition).** Let $(w, s) : U \leftrightarrow V$ and $(w', s') : V \leftrightarrow W$ be Galois connections. Then $(w' \circ w, s \circ s') : U \leftrightarrow W$ is a Galois connection.

Informally, any two Galois connections with a common domain can be composed on their common domain to create a new Galois connection.

### 3.2 Galois Contracts

**Contract Domain** Define a contract domain as a join semi-lattice $(U, \leq_U, \sqcup_U)$ For convenience we drop subscripts, and refer to $(U, \leq_U, \sqcup_U)$ as $U$, where there is no ambiguity. Refer to $\leq$ as conformance and say that $c_1$ conforms to (strengthens or refines) $c_2$ iff $c_1 \leq c_2$, or equivalently that $c_2$ weakens or generalizes $c_1$. We refer to $\sqcup$ as strongest covered contract. Further, given $c_1, c_2 \in U$ say $c_1 \equiv c_2$ iff $c_1 \leq c_2$ and $c_2 \leq c_1$. Say $c_1 < c_2$ iff $c_1 \leq pc$ and not $c_2 \leq c_1$.

**Galois Contract** Let $(U, \leq_U, \sqcup_U)$ and $(V, \leq_V, \sqcup_V)$ be contract domains, and let $w : U \rightarrow V$ and $s : V \rightarrow U$ be functions such that $(w, s)$ is a Galois connection. Then $(w, s)$ is a Galois contract with respect to $(U, V)$, written $(w, s) : U \leftrightarrow V$, or simply $(w, s)$ where there is no ambiguity.

To further support adaptation (see later) we explicitly allow that $w$ and $s$ be defined over larger domains than $U$ and $V$. Thus a pair of functions $(w, s)$ may constitute multiple Galois contracts, constrained with respect to some combination of domains, in which case $U$ and $V$ must be given explicitly.

Where $U$ and $V$ are implied by $w$ and $s$, an alternative form of specification may be used. Let $u \in U$ and $v \in V$. Then $(w, s) : u \leftrightarrow v$ denotes $(w, s) : U' \leftrightarrow V'$ where $U'$ is the so-called filter of $U$ with respect to $u$, that is $U' = \{u' | u' \in U, u \leq u'\}$, and similarly for $V'$. Call $u$ and $v$ the bounds of $(w, s)$.

Mathematically, consistency of the above representation is intrinsic, that is, $w$ and $s$ are applicable to all values in their domain and return values in their codomains (respectively). In practice consistency may not be guaranteed. This is not an issue for contract tables but becomes an issue for richer models (later). Although out of the scope of this paper, consistency checking is also required when modeling composition over multiple contracts via multiple ports and connectors.

- There is a natural notion of conformance (generalization) between GCs:

**GC Conformance (Generalization)** Let $U, U', V, V'$ be contract lattices and let $c = (w, s) : U \leftrightarrow V$ and $c' = (w', s') : U' \leftrightarrow V'$. Then $c'$ conforms to (strengthens) $c$, written $c' \leq c$, iff $U \subseteq U'$ and $V' \subseteq V$ and for all $u \in U$, $w'(u) \leq_U w(u)$, and for all $v \in V$, $s(v) \leq_V s'(v)$. Likewise we say that $c$ generalizes $c'$.

---

1 For example formal concept analysis provides a method known as conceptual scaling for this purpose [10].
Informally, \( c' \) conforms to (strengthens) \( c \) if and only if \( c' \) provides not less and requires not more than \( c \), both domain-wise and element-wise. That is, \( c' \) provides a not-smaller static contract domain than \( c \), and requires a not-larger static contract domain than \( c \), and, for any given static contract which \( c \) can provide, \( c' \) requires no more than \( c \) would, and, for any given static contract which \( c \) requires, \( c' \) provides no less than \( c \) would.

Although the above notion of conformance appears to be context-free, defining only whether one GC would conform to another in all contexts, a constraint-based notion of context-dependent conformance follows immediately, as shown in prior work. Consider a component modeled by \( C_1 : U \leftrightarrow V \) and another component \( C_2 \), where it is not the case that \( C_2 \subseteq C_1 \). After deployment, what is actually required of \( C_1 \) and what can actually be provided to \( C_1 \) are partially ordered by \( \leq \) and are naturally conformed to \( C_1 \) if and only if \( C_2 \subseteq (C_1 : U \leftrightarrow V) \). Where the architectural context is fully known, it is necessary only to check the pointwise case by applying the respective values of \( w \) and/or \( s \). (As for Meyer’s Design by Contract, it may not be feasible to completely check conformance at substitution time, however substitution nevertheless comes with additional conformance proof obligations that may be deferred, and such deferred obligations form part of the trustworthiness profile for a component.)

Simplification: The dual notions of fidelity and simplicity provide useful metrics for tradeoffs between accuracy and analytical cost. Fidelity (i.e. faithfulness or accuracy) is the ratio of kernel size to the size of the largest domain. Simplicity (i.e. representation cost and analysis time) is the size of the kernel. For example, a canonically simple, but uselessly imprecise contract would always provide \( \top \) (weakest static contract) while always requiring \( \bot \) (strongest static contract).

4. TABLES SPECIALISE GCS

Every feature dependence relation (and therefore every contract table) generates a Galois contract.

Feature Dependence Relation Let \( P \) and \( R \) be mutually disjoint sets, and \( D \) any relation \( D \subseteq P \times R \). We refer to \( D \) a feature dependence relation.

Define functions \( w_D : 2^P \rightarrow 2^R \) and \( s_D : 2^R \rightarrow 2^P \) where

\[
w_D(P') = P' \circ D.
\]

and

\[
s_D(R') = \{ y \in P \mid yD. \subseteq R' \}
\]

where \( yD. = \{ x \mid yDx \} \).

**Theorem 4.1.** \((w_D, s_D) : 2^P \leftrightarrow 2^R \) (where \( 2^P \) and \( 2^R \) are partially ordered by \( \supseteq \)).

Every feature dependence relation over \( P \) and \( R \) generates a Galois contract \((w, s)\) with respect to the contract domains \((2^P, \supseteq, \cap)\) and \((2^R, \supseteq, \cap)\). Thus feature dependence relations provide a simple notation to create well-formed GCs by construction.

The interpretation of \( D \) is asymmetric. We justify this by emphasising the asymmetry in the basic notion of feature dependence, where we reason from a set \( P' \) of (provided) features actually required, to all elements required by any of them, and conversely from a set \( R' \) of (required) functions actually available, to all features requiring only those in \( R' \).

### 4.1 Cell Contract Table as GC

Table 1 can be derived from other models or empirically as follows. Define a relation \( \text{time} \subseteq P \times R \) associating features with possibly multiple execution times, as would be expected if there is variability (e.g. due to non-determinism) in execution time associated with a given feature. At this point we have Table 2. Interpreting \( \text{time} \) as a FDR we can derive a GC \( \text{Time} : 2^P \leftrightarrow 2^R \) as shown above. The classical (partial) function \( \text{max} : 2^R \rightarrow R \) is also a Galois connection \( \text{max} : 2^R \leftrightarrow R \cup \text{undefined} \) where \( R \) is the semilattice ordered by \( \geq \). Since both \( \text{Time} \) and \( \text{max} \) share the common domain \( 2^P \), \( \text{max}(T) = \text{def} \{ \text{max} \circ \text{Time} : 2^P \leftrightarrow R \cup \text{undefined} \} \) is also a Galois connection, as shown in Table 1.

A graphical representation of the GC is given in Figure 2. The figure shows Hasse diagrams of the provided domain (feature sets, left), required domain (execution times, right) and the bijection arising from the Galois connection (horizontal dashed lines connecting feature sets to execution times) indicating the four canonical cases. All other scenarios are mapped to a canonical case by the GC as follows: Given a set of features \( U \), find the canonical case with the smallest superset of \( U \) and smallest corresponding WCCT. Given a WCCT, find the case with the next-largest WCCT.

For example, \{safe\} is canonicalised to \{fast, safe\}. (Note: this representation is only to give the intuition of the Galois connection and in particular the existence of canonical elements. The ambiguity which appears to arise in canonicalising elements is an artifact of the representation only, and is not present in the Galois connection itself.)

The figure and example above highlight what may not be intuitively clear about the cell: there is no need for a distinct case corresponding to safe synchronisation alone. While safe and fast synchronisation have no common behaviour, they are strictly ordered by WCCT. That is, given a WCCT “budget” of \( t_1 \), only fast synchronisation is feasible, but given a
budget of $\Omega$, both safe and fast synchronisation are feasible. Of course only the presence of the safe and fast features and WCCT are being taken into account here. For more precise GCs the clusterings may be less coarse.

5. TRACING AUTOMATA SPECILISE GCs

It is very common to represent contracts using state machines or extensions. It is not as common that contract representations cater for true concurrency or properties such as timing over multiple synchronised clocks, or that representations support of bidirectional reasoning (e.g. from usage to a property such as time and vice versa). In this section we instantiate Galois Contracts for trace languages, for which bidirectional reasoning is feasible under certain conditions using trace automata.

In particular we show that a trace translation relation (derived from a trace-theoretic behaviour description), can be interpreted as a GC. This is of particular relevance to our “wicked” problems where real parallelism, concurrency and extra functional properties are intertwined. Since FDRs (regardless of what a feature is), and trace dependence relations model GCs, the same theory can be applied to both. In particular pre-existing notions of consistency and conformance and property analysis methods [19] may be applied uniformly to both FDRs and trace translations relations, and trace translation relations can be approximated by FDRs. Trace dependence relations were not expressible in the prior formalisation of parameterised contract as bijection.

5.1 Background

Hereafter, let $U_\Sigma$ be a set (of symbols), called the universal symbol set.

**Dependence, Trace, Trace Language** [17] Let $\Sigma \subseteq U_\Sigma$, and $D$ a finite, reflexive and symmetric relation $D \subseteq \Sigma \times \Sigma$. Then $D$ is called a dependence relation (or dependency). Re- refer to the domain of $D$ as $\Sigma_D$ and call it the alphabet of $D$. Call the relation $I_D = (\Sigma_D \times \Sigma_D) - D$ the independence relation (or independency). Omit subscripts where there is no ambiguity. Denote by $M(\Sigma_D, I_D)$ the partition of $\Sigma^n$ into sets of words which are equivalent under arbitrary composition of independent adjacent symbol pairs. Abbreviate $M(\Sigma_D, I_D)$ as $M(\Sigma, I)$ where there is no ambiguity. Call elements of $M(\Sigma_D, I_D)$ traces (over $D$) and for word $w$ let $[w]_D$ be the trace over $D$ corresponding to all interleavings of $w$ with respect to $I$. Call any subset of $\Sigma^n$ a language, and any subset of $M(\Sigma, I)$ a trace language.

**Projection** Let $C$ and $D$ be dependence relations where $C \subseteq D$. Let $t$ be a trace over $D$. Informally define the trace projection of $t$ to $C$ [17] as the trace arising from the removal of all symbols in all words of $t$ which are not in $C$. Use the notation $\pi_C(t)$. Trace projection extends in the natural way to trace languages.

**Synchronisation** The synchronisation [17] of the trace language $T_1$ over $D_1$ with the trace language $T_2$ over $D_2$, is the trace language $(T_1 \parallel T_2)$ over $(D_1 \cup D_2)$ such that

$$t \in (T_1 \parallel T_2) \iff \pi_{D_1}(t) \in T_1 \land \pi_{D_2}(t) \in T_2$$

**Trace Automaton** A trace automaton [12] is a finite automaton $A$ where $\Sigma$ is equipped with a dependence relation $D$. $A$ denotes a trace language by interpreting each of its accepted words as the representative of a trace, and taking the union of such traces as the denoted trace language.

5.2 Trace Translation Relation as GC

Let $\Sigma_P, \Sigma_H$ and $\Sigma_R$ be dependence relations where $\Sigma_H$ and $\Sigma_P$ are disjoint and $\Sigma_H$ and $\Sigma_R$ are disjoint. Let $\Sigma = \Sigma_P \cup \Sigma_H \cup \Sigma_R$. Let $T$ be a trace language over $\Sigma$. Let $T_1 \subseteq T$ and $T_2 \subseteq T$. Define $\subseteq \subseteq T \times T$, where $T_1 \subseteq T_2$ iff $T_1 \supseteq T_2$.

Given that every relation may be interpreted as a GC, every trace translation may be interpreted as an FDR interpreting traces as features—call it a trace GC. The trace GC maps back and forth between sets of traces, i.e. trace languages. Since trace languages may be infinite, trace GCs may be defined with respect to infinite domains.

When a trace-theoretic GC is abstracted by a contract table, it is natural to interpret whole trace languages as features. This can be understood as an extension of FDRs where a new provided feature may be introduced as the name for a collection of other provided features; the new feature requires all the required features of any of the collection of provided features.

For trace GCs we use the notation $(w_E, s_E) : E_I \rightarrow E_O$, where $E$ is a regular expression implicitly defining an alphabet $\Sigma$ and $E_I$ and $E_O$ are regular expressions implicitly defining alphabets $\Sigma_I$ and $\Sigma_O$, to denote a Galois connection based on the trace relation $\Theta_E \subseteq \Sigma_I \times \Sigma_O$ (and trace languages denoted by $E_I$ and $E_O$ as upper bounds).

In prior work [23] the authors and others showed that for this example, involving deadlock-free synchronisation of suitable simple-cyclic automata satisfying a free-choice property, a state machine decomposable Petri net representation of automata can be straightforwardly constructed, and properties such as worst-case cycle time can be computed from a version of the nets with loops cut, based on further parameterisation of low-level times for the various actions such as press, press. This approach avoids the state explosion problem entirely. We refer readers to the prior paper for more fully worked examples of WCCCT calculation.

Consistency checking may be useful when using trace automata to represent functions and bounds separately. In previous work [22], Dependent Finite State Machines were explicitly specified and required certain inclusion checks to ensure consistency.

5.3 Cell Trace GCs: Use and Conformance

Table 3 shows protocols and internal synchronisation of the cell and its components specified by regular expressions.

3When $I = \{\}$ this is an extension of ordering over regular languages. Since no commutation is possible, traces and words are isomorphic.
The symbol load (as in arm1.load) stands for the sequence load!start!load!end (and similarly for unload). Below we will interpret a regular expressions either as a trace language or as a minimal deterministic finite trace automaton, as required by the context. The trace languages in this example are closely related to those in the earlier work involving the authors [22, 23].

Let openCell1 = arm1P|arm2P|baseP|pressP, and fast = openCell|robotPressTwoArms2|baseArmsFast, safe = openCell|robotPressTwoArms2|baseArmsSafe, 1Arm = openCell|robotPressOneArm|base1ArmSync, and finally cell = fast|safe|1Arm.

Let cellR = pressP.

Define GCs $C_{cell} = (w_{cell}, s_{cell}) : cellP \leftrightarrow cellR$ and $C_{1Arm} = (w_{1Arm}, s_{1Arm}) : cellP \leftrightarrow cellR$.

As an example of analysis, $w_{cell}$ can compute all the traces that may be required of the press to consume one blank and emit one blank:

$w_{cell}(in,out) = \{
\text{press.down, press.up}\?, \text{press.hold, press.press}\\}.$

That is, in its most general form, cell may (or may not) require the use of the lower press position. Correspondingly

$w_{1Arm}(in,out) = \text{press.hold, press.press}.$

As an example of conformance, $C_{1Arm} \preceq C_{cell}$, since it provides all of cellP and requires only the restricted version of the press.

### 5.4 Cell Worst-Case Time GC

Let the relation $t \leq T \times \mathbb{R}$ associate finite traces with (possibly multiple) execution times (according to some fixed assignment of times to primitive actions) such that:

- $t(ab) \leq (t(a) + t(b))$ if $aDb$, and
- $t(ab) \leq \max(t(a), t(b))$ if $aIb$.

Thus time takes into account independence. Then there exists a Galois contract WCCT$_{cell}$ mapping all finite-trace sublanguages of cell to WCCT bounds and back. However WCCT$_{cell}$ may be too complex to model and reason about directly. In particular, computing the inverse, which gives all finite trace languages whose WCCT are less then a given bound, seems computationally prohibitive.

Nevertheless we can abstract WCCT$_{cell}$ by conservative approximation of the time relation, associating certain traces with larger time values than could actually be observed. A GC associated with such a relation abstracts WCCT$^\star_{cell}$ (according to our definition above), since it requires no less time for any given trace, and provides more traces for any given time. At least two (extreme) cases should be considered for effective abstraction. The trace language $T = \{\}$ corresponds to completely restricted usage of the cell, for which there are no traces and thus no defined times. The trace language cell corresponds to unrestricted usage, whose WCCT is the context-free worst-case time $t_{max}$ for cell.

Therefore initially we conservatively assume that there is a single time $t_{max}$ in the codomain and relate every trace to $t_{max}$. (It is still necessary to determine the value of $t_{max}$ analytically by evaluating time for the conservative case cell using a valid WCET technique.)

Such an abstraction is simplistic and too imprecise for many scenarios. However a contract $C$ approximating a component, where previously acceptable in an architecture, can safely be replaced by a conformant contract $C'$. Thus a component with performance abstracted by contract $C$ may be reviewed, and, if the component is shown to conform to the stricter $C'$, then one can safely reason with the improved $C'$ instead.

Three additional trace languages are of interest—fast, safe and 1Arm. Each of these correspond to a restriction and refinement of cell. That is, contracts based on these are only substitutable for cell in certain architectural contexts, but, when substituted, they have reduced WCCT values: call these $t_1, t_2$ and $t_3$ respectively and assume $t_1 < t_2 < t_3 \leq T_{max}$. We model such context-dependent refinement in Table 3 which refines Table 1.

Table 3 has a more precise interpretation than Table 1, since each feature corresponds to a trace language restriction of cell. It is thus possible to “guard” each entry in the table with a corresponding automaton or other suitable representation of the restriction, and use the table as part of an automated analysis of assemblies involving cell.

<table>
<thead>
<tr>
<th>Component</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>safe</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1Arm</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 3: Refined Contract Table for Cell

The advantage of this method is that generalisations of any Galois connection can always be constructed and refined, simply, and only as needed.

Thus we have demonstrated that, even where calculating $w$ (weakest requirement) is prohibitive, one can construct a conservative (therefore safe) ad hoc approximation of the reverse function, incrementally on demand. We demonstrated that approximation can be achieved between representations, by generalising a trace-based GC to a contract table. Although we have used a contract table as the approximation for demonstration purposes, it should also be feasible to approximate contracts using conservatively constructed finite automata.

### 6. RELATED WORK

Our main contribution is to elaborate a formal generalisation of parameterised contracts, from bijections to Galois connections, and from domains modeled by regular languages to domains modeled by semilattices, and to show its benefits.

Reussner introduced a notion of parameterised contracts for components [20]. The second author and others including Reussner [22] formalized parameterised contracts as total, bijective (thus invertible) and monotone functions with respect to powersets of trace automata. Our work extends bijections to Galois connections, thus the invertible function is then an invertible “adaptation”. To our knowledge there is no prior work generalising parameterised contracts to Galois connections, or elaborating the benefits for component contracts and architectural reasoning. In particular this generalisation then covers many classes of automata that permit

---

3The cross table version underlying Table 1 (implicitly) relates 1Arm to all cycle times whereas that of Table 3 relates 1Arm only to cycle times less than $t_3$. 
<table>
<thead>
<tr>
<th>Name</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellP</td>
<td>([in,out])</td>
</tr>
<tr>
<td>lArmP</td>
<td>halfpress.([in,out])</td>
</tr>
<tr>
<td>arm1P</td>
<td>([arm1.load!start,arm1.ext,arm1.take,arm1.ret,arm1.load!end,arm1.unload!end])</td>
</tr>
<tr>
<td>arm2P</td>
<td>([arm2.load!start,arm2.ext,arm2.take,arm2.ret,arm2.load!end,arm2.unload!end])</td>
</tr>
<tr>
<td>baseP</td>
<td>([base.tr],[base.tr])</td>
</tr>
<tr>
<td>pressP</td>
<td>([press.down,press.up],[press.hold,press.press])</td>
</tr>
<tr>
<td>halfPressP</td>
<td>([press.hold,press.press])</td>
</tr>
<tr>
<td>robotPress2Arms</td>
<td>fullpress.([press.down,arm2.load,press.up,arm1.unload],[press.hold,press.press])</td>
</tr>
<tr>
<td>baseArmsFast</td>
<td>([arm1.take,in,base.tr,arm2.take,base.tr,arm1.drop,base.tr,arm2.drop,out,rbase.tr,rbase.tr,rbase.tr])</td>
</tr>
<tr>
<td>baseArmsSafe</td>
<td>([arm1.load,arm1.load.end,in,rbase.tr,arm2.load,rbase.tr,arm1.unload,rbase.tr,arm2.unload.out,rbase.tr,rbase.tr,rbase.tr])</td>
</tr>
<tr>
<td>robotPress1Arm</td>
<td>halfpress.([arm1.ext,arm1.take,arm1.ret,arm1.ext,arm1.drop,arm1.ret,press,press,arm1.ext,arm1.take,arm1.ret,arm1.ext,arm1.drop,arm1.ret,out,rbase.tr,rbase.tr,rbase.tr])</td>
</tr>
<tr>
<td>baseArm1Sync</td>
<td>([arm1.ext,arm1.take,arm1.ret,in,rbase.tr,arm1.ext,arm1.drop,arm1.ret,arm1.ext,arm1.take,arm1.ret,arm1.ext,arm1.drop,arm1.ret,out,rbase.tr,rbase.tr,rbase.tr])</td>
</tr>
</tbody>
</table>

Figure 3: Cell and Subcomponent Behaviours

A simulation or bisimulation notion serving as a partial order for the core conformance/generalization relationship.

In this paper we also generalize contract domains, modeled by trace automata with a regular language intersection operation, to join semilattices. While discussed semi-formally in prior work [21], to our knowledge this was never explicitly formalized or elaborated. In this paper we show that semilattices may be instantiated as feature sets ordered by the subset relation, thus creating simple and intuitive contract representations such as feature dependence relations and contract tables. We also showed how contract tables are an intuitive general model for Galois connections over more general semilattices. We believe this formalisation may serve numerous tabular representations used in practice.

Most importantly, the bidirectional nature of Galois connections allow forward and backward analysis, not only with a focus on consistency and conformance in the architecture of a single software product, but for the purpose of adaptation and variation in software product lines. Beyond the widely celebrated substitutability (found in some notion of semantic subtyping related to conformance/generalization for contracts) such variation includes contract restrictions and enrichments. We have shown here in the example of the production cell how various different behavioural adaptations rooted in physical system reconfigurations relate to context-sensitive and thus parameterised contract variation in the architecture of the product line.

We have shown that Galois contracts may be modeled using tables and trace automata, and that models in each representation may faithfully abstract or refine each other.

In this paper we generalize the notion of a translation relation more fully to traces, with explicit reference to trace synchronisation and projection as defined by Mazurkiewicz [17]. While prior work [22] introduces trace automata and develops a notion of behavioural subtyping for our architecture definition language Radl, the translation relation defined there does not explicitly refer to trace-theoretic synchronisation and projection.

Our framework also works towards compatibility with the work of Fredriksson et al [9] where contract-based reusable WCET estimates were constructed using clustering techniques to drive context-qualified empirical measurement of WCET performance, expressed using tables. We have provided a formal basis for the interpretation of certain kinds of tables, and the means to link the semantics of tables with richer models such as trace automata and logics.

Use of Galois Connections in modeling and reasoning about computation has a long-standing tradition in mathematics and computer science, for example in various weakest-precondition calculi in program verification [6], abstract interpretation, and assume-guarantee verification. Assume-guarantee reasoning is a widely studied technical for compositional analysis of systems, including automatic generation of so-called weakest assumptions [11]. Moreover several of the theorems we use in this paper are believed to be part of computer science and mathematics folklore. Nevertheless the application of Galois connections to the modeling context-dependent component contracts appears to be novel. Thus our approach may provide a bridge to richer approaches providing semantics aiming at correctness proofs for software, while our formalisation aims to underpin real-time behavioural and extra-functional analysis in software product lines. Our framework also has the prospect of providing a unified view of the four levels of component contracts observed by Beugnard [5]. We showed how a simple syntactic view involving sets of feature names, and a richer view combining aspects of synchronisation and extra-functional properties, could be unified, spanning several of Beugnard’s four levels.

Our approach is slightly different in emphasis to assume-guarantee verification, where recent work focuses on compositional verification using automated inference of what we call “contracts” at component boundaries. Our approach is characterised by the assumption that engineers may have useful partial intuitions about behaviour and properties which are worth encoding, and may even be useful in hybrid approaches to contract inference or contract recovery. We explicitly excluded the modeling of composition from the scope of this paper. Composition involves handling several related issues e.g.: multiplicity of ports, ensuring consistency at contractual and higher semantic levels, and adapta-
We emphasise that our formulation of Galois contracts has been designed for backward compatibility with earlier techniques for modeling composition and handling consistency. For example, in many ADLs components have multiple ports which model interfaces which are independently-connectable. In prior work on parameterised contracts, multiple ports are modeled as product lattices of regular language subsets [22]. Our approach may inherit the product lattice technique without change.

Adaptation is necessary where two protocols are related but not identical. In prior work on parameterised contracts domains are implicitly join semilattices (explicitly powersets of trace automata with trace language intersection). The join operation models a form of adaptation, where even if the whole of what is required could not be provided by a given contract, the portion which could be provided can be calculated using the join operator. Our work is fully compatible with such an approach. In the case of contract tables this is useful in situations where some, but not all, of a requested set of features cannot be provided by a contract under any circumstances. The join operator, realized as set intersection, trims away those features that cannot be provided, protecting the contract from reasoning about spurious features outside its domain.

In prior work on adaptation [24], it was shown that regular language operations provide useful models for adaptation. Such adaptation can be modeled by the notion of Galois connection, affording compatibility between static contract domains with different cardinalities and modeling the clustering of static contracts where certain distinctions are irrelevant.

Several early ideas from parameterised contracts influenced the design of Palladio component model, which uses service effect machines resembling UML state charts to reason about probability distributions over execution times for component-based systems. Research around Palladio has also tackled theoretical and practical issues relating to the modeling of concurrent, probabilistic systems. Palladio’s notion of conformance is defined as follows: component \( C_1 \) conforms to \( C_2 \) iff \( C_1 \) provides at least as many services as \( C_2 \) and requires no more services than \( C_1 \). This notion of conformance is likely to be extended to protocol language inclusion, thus Palladio uses essentially the same definition of conformance as in the work we are extending. To our knowledge the design of Palladio does not explicitly incorporate the notion of a Galois connection as a generic model for contracts, nor the use of simple formalisms such as contract tables.

Architectural languages such as Darwin/LTSA [16], Palladio [4] and Radl differ from numerous others by not only characterising the interfaces—i.e., essentially connector conformance—but also modeling the abstract behaviour of components themselves to capture global architectural dependencies between interfaces essential for global system properties such as safety and reliability. Our generic Galois contracts provide a framework for modelling these dependencies across architectural networks, largely independently of whether these are modelled in formal logic, process definition languages, automata or relation tables. The essence to which we reduce the dependencies is the lattice structure that underlies the conformance ordering of the contracts, a minimal set of (lattice) operations serving to vary these contracts (walking the lattice), and Galois functions mapping from one formalism (contract domain) to another, thus underpinning dependencies between hybrid architectural descriptions which are becoming increasingly important with distributed systems and advances in multiple different methods for automated software engineering for this domain.

7. CONTEXT AND FUTURE WORK

The theory elaborated here and its instantiation for trace automata has been incorporated into our Rich Architectural Description Language Radl as part of an effort to bridge Radl to various ADLs and modeling tools. We have implemented bridges to the Tuscany [1] Service Component Architecture (SCA) [3], supported by the Eclipse [2] platform. Other extensions include a bridge between Upaal and Radl. In our work on grid fault-tolerance modeling [25] we also use Prism for probabilistic automata dependencies. An extension of the work presented in this paper to Prism/PCTL [13] will be future work.

Other future work includes:

- verifying that the generic framework can be instantiated for other common, but possibly complex, notations for contracts, such as temporal logics. For example new parametric probabilistic logics, such as those proposed by Dawes [7], may be needed;
- documenting the framework as it applies in the wider context of composition, i.e. including notions of multiple ports, inter-port connection and adaptation and semantic consistency as summarised by de Roever [8]. In particular we believe that the theory of GCs can be applied equally to models of connectors;

8. CONCLUSION

In this paper we extended the prior work of Schmidt, Reussner and Poernomo, to generalize parameterised contracts to Galois contracts. We showed (a) how such an approach leads to more general notions of parameterisation, conformance vs generalisation and adaptation; (b) that a generic approach is also practical since it enables a range of simplicity-accuracy tradeoffs; (c) how the framework can be instantiated for two common notations—tables and trace automata, and (d) that contracts might be combined heterogeneously, that is, different specific notions of contract (different domains in the underlying lattices) can serve as abstractions or refinements of each other based on their connection by Galois contracts.

9. REFERENCES


