Synthesising Features by Games

Dimitar P. Guelev 1
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Sofia, Bulgaria

Mark Ryan 2
School of Computer Science
University of Birmingham
Birmingham, United Kingdom

Pierre Yves Schobbens 3
Institut d’Informatique
Facultés Universitaires de Namur
Namur, Belgium

Abstract
We describe an algorithmic method for the synthesis of features. The method takes as input a base system, a triggering condition for the feature, a set of system variables which the feature is allowed to update, and a requirement on the result of integrating it. It computes whether a feature of the given form and with the desired property exists and, if so, a construction of it. The method is based on the theory of infinite parity games.

Key words: feature synthesis, infinite parity games

Introduction
The concept of feature has emerged as a popular way of structuring user-oriented descriptions of certain kinds of systems. Updating a system by adding

1 Email: guelevdp@math.bas.bg
2 Email: M.D.Ryan@cs.bham.ac.uk
3 Email: pys@info.fundp.ac.be

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs
new features to it is a technique which enables designs and code to be reused. It started to become popular when telephone companies began to introduce features such as call-forwarding and ring-back-when-free into plain old systems which did not support that functionality. This process of feature addition is well-known to be *non-monotonic*: adding a feature does not necessarily preserve the temporal properties of the system. Usually features are designed in isolation from one another, and putting several of them together in a phone system may lead to them interfering with each other in undesirable ways. This is known as the ‘feature interaction problem’, and is currently gaining considerable attention from academic and industrial researchers [CM00,GR01,AL03].

One may understand a feature at two levels of abstraction. At the lower (programming) level, it is code to be added to a base system, or a transformation of code already part of the base system, which brings about the intended functionality. At the higher (property) level, it is a characterisation of the intended functionality, for example by a temporal logic property. ‘Feature interaction’ can be understood as the failure of the feature code to enforce the intended feature property [PR01]. Considering features as having these two aspects has naturally led to using model checking to analyse feature interactions [CM01,PR01,dB99,BZ92].

In this paper we address the problem of synthesising feature code from abstract properties expressed in temporal logic. We show that, given a requirement \( \varphi \), a base system \( S \), and a subset of its states \( c \) at which potential features can be triggered, it can be decided whether a feature \( F \) such that \( S + F \models \varphi \) exists, where \( S + F \) stands for the result of integrating \( F \) into \( S \). Furthermore, if a feature with this property exists, then it can be synthesised in a concrete form, which includes the way it affects the transitions of the system and the variables it needs to have added and maintained upon its integration for doing so.

Our results are based on the formalism we presented in [GRS04], where we introduced two classes of features and showed how to model their integration in systems described in terms of states and transitions. These are the classes of *precomposed* features and *postcomposed* features, which affect the behaviour of the corresponding base systems by revising its transitions depending on conditions on their source and destination states, respectively. In [GRS04] we proposed some methods for verifying that features of these types preserve given properties of their base systems.

The study of algorithmic controller synthesis from requirements specified by formal languages was started by Ramadge and Wonham. A survey of their work can be found in [RW89]. They focused on the building of controllers represented as transition systems which satisfy requirements on finite behaviours specified by regular languages. The recent works [KMTV00,KV00,AVW03] extend this approach by studying requirements written in Computation Tree Logics (*CTL* and *CTL* [CE81]), the modal \( \mu \)-calculus ([Koz83], cf. e.g. [AN01]) and infinite behaviours. The underlying theory in these works is that
of parity $\omega$-automata and parity games (cf. e.g. [Maz02,Kûs02] in [GTW02]) of various forms.

We show that the solution of the problem of synthesising features can be obtained using the theory of parity games too: the given base system $S$, the triggering condition $c$ and the restrictions on the access of the prospective feature $F$ to the variables of $S$ can be used to define a game so that the feature-contributed transitions can be obtained as a winning strategy for that game, if such a strategy exists.

Structure of the paper
We first give brief preliminaries on the way of describing systems and features, the $\omega$-languages used to specify requirements, automata on infinite languages and parity automata in particular, and the relevant results on parity games. Then we show how the problem of feature synthesis can be formulated in terms of parity games to obtain our result. Finally, we point to the related issue of inheriting properties from base systems, the more general problem of obtaining properties by introducing both precomposed and postcomposed features simultaneously, and make some concluding remarks.

1 Preliminaries

1.1 Descriptions of systems

We assume that observable states of a system $S$ are described as valuations of its set of variables $P_S$, which we assume to be all boolean for the sake of simplicity. The possible states of $S$ are the valuations of $P_S$. We denote the set $\{P_S \rightarrow \{0, 1\}\}$ of these states by $W_S$. Behaviours of $S$ are infinite sequences

$$ (1) \quad b = b_0 b_1 \ldots b_n \ldots $$

of states $b_i \in W_S$. We define the relation $R_S \subseteq W_S^2$ by putting $R_S(s, s')$ if $S$ can move from $s$ directly to $s'$. We denote the set of the initial states of $S$ by $I_S$. A sequence of the form (1) is a behaviour of $S$ if and only if $b_0 \in I$ and $R_S(b_i, b_{i+1})$ for all $i < \omega$. To guarantee the infiniteness of behaviours, we require $R_S$ to be serial, that is, to satisfy $(\forall s \in W_S)(\exists s' \in W_S)R_S(s, s')$. A system $S$ is described completely by the triple $\langle W_S, I_S, R_S \rangle$. We identify systems with their descriptions of this form.

In the sequel we regard system states as models for the propositional language based on the set of variables of the system. In particular, given a propositional formula $\varphi$, we denote the set of states which satisfy it without regard of their accessibility by $\llbracket \varphi \rrbracket$. 

3
1.2 ω-Regular Languages

Sets of infinite sequences states of the form (1) are termed ω-languages in the literature. In this paper we assume that the properties which systems with features are required to have are described as ω-languages which are recognised by ω-automata (cf. e.g. [Far02]). Such languages are called ω-regular. All languages which can be defined by formulas in propositional Linear Temporal Logic (LTL, cf. e.g. [GHR94]) on the natural numbers as the model of time are ω-regular languages. The converse does not hold. ω-regular languages have the general form

$$\bigcup_{i=1}^{n} U_{i} \cdot V_{i}^{\omega}$$

where $U_{i}, V_{i}$ are some regular languages, $i = 1, \ldots, n$, and $X^{\omega}$ stands for the set of all the infinite concatenations of words from $X$. Given an alphabet $\Sigma$, $\Sigma^{\omega}$ stands for the set of all infinite words in $\Sigma$ and $\text{Inf}(\alpha)$ denotes the set $\{ \alpha \in \Sigma : \alpha_{i} = \alpha \text{ for infinitely many } i < \omega \}$ for $\alpha \in \Sigma^{\omega}$.

Given a subset $\Sigma_{0}$ of the alphabet $\Sigma$ of a language, which in our setting is the state space of a system, the ω-language

$$L_{\Sigma_{0}} = \{ \alpha \in \Sigma^{\omega} : \text{Inf}(\alpha) \cap \Sigma_{0} \neq \emptyset \},$$

which consists of all the words which have infinitely many occurrences of symbols from $\Sigma_{0}$, is ω-regular. Furthermore, if $L$ is an ω-regular language and $L \subseteq L_{\Sigma_{0}}$, then the language obtained by deleting the occurrences of symbols outside $\Sigma_{0}$ from the words of $L$ is an ω-regular language too. The requirement $L \subseteq L_{\Sigma_{0}}$ is necessary, because otherwise deleting the non-$\Sigma_{0}$ symbols can render some words from $L$ finite. Conversely, if $L'$ is an ω-regular language in the alphabet $\Sigma_{0}$, then the language

$$(2)\{ \alpha \in \Sigma^{\omega} : \text{deleting the symbols from } \Sigma \setminus \Sigma_{0} \text{ in } \alpha \text{ produces a word from } L' \}$$

is ω-regular too. Given $L'$ and $\Sigma_{0}$, we denote (2) by $L'/\Sigma_{0}$ and call it the converse projection of $L$ onto $\Sigma_{0}$ in this paper. Similar projection operations appear in various other specification formalisms, such as interval temporal logic [HMM83] and the language ForSpec [AFF+02]. If $L$ is definable by an LTL formula, then so is $L/\Sigma_{0}$ (see e.g. [GRS04]).

The set of the possible behaviours of a system $S = \langle W_{S}, I_{S}, R_{S} \rangle$ is an ω-language over the alphabet $W_{S}$. In the sequel we denote this language by $[S]$.

1.3 Parity automata

Finite automata can be used to define ω-regular languages much like the way they are used to define regular languages that consist of finite words. The principal difference is the form of the acceptance condition for such automata, which, unlike the simple sets of final states in the case of regular languages, must deal with infinite words. Finite automata that accept languages of in-
finite words are called $\omega$-automata. An introduction to $\omega$-automata can be found in [Far02]. Here we only give the notions and results needed for our work.

**Definition 1.1** An $\omega$-automaton is a tuple of the form $\langle Q, \Sigma, \delta, q_I, \text{Acc} \rangle$ where $Q$ is a finite set of states, $\Sigma$ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the state transition function, $q_I \in Q$ is the initial state, and Acc is the acceptance component.

**Definition 1.2** Given $\alpha \in \Sigma^\omega$ and an automaton $A = \langle Q, \Sigma, \delta, q_I, \text{Acc} \rangle$, $\sigma \in Q^\omega$ is a run of $A$ on $\alpha$ if:

$$
\sigma_0 = q_I; \\
\sigma_{i+1} \in \delta(\sigma_i, \alpha_i), \ i < \omega.
$$

The various kinds of $\omega$-automata differ by the form of Acc.

**Definition 1.3** A parity automaton is an $\omega$-automaton $\langle Q, \Sigma, \delta, q_I, c \rangle$, where the acceptance component is a function $c : Q \to \{1, \ldots, k\}$ for some $k < \omega$. Parity automaton $A$ accepts a word $\alpha \in \Sigma^\omega$ iff

$$
\min\{c(q) : q \in \text{Inf}(\sigma)\}
$$

is even for some run $\sigma$ of $A$ on $\alpha$.

Every $\omega$-regular language can be defined by a deterministic parity automaton, that is, a parity automaton whose state transition function determines unique successor states.

### 1.4 Parity games

Parity games are infinite 2-player games in which the winning condition is like the acceptance condition of parity automata. A broad introduction to infinite games can be found in [Maz02]. Again, we only cover the issues needed for our work here. To define parity games, we need the underlying notion of arena.

**Definition 1.4** An arena is a triple of the form $A = \langle V_0, V_1, E \rangle$, where $V_0$ and $V_1$ are disjoint sets of 0-vertices and 1-vertices respectively, and $E$ is a relation on $V_0 \cup V_1$.

A play is an infinite sequence of states $v \in (V_0 \cup V_1)^\omega$ such that $E(v_i, v_{i+1})$ for all $i < \omega$. Transitions from states $v_i \in V_0$ in a play $v$ are chosen by player 0 and transitions from $v_i \in V_1$ are chosen by player 1. The winning condition of a game is the set of plays which are regarded as won by player 0. In the sequel we consider winning conditions which are $\omega$-regular languages in the alphabet $V_0 \cup V_1$. In general an arena can have both finite and infinite plays. We consider only infinite plays in this paper and therefore assume that $E(v) \neq \emptyset$ for all $v \in V_0 \cup V_1$. 


Definition 1.5 A game is a pair of the form \( \langle A, W \rangle \) where \( A \) is an arena and \( W \) is a winning condition on it.

In this paper we use \emph{parity games} which have their winning conditions specified in the form of the acceptance conditions of parity automata:

\textbf{Definition 1.6} A game \( G = \langle \langle V_0, V_1, E \rangle, W \rangle \) is a \emph{parity game}, if \( W \) has the form

\[ \{ v \in (V_0 \cup V_1)^\omega : \min\{ c(a) : a \in \text{Inf}(v) \} \text{ is even} \} \]

for some appropriate mapping \( c : V_0 \cup V_1 \to \{1, \ldots, k\} \), which is called a \emph{colouring function} for \( G \).

In this paper we only consider \emph{initialised games} \( \langle G, v_i \rangle \), all plays of which start at a distinguished vertex \( v_i \). The main result that we need for this paper concerns winning strategies for parity games.

\textbf{Definition 1.7} Given an arena \( A = \langle V_0, V_1, E \rangle \), a strategy for player \( i \) is a function \( \mu : (V_0 \cup V_1)^* \to V_0 \cup V_1 \) such that \( \mu(v_1 \ldots v_k) \in E(v_k) \) for all nonempty words \( v_1 \ldots v_k \in (V_0 \cup V_1)^* \) such that \( v_k \in V_i \), \( i = 0, 1 \). Given a winning condition \( W \), a strategy \( \mu \) is \emph{winning} for player 0 if any play \( v \) which satisfies \( v_{k+1} = \mu(v_0 \ldots v_k) \) for \( k \) such that \( v_k \in V_0 \) is in \( W \).

Winning strategies for player 1 are defined symmetrically. A game is \emph{determined} if either player 0 or player 1 has a winning strategy. Parity games are determined.

\textbf{Definition 1.8} A strategy \( \mu \) is \emph{memoryless} if \( \mu(v_0 \ldots v_k) \) depends only on the last symbol \( v_k \) of the word \( v_0 \ldots v_k \).

Here follows the result on infinite games which is most important for our work:

\textbf{Theorem 1.9} If a player has a winning strategy for a parity game, then it has a memoryless winning strategy.

Proofs of this result can be found in [Küs02]. In the sequel we use the following simple observation about parity automata and games.

\textbf{Proposition 1.10} Let \( G = \langle \langle V_0, V_1, E \rangle, W, v_i \rangle \) be an initialised infinite game and \( W \) be \( \omega \)-regular. Let \( A = \langle Q, V_0 \cup V_1, \delta, q_i, c \rangle \) be a deterministic parity automaton for \( W \). Then the product \( G \times A = \langle \langle V_0', V_1', E', W', v_i' \rangle \rangle \) whose components are defined by the equalities

\[ V_i' = V_i \times Q, \ i = 0, 1, \]

\[ E'((u, q), (v, r)) \leftrightarrow E(u, v) \land v = \delta(q, u), \]

\[ W' = \{ v_0 \ldots v_k \ldots : v_0 \ldots v_k = v \in W \}, \]

\[ v_i' = (v_i, q_0), \]
is a parity game for which a colouring function $c'$ can be defined by the equality $c'(\langle v, q \rangle) = c(q)$.

Furthermore, player 0 has a winning strategy for $G$ iff it has a winning strategy for $G \times A$.

This observation becomes useful in conjunction with the fact that parity games admit memoryless strategies. Since any infinite game with an $\omega$-regular winning condition can be transformed into an equivalent parity game by multiplying it with a parity automaton which defines its winning condition, the amount of “memory” needed for storing a state of this automaton is sufficient for the winning strategies for the game.

1.5 Abbreviations for restrictions of relations and projections of states, etc.

Given a system $S$, $s \in W_S$ and $P \subseteq P_S$, $s|_P$ stands for the restriction of $s$ to the variables from $P$. Given a relation $R \subseteq W_S \times W_S$, $R|_U$ and $R|_V$ denote the restrictions $R \cap (U \times W_S)$ and $R \cap (W_S \times V)$ of the binary relation $R$ on $W_S$ to the domain $U$ and the range $V$, respectively. We denote the complement $W_S \setminus X$ of a subset $X$ of $W_S$ relative to $W_S$ by $\overline{X}$. Similarly, we denote the complement $P_S \setminus P$ of a subset $P$ of $P_S$ relative to $P_S$ by $\overline{P}$.

1.6 Features

Informally, a feature is an addition to a system of limited calibre meant to improve the functionality of the system. The result of integrating a feature $F$ into a system $S$, which is an (enhanced) system, is denoted by $S + F$. $F$ can bring in its own variables upon integration into $S$. The behaviours of $S$ and $S + F$ can also differ as observed in terms of the variables of $S$. A system can undergo the successive integration of several features. A feature $F$ which both adds variables and changes behaviour can be seen as a pair of features $F_1$ and $F_2$ to be integrated successively, $F_1$ being just an addition of variables, and $F_2$ carrying both the description of the behaviour of the new variables and the changes to the behaviour of the base system, but no more new variables. Clearly, properties of $S + F_1 + F_2$ written in the vocabulary $P_S$ can only be affected upon adding $F_2$. In this paper we restrict ourselves to features like $F_2$, which only change behaviour without contributing variables. If $F$ has this form, then $P_{S+F} = P_S$ and $W_{S+F} = W_S$. We assume $I_{S+F} = I_S$ for the sake of simplicity too. Then the integration of $F$ amounts to replacing $R_S$ by a new transition relation $R_{S+F}$.

Example 1.11 Consider the lift system [PR01]. It consists of a lift travelling between $n$ floors. There is a button on each floor (for calling the lift) and $n$ buttons inside the lift. The overloaded feature adds some vocabulary to the system, namely a boolean representing whether the lift is overloaded, and some new behaviour: the lift refuses to close its doors if it is overloaded.
Features can be classified into several categories, depending on their effect on the behaviour of the respective base systems [Kat93]. Features which only impose constraints on the behaviour of the variables added upon their integration are called *spectative*. Such features do not change the behaviour of the base system, but the variables they introduce can be used by subsequent features which do. Features which only rule out some of the behaviours of their base system are called *regulative*. Features which affect system behaviour in more general ways are called *invasive*. (The feature of the example is invasive.)

A feature $F$ affects the working of its base system $S$ only at transitions at which it becomes *triggered*. Let the current state of $S + F$ be $s$ and $R_S(s, s')$ for some $s' \in W_{S+F}$. Then, unless $F$ is triggered, $S + F$ can simply make the transition $\langle s, s' \rangle$. $F$ can be triggered by a condition on $s$, on $s'$, or on both $s$ and $s'$. In this paper we focus on $F$ which have triggering conditions of the first two kinds and call them *precomposed* and *postcomposed* features, respectively. The triggering condition of such an $F$ is a propositional formula. We denote it by $c_F$ and call it the *guard* of $F$.

In general it would be too crude to assume that the triggering of a feature $F$ can affect any variable of $S + F$. For this reason, we assume that the description of $F$ includes the set of the variables $P_F$ which $F$ can update differently from $S$ when triggered. The effect of a feature $F$ on a pending transition $\langle s_1, s_2 \rangle \in R_S$ is as follows:

A *precomposed* $F$ evaluates its guard $c_F$ at state $s_1$. If $s_1 \models c_F$, then $F$ cancels the transition to $s_2$ and first takes $S + F$ to some other state $s'_1$ such that an appropriate relation $R_F$ holds between $s_1$ and the restriction $s'_1|_{P_F}$ of $s'_1$ to the variables from $P_F$ which $F$ is allowed to change when triggered. The values of the variables outside $P_F$ remain the same upon the transition from $s_1$ to $s'_1$. Then $F$ allows a transition from $s'_1$ to be made by $S$. The externally observed transition resulting from this is from $s_1$ to the state $s'_2$ to which $S$ takes $S + F$ from $s'_1$.

A *postcomposed* $F$ evaluates its guard $c_F$ at the destination state $s_2$ of the pending transition $\langle s_1, s_2 \rangle$. If $s_2 \models c_F$, then $F$ prevents the transition to $s_2$ from being observed. Instead it uses $s_2$ to choose a state $s'_2$ such that $R_F(s_2, s'_2|_{P_F})$ and the values of the variables from $P_F$ at $s'_2$ are the same as at $s_2$. The externally observed transition is from $s_1$ to $s'_2$ again.

A feature $F$ can be described as the triple $\langle c_F, P_F, R_F \rangle$, where $R_F \subseteq W_{S+F} \times (P_F \rightarrow \{0, 1\})$ is the relation describing the $F$-specific updates of the variables from $P_F$ in transitions which trigger $F$. It can be assumed that $\text{dom} R_F$ is exactly $[c_F]$. Given $\langle c_F, P_F, R_F \rangle$ and $S$, we can define $R_{S+F}$ by the equalities

$$
(3) \quad R_{S+F} = R_S|_{c_F} \cup R'_F \circ R_S \text{ for precomposed } F,
$$

$$
(4) \quad R_{S+F} = R_S|_{c_F} \cup R_S \circ R'_F \text{ for postcomposed } F,
$$
where $R'_F$ is defined by the equivalence

$$R'_F(s, s') \leftrightarrow R_F(s, s'|_{P_F}) \land s'|_{P_F} = s|_{P_F}.$$  

**Example 1.12** (continued) Let $S$ be the lift system. Suppose $P_S$ already contains the boolean overloaded. The overloaded feature is described by the tuple $(c_F, P_F, R_F)$ where $c_F = \neg \text{doors\_open}$, $P_F = \{\text{doors\_open}\}$, and

$$R_F = [[\text{overloaded}] \times \{\text{doors\_open} \mapsto 1\} \cup \{(s, s|_{\text{doors\_open}}) : s \in [\neg \text{overloaded}]\}.$$  

This feature should be postcomposed.

Note that both the class of precomposed features and that of postcomposed features contain a neutral feature, which can be represented using the relation

$$\text{Id}_{c_F, P_F}(s, s') \leftrightarrow s \in [c_F] \land s' = s|_{P_F}$$
as $R_F$.

## 2 Separating system- and feature-contributed transitions

In this section we propose a transformation of system and feature descriptions which leads to a clear separation between the contribution of features and base systems to the behaviour of their combination. We first introduced this separation in [GRS04] where, however, we considered requirements written in LTL. Requirements have to be transformed into equivalent forms which apply to transformed feature and system descriptions too.

The definition (3) of $R_{S+F}$ for precomposed $F$ shows that the states $s$ of $S$ can be partitioned into three subsets with respect to the possible outgoing transitions of $S + F$:

- $s \not\in c_F$;
- $s \models c_F$, and $s$ triggers $F$;
- $s \models c_F$, but does not trigger $F$, because $s$ is the destination of a transition made by $F$.

In general, states from the second and the third kinds cannot be told apart out of the context of particular behaviours. States from the third set do not occur in observable behaviours, according to our definition of the working of precomposed features. However, (3) suggests that being aware of these states can simplify the separation between the contributions of $F$ and $S$ to the behaviour of $S+F$. We transform the descriptions of $S$ and $F$ so that these states become observable. This facilitates the considered separation at the cost of one additional variable, which we call $h$ (for hidden). The components of the transformed descriptions $S'$ and $F'$ of $S$ and $F$, respectively, are defined
as follows:

\[ P_{S'} = P_S \cup \{h\} \text{ and } P_{F'} = P_F \cup \{h\}; \]
\[ I_{S'} = \{s \in W_{S'} : s|_{P_S} \in I_s, s \in [\neg h]\}; \]
\[ c_{F'} = c_F \land \neg h; \]
\[ R_{S'}(s, s') \leftrightarrow R_S(s|_{P_S}, s'|_{P_S}) \land (s' \notin [h]); \]
\[ R_{F'}(s, s') \leftrightarrow R_F(s|_{P_S}, s'|_{P_F}) \land (s \notin [h]) \land (s' \in [h]). \]

In words, \( R_{S'} \) takes \( S' + F' \) from any state to a visible state, \( F \) becomes triggered only at visible states and \( R_{F'} \) takes \( S' + F' \) to hidden states. In all other aspects \( R_{S'} \) and \( R_{F'} \) are like \( R_S \) and \( R_F \), respectively. Obviously a sequence of states \( s_0s_1 \ldots s_n \ldots \) is a behaviour or \( S + F \) iff a behaviour of \( S' + F' \) can be obtained from it by appropriately inserting states which satisfy \( h \) and setting the value of \( h \) at the original states to \( 0 \).

Using \( W_{S'+F'} \) as an alphabet, properties of \( S' + F' \) behaviours define sublanguages of \( W_{S'+F'} \). Sublanguages of \( W_{S'+F'} \) which are closed under the replacement of \( h \)-similar valuations in their words can be regarded as properties of the behaviours of \( S + F \) as well. \( S + F \) has the property defined by the language \( L \) iff \( S' + F' \) has the property defined by \( L/\neg [h] \).

Symmetrically, \( S' \) and \( F' \) can be defined for postcomposed \( F \) as follows:

\[ I_{S'} = \{s \in W_{S'} : s|_{P_S} \in I_s, s \in [h]\}; \]
\[ c_{F'} = c_F \land h; \]
\[ R_{S'}(s, s') \leftrightarrow R_S(s|_{P_S}, s'|_{P_S}) \land (s' \in [h]); \]
\[ R_{F'}(s, s') \leftrightarrow R_F(s|_{P_S}, s'|_{P_F}) \land (s \in [h]) \land (s' \notin [h]). \]

\( P_{S'} \) and \( P_{F'} \) are as for precomposed \( F \).

Just like in the case of precomposed features, \( S+H \) has the property defined by the language \( L \) iff \( S' + F' \) has the property defined by \( L/\neg(h \land c_F) \).

Moving to \( S' \) and \( F' \) and the assumption of the visibility of all states leads to the simple form

\[ R_{S'+F'} = R_{S'}[\neg [h]] \cup R_{F'}, \]

of both (3) and (4), where \( R_{F'} \) is as in (5).

### 3 Features as strategies

In this section we explain the correspondence between winning strategies for infinite games and features which, if integrated in a system, cause its behaviours to have some given property. We present the main result of this paper, which is to show how the construction of a winning strategy of appropriate parity games can be used to synthesise a feature \( F \) for a given system \( S \)
and with a given guard so that $S + F$ satisfies a property specified by a given ω-regular language.

Let $S = \langle W_S, I_S, R_S \rangle$ be a given system and $P_S$ be the set of its variables. Let $P_F \subseteq P_S$ and $F = \langle c_F, P_F, R_F \rangle$ be a precomposed feature. Assume that $S$ and the guard $c_F$ of $F$ are fixed. Let $L \subseteq W_S^*$ represent a desirable property for $S + F$. Our goal is to find a transition relation $R_F$ for $F$ such that $[S + F] \subseteq L$, if one exists.

Let, for the sake of simplicity, $S$ have a unique initial state: $I_S = \{ s_I \}$. Let $S'$ and $F'$ denote the descriptions of $S$ and $F$ introduced in Section 2, respectively. Consider the initialised infinite game

$$G_{S,L,c_F} = \langle \langle V_0, V_1, E \rangle, W, v_I \rangle$$

whose components are defined by the equalities

$$V_0 = \{ c_{P'} \}, \quad V_1 = W_{S'} \setminus \{ c_{P'} \},$$

$$E(s, s') \leftrightarrow s' \in \llbracket c_{P'} \rrbracket \land R_{S'}(s, s') \lor s' \in \llbracket c_{P'} \rrbracket \land (s' \in [h]) \land s_{[P']_s} = s'_{[P']_s},$$

$$W = L/[-h],$$

$$v_I = s'_I, \text{ where } \{ s'_I \} = I_{S'}.$$

The arena of this game consists of the states of $S'$, which, since $P_F \subseteq P_S$, are also the states of $S' + F'$. The moves for player 1 are determined by the transition relation $R_{S'}$ of $S'$ and represent the contribution of $S'$ to the behaviour of $S' + F'$. The moves for player 0 are restricted only by the requirements on $F'$ to take the system to hidden states and not to alter variables outside $P_{F'}$. Hence player 1, who represents $F$, is free to choose its moves in order to satisfy the winning condition, which is the converse projection of the property formulated for $S + F$ onto the originally visible states of $S' + F'$ which do not satisfy $h$, and therefore is equivalent to $L$ for behaviours in which only these states are accounted of.

If player 0 has a winning strategy $\mu$ for $G_{S,L,c_F}$, then it can force any play of $G_{S,L,c_F}$ to satisfy the $S'$ counterpart $L/[-h]$ of $L$ by choosing the moves from the states which are supposed to trigger $F'$. For such a strategy to enable the definition of the transition relation $R_{F'}$ of a feature which achieves the same, it must be memoryless. For a memoryless $\mu$ the corresponding $R_{F'}$ can be defined by the equivalence

$$R_{F'}(s, s') \leftrightarrow \mu(s_0 \ldots s_n s) = s', \tag{8}$$

which defines $R_{F'}$ correctly, because $\mu(s_0 \ldots s_n s)$ depends only on $s$ for memoryless $\mu$. Note that (8) immediately implies that if a feature $F$ satisfying $[S + F] \subseteq L$ for the given $S$, $L$ and fixed $c_F$ exists, then player 0 has a memoryless winning strategy for $G_{S,L,c_F}$. Since parity games admit memoryless strategies, we next use Proposition 1.10 to move from $G_{S,L,c_F}$ to the parity game $G' = G_{S,L,c_F} \times A_L/[-h]$, where $A_L/[-h]$ stands for some parity automaton for $L/[-h]$. 

11
Now let us examine the effect of moving to $G_{S,L,F} \times A_{L[1-h]}$ on the correspondence with our given system and feature guard. The vertices of the arena of $G'$ consist of $S'$ states combined with states of $A_{L[1-h]}$. The moves of $G'$ are also combinations of $G$-moves, which are either transitions contributed by $S'$, or by $F'$, depending on the source state, augmented with transformations of the state of $A_{L[1-h]}$. We can assume that $A_{L[1-h]}$ is deterministic, which allows us to have a unique move in $G'$ correspond to each transition contributed by the given system $S'$. As for the transitions to be contributed by the feature, the $A_{L[1-h]}$-state components of the vertices provide the data needed in order to choose them so that the resulting set of behaviours is in $L[1-h]$, if this is possible at all.

This means that we have the following solution to our problem of synthesising features to achieve given properties:

Let $S$ be and let $L \subseteq W^S$ be $\omega$-regular. Let $c_F$ define a set of states of $S$. Let $S'$ be the extension of $S$ described in Section 2, $h$ be the variable involved in it and $A_{L[1-h]}$ be a deterministic parity automaton defining $L \subseteq W^S$. Let $S''$ be the extension of $S'$ by a speculative feature, which contributes a variable to hold a state of $A_{L[1-h]}$ and updates it according to the transitions of $A_{L[1-h]}$ that correspond to the transitions taken by $S'$. Then a precomposed feature $F$ which is triggered at states satisfying $c_F$ and is such that $S'' + F'$ has the property $L[1-h]$ exists iff player 0 has a (memoryless) winning strategy for the game $G'$ described above. In this case the transition relation of $F'$ can be defined by (8). If there is no such strategy, then no speculative extension of $S$ admits a feature implementing the property defined by $L$.

The construction for postcomposed features is similar.

4 Some related issues and a generalisation

Note that in general the system $S+F$ as described in Section 3 is not guaranteed to inherit any properties from $S$. The preservation of properties of $S$ by $S+F$ can be checked using techniques from [GRS04]. Sometimes the preservation of $S$ properties can be derived from the restrictions on $F$ imposed through its guard and the variables it can update. Given these, to check whether a feature $F$ exists such that $S+F$ does not have a certain property is equivalent to checking whether $S+F_{\text{max}}$ has the property, where

$$F_{\text{max}} = \langle c_F, P_F, W_S \times (P_F \rightarrow \{0,1\}) \rangle$$

is the feature which enables every behaviour that a system of the form $S+F$ can have.

---

4 Strictly speaking, these have to be several propositional variables, as many as necessary to accommodate the binary representation of a state of $A_{L[1-h]}$. 

12
A property may take adding both precomposed and postcomposed features to achieve. While the resulting system can be described in the form $S + F_1 + F_2 + \ldots$, adding the features one by one is not an option regarding synthesis, because a property of the final system is aimed. Hence we need to consider features which are both precomposed and postcomposed. The techniques described in Sections 2 and 3 can be adapted to this case too by means of two variables to mark the hidden states contributed by each feature.

Concluding remarks

We have described an algorithmic method for synthesising features that have given triggering conditions and access to specified variables and are required to cause the respective systems to satisfy given requirements. It allows both to decide the possibility to synthesise a feature of the given form and with the required properties and to determine how to extend the respective base system to allow the integration of the desired feature. The method is based on the theory of $\omega$-automata and infinite games, which has been developed for more general purposes. The results from this theory turn out to be straightforwardly applicable to our problem and allow us to propose an exhaustive solution.

Acknowledgements

We thank an anonymous referee for pointing out that our method can be implemented using games and deterministic automata with other forms of winning and accepting conditions such as those due to Rabin and Streett as well, and that there are complexity trade-offs between the choices.

The presentation of this paper at AVoCS by Dimitar Guelev was supported by the Paul and Yuanbi Ramsay Fund at the University of Birmingham.

References


