Resource Interfaces*

Arindam Chakrabarti[1], Luca de Alfaro[2], Thomas A. Henzinger[1], and Mariëlle Stoelinga[2]

[1] Electrical Engineering and Computer Sciences, UC Berkeley
[2] Computer Engineering, UC Santa Cruz

Abstract. We present a formalism for specifying component interfaces that expose component requirements on limited resources. The formalism permits an algorithmic check if two or more components, when put together, exceed the available resources. Moreover, the formalism can be used to compute the quantity of resources necessary for satisfying the requirements of a collection of components. The formalism can be instantiated in several ways. For example, several components may draw power from the same source. Then, the formalism supports compatibility checks such as: can two components, when put together, achieve their tasks without ever exceeding the available amount of peak power? or, can they achieve their tasks by using no more than the initially available amount of energy (i.e., power accumulated over time)? The corresponding quantitative questions that our algorithms answer are the following: what is the amount of peak power needed for two components to be put together? what is the corresponding amount of initial energy? To solve these questions, we model interfaces with resource requirements as games with quantitative objectives. The games are played on state spaces where each state is labeled by a number (representing, e.g., power consumption), and a play produces an infinite path of labels. The objective may be, for example, to minimize the largest label that occurs during a play. We illustrate our approach by modeling compatibility questions for the components of robot control software, and of wireless sensor networks.

1 Introduction

In component-based design, a central notion is that of *interfaces*: an interface should capture those facts about a component that are necessary and sufficient for determining if a collection of components fits together. The formal notion of interface, then, depends on what "fitting together" means. In a simple case, an interface exposes only type information about the component's inputs and outputs, and "fitting together" is determined by type checking. In a more ambitious case, an interface may expose also temporal information about inputs and outputs. For example, the temporal interface of a file server may specify that the

* This research was supported in part by the DARPA grant F33615-00-C-1693, the MARCO grant 98-DT-660, the ONR grant N00014-02-1-0671, and the NSF grants CCR-0085949, CCR-0132780, CCR-0234690, and CCR-9988172.

`open_file` method must be called before the `read_file` method is invoked. If a client, instead, calls `read_file` before `open_file`, then an interface violation occurs. In [2], we argued that temporal interfaces are *games*. There are two players, Input and Output, and an objective, namely, the absence of interface violations. Then, an interface is *well-formed* if the corresponding component can be used in some environment; that is, player Input has a strategy to achieve the objective. Moreover, two interfaces are *compatible* if the corresponding components can be used together in some environment; that is, the composition of the two games is well-formed, and specifies the composite interface.

Here, we develop the theory of interfaces as games further, by introducing interfaces that expose resource information. Consider, for example, components whose power consumption varies. We model the interface of such a component as a control flow graph whose states are labeled with integers, which represent the power consumption while control is at that state. For instance, in the thread-based programming model for robot motor control presented in Section 5, the power consumption of a program region depends on how many motors and other devices are active. Now suppose that we want to put together two threads, each of which consumes power, but the overall amount of available peak power is limited to a fixed amount Δ. The threads are controlled by a scheduler, which at all times determines the thread that may progress. Then the two threads are *Δ-compatible* if the scheduler has a strategy to let them progress in a way so that their combined power consumption never exceeds Δ. In more detail, the game is set up as follows: player Input is the scheduler, and player Output is the composition of the two threads. At each round of the game, player Input determines which thread may proceed, and player Output determines the successor state in the control flow graph of the scheduled thread. In this game, in order to avoid a safety violation (power consumption greater than Δ), player Input may not let any thread progress. To rule out such trivial schedules, one may augment the safety objective with a secondary, liveness objective, say, in the form of a Büchi condition, which specifies that the scheduler must allow each thread to progress infinitely often. The resulting compatibility check, then, requires the solution of a Büchi game: the two threads are Δ-compatible iff player Input has a strategy to satisfy the Büchi condition without exceeding the power threshold Δ.

The basic idea of formalizing interfaces as such *Büchi threshold games* on integer-labeled graphs has many applications besides power consumption. For example, access to a mutex resource can be modeled by state labels 0 and 1, where 1 represents usage of the resource. Then, if we choose $\Delta = 1$, two or more threads are Δ-compatible if at any time at most one of the threads uses the resource. In Section 5, we will also present an interface model for the clients of a wireless network, where each state label represents the number of active messages at a node of the network, and Δ represents the buffer size. In this example, the Δ-compatibility check synthesizes not a scheduling strategy but a routing protocol that keeps the message buffers from overflowing.

A wide variety of other formalisms for the modeling and analysis of resource constraints have been proposed in the literature (e.g., [7–9, 11]). The essential difference between these papers and our work is that our models and analysis techniques are based on games. Once resource interfaces are modeled as games on graphs with integer labels, in addition to the *boolean* question of Δ-compatibility, for fixed Δ, we can also ask a corresponding *quantitative* question about resource requirements: What is the minimal resource level (peak power, buffer size, etc.) Δ necessary for two or more given interfaces to be compatible? To formalize the quantitative question, we need to define the *value* of an outcome of the game, which is the infinite sequence of states that results from playing the game for an infinite number of rounds. For Büchi threshold games, the value of an outcome is the supremum of the power consumption over all states of the outcome. The player Input (the scheduler) tries to minimize the value, while the player Output (the thread set) tries to maximize. The quantitative question, then, asks for the inf-sup of the value over all player Input and Output strategies.

The threshold interfaces, where an interface violation occurs if a power threshold is exceeded at any one time, provide but one example of how the compatibility of resource interfaces may be defined. We also present a second use of resource interfaces, where a violation occurs when an initially available amount Δ of energy (given, say, by the capacity of a battery) is exhausted. In this case, the value u of a finite outcome is defined as the *sum* (rather than maximum) over all labels of the states of the outcome, and player Input (the scheduler) wins if it can keep $\Delta - u$ nonnegative forever, or until a certain task is achieved. Note that in this game, negative state labels can be used to model a recharging of the energy source. Achieving a task might be modeled again by a Büchi objective, but for variety's sake, we use a quantitative *reward* objective in our formalization of such *energy interfaces*. For this purpose, we label each state with a second number, which represents a reward, and the objective of player Input is to obtain a total accumulated reward of Λ. The boolean Δ-compatibility question, then, asks if Λ can be obtained from the composition of two interfaces without exceeding the initial energy supply Δ. The corresponding quantitative resource-requirement question asks for the minimum initial energy supply Δ necessary to achieve the fixed reward Λ. Dually, a similar algorithm can be used to determine the maximal achievable reward Λ given a fixed initial energy supply Δ. In particular, if every state offers reward 1, this asks for the maximum runtime of a system (in number of state transitions) that a scheduler can achieve with a given battery capacity.

The paper is organized as follows. Section 2 reviews the definitions needed for modeling temporal (resourceless) interfaces as games and Section 3 adds resources to these games: we introduce integer labels on states to model resource usage, and we define boolean as well as quantitative objective functions on the outcomes of a game. As examples, we define four specific resource-interface theories: threshold games without and with Büchi objectives, and energy games without and with reward objectives. For these four theories, Section 4 gives algorithms for solving the boolean Δ-compatibility and the quantitative resource-requirement questions. These interface theories are also used in the two case studies of Section 5, one on scheduling embedded threads for robot control, and the other on routing messages across wireless networks.

2 Preliminaries

An *interface* is a system model that represents both the behavior of a component, and the behavior the component expects from its environment [2]. An interface communicates with its environment through input and output variables. The interface consists of a set of states. Associated with each state is an input assumption, which specifies the input values that the component is ready to accept from the environment, and an output guarantee, which specifies the output values that the component can generate. Once the input values are received and the output values are generated, these values cause a transition to a new state. In this way, both input assumptions and output guarantees can change dynamically. Formally, an *assume-guarantee* (A/G) *interface* [3] is a tuple $M = \langle V^i, V^o, Q, \hat{q}, \phi^i, \phi^o, \rho \rangle$ consisting of:

- Two finite sets V^i and V^o of boolean *input* and *output variables*. We require that $V^i \cap V^o = \emptyset$.
- A finite set Q of *states*, including an initial state $\hat{q} \in Q$.
- Two functions ϕ^i and ϕ^o which assign to each state $q \in Q$ a satisfiable predicate $\phi^i(q)$ on V^i, called *input assumption*, and a satisfiable predicate $\phi^o(q)$ on V^o, called *output guarantee*.
- A function ρ which assigns to each pair $q, q' \in Q$ of states a predicate $\rho(q, q')$ on $V^i \cup V^o$, called the *transition guard*. We require that for every state $q \in Q$, we have (1) $(\phi^i(q) \wedge \phi^o(q)) \Rightarrow \bigvee_{q' \in Q} \rho(q, q')$ and (2) $\bigwedge_{q', q'' \in Q}((\rho(q, q') \wedge \rho(q, q'')) \Rightarrow (q' = q''))$. Condition (1) ensures nonblocking; condition (2) ensures determinism.

We refer to the states of M as Q_M, etc. Given a set V of boolean variables, a *valuation* v for V is a function that maps each variable $x \in V$ to a boolean value $v(x)$. A valuation for the input (resp. output) variables V^i is called *input* (resp. *output*) *valuation*. We write $\mathbb{V}^i$ and $\mathbb{V}^o$ for the sets of input and output valuations.

Interfaces as games. An interface is best viewed as a game between two players, Input and Output. The game $G_M = \langle Q, \hat{q}, \gamma^i, \gamma^o, \delta \rangle$ associated with the interface M is played on the set Q of states of the interface. At each state $q \in Q$, player Input chooses an input valuation v^i that satisfies the input assumption, and simultaneously and independently, player Output chooses an output valuation v^o that satisfies the output guarantee; that is, at state q the moves available to player Input are $\gamma^i(q) = \{v \in \mathbb{V}^i \mid v \models \phi^i(q)\}$, and the moves available to player Output are $\gamma^o(q) = \{v \in \mathbb{V}^o \mid v \models \phi^o(q)\}$. Then the game proceeds to the state $q' = \delta(q, v^i, v^o)$, which is the unique state in Q such that $(v^i \uplus v^o) \models \rho(q, q')$. The result of the game is a run. A *run* of M is an infinite sequence $\pi = q_0, (v_0^i, v_0^o), q_1, (v_1^i, v_1^o), q_2, \ldots$ of alternating states $q_k \in Q$, input valuations $v_k^i \in \mathbb{V}^i$, and output valuations $v_k^o \in \mathbb{V}^o$, such that for all $k \geq 0$, we have (1) $v_k^i \in \gamma^i(q_k)$ and $v_k^o \in \gamma^o(q_k)$, and (2) $q_{k+1} = \delta(q_k, v_k^i, v_k^o)$. The run π is *initialized* if $q_0 = \hat{q}$. A *run prefix* is a finite prefix of a run which ends in a state. Given a run prefix π, we write $last(\pi)$ for the last state of π.

In a game, the players choose moves according to strategies. An *input strategy* is a function that assigns to every run prefix π an input valuation in $\gamma^i(last(\pi))$, and an *output strategy* is a function that assigns to every run prefix π an output valuation in $\gamma^o(last(\pi))$. We write Σ^i and Σ^o for the sets of input and output strategies. Given a state $q \in Q$, and a pair $\sigma^i \in \Sigma^i$ and $\sigma^o \in \Sigma^o$ of strategies, the *outcome* of the game *from* q is the run $out(q, \sigma^i, \sigma^o) = q_0, (v_0^i, v_0^o), q_1, (v_1^i, v_1^o), \ldots$ such that (1) $q_0 = q$ and (2) for all $k \geq 0$, we have $v_k^i = \sigma^i(q_0, \ldots, q_k)$ and $v_k^o = \sigma^o(q_0, \ldots, q_k)$.

The *size* of the A/G interface M is taken to be the size of the associated game G_M: define $|M| = \sum_{q \in Q} |\gamma^i(q)| \cdot |\gamma^o(q)|$. Since the interface M is specified by predicates on boolean variables, the size $|M|$ may be exponentially larger than the syntactic description of M.

Compatibility and composition. The basic principle is that two interfaces are compatible if they can be made to work together correctly. When two interfaces are composed, the outputs of one interface may be fed as inputs to the other interface. Thus, the possibility arises that the output behavior of one interface violates the input assumptions of the other. The two interfaces are called compatible if the environment can ensure that no such I/O violations occur. The assurance that the environment behaves in a way that avoids all I/O violations is, then, the input assumption of the composite interface. Formally, given two A/G interfaces M and N, define $V^o = V_M^o \cup V_N^o$ and $V^i = (V_M^i \cup V_N^i) \setminus V^o$. Let $Q = Q_M \times Q_N$ and $\hat{q} = (\hat{q}_M, \hat{q}_N)$. For all $(p, q), (p', q') \in Q_M \times Q_N$, let $\phi^o(p, q) = (\phi_M^o(p) \wedge \phi_M^o(q))$ and $\rho((p, q), (p', q')) = (\rho_M(p, p') \wedge \rho_N(q, q'))$. The interfaces M and N are *compatible* if (1) $V_M^o \cap V_N^o = \emptyset$ and (2) there is a function ψ^i that assigns to all states $(p, q) \in Q$ a satisfiable predicate on V^i such that:

> For all initialized runs $(p_0, q_0), (v_0^i, v_0^o), (p_1, q_1), (v_1^i, v_1^o), \ldots$ of the A/G interface $\langle V^i, V^o, Q, \hat{q}, \psi^i, \phi^o, \rho \rangle$ and all $k \geq 0$, we have $(v_k^i \uplus v_k^o) \models (\phi_M^i(p_k) \wedge \phi_N^i(q_k))$. (†)

If M and N are compatible, then the *composition* $M \| N = \langle V^i, V^o, Q, \hat{q}, \phi^i, \phi^o, \rho \rangle$ is the A/G interface with the function ϕ^i that maps each state $(p, q) \in Q$ to a satisfiable predicate on V^i such that for all functions ψ^i that satisfy the condition (†), and all $(p, q) \in Q$, we have $\psi^i(p, q) \Rightarrow \phi^i(p, q)$; i.e., the input assumptions ϕ^i are the weakest conditions on the environment of the composite interface $M \| N$ which guarantee the input assumptions of both components. Algorithms for checking compatibility and computing the composition of A/G interfaces are given in [3]. These algorithms use the game representation of interfaces.

3 Resource Interfaces

Let $\mathbb{Z}_\infty = \mathbb{Z} \cup \{\pm\infty\}$. A *resource algebra* is a tuple $A = \langle \mathbb{L}, \oplus, \Theta \rangle$ consisting of:

– A set $\mathbb{L}$ of *resource labels*, each signifying a level of consumption or production for a set of resources.
– A binary *composition operator* $\oplus$: $\mathbb{L}^2 \to \mathbb{L}$ on resource labels.
– A *value function* Θ: $\mathbb{L}^\omega \to \mathbb{Z}_\infty$, which assigns an integer value (or infinity) to every infinite sequence of resource labels.

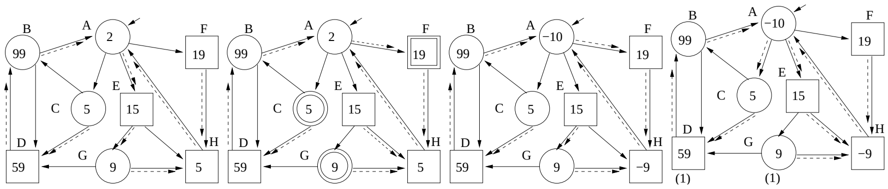
A *resource interface* over A is a pair $R = (M, \lambda)$ consisting of an A/G interface $M = \langle \cdot, \cdot, Q, \hat{q}, \cdot, \cdot, \cdot \rangle$ and a labeling function λ: $Q \to \mathbb{L}$, which maps every state of the interface to a resource label. The *size* of the resource interface is $|R| = |M| + \sum_{q \in Q} |\lambda(q)|$. The runs of R are the runs of M, etc. Given a run $\pi = q_0, (v_0^i, v_0^o), q_1, (v_1^i, v_1^o), \ldots$, we write $\lambda(\pi) = \lambda(q_0), \lambda(q_1), \ldots$ for the induced infinite sequence of resource labels. Given a state $q \in Q$, the *value at* q is the minimum value that player Input can achieve for the outcome of the game from q, irrespective of the moves chosen by player Output: $\mathit{val}(q) = \inf_{\sigma^i \in \Sigma^i} \sup_{\sigma^o \in \Sigma^o} \Theta(\lambda(\mathit{out}(q, \sigma^i, \sigma^o)))$. The state q is *Δ-compliant*, for $\Delta \in \mathbb{Z}_\infty$, if $\mathit{val}(q) \leq \Delta$. We write $Q_\Delta^{rc} \subseteq Q$ for the set of Δ-compliant states. The resource interface R is *Δ-compliant* if the initial state $\hat{q}$ is Δ-compliant, and the *value* of R is $\mathit{val}(\hat{q})$.

Given two resource interfaces R and S over the same resource algebra A, define λ: $Q_R \times Q_S \to \mathbb{L}$ such that $\lambda(p, q) = \lambda_R(p) \oplus \lambda_S(q)$. The resource interfaces R and S are *Δ-compatible*, for $\Delta \in \mathbb{Z}_\infty$, if (1) the underlying A/G interfaces M_R and M_S are compatible, and (2) the resource interface $(M_R \| M_S, \lambda)$ over A is Δ-compliant. Note that Δ-compatibility does not require that both component interfaces R and S are Δ-compliant. Indeed, if R consumes a resource produced by S, it may be the case that R is not Δ-compliant on its own, but is Δ-compliant when composed with S. This shows that different applications call for different definitions of composition for resource interfaces, and we refrain from a generic definition. We use, however, the abbreviation $R \| S = (M_R \| M_S, \lambda)$.

The class of resource interfaces over a resource algebra A is denoted $\mathcal{R}[A]$. We present four examples of resource algebras and the corresponding interfaces.

Pure threshold interfaces. The resource labels of a threshold interface specify, for each state q, an amount $\lambda(q) \in \mathbb{N}$ of resource usage in q (say, power consumption). When the states of two interfaces are composed, their resource usage is additive. The number $\Delta \geq 0$ provides an upper bound on the amount of resource available at every state. A state q is Δ-compliant if player Input can ensure that, when starting from q, resource usage never exceeds Δ. The value at q is the minimum amount Δ of resource that must be available at all states for q to be Δ-compliant. Formally, the *pure threshold algebra* A^t is the resource algebra with $\mathbb{L}^t = \mathbb{N}$ and $\oplus^t = +$ and $\Theta^t(n_0, n_1, \ldots) = \sup_{k \geq 0} n_k$. The resource interfaces in $\mathcal{R}[A^t]$ are called *pure threshold interfaces*. Throughout the paper, we assume that all numbers, including the state labels $\lambda(q)$ of pure threshold interfaces as well as Δ, can be stored in space of a fixed size. It follows that the size of a pure threshold interface $R = (M, \lambda)$ is equal to the size of the underlying A/G interface M.

Example 1 Figure 1(a) shows the game associated with a pure threshold interface. For simplicity, the example is a turn-based game in which player Input



(a) (b) (c) (d)

Fig. 1. Games illustrating the four classes of resource interfaces.

makes moves in circle states, and player Output makes moves in square states. The numbers inside the states represent their resource labels. The solid arrows show the moves available to the players, and the dashed arrows indicate the optimal strategies for the two players. Note that at the initial state A, state E is a better choice than C for player Input in spite of having a greater resource label. It is easy to see that the value of the game (at A) is 15. □

Büchi threshold interfaces. While pure threshold interfaces ensure the safe usage of a threshold resource, they may allow some systems to never use the resource by not doing anything useful. To rule out this possibility, we may augment the pure threshold algebra with a generalized Büchi objective, which requires that certain state labels be visited infinitely often. A state q, then, is Δ-compliant if player Input can ensure that, when starting from q, the Büchi conditions are satisfied *and* resource usage never exceeds Δ. The formal definition of Büchi conditions within a resource algebra is somewhat technical. The *Büchi threshold algebra* A^{bt} is defined as follows:

- $\mathbb{L}^{bt}$ consists of triples $\langle n, \alpha, \beta \rangle$, where $n \in \mathbb{N}$ indicates the current level of resource usage, α is a set of state labels that need to be repeated infinitely often, and β is the set of state labels that are satisfied in the current state.
- $\langle n, \alpha, \beta \rangle \oplus^{bt} \langle n', \alpha', \beta' \rangle = \langle n + n', \alpha \uplus \alpha', \beta \uplus \beta' \rangle$.
- We distinguish two cases, depending on whether or not the generalized Büchi objective is violated: $\Theta^{bt}(\langle n_0, \alpha_0, \beta_0 \rangle, \langle n_1, \alpha_1, \beta_1 \rangle, \ldots) = +\infty$ if there is an $\ell \in \alpha_0$ and a $k \geq 0$ such that for all $j \geq k$, we have $\ell \notin \beta_j$; otherwise, $\Theta^{bt}(\langle n_0, \alpha_0, \beta_0 \rangle, \langle n_1, \alpha_1, \beta_1 \rangle, \ldots) = \sup_{k \geq 0} n_k$.

The resource interfaces in $\mathcal{R}[A^{bt}]$ are called *Büchi threshold interfaces.* The *number of Büchi conditions* of a Büchi threshold interface $R = (M, \lambda)$ is $|\hat{\alpha}|$, where $\hat{\alpha}$ is the second component of the label $\lambda(\hat{q})$ for the initial state $\hat{q}$ of M.

Example 2 Figure 1(b) shows a Büchi threshold game with a single Büchi condition. The graph is the same as in Example 1. The states with double borders are Büchi states, i.e., one of them needs to be repeated infinitely often. Note that the optimal output strategy at E has changed, because C is a Büchi state but H is not. This forces player Input to prefer at A state F over E in order to satisfy the Büchi condition. The value of the game is now 19. □

Pure energy interfaces. The resource labels of an energy interface specify, for each state q, the amount of energy $\lambda(q) \in \mathbb{Z}$ that is produced (if $\lambda(q) > 0$) or consumed (if $\lambda(q) < 0$) at q. When the states of two interfaces are composed, their energy expenditures are added. The number $\Delta \geq 0$ provides the initial amount of energy available. A state q is Δ-compliant if player Input can ensure that, when starting from q, the system can run forever without the available energy dropping below 0. The value at q is the minimum amount Δ of initial energy necessary for q to be Δ-compliant. Formally, the *pure energy algebra* A^e is the resource algebra with $\mathbb{L}^e = \mathbb{Z}$ and $\oplus^e = +$ and $\Theta^e(d_0, d_1, \ldots) = -\inf_{k \geq 0} \sum_{0 \leq j \leq k} d_j$. The resource interfaces in $\mathcal{R}[A^e]$ are called *pure energy interfaces*. The *maximal energy consumption* of a pure energy interface $R = (M, \lambda)$ is 1 if $\lambda(q) \geq 0$ for all states $q \in Q$, and $-\min_{q \in Q} \lambda(q)$ otherwise.

Example 3 Figure 1(c) shows a pure energy game. Player Input has a strategy to run forever when starting from the initial state A with 9 units of energy, but 8 is not enough initial energy; thus the game has the value 9. □

Reward energy interfaces. Some systems have the possibility of saving energy by doing nothing useful. To rule out this possibility, we may use a Büchi objective as in the case of threshold interfaces. For variety's sake, we provide a different approach. We label each state q not only with an energy expenditure, but also with a reward, which represents the amount of useful work performed by the system when visiting q. A reward energy algebra specifies a minimum acceptable reward Λ. A state q, then, is Δ-compliant if player Input can ensure that, when starting from q, the reward Λ can be obtained without the available energy dropping below 0. For $\Lambda \in \mathbb{N}$, the *Λ-reward energy algebra* A^{re}_Λ is defined as follows:

- $\mathbb{L}^{re} = \mathbb{Z} \times \mathbb{N}$. The first component of each resource label represents an energy expenditure; the second component represents a reward.
- $\langle d, n \rangle \oplus^{re} \langle d', n' \rangle = \langle d + d', n + n' \rangle$.
- There are two cases: $\Theta^{re}_\Lambda(\langle d_0, n_0 \rangle, \langle d_1, n_1 \rangle, \ldots) = +\infty$ if $\sum_{j \geq 0} n_j < \Lambda$; otherwise, let $k^* = \min_{k \geq 0}(\sum_{0 \leq j \leq k} n_j \geq \Lambda)$ and define $\Theta^{re}_\Lambda(\langle d_0, n_0 \rangle, \langle d_1, n_1 \rangle, \ldots) = -\inf_{0 \leq k \leq k^*} \sum_{0 \leq j \leq k} d_j$.

The resource interfaces in $\mathcal{R}[A^{re}_\Lambda]$ are called *Λ-reward energy interfaces*. The *maximal energy consumption* of a reward energy interface is defined as for pure energy interfaces, with the proviso that only the energy (i.e., first) components of resource labels are considered.

Example 4 Figure 1(d) shows a Λ-reward energy game with $\Lambda = 1$. The numbers in parentheses represent rewards; states that are not labeled with parenthesized numbers have reward 0. The optimal choice of player Input at state A is E, precisely the opposite of the pure energy case. If player Output chooses G at E, then the reward 1 is won, and player Input's objective is accomplished. If player Output instead chooses H at E, then 4 units of energy are gained in the cycle A,E,H,A. By pumping this cycle, player Input can gain sufficient energy to eventually choose the path A,C,D and win the reward 1. Hence the game has the value 5. Note that this example shows that reward energy games may not have memoryless winning strategies. □

4 Algorithms

Let A be a resource algebra. We are interested in the following questions:

Verification Given two resource interfaces $R, S \in \mathcal{R}[A]$, and $\Delta \in \mathbb{Z}_\infty$, are R and S Δ-compatible?

Design Given two resource interfaces $R, S \in \mathcal{R}[A]$, for which values $\Delta \in \mathbb{Z}_\infty$ are R and S Δ-compatible?

To answer these questions, we first need to check the compatibility of the underlying A/G interfaces M_R and M_S. Then, for the qualitative verification question, we need to check if the resource interface $R \| S \in \mathcal{R}[A]$ is Δ-compliant, and for the quantitative design question, we need to compute the value of $R \| S$. Below, for $A \in \{A^t, A^{bt}, A^e, A^{re}\}$, we provide algorithms for checking if a given resource interface $R \in \mathcal{R}[A]$ is Δ-compliant, and for computing the value of R. We present the algorithms in terms of the game representation $G_R = \langle Q, \hat{q}, \gamma^i, \gamma^o, \delta \rangle$ of the interface. The algorithms have been implemented in our tool CHIC [12].

Pure threshold interfaces. For $n \in \mathbb{N}$, let $Q_{\leq n} = \{q \in Q \mid \lambda(q) \leq n\}$. For $\Delta \geq 0$, a pure threshold interface R is Δ-compliant iff player Input can win a game with the safety objective of staying forever in $Q_{\leq \Delta}$. Such safety games can be solved as usual using a *controllable predecessor* operator *CPre*: $2^Q \to 2^Q$, defined for all $X \subseteq Q$ by $\mathit{CPre}(X) = \{q \in Q \mid \exists v^i \in \gamma^i(q). \forall v^o \in \gamma^o(q). \delta(q, v^i, v^o) \in X\}$. The set of Δ-compliant states can then be written as the limit $Q^{rc}_\Delta = \lim_{k\to\infty} X_k$ of the sequence defined by $X_0 = Q$ and, for $k \geq 0$, by $X_{k+1} = Q_{\leq \Delta} \cap \mathit{CPre}(X_k)$. This algorithm can be written in μ-calculus notation as $Q^{rc}_\Delta = \nu X. (Q_{\leq \Delta} \cap \mathit{CPre}(X))$, where ν is the greatest fixpoint operator.

To compute the value of R, we introduce two mappings *lmax*: $2^Q \to \mathbb{N}$ and *below*: $2^Q \to 2^Q$. For $X \subseteq Q$, let $\mathit{lmax}(X) = \max\{\lambda(q) \mid q \in X\}$ be the maximum label of a state in X, and let $\mathit{below}(X) = \{q \in X \mid \lambda(q) < \mathit{lmax}(X)\}$ be the set of states with labels below the maximum. Then, define $X_0 = Q$ and, for $k \geq 0$, define $X_{k+1} = \nu X. (\mathit{below}(X_k) \cap \mathit{CPre}(X))$. For $k \geq 0$ and $q \in X_k \setminus X_{k+1}$, we have $\mathit{val}(q) = \mathit{lmax}(X_k)$.

While it may appear that computing the fixpoint $\nu X. (Q_{\leq \Delta} \cap \mathit{CPre}(X))$ requires quadratic time (computing *CPre* is linear in $|R|$, and we need at most $|Q|$ iterations), this can be accomplished in linear time. The trick is to use a refined version of the algorithm, where each move pair $\langle v^i, v^o \rangle$ is considered at most once. First, we remove from the fixpoint all states q' such that $\lambda(q') > \Delta$. Whenever a state $q' \in Q$ is removed from the fixpoint, we propagate the removal backward, removing for all $q \in Q$ any move pair $\langle v^i, v^o \rangle \in \langle \gamma^i(q), \gamma^o(q) \rangle$ such that $\delta(q, v^i, v^o) = q'$ and, whenever $\langle v^i, v^o \rangle$ is removed, removing also $\langle v^i, \hat{v}^o \rangle$ for all $\hat{v}^o \in \gamma^o(q)$. The state q is itself removed if all its move pairs are removed. Once the removal propagation terminates, the states that have not been removed are precisely the Δ-compliant states. In order to implement efficiently the algorithm for computing the value of a threshold interface, we compute X_{k+1} from X_k by removing the states having the largest label, and then back-propagating the removal. In order to compute $below(X_k)$ efficiently for all k, we construct a list of states sorted according to their label.

Theorem 1 *Given a pure threshold interface R of size n, and $\Delta \in \mathbb{Z}_\infty$, we can check the Δ-compliance of R in time $O(n)$, and we can compute the value of R in time $O(n \cdot \log n)$.*

Büchi threshold interfaces. Given a Büchi threshold interface R, let $\lambda(\hat{q}) = \langle \hat{n}, \hat{\alpha}, \hat{\beta} \rangle$, $|\hat{\alpha}| = m$, and $\hat{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$. Let $B^i = \{q \in Q \mid \lambda(q) = \langle n^q, \alpha^q, \beta^q \rangle$ and $\alpha_i \in \beta^q\}$ be the i-th set in the generalized Büchi objective, for $1 \le i \le m$. We can compute the set of Δ-compliant states of R by adapting the fixpoint algorithm for solving Büchi games [5] as follows. Given two sets $Z, T \subseteq Q$ of states, we define $Reach(Z,T) \subseteq Q$ as the set of states from which player Input can force the game to T while staying in Z. Formally, define $Reach(Z,T) = \lim_{k \to \infty} W_k$, where $W_0 = \emptyset$ and $W_{k+1} = Z \cap (T \cup CPre(W_k))$ for $k \ge 0$. Then, for $Z \subseteq Q$ and $1 \le i \le m$, we compute the sets $Y^i \subseteq Q$ as follows. Let $i' = (i \mod m) + 1$ be the successor of i in the cyclic order $1, 2, \ldots, m, 1, \ldots$ Let $Y_0^i = Q$, and for $j \ge 0$, let $Y_{j+1}^i = Reach(Z, B^i \cap CPre(Y_j^{i'}))$. Intuitively, the set Y_{j+1}^i consists of the states from which Input can first reach B^i, and then go to $Y_j^{i'}$ while staying in Z. For $1 \le i \le m$, let the fixpoint be $Y^i = \lim_{j \to \infty} Y_j^i$: from Y^i, Input can reach B^i while staying in Z; moreover, once at B^i, Input can proceed to $Y^{i'}$. Hence, Input can visit the sets $B^1, B^2, \ldots, B^m, B^1, \ldots$ cyclically, satisfying the generalized Büchi acceptance condition. Denoting by $GBüchi(Z, B^1, \ldots, B^m) = Y^1 \cup Y^2 \cup \ldots \cup Y^m$, we can write the set of Δ-compliant states of the interface as $Q_\Delta^{rc} = GBüchi(Q_{\le \Delta}, B^1, \ldots, B^m)$.

The algorithm for computing the value of a Büchi threshold interface can be obtained by adapting the algorithm for Δ-compliance, similarly to the case for pure threshold interfaces. Let $X_0 = Q$, and for $k \ge 1$, let $X_{k+1} = GBüchi(below(X_k), B^1, \ldots, B^m)$. Then, for a state $q \in X_k \setminus X_{k+1}$, we have $val(q) = lmax(X_k)$.

Since the set $Reach(Z,T)$ can be computed in time $O(m \cdot |R|)$, using again a backward propagation procedure, the computation of the set of Δ-compliant states of the interface requires time $O(m \cdot |R|^2)$, in line with the complexity for solving Büchi games. The value of Büchi threshold games can also be computed in the same time. In fact, Y^i for iteration $k+1$ (denoted $Y^i(k+1)$) can be obtained from Y^i for k (denoted $Y^i(k)$) by $Y_0^i(k+1) = Y^i(k)$ and, for $j \ge 0$, by $Y_{j+1}^i(k+1) = Reach(X_k \cap Y^i(k), B^i \cap CPre(Y_j^{i'}(k+1)))$. We then have $Y^i(k+1) = \lim_{j \to \infty} Y_j^i(k+1)$. Hence, for $1 \le i \le m$, the sets $Y^i(0)$, $Y^i(1)$, $Y^i(2)$, ... can be computed by progressively removing states. As each removal (which requires the computation of $Reach$) is linear-time, the overall algorithm is quadratic.

Theorem 2 *Given a Büchi threshold interface R of size n with m Büchi conditions, and $\Delta \in \mathbb{Z}_\infty$, we can check the Δ-compliance of R and compute its value in time $O(n^2 \cdot m)$.*

Pure energy interfaces. Given a pure energy interface R, the value at state $q \in Q$ is given by $val(q) = \inf_{\sigma^i \in \Sigma^i} \sup_{\sigma^o \in \Sigma^o} \{\Theta(\lambda(out(q, \sigma^i, \sigma^o)))\}$. This problem is technically similar to the solution of games with respect to the *total reward* criterion [6]. To solve it, we define an *energy predecessor operator* $EPre$: $(Q \to \mathbb{Z}_\infty) \to (Q \to \mathbb{Z}_\infty)$, defined for all $f\colon Q \to \mathbb{Z}_\infty$ and $q \in Q$ by

$$EPre(f)(q) = -\lambda(q) + \max\{0, \min_{v^i \in \gamma^i(q)} \max_{v^o \in \gamma^o(q)} f(\delta(q, v^i, v^o))\}.$$

Intuitively, $EPre(f)(q)$ represents the minimum energy Input needs for performing one step from q without exhausting the energy, and then continuing with energy requirement f. Consider the sequence of functions $f_0, f_1, \ldots\colon Q \to \mathbb{Z}_\infty$, where f_0 is the constant function such that $f_0(q) = -\infty$ for all $q \in Q$, and where $f_{k+1} = EPre(f_k)$ for $k \geq 0$. The functions in the sequence are pointwise increasing: for all $q \in Q$ and $k \geq 0$, we have $f_k(q) \leq f_{k+1}(q)$. Hence the limit $f_* = \lim_{k \to \infty} f_k$ (defined pointwise) always exists. From the definition of $EPre$, it can be shown by induction that $f_*(q) = val(q)$. The problem is that the sequence $f_0, f_1, \ldots$ may not converge to f_* in a finite number of iterations. For example, if the game has a state q with $\lambda(q) < 0$ and whose only transitions are self-loops, then $f_*(q) = +\infty$, but the sequence $f_0(q), f_1(q), \ldots$ never reaches $+\infty$. To compute the limit in finitely many iterations, we need a stopping criterion that allows us to distinguish between divergence to $+\infty$ and convergence to a finite value. The following lemma provides such a stopping criterion.

Lemma 1. *For all states q of a pure energy interface, either $val(q) = +\infty$ or $val(q) \leq -\sum_{p \in Q} \min\{0, \lambda(p)\}$.*

This lemma is proved in a fashion similar to a theorem in [4], by relating the value of the energy interface to the value along a loop in the game. Let $v^+ = -\sum_{p \in Q} \min\{0, \lambda(p)\}$. If $f_k(q) > v^+$ for some $k \geq 0$, we know that $f_*(q) = +\infty$. This suggests the definition of a modified operator $ETPre$: $(Q \to \mathbb{Z}_\infty) \to (Q \to \mathbb{Z}_\infty)$, defined for all $f\colon Q \to \mathbb{Z}_\infty$ and $q \in Q$ by

$$ETPre(f)(q) = \begin{cases} EPre(f)(q) & \text{if } EPre(f)(q) \leq v^+, \\ +\infty & \text{otherwise.} \end{cases}$$

We have $f_* = \lim_{k \to \infty} f_k$, where $f_0(q) = -\infty$ for all $q \in Q$, and $f_{k+1} = ETPre(f_k)$ for $k \geq 0$. Moreover, there is $k \in \mathbb{N}$ such that $f_k = f_{k+1}$, indicating that the limit can be computed in finitely many iterations. Once f_* has been computed, we have $val(q) = f_*(q)$ and $Q^{rc}_\Delta = \{q \in Q \mid f_*(q) \leq \Delta\}$.

Let ℓ be the maximal energy consumption of R. We have $v^+ \leq |Q| \cdot \ell$. Consider now the sequence $f_0, f_1, \ldots$ converging to f_*: for all $k \geq 0$, either $f_{k+1} = f_k$ (in which case $f_* = f_k$ and the computation terminates), or there must be $q \in Q$ such that $f_k(q) < f_{k+1}(q)$. Thus, the limit is reached in at most $v^+ \cdot |Q| \leq |Q|^2 \cdot \ell$ iterations. Each iteration involves the evaluation of the $ETPre$ operator, which requires time linear in $|R|$.

Theorem 3 *Given a pure energy interface R of size n with maximal energy consumption ℓ, and $\Delta \in \mathbb{Z}_\infty$, we can check the Δ-compliance of R and compute its value in time $O(n^3 \cdot \ell)$.*

Reward energy interfaces. Given a Λ-reward energy interface R and $\Delta \in \mathbb{Z}$, to compute Q^{rc}_Δ and *val*, we use a dynamic programming approach reminiscent of that used in the solution of shortest-path games [6]. We iterate over a set of *reward-energy allocations* $\mathcal{E}$: $Q \to (\{0, \dots, \Lambda\} \to \mathbb{Z}_\infty)$. Intuitively, for $f \in \mathcal{E}$, $q \in Q$, and $r \in \{0, \dots, \Lambda\}$, the value $f(q)(r)$ indicates the amount of energy necessary to achieve reward r before running out of energy. For $e_1, e_2 \in \mathbb{Z}$, let $\mathrm{Mxe}(e_1, e_2) = \max\{e_1, e_2\}$ if $\max\{e_1, e_2\} \leq v^+$, and $\mathrm{Mxe}(e_1, e_2) = +\infty$ otherwise. For $r \in \mathbb{N}$, let $\mathrm{Mxr}(r) = \max\{0, r\}$. For $q \in Q$, use $\lambda(q) = \langle d(q), n(q) \rangle$. We define an operator *ERPre*: $\mathcal{E} \to \mathcal{E}$ on energy-reward allocations by letting $g = \mathit{ERPre}(f)$, where $g \in \mathcal{E}$ is such that for all $q \in Q$ we have $g(q)(0) = 0$, and for all $r \in \{0, \dots, \Lambda - 1\}$,

$$g(q)(r) = \mathrm{Mxe}(-d(q), -d(q) + \min_{v^i \in \gamma^i(q)} \max_{v^o \in \gamma^o(q)} f(\delta(q, v^i, v^o))(\mathrm{Mxr}(r - n(q)))).$$

Intuitively, given an energy-reward allocation f, a state q, and a reward r, $\mathit{ERPre}(f)(q)(r)$ represents the minimum energy to achieve reward r from state q given that the next-state energy-reward allocation is f. Let $f_0 \in \mathcal{E}$ be defined by $f_0(q)(r) = +\infty$, for $q \in Q$ and $r \in \{0, \dots, \Lambda\}$, and for $k \geq 0$, let $f_{k+1} = \mathit{ERPre}(f_k)$. The limit $f_* = \lim_{k \to \infty} f_k$ (defined pointwise) exists; in fact, for all $q \in Q$ and $r \in \{0, \dots, \Lambda\}$, we have $f_{k+1}(q)(r) \leq f_k(q)(r)$. For all $q \in Q$, we then have $\mathit{val}(q) = f_*(q)(\Lambda)$, and $q \in Q^{rc}_\Delta$ if $f_*(q)(\Lambda) \leq \Delta$.

The complexity of this algorithm can be characterized as follows. For all $q \in Q$, $r \in \{0, \dots, \Lambda\}$, and $f \in \mathcal{E}$, the energy $f(q)(r)$ can assume at most $1 + v^+ \leq 1 + \ell \cdot |Q|$ values, where ℓ is the maximal energy consumption in R. Since each of these values is monotonically decreasing, the limit f_* is computed in at most $O(|Q|^2 \cdot \ell \cdot \Lambda)$ iterations. Each iteration has cost $|R| \cdot \Lambda$.

Theorem 4 *Given a Λ-reward energy interface R of size n with maximal energy consumption ℓ, and $\Delta \in \mathbb{Z}_\infty$, we can check the Δ-compliance of R and compute its value in time $O(n^3 \cdot \ell \cdot \Lambda^2)$.*

5 Examples

We sketch two small case studies that illustrate how resource interfaces can be used to analyze resource-constrained systems.

5.1 Distribution of resources in a Lego robot system

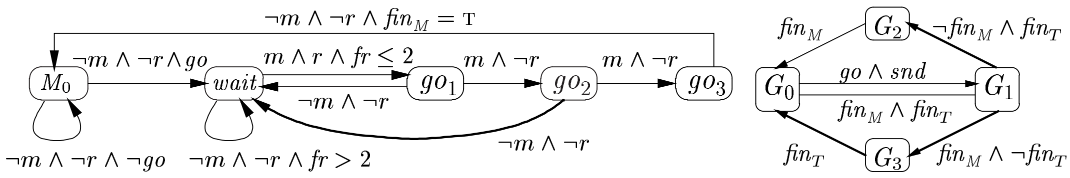
We use resource interfaces to analyze the schedulability of a lego robot executing various parallel threads. In this set up, the Input player is a "resource broker" who distributes the resources over the threads. The system is compatible if Input can ensure that all resource constraints are met.

The Lego robot. We have modeled a lego robot that must execute various commands it gets from the base station through infrared (ir) communication, while it also has to recover from bumping into obstacles. Its software is organized in 5 parallel threads, interacting via a central repository. The thread Scan Sensors (S) scans the values of the sensors and puts these into the repository, Motion (M) executes the tasks from the base station, Bump Recovery (B) is executed when the robot bumps into an object, Telemetry (T) is responsible for communication with the base station and the Goal Manager (G) manages the various goals. There are 3 mutex resources: the motor (m), the ir sensor (s) and the central repository (c). Furthermore, energy is consumed by the motor and ir sensor. This system is open, as more threads can be added later.

Checking schedulability using pure threshold interfaces. First, we discard the energy consumption and consider the question whether all the mutex requirements can be met. To this end, we model each thread i as a threshold interface (M^i, λ^i) with threshold value $\Delta = 1$. The resource labeling $\lambda^i = (\lambda^i_m, \lambda^i_c, \lambda^i_s)$ is such that $\lambda^i_R(q)$ indicates whether, in state q, thread i owns resource R. The underlying A/G interface M_i has, for each resource $R \in \{m, c, s\}$, a boolean input variable gr^i_R (abbreviated R in the figures) indicating whether Input grants R to i. We also model a resource interface (M^E, λ^E) for the environment, expressing that bumps do not occur too often. This interface does not use any resources, i.e. $\lambda^E_r(q) = 0$ for all states q and all resources R. Then these resource interfaces are 1-compatible iff all mutex requirements are met.[1]

Due to space limitations, Figure 2 only presents the A/G interfaces for Motion and Goal Manager; the others be modeled in a similar fashion. Also, rather than with $\rho(p, q)$, we label the edges $\rho(p, q) \wedge \phi^i(p) \wedge \phi^o(q)$. The tread Motion in Figure 2(a) has one boolean output variable fin_M, indicating whether it has finished a command from the base station. Besides the input variables gr^M_m, gr^M_c and gr^M_s discussed above, Motion has an input variable fr which is controlled by Scan Sensors and counts the steps since the last scanning of the sensors. In the initial location M_0, Motion waits for a command go from the Goal Manager. Its input assumption is $\neg m \wedge \neg c$, indicating that Motion needs neither the motor nor the repository. When receiving a command, Motion moves to the location *wait*, where it tries to get hold of the motor and the repository. Since Motion needs fresh sensor values, it requires $fr \leq 2$ to move on the next location; otherwise it does not need either resources. In the locations go_1, go_2 and go_3, Motion executes the command. It needs the motor and repository in go_1, and the motor only in go_2 and go_3. If, in locations go_1 or go_2, the motor is retrieved from Motion (input $\neg m \wedge \neg c$, typically if Bump Recovery needs the motor), it starts from the location *wait*. When leaving location go_3, Motion sets $fin_M = \mathrm{T}$, indicating the completion of a command. We let $fin_M = \mathrm{F}$ on all other transitions. The labeling λ^M_r for $r \in R$ is given by: $\lambda^M_m(go_1) = \lambda^M_m(go_2) = \lambda^M_m(go_3) = \lambda^M_c(go_1)$. $\lambda^M_r(q) = 0$ in all other cases. (Note that $\lambda^i_R(q)$ is derivable from gr^i_R by con-

[1] Note that the resource compliance of (Büchi) threshold games with multiple resource labelings can be checked along the same lines as the resource compliance of threshold games with single resource labelings.



(a) Motion. (b) Goal manager.

Fig. 2. A/G interfaces modeling a Lego robot.

sidering edges leading to q.) The interface for *Goal Manager* (Figure 2(b)) has output variables *go* and *snd* through which it starts up the threads Motion and Telemetry in location G_0 and then waits for them to be finished. It does not use any resources.

Checking schedulability using Büchi threshold interfaces. The threshold interfaces before express safety, but not liveness: the resource broker is not forced to grant ever the motor to Motion or Telemetry, in which case they stay forever in the locations *wait* or $wait_1$ respectively. To enforce the progress of the threads, we add a Büchi condition expressing that the locations G_0 should be visited infinitely often. Thus, each state is a state label and we define the location labeling of thread G by $\kappa^G(q) = (\lambda^G(q), \{G_0\}, \{q\})$ and for $i \in \{S, M, B, T, E\}$ by $\kappa^i(q) = (\lambda^i(q), \emptyset, \emptyset)$, where λ^i is as before. Then all mutex requirements can be met, with the state G_0 being visited infinitely often, iff the resource interfaces are 1-compatible.

Analyzing energy consumption using reward energy interfaces. Energy is consumed by the motor and the ir sensor. We define the energy expense for thread i at state q as $\lambda^i_e(q) = 5\lambda^i_m(q) + 2\lambda^i_s(q)$, expressing that the motor uses 5 energy units and the ir sensor 2. Currently, the system will always run out of energy because it is never recharged, but it is easy to add an interface for that. To prevent the system from saving energy by doing nothing at all, we specify a reward. A naive attempt would be to assign the reward to each location in each thread and sum the rewards upon composition. However, suppose that the reward acquired per energy unit is higher when executing Motion than when executing Telemetry. Then, the highest reward is obtained by always executing Motion and never doing Telemetry. This phenomenon is not a deficit of the theory, it is inherent when managing various goals. Since the latter is exactly the task of the goal manager, we reward the completion of a round of the goal manager. That is, we put $\lambda^G_r(G_0) = 1$ and $\lambda^i_r(q) = 0$ in all other cases. Then all mutex requirements can be met, while the system never runs out of power, iff the threshold interfaces interfaces (as defined before) are 1-compatible and their composition is 0-compliant as an energy reward interface.

5.2 Resource accounting for the PicoRadio network layer

The PicoRadio [1] project aims to create large-scale, self-organizing sensor networks using very low-power, battery-operated *piconodes* that communicate over wireless links. Intended application scenarios make it infeasible to connect nodes individually to power lines or to periodically replace their batteries. Energy-aware routing [10] strategies have been found necessary to optimize use of scarce energy resources.We show how our methodology can be profitably applied to evaluate networks and synthesize optimal routing algorithms.

A PicoRadio network. A *piconet* consists of a set of piconodes that can create, store, or forward packets using multi-hop routing. The piconet *topology* describes the position, maximal packet-creation rate, and packet-buffer capacity at each piconode, and capacity of each link. Each packet has a *destination*, which is a node in the network. A *configuration* of the network represents the number of packets of each destination currently stored in the buffer of each piconode. A configuration that assigns more packets to a node than its buffer size is not *legal*.The network moves from one configuration to another in a *round*. We assume that a piconode always uses its peak transmission capacity on an outgoing link as long as it has enough packets to forward on that link. Wireless transmission costs energy. Each piconode starts with an initial amount of energy, and can possibly gain energy by scavenging.

We are given a piconet with known topology and initial energy levels at each piconode. We wish to find a routing algorithm that makes the network satisfy a certain safety property, e.g., that buffer overflows do not occur (or that whenever a node has a packet to forward, it has enough energy to do so). A piconet together with such a property represents a concurrent finite-state safety game between player Packet Generator, and player Router. Each legal configuration of the network is represented by a game state; the state ERROR represents all illegal configurations. The state transitions reflect the configuration changes the network undergoes from round to round as the players concurrently make packet creation and routing choices under the constraints imposed by the network topology. The state ERROR has no outgoing transitions. The initial state corresponds to the network configuration that assigns 0 packets to each node. The winning condition is derived from the property the network must satisfy. If player Router has a winning strategy σ, a routing algorithm that makes the network satisfy the given property under the constraints imposed by the topology exists and can be found from σ; else no such routing algorithm exists. We present several examples.

Finding a routing strategy to prevent buffer overflows. Let $\lambda(q_c) = 0$ for each state q_c that represents a legal configuration c, and let $\lambda(\text{ERROR}) = 1$. If the pure threshold interface thus constructed is Δ-compliant for $\Delta = 0$, then a routing algorithm that prevents buffer overflow exists and can be synthesized from a winning strategy for player Router.

Finding the optimal buffer size for a given topology. We wish to find out the smallest buffer capacity (less than a given bound) each piconode must have so that there exists a routing algorithm that prevents buffer overflows. Let $\lambda(q_c) = \max_i \sum_j c_{ij}$ for all nodes i and packet destinations j, where c_{ij} is the number of packets with destination j in node i in configuration c. The value of the pure threshold interface thus constructed gives the required smallest buffer size.

Checking if the network runs forever using energy interfaces. We wish to find if there exists a routing algorithm A_f that enables a piconet to run forever, assuming each piconode starts with energy e. Let e_{sc} be the energy scavenged by a piconode in each round. Let $\lambda(q_c) = e_{sc} - \max_i \sum_j (p \cdot \min(c_{ij}, l_i(r_i(j))))$, where q_c, i, j, and c_{ij} are as above, p is the energy spent to transmit a packet, $l_i(x)$ is the capacity of the link from node i to node x, and r_i is the routing table at node i. If the pure energy interface thus constructed is Δ-compliant for $\Delta = e$, then A_f exists and is given by the Router strategy.

Finding the minimum energy required to achieve a given lifetime. We wish to find the minimum initial energy e such that there exists a routing algorithm A_r that makes each piconode run for at least r rounds. Let $\lambda(q_c) = (e_c, 1)$, where e_c is the energy label of configuration q_c defined in the pure energy interface above, and 1 is a reward. For $\Lambda = r$, the value of the Λ-reward energy interface thus constructed gives e, and the Router strategy gives A_r.

References

1. J.L. da Silva Jr., J. Shamberger, M.J. Ammer, C. Guo, S. Li, R. Shah, T. Tuan, M. Sheets, J.M. Rabaey, B. Nikolic, A. Sangiovanni-Vincentelli, and P. Wright. Design methodology for pico-radio networks. In *Proc. Design Automation and Test in Europe*, pp. 314–323. IEEE, 2001.
2. L. de Alfaro and T.A. Henzinger. Interface automata. In *Proc. Foundations of Software Engineering*, pp. 109–120. ACM, 2001.
3. L. de Alfaro and T.A. Henzinger. Interface theories for component-based design. In *Embedded Software*, LNCS 2211, pp. 148–165. Springer, 2001.
4. A. Ehrenfeucht and J. Mychielski. Positional strategies for mean-payoff games. *Int. J. Game Theory*, 8:109–113, 1979.
5. E.A. Emerson and C.S. Jutla. Tree automata, μ-calculus, and determinacy. In *Proc. Foundations of Computer Science*, pp. 368–377. IEEE, 1991.
6. J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.
7. M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous π-calculus. In *Proc. Automata, Languages, and Programming*, LNCS 1853, pp. 415–427. Springer, 2000.
8. I. Lee, A. Philippou, and O. Sokolsky. Process-algebraic modeling and analysis of power-aware real-time systems. *Computing and Control Engineering J.*, 13:180–188, 2002.
9. M. Núñez and I. Rodríguez. Pamr: a process algebra for the management of resources in concurrent systems. In *Proc. Formal Techniques for Networked and Distributed Systems*, pp. 169–184. Kluwer, 2001.
10. R. Shah and J.M. Rabaey. Energy-aware routing for low-energy ad-hoc sensor networks. In *Proc. Wireless Communications and Networking Conference*, pp. 812–817. IEEE, 2002.
11. D. Walker, K. Crary, and G. Morrisett. Typed memory management via static capabilities. *ACM Trans. Programming Languages and Systems*, 22:701–771, 2000.
12. Chic: Checker for Interface Compatibility. www.eecs.berkeley.edu/~tah/Chic.