ASM and Controller Synthesis

Richard Banach[1]*, Huibiao Zhu[2]**, Wen Su[2], and Xiaofeng Wu[2]

[1]School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
banach@cs.man.ac.uk

[2]Software Engineering Institute, East China Normal University,
3663 Zhongshan Road North, Shanghai 200062, P.R. China.
{hbzhu,wensu,xfwu}@sei.ecnu.edu.cn

Abstract. While many systems are naturally viewed as the interaction between a controller subsystem and a controlled, or plant subsystem, they are often most easily understood and designed monolithically. A practical implementation needs to separate controller from plant. We study the problem of when a monolithic ASM system can be split into controller and plant subsystems along syntactic lines derived from variables' natural affiliations. We give restrictions that enable the split to be carried out cleanly, and we give conditions that ensure that the resulting pair of controller and plant subsystems have the same behaviours as the original design. We illustrate the theory with a case study concerning eating with chopsticks. This leads to an extension of controller synthesis for continuous ASM systems, which are briefly covered. The case study is then extended into the continuous sphere.

1 Introduction

Today, when one considers the ubiquity of embedded controllers, which take on the digital role in the interaction of a digital and an external system, it becomes clear that many systems are naturally viewed as the interaction between a controller subsystem and a controlled, or plant subsystem. Such systems are often most easily and conveniently understood and designed monolithically — this allows the bulk of the design activity to focus on the overall system goals rather than lower level detail. However, a practical implementation needs to separate the controller from the plant, since it is the controller which behaves according to a human-created digital design, and the plant behaves according to patterns determined by the laws of nature. In this paper we study the problem of when a monolithic ASM system design, embodying this dual controller/plant nature, can be split into separate controller and plant subsystems along generic syntactic lines

* The majority of the work reported in this paper was done while the first author was a visiting researcher at the Software Engineering Institute at East China Normal University. The support of ECNU is gratefully acknowledged.

** Huibiao Zhu is supported by National Basic Research Program of China (No. 2011CB302904), National High Technology Research and Development Program of China (No. 2011AA010101 and No. 2012AA011205), National Natural Science Foundation of China (No. 61061130541 and No. 61021004).

derived from the most natural associations of the system variables to one or other subsystem. This requires that the monolithic design satisfies some simple criteria *ab initio*.

The rest of the paper is as follows. Section 2 describes the controller synthesis problem in abstract terms, focusing on the specific way that controller and plant are to be separated. A sufficient condition for the desired controller/plant separation is formulated and proved. The undecidability of controller synthesis is also briefly discussed by reduction to the Halting Problem in Section 2.1. In Section 3 we consider a computable subset of the controller synthesis problem and argue that it is adequate for practical purposes. Section 4 discusses an example based on the idea of picking up food with chopsticks, viewed as a control problem. Section 5 extrapolates the preceding ideas to the case of continuous ASM, in which smoothly changing (as well as discretely changing) behaviours are admitted. Section 6 extends the discussion of the chopsticks case study by taking on board the continuous notions. Section 7 concludes.

2 The Controller Synthesis Problem

We consider a generic ASM system consisting of basic ASM rules using straightforward single variable locations and a simple element of nondeterminism. Following [2], for our purposes, such a rule can be written as:

```
OP($pars$) =                                                      (1)
if $guard(xs, pars)$ then choose $xs'$ with $rel(xs', xs, params)$
   do $xs$ := $xs'$
```

In (1), $pars$ are the input parameters (as needed) and xs are the variables modified by the rule. The rule's guard is $guard$, and rel represents the relationship that is to hold between the parameters, the before-values of the variables xs, and their after-values referred to as xs', when the rule fires. As usual, in a single step of a run of the system, all rules which are enabled (i.e. their guards are true) fire simultaneously, provided that the totality of updates defined thereby is consistent, else the run aborts.

In this paper we are interested in control applications, and we envisage the design done in a monolithic way at the outset, addressing system-wide design goals before plunging into the details of subsystem design. Thus the design may start by being expressed using system-wide variables. However, by a process of gradual refinement, the collection of variables will eventually end up such that each variable can be identified as belonging to either the controller-subsystem-to-be, or the plant-subsystem-to-be. Nevertheless, a legacy of the top-down design process is that many, or even all of the rules will still involve variables of both kinds.

The controller synthesis problem is the problem of taking such a collection of rules (call it Sys), and separating it into one set of rules for the controller (call it Con) and another set for the plant (call it Pla), each reading only the variables accessible to it, and each modifying only its own variables, such that the combination of the rules in Con and Pla generates the same behaviour (i.e. the same set of runs) as the original ruleset Sys.[1]

[1] In [2], the importance of distinguishing *controlled* functions from *monitored* ones is stressed, in a sense solving the controller synthesis problem right at the outset since the distinction

We perform the separation in a systematic manner. We assume that the variables Var of Sys can be partitioned into $xs_C \subseteq Var_C$, the variables for which the controller has write access, and $xs_P \subseteq Var_P$, the variables for which the plant has write access, with $Var_C \cap Var_P = \emptyset$. We assume that for each rule $\textsc{Op}(params) \in Sys$, the guard can be written in the form $guard(xs, pars) \equiv guard_C(xs_C, xs_P^c, pars_C) \wedge guard_P(xs_P, xs_C^p, pars_P)$, where xs_P^c are the plant variables to which the controller has read access, and xs_C^p are the controller variables to which the plant has read access. We also assume that for each rule, $rel(xs', xs, pars)$ can be written in the form $rel(xs', xs, pars) \equiv rel_C(xs_C, xs_P^c, pars_C) \wedge rel_P(xs_P, xs_C^p, pars_P)$. We say that a system is **admissible** iff the above hold.

Under the above assumptions, the desired construction is relatively clear. For each rule like (1) in Sys, we generate two fresh rules:

$$\begin{array}{l} \textsc{Op}_C(pars) \ = \\ \textbf{if} \ \ guard_C(xs_C, xs_P^c, pars_C) \ \ \textbf{then choose} \ \ xs_C' \\ \quad \textbf{with} \ \ rel_C(xs_C', xs_C, xs_P^c, pars_C) \ \ \textbf{do} \ \ xs_C \ := \ xs_C' \end{array} \tag{2}$$

$$\begin{array}{l} \textsc{Op}_P(pars) \ = \\ \textbf{if} \ \ guard_P(xs_P, xs_C^p, pars_P) \ \ \textbf{then choose} \ \ xs_P' \\ \quad \textbf{with} \ \ rel_P(xs_P', xs_P, xs_C^p, pars_P) \ \ \textbf{do} \ \ xs_P \ := \ xs_P' \end{array} \tag{3}$$

Of these, (2) goes into Con and (3) goes into Pla.

With Con and Pla thus constructed, and with initial states correspondingly constructed by restricting the initial states of Sys to the variables in Var_C and Var_P respectively (by existentially quantifying out $Var - Var_P$ in Con, and $Var - Var_C$ in Pla, provided there are no non-trivial joint initial properties), it is evident that whenever a rule $\textsc{Op}$ of Sys is enabled, the corresponding rules $\textsc{Op}_C$ and $\textsc{Op}_P$ of Sys_C and Sys_P will also be enabled (since their guards are just weakenings of $\textsc{Op}$'s guard). If we thus consider the system Sys_{C+P}, which consists of the variables and initial states of Sys,[2] and whose rules are the union of the $\textsc{Op}_C$ and $\textsc{Op}_P$ rules, then whenever a rule $\textsc{Op}$ of Sys is enabled, it follows that in Sys_{C+P}, $\textsc{Op}_C$ and $\textsc{Op}_P$ will be enabled and both will be scheduled simultaneously by the ASM scheduling policy, replicating the update performed by $\textsc{Op}$ in Sys. So the runs of Sys are a subset of the runs of Sys_{C+P}.

On the other hand, they may be a *proper* subset since the guards of the individual $\textsc{Op}_C$ and $\textsc{Op}_P$ rules are weaker than the guard of $\textsc{Op}$, and so may enable one or other of $\textsc{Op}_C$ and $\textsc{Op}_P$ without the other being enabled. This is highly undesirable from a requirements point of view since the overall objective was to achieve the behaviour of Sys, and not to introduce some spurious additional behaviours.

Definition 1. *A system Sys, with $Var = Var_C \uplus Var_P$ which is admissible, has a resolvable controller synthesis problem iff, after the construction above, the runs of Sys_{C+P} are exactly the runs of Sys.*

already separates the controller from the plant. Our perspective is slightly different however, since it permits this aspect to be ignored for a portion of the development, and asks under what conditions the separation can be done later in a systematic way.

[2] The initial states are recovered by conjoining initial states of Sys_C and Sys_P.

Theorem 1. *Suppose a system Sys is admissible. Then Sys has a resolvable controller synthesis problem if:*

$$\begin{aligned}&\textit{For all rules } \textsc{Op}\textit{, their derived rules } \textsc{Op}_C \textit{ and } \textsc{Op}_P\textit{, and reachable states } xs\ \bullet\\ &\quad [\ Domain(xs) \wedge guard_C(xs_C, xs_P^c, pars_C) \Rightarrow guard(xs, pars)\ ] \wedge\\ &\quad [\ Domain(xs) \wedge guard_P(xs_P, xs_C^p, pars_P) \Rightarrow guard(xs, pars)\ ]\end{aligned} \tag{4}$$

where $Domain(xs)$ is the domain theory for the development of Sys.

Proof: To get the result, it is sufficient to show that when (4) holds, every run of Sys_{C+P} is a run of Sys, since we argued above that all Sys runs are Sys_{C+P} runs anyway. We proceed by induction on the length of the run. The base case is trivial since the initial states of Sys and of Sys_{C+P} are identical. Suppose then that we have the result for all Sys_{C+P} runs of length n or less. Choose a run rr of length n which is extendable. This means that there is some rule, $\textsc{Op}_C$ say, that is enabled in the final state xs reached by rr (the argument is symmetrical if it is $\textsc{Op}_P$ that is enabled). Since $\textsc{Op}_C$ is enabled in xs, $guard_C$ holds, whence $guard$ holds by (4). Since $guard_P$ weakens $guard$, $guard_P$ holds, whence $\textsc{Op}_P$ is enabled. Since both $\textsc{Op}_C$ and $\textsc{Op}_P$ are enabled, the update of Sys is emulated by Sys_{C+P} in the next step of the run. The same argument applies for all rules of Sys_{C+P} enabled in xs, so that the next Sys_{C+P} step from rr exactly mirrors a corresponding step of Sys. Doing the same for all possible ways of extending all extendable runs of length n completes the inductive step. □

2.1 Undecidability of Controller Synthesis

The presence of reachability in (4) makes the undecidability of the controller synthesis problem relatively unsurprising, so we just briefly sketch a reduction of the Halting Problem. Let TM be an arbitrary Turing Machine. Let TM_C^0 be an emulation of TM by an ASM constructed in a rather obvious way: i.e. there is an alphabet of states, another of tape symbols, a variable for the current state, a data structure for the tape, and a separate rule for each transition in the transition relation of TM. Let TM_P^0 be another such ASM emulation, isomorphic to TM_C^0, but with all alphabets and variables completely disjoint from those of TM_C^0. Consider the ASM TM_{C+P}^0 constructed as in the previous section. It has twice as many rules as TM has transitions, but they are enabled pairwise at exactly the same moments, so TM_{C+P}^0 just emulates two disjoint copies of TM running in lockstep. Consider the ASM $TM_{C\wedge P}^0$ constructed by fusing each corresponding pair of rules of TM_{C+P}^0 into a single rule by conjoining the guards, and combining the updates. It has exactly as many rules as TM has transitions. $TM_{C\wedge P}^0$ and TM_{C+P}^0 are bisimilar to each other and to TM. Now we modify TM_C^0, and modify TM_P^0, as follows.

Since TM is arbitrary, it may contain halting before-configs —i.e. pairs (t, s) where t is a tape symbol and s is a state— from which no transition issues. If TM has a halting before-config (t, s), we do the following. Let (t_C, s_C) be the counterpart of (t, s) in TM_C^0. To TM_C^0 we add a rule that implements a self-loop guarded on (t_C, s_C) (without moving the tape head), getting TM_C. Let (t_P, s_P) be the counterpart of (t, s)

in TM_P^0. To TM_P^0 we add a rule that implements a self-loop guarded on s_P alone (i.e. ignoring the tape symbol, and without moving the tape head), getting TM_P.

Now consider the two ASM systems $TM_{C \wedge P}$ and TM_{C+P}. In $TM_{C \wedge P}$ (which plays the role of Sys above), the stronger guard of the TM_C rule in effect subsumes the weaker one of the TM_P rule, and the fused rule is only enabled exactly when the TM_C rule is enabled. However in TM_{C+P} (which plays the role of Sys_{C+P} above), this is not the case. There, the TM_P rule exists independently, and if the computation of TM reaches a machine configuration in which the tape symbol and state are (t, s), then the TM_P rule is also enabled when the tape symbol and state are $(\tilde{t}, s)$, for some $\tilde{t} \neq t$, giving rise to behaviours not reflected in $TM_{C \wedge P}$.

3 Computable Controller Synthesis

Restricting to a safe approximation to reachability, we get a computable version of (4), which we argue will be adequate for all practical purposes.

Theorem 2. *Suppose a system Sys is admissible and XS is a set of states that includes all reachable states. Then Sys has a resolvable controller synthesis problem if:*

For all rules $\textsc{Op}$, *their derived rules* $\textsc{Op}_C$ *and* $\textsc{Op}_P$, *and all* $xs \in XS$ $\bullet$

$$\begin{aligned}&[\, Domain(xs) \wedge guard_C(xs_C, xs_P^c, pars_C) \vdash guard(xs, pars) \,] \wedge \\ &[\, Domain(xs) \wedge guard_P(xs_P, xs_C^p, pars_P) \vdash guard(xs, pars) \,] \end{aligned} \qquad (5)$$

where $Domain(xs)$ is the domain theory for Sys and $\vdash$ is provability in a suitable system.

4 An Example: Eating with Chopsticks

We now look at a simple example of the preceding theory: eating food with chopsticks. Fig. 1 shows the forces involved in grasping a morsel of food with chopsticks.

4.1 Food and Chopsticks

In a statically stable situation, the chopsticks extert forces on the food, and the food exerts equal and opposite forces on the chopsticks. The forces exerted by the food are $\boldsymbol{f}_{\text{FU}}$ on the upper chopstick and $\boldsymbol{f}_{\text{FL}}$ on the lower chopstick. For simplicity we assume that these forces sum to zero (else the food would accelerate) and colinear.[3] Reacting to $\boldsymbol{f}_{\text{FU}}$ and $\boldsymbol{f}_{\text{FL}}$, the chopsticks exert their forces $\boldsymbol{f}_{\text{HCU}}$ and $\boldsymbol{f}_{\text{HCL}}$, equal and opposite to $\boldsymbol{f}_{\text{FU}}$ and $\boldsymbol{f}_{\text{FL}}$. So we have:

$$\boldsymbol{f}_{\text{FU}} + \boldsymbol{f}_{\text{FL}} = \mathbf{0} \qquad (6)$$

[3] In reality, slight deviations from colinearity are compensated for by forces of friction and deformation arising from the food, aided where appropriate, by surface tension forces coming from any sauce that the food might be prepared in.

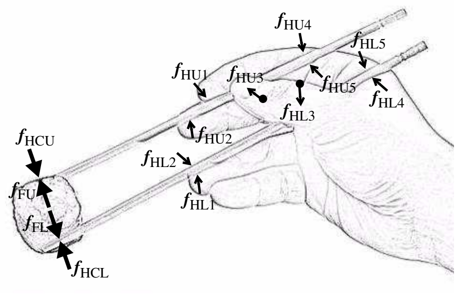

Fig. 1. Forces involved in grasping a piece of food with chopsticks.

$$\boldsymbol{f}_{\text{HCU}} + \boldsymbol{f}_{\text{HCL}} = \mathbf{0} \tag{7}$$
$$\boldsymbol{f}_{\text{FU}} + \boldsymbol{f}_{\text{HCU}} = \mathbf{0} \tag{8}$$
$$\boldsymbol{f}_{\text{FL}} + \boldsymbol{f}_{\text{HCL}} = \mathbf{0} \tag{9}$$
$$|\boldsymbol{f}_{\text{FU}}| = |\boldsymbol{f}_{\text{FL}}| = |\boldsymbol{f}_{\text{HCU}}| = |\boldsymbol{f}_{\text{HCL}}| \geq D \tag{10}$$

The last of these (10), expresses a constraint that the forces mentioned have to be large enough (D) that they generate additional frictional forces (which can be taken to be proportional to them), sufficient to counteract gravity (which we have not taken into account), thereby to stop the food from dislodging from the chopsticks when lifted.

We can write this as an ASM model, with a rule:

```
GRASPFOOD =                                                                    (11)
choose $\boldsymbol{f}'_{\text{FU}}, \boldsymbol{f}'_{\text{FL}}, \boldsymbol{f}'_{\text{HCU}}, \boldsymbol{f}'_{\text{HCL}}$
    with $\boldsymbol{f}'_{\text{FU}} + \boldsymbol{f}'_{\text{FL}} = \boldsymbol{f}'_{\text{HCU}} + \boldsymbol{f}'_{\text{HCL}} = \boldsymbol{f}'_{\text{FU}} + \boldsymbol{f}'_{\text{HCU}} = \boldsymbol{f}'_{\text{FL}} + \boldsymbol{f}'_{\text{HCL}} = \mathbf{0} \wedge$
         $|\boldsymbol{f}'_{\text{FU}}| = |\boldsymbol{f}'_{\text{FL}}| = |\boldsymbol{f}'_{\text{HCU}}| = |\boldsymbol{f}'_{\text{HCL}}| \geq D$
    do $\boldsymbol{f}_{\text{FU}} := \boldsymbol{f}'_{\text{FU}}, \boldsymbol{f}_{\text{FL}} := \boldsymbol{f}'_{\text{FL}}, \boldsymbol{f}_{\text{HCU}} := \boldsymbol{f}'_{\text{HCU}}, \boldsymbol{f}_{\text{HCL}} := \boldsymbol{f}'_{\text{HCL}},$
       $grasped$ := TRUE
```

There will be another rule DISLODGEFOOD, differing from (11) in the replacement of '$\geq D$' by '$< D$' and of TRUE by FALSE, regarding dislodgement of food as being due to inadequate force, and disregarding any other maladroitness on the part of the user. Given the similarity of the two rules, we will not mention DISLODGEFOOD further, unless it is unavoidable.

We can regard GRASPFOOD (and DISLODGEFOOD) as a simple design for a control system — the chopsticks are intended to control the food by grasping it. Thus we can pursue our earlier strategy by separating the system into plant (food) and controller (chopsticks) subsystems. The GRASPFOOD rule separates into GRASPFOOD$_C$ and GRASPFOOD$_P$:

```
GRASPFOOD$_C$ =                                                                (12)
choose $\boldsymbol{f}'_{\text{HCU}}, \boldsymbol{f}'_{\text{HCL}}$
    with $\boldsymbol{f}'_{\text{HCU}} + \boldsymbol{f}'_{\text{HCL}} = \mathbf{0} \wedge |\boldsymbol{f}'_{\text{HCU}}| = |\boldsymbol{f}'_{\text{HCL}}| \geq D$
    do $\boldsymbol{f}_{\text{HCU}} := \boldsymbol{f}'_{\text{HCU}}, \boldsymbol{f}_{\text{HCL}} := \boldsymbol{f}'_{\text{HCL}},$
       $grasped$ := TRUE
```

```
$\text{GraspFood}_P$ =                                              (13)
choose $\boldsymbol{f}'_{\text{FU}}, \boldsymbol{f}'_{\text{FL}}$
    with $\boldsymbol{f}'_{\text{FU}} + \boldsymbol{f}'_{\text{FL}} = \mathbf{0}$
    do $\boldsymbol{f}_{\text{FU}} := \boldsymbol{f}'_{\text{FU}}, \boldsymbol{f}_{\text{FL}} := \boldsymbol{f}'_{\text{FL}}$
```

In (12) and (13) we see that GraspFood_C only 'owns' $\boldsymbol{f}_{\text{HCU}}$ and $\boldsymbol{f}_{\text{HCL}}$, so only assigns to those variables, and GraspFood_P only 'owns' $\boldsymbol{f}_{\text{FU}}$ and $\boldsymbol{f}_{\text{FL}}$, so only assigns to them. We also observe that some pieces of GraspFood are not present in either GraspFood_C or GraspFood_P, namely the terms that relate the food forces to the chopstick forces. This is explained by the observation that the relevant equations are part of the domain theory of statics: action and reaction are *always* equal statically, by Newton's Law. Additionally, that successful grasping needs adequate force is also part of the domain, so we can write:

$$\begin{aligned} Domain_{FHC} \equiv\ & \boldsymbol{f}_{\text{FU}} + \boldsymbol{f}_{\text{HCU}} = \mathbf{0} \wedge \boldsymbol{f}_{\text{FL}} + \boldsymbol{f}_{\text{HCL}} = \mathbf{0} \wedge \\ & (grasped = \text{TRUE} \Leftrightarrow |\boldsymbol{f}_{\text{HCL}}| \geq D) \end{aligned} \tag{14}$$

Now, in the context of (14), it is easy to see that:

$$Domain_{FHC} \wedge guard_{\text{GraspFood}_C} \vdash guard_{\text{GraspFood}} \tag{15}$$

$$Domain_{FHC} \wedge guard_{\text{GraspFood}_P} \vdash guard_{\text{GraspFood}} \tag{16}$$

4.2 Chopsticks and Hand

The preceding was rather elementary. In particular, it presumed that chopsticks somehow grasp food by themselves, which is silly. In reality, chopsticks are held in the right hand, which causes them to exert the forces spoken of previously. We now enrich our model by considering the hand-chopstick system as a further control system, and decomposing it further into a plant subsystem (the chopsticks themselves) and a controller subsystem (the hand).

We refer to Fig. 1 again. For a solid object to remain stable in 3D space, it needs to have four non-colinear forces summing to zero acting on it. If gravity is acting (as it normally is) then it supplies one force, and we derive the well-known fact that an object needs to be supported from underneath by three or more forces for stability.

This applies to the hand-chopstick system, where for simplicity, we can ignore gravity. Given how chopstick are disposed with respect to the hand, it is in fact convenient to view the hand as exerting five forces per chopstick. Fig. 1 shows the forces involved.

The middle of the lower chopstick is held steady on the ring finger. Typically it is gently wedged in the angle between the edge of the fingernail and the side of the fleshy pad of the fingertip, which we model by the forces $\boldsymbol{f}_{\text{HL1}}$ and $\boldsymbol{f}_{\text{HL2}}$ in Fig. 1. These are predominantly directed in the plane of the diagram, with a small component at right angles, out of the plane of the diagram, towards the reader. The back end of the lower chopstick is held on the fleshy part between the thumb and palm, and the forces are modeled by $\boldsymbol{f}_{\text{HL4}}$ and $\boldsymbol{f}_{\text{HL5}}$. Again these are mostly in the plane of the diagram, with a small component outwards, towards the reader. Opposing all the outwards components is $\boldsymbol{f}_{\text{HL3}}$ (the force drawn with the blob at its tail in Fig. 1), which is exerted by the lower end of the thumb, predominantly inwards into the diagram.

If the chopstick is merely being held steady, then these forces sum to zero. However, if food is being held, then the user adjusts the individual forces so that they sum to $\boldsymbol{f}_{\mathrm{HCL}}$:

$$\boldsymbol{f}_{\mathrm{HL1}} + \boldsymbol{f}_{\mathrm{HL2}} + \boldsymbol{f}_{\mathrm{HL3}} + \boldsymbol{f}_{\mathrm{HL4}} + \boldsymbol{f}_{\mathrm{HL5}} \ = \ \boldsymbol{f}_{\mathrm{HCL}} \tag{17}$$

The story for the upper chopstick is similar. The forces $\boldsymbol{f}_{\mathrm{HU1}}$ and $\boldsymbol{f}_{\mathrm{HU2}}$, formed by the more pronounced wedge between first and second fingers, serves to firmly hold and direct the middle of the chopstick in order to open and close the chopsticks for grasping food. Forces $\boldsymbol{f}_{\mathrm{HU4}}$ and $\boldsymbol{f}_{\mathrm{HU5}}$, exerted by the dip between the palm knuckle and first knuckle of the index finger, support the back of the chopstick. And vertical movement is restrained by $\boldsymbol{f}_{\mathrm{HU3}}$, once more indicated with a blob at its tail in Fig. 1, exerted by the upper part of the thumb. Again, if the chopstick is just being held steady, then these forces sum to zero. However, if food is being grasped, then they sum to $\boldsymbol{f}_{\mathrm{HCU}}$:

$$\boldsymbol{f}_{\mathrm{HU1}} + \boldsymbol{f}_{\mathrm{HU2}} + \boldsymbol{f}_{\mathrm{HU3}} + \boldsymbol{f}_{\mathrm{HU4}} + \boldsymbol{f}_{\mathrm{HU5}} \ = \ \boldsymbol{f}_{\mathrm{HCU}} \tag{18}$$

(N.B. In reality, many guides to eating with chopsticks recommend all sorts of alternative configurations for holding chopsticks (see eg. [3]), but the configuration described here is the only one that the first author has found to permit both adequate chopstick maneuvrability and sufficient deployable resultant force, especially when it comes to bigger pieces of food.)

With these observation, we can decompose the GRASPFOOD$_C$ function into its plant and controller subsystems, rules CHOPSTICK$_P$ and HAND$_C$.

In those rules, we have singled out $\boldsymbol{f}_{\mathrm{CU}}$ and $\boldsymbol{f}_{\mathrm{CL}}$ as output parameters in the signature of HAND$_C$ for emphasis. They are quantities derived from the underlying hand forces, which the chopsticks react to by setting their forces appropriately. The equalities $\boldsymbol{f}_{\mathrm{HCU}} = \boldsymbol{f}_{\mathrm{CU}}$ and $\boldsymbol{f}_{\mathrm{HCL}} = \boldsymbol{f}_{\mathrm{CL}}$ again become part of the domain theory of statics.

```
CHOPSTICK$_P$ =                                                    (19)
choose $\boldsymbol{f}'_{\mathrm{HCU}}, \boldsymbol{f}'_{\mathrm{HCL}}$
    with $\boldsymbol{f}'_{\mathrm{HCU}} + \boldsymbol{f}'_{\mathrm{HCL}} = \mathbf{0}$
    do $\boldsymbol{f}_{\mathrm{HCU}} := \boldsymbol{f}'_{\mathrm{HCU}}, \boldsymbol{f}_{\mathrm{HCL}} := \boldsymbol{f}'_{\mathrm{HCL}}$
```

```
HAND$_C$(out $\boldsymbol{f}_{\mathrm{CU}}, \boldsymbol{f}_{\mathrm{CL}}$) =                                      (20)
choose $\boldsymbol{f}'_{\mathrm{HU1}}, \boldsymbol{f}'_{\mathrm{HU2}}, \boldsymbol{f}'_{\mathrm{HU3}}, \boldsymbol{f}'_{\mathrm{HU4}}, \boldsymbol{f}'_{\mathrm{HU5}}, \boldsymbol{f}'_{\mathrm{HL1}}, \boldsymbol{f}'_{\mathrm{HL2}}, \boldsymbol{f}'_{\mathrm{HL3}}, \boldsymbol{f}'_{\mathrm{HL4}}, \boldsymbol{f}'_{\mathrm{HL5}}$
    with $\boldsymbol{f}'_{\mathrm{HU1}} + \boldsymbol{f}'_{\mathrm{HU2}} + \boldsymbol{f}'_{\mathrm{HU3}} + \boldsymbol{f}'_{\mathrm{HU4}} + \boldsymbol{f}'_{\mathrm{HU5}} +$
         $\boldsymbol{f}'_{\mathrm{HL1}} + \boldsymbol{f}'_{\mathrm{HL2}} + \boldsymbol{f}'_{\mathrm{HL3}} + \boldsymbol{f}'_{\mathrm{HL4}} + \boldsymbol{f}'_{\mathrm{HL5}} = \mathbf{0}$
         $|\boldsymbol{f}'_{\mathrm{HU1}} + \boldsymbol{f}'_{\mathrm{HU2}} + \boldsymbol{f}'_{\mathrm{HU3}} + \boldsymbol{f}'_{\mathrm{HU4}} + \boldsymbol{f}'_{\mathrm{HU5}}| =$
         $|\boldsymbol{f}'_{\mathrm{HL1}} + \boldsymbol{f}'_{\mathrm{HL2}} + \boldsymbol{f}'_{\mathrm{HL3}} + \boldsymbol{f}'_{\mathrm{HL4}} + \boldsymbol{f}'_{\mathrm{HL5}}| \geq D$
    do $\boldsymbol{f}_{\mathrm{HU1}} := \boldsymbol{f}'_{\mathrm{HU1}} \dots \boldsymbol{f}_{\mathrm{HU5}} := \boldsymbol{f}'_{\mathrm{HU5}}, \boldsymbol{f}_{\mathrm{HL1}} := \boldsymbol{f}'_{\mathrm{HL1}} \dots \boldsymbol{f}_{\mathrm{HL5}} := \boldsymbol{f}'_{\mathrm{HL5}},$
       $\boldsymbol{f}_{\mathrm{CU}} := \boldsymbol{f}'_{\mathrm{HU1}} + \boldsymbol{f}'_{\mathrm{HU2}} + \boldsymbol{f}'_{\mathrm{HU3}} + \boldsymbol{f}'_{\mathrm{HU4}} + \boldsymbol{f}'_{\mathrm{HU5}},$
       $\boldsymbol{f}_{\mathrm{CL}} := \boldsymbol{f}'_{\mathrm{HL1}} + \boldsymbol{f}'_{\mathrm{HL2}} + \boldsymbol{f}'_{\mathrm{HL3}} + \boldsymbol{f}'_{\mathrm{HL4}} + \boldsymbol{f}'_{\mathrm{HL5}},$
       $grasped$ := TRUE
```

5 Continuous Controller Synthesis

The reader may well have noticed that there are some slightly unnatural aspects of the account of chopstick use that we gave. The ASM rules in the preceding section were the usual kind of discrete ASM rules. However, grasping via chopsticks is not the usual kind of discrete event control system. In particular, both the chopsticks and the food react instantaneously to the force exerted by the other, and not to the previous value maintained by the other, as one would expect in a normal discrete event control system. We handled this via the domain theory, which demanded that the opposed forces exactly matched, without giving any inkling as to how this might be accomplished.

In a more realistic account, the force applied by the chopsticks to the food moves smoothly from zero to a value sufficient to ensure grasping, and the food senses this and smoothly reacts by offering a matching resistive force. The sudden assignment to equal and opposite values in the discrete picture is replaced by a pair of differential equations which state that the derivatives of the chopstick and food forces are equal and opposite over time, which together with initial conditions stating that both are zero, guarantees that the forces themselves remain equal and opposite.

Incorporating these insights into the ASM framework requires an extension of ASM to include continuously varying behaviours as well as discrete changes. In [1] the authors give such an extension which we briefly recapitulate now.

5.1 Continuous ASM

We partition the variables into two subsets: the **mode variables**, whose types are discrete sets, and the **pliant variables**, whose types include topologically dense sets, and which are permitted to evolve both continuously and via discrete changes. By restricting to mode variables alone, we recover the conventional discrete ASM framework.

Time is modelled as an interval $\mathcal{T}$ of the real numbers $\mathbb{R}$, with a finite left endpoint for the initial state, and with a right endpoint which is finite or infinite, as needed. $\mathcal{T}$ partitions into a sequence of left-closed right-open intervals, $\langle [t_0 \ldots t_1), [t_1 \ldots t_2), \ldots \rangle$, the coarsest partition such that all discontinuous changes take place at some boundary point t_i. Mode variables are constant on each of these intervals, while pliant variables evolve continuously. Otherwise arbitrary continuous evolution is constrained within reasonable bounds by three main restrictions:

I Zeno: there is a constant δ_{Zeno}, such that for all i needed, $t_{i+1} - t_i \geq \delta_{\mathsf{Zeno}}$.

II Limits: for every variable x, for every time $t \in \mathcal{T}$, and with $\delta > 0$, the left limit $\lim_{\delta \to 0} x(t - \delta)$ written $\overrightarrow{x(t)}$ and right limit $\lim_{\delta \to 0} x(t + \delta)$, written $\overleftarrow{x(t)}$ exist, and for every t, $x(t) = \overleftarrow{x(t)}$.

III Differentiability: The behaviour of every pliant variable x in the interval $[t_i \ldots t_{i+1})$ is given by the solution of a well posed initial value problem $\mathcal{D}\, xs = \phi(xs, t)$ (where $\mathcal{D}$ is the time derivative).

The two kinds of variable (mode and pliant) are reflected in two kinds of transitions: mode and pliant. Mode transitions, given by rules of the form (21), just record discrete transitions from before-values to after-values of variables, with the use of the left limit for before-values and right limit for after-values making the semantics of these transitions instantaneous. Both kinds of variable can be subject to a mode transition, and in (21), where we decorate the variables with this limit information, we single out inputs is and outputs os in the signature of OP.

$$\begin{aligned}&\textsc{Op}(\textbf{in}\ \overrightarrow{is}, \textbf{out}\ \overleftarrow{os})\ = \qquad\qquad (21)\\ &\textbf{if}\ \ guard(\overrightarrow{xs}, \overrightarrow{is})\ \ \textbf{then choose}\ \ \overleftarrow{xs}, \overleftarrow{os}\ \ \textbf{with}\ \ rel(\overleftarrow{xs}, \overrightarrow{xs}, \overrightarrow{is}, \overleftarrow{os})\\ &\quad \textbf{do}\ \ xs, os\ := \overleftarrow{xs}, \overleftarrow{os}\end{aligned}$$

Pliant transitions describe continuous changes for pliant variables. While a mode transition captures a single before-/after-value pair, a pliant transition is a family of before-/after-value pairs parameterized by the relevant time interval $[t_i \dots t_{i+1})$. The before-value is, in each case, the value at t_i, while the after-value refers to an arbitrary time in the interval, so the two values are separated in time. A rule for a pliant transition can be written as in (22), where the symbol $\overset{\text{c}}{=}$ syntactically distinguishes a pliant transition from a mode transition.

$$\begin{aligned}&\textsc{PliOp}(\textbf{in}\ is(t \in (t_{\mathrm{L}(t)} \dots t_{\mathrm{R}(t)})), \textbf{out}\ os(t \in (t_{\mathrm{L}(t)} \dots t_{\mathrm{R}(t)})))\ \overset{\text{c}}{=} \qquad (22)\\ &\textbf{if}\ \ IV(xs(t_{\mathrm{L}(t)}))\ \ \textbf{and}\ \ guard(xs(t_{\mathrm{L}(t)}))\ \ \textbf{then with}\ \ rel(xs, is, os, t)\\ &\quad \textbf{do}\ \ xs(t), os(t)\ :=\ \textbf{solve}\ DE(xs(t), is(t), os(t), t)\end{aligned}$$

In (22), $\mathrm{L}(t) = \max\{i \,|\, t_i \le t\}$ and $\mathrm{R}(t) = \min\{i \,|\, t_i > t\}$ so that we do not have to statically know the index i for the interval $[t_i \dots t_{i+1})$, thus making the notation generic. Furthermore, IV and $guard$ refer to the initial value and any additional guard restriction that apply for the initial value problem in $[t_i \dots t_{i+1})$. DE is the differential equation of the initial value problem, while rel expresses any additional constraints that must hold beyond DE. Inputs is and outputs os (shown as depending on the whole interval $(t_{\mathrm{L}(t)} \dots t_{\mathrm{R}(t)})$) again appear in the signature. If, as can often happen, we know the form of the continuous behaviour that we want (in contrast to merely knowing a differential equation for it), then we can replace the **solve** clause with a straightforward assignment using a **do**.

A continuous ASM ruleset, consisting of rules as we have described, is **well formed** iff the initial transition is a mode transition, every mode transition enables a pliant transition (but no mode transition), and every pliant transition (except perhaps for a final one) enables a mode transition (which, during runtime, preempts it).

Given a conventional discrete ASM system, we can rather trivially turn it into a continuous ASM system, as follows:

– consider the original discrete ASM rules as mode rules,
– decide on a fixed duration δ_t,
– determine that each state of the discrete event ASM system will persist for δ_t,
– add continuous ASM rules setting time derivatives of all ASM state variables to 0,
– add a time variable, and enable all mode transitions after integral multiples of δ_t.

5.2 Continuous Controller Synthesis

We can ask how the process of separating a set of rules into controller and plant rules goes, when we have pliant as well as mode transitions. In fact, the process is very similar to what went before. Since mode rules are identical to the rules we considered earlier, there is nothing new for them. For pliant rules, they also have a $guard$ and a rel, and for these we demand the same conditions as previously. But there is also the **solve** clause. We need to stipulate that it separates cleanly into controller and plant in the same way that $guard$ and rel do so that the rule as a whole splits neatly.

The tuple of differential equations $\mathcal{D}\, xs = \phi(xs, t)$ contained in the **solve** clause naturally splits into two: $\mathcal{D}\, xs_C = \phi_C(xs, t)$ and $\mathcal{D}\, xs_P = \phi_P(xs, t)$. But there is no *a priori* guarantee that $\phi_C(xs, t)$ contains only the variables xs_C, xs_P^c, and $\phi_P(xs, t)$ contains only the variables xs_P, xs_C^p. So this is what we must additionally demand for admissibility.

It is clear that the embedding of discrete ASMs into continuous ASMs at the end of the last section is admissible in the extended sense just discussed, provided the original discrete ASM system is admissible, so that the properties derived for controller synthesis in Sections 2 and 3 carry through essentially unchanged.

6 Continuous Grasping

Let us revisit the chopsticks case study in the continuous ASM framework to see how the latter can lend it a more persuasive air.

As before, we restrict the modeling to that of forces only (albeit now allowing them to vary continuously). This avoids complications arising from having to consider movement of either the food or the chopsticks, or distortions of the shape of either the food or chopsticks consequent on them experiencing the forces that we model, and keeps the model within a relatively limited space.

We concentrate on elaborating the simpler model in Section 4.1. Time $t = 0$ triggers the intial mode rule:

```
START =                                                        (23)
if  $t = 0$  then
    do  $mode$ := $grasping$, $grasped$ := $undef$,
        $\boldsymbol{f}_{\mathrm{FU}}$ := $\mathbf{0}$, $\boldsymbol{f}_{\mathrm{FL}}$ := $\mathbf{0}$, $\boldsymbol{f}_{\mathrm{HCU}}$ := $\mathbf{0}$, $\boldsymbol{f}_{\mathrm{HCL}}$ := $\mathbf{0}$
```

The $grasping$ mode enables the following pliant rule:

```
GRASPING $\overset{\mathrm{c}}{=}$                                             (24)
if  $mode = grasping$  then
    do  $\boldsymbol{f}_{\mathrm{FU}}, \boldsymbol{f}_{\mathrm{FL}}, \boldsymbol{f}_{\mathrm{HCU}}, \boldsymbol{f}_{\mathrm{HCL}}$ :=
        solve $[\, \mathcal{D}\boldsymbol{f}_{\mathrm{FU}}, \mathcal{D}\boldsymbol{f}_{\mathrm{FL}}, \mathcal{D}\boldsymbol{f}_{\mathrm{HCU}}, \mathcal{D}\boldsymbol{f}_{\mathrm{HCL}} \,] = [\, \mathbf{e}_{\mathrm{z}}, -\mathbf{e}_{\mathrm{z}}, -\mathbf{e}_{\mathrm{z}}, \mathbf{e}_{\mathrm{z}} \,]$
```

This rule causes the forces $\boldsymbol{f}_{\mathrm{FU}}, \boldsymbol{f}_{\mathrm{FL}}, \boldsymbol{f}_{\mathrm{HCU}}, \boldsymbol{f}_{\mathrm{HCL}}$ to acquire suitable pairwise equal and opposite rates of change, of magnitude 1, oriented along the unit vector of the z axis. This causes these forces to change continuously (although in fact non-smoothly[4]) away from zero at a uniform rate. The continuous grasping persists until a time t_{STOP}, when it is determined whether enough force has been applied to hold the food:

```
STOPGRASPED = if  $t = t_{\mathrm{STOP}} \wedge \boldsymbol{f}_{\mathrm{HCU}} \geq D$  then     (25)
    do  $mode$ := $stop$, $grasped$ := TRUE
```

[4] Since the derivatives of the forces jump discontinuously at $t = 0$, the forces themselves, though continuous, experience a kink at $t = 0$.

$$\textsc{StopDislodged} = \textbf{if } t = t_{\text{STOP}} \wedge \boldsymbol{f}_{\text{HCU}} < D \textbf{ then} \\ \quad \textbf{do } mode := stop, grasped := \textsc{false} \tag{26}$$

The stopped mode just enters a pliant final state:

$$\textsc{F-Idle} \overset{\text{c}}{=} \textbf{if } mode = stop \textbf{ then do } \mathsf{skip} \tag{27}$$

The above is all consistent with the domain theory (14), although the theory would have to be augmented by various facts concerning time and the additional variables introduced above, in order that the natural continuous counterparts of the statements in (5) could hold.[5]

6.1 Decomposing Continuous Grasping

We now look at applying the decomposition strategy discussed earlier to the above integrated model. We asssume that the chopsticks, as controller, are in charge, and own variables like $mode$ and $grasped$. We decompose the rules above one by one, starting with START:

$$\textsc{Start}_C = \\ \textbf{if } t = 0 \textbf{ then} \\ \quad \textbf{do } mode := grasping, grasped := undef, \boldsymbol{f}_{\text{HCU}} := 0, \boldsymbol{f}_{\text{HCL}} := 0 \tag{28}$$

$$\textsc{Start}_P = \\ \textbf{if } t = 0 \textbf{ then do } \boldsymbol{f}_{\text{FU}} := 0, \boldsymbol{f}_{\text{FL}} := 0 \tag{29}$$

Next, the decomposition of the GRASPING rule. This yields:

$$\textsc{Grasping}_C(\textbf{out } \boldsymbol{of}_{\text{HCU}}, \boldsymbol{of}_{\text{HCL}}) \overset{\text{c}}{=} \\ \textbf{if } mode = grasping \textbf{ then} \\ \quad \textbf{do } \boldsymbol{f}_{\text{HCU}}, \boldsymbol{f}_{\text{HCL}} := \textbf{solve } [\, \mathcal{D}\boldsymbol{f}_{\text{HCU}}, \mathcal{D}\boldsymbol{f}_{\text{HCL}} \,] = [\, -\mathbf{e}_{\text{z}}, \mathbf{e}_{\text{z}} \,], \\ \quad\quad \boldsymbol{of}_{\text{HCU}} := \boldsymbol{f}_{\text{HCU}}, \boldsymbol{of}_{\text{HCL}} := \boldsymbol{f}_{\text{HCL}} \tag{30}$$

$$\textsc{Grasping}_P(\textbf{in } \boldsymbol{if}_{\text{HCU}}, \boldsymbol{if}_{\text{HCL}}) \overset{\text{c}}{=} \\ \textbf{if } mode = grasping \textbf{ then do } \boldsymbol{f}_{\text{FU}} := -\boldsymbol{if}_{\text{HCU}}, \boldsymbol{f}_{\text{FL}} := -\boldsymbol{if}_{\text{HCL}} \tag{31}$$

The above rules display a slightly more complex manner of decomposition than we have considered hitherto. Instead of merely partitioning the variables and determining that subsystem B has read access to some of the variables owned by subsystem A, we have introduced input and output variables that do this job explicitly. So the chopsticks have output variables $\boldsymbol{of}_{\text{HCU}}$ and $\boldsymbol{of}_{\text{HCL}}$, which are just copies of variables $\boldsymbol{f}_{\text{HCU}}$ and $\boldsymbol{f}_{\text{HCL}}$, and the food has input variables $\boldsymbol{if}_{\text{HCU}}$ and $\boldsymbol{if}_{\text{HCL}}$, which are used to read the relevant values in. Thus, the modeling is a now little different in that the food explicitly reacts to the forces it senses (by generating equal and opposite forces of its own) — we have substituted equals for equals, but have gone beyond the simple syntactic transformation described earlier in the paper. It is a natural temptation to do this at the more realistic and

[5] The domain theory would also have to be supplemented with a background theory of facts about calculus, continuous mathematics etc., as needed.

practical level of modeling that we have reached. Since the new variables are just copies of existing ones, only trivial modifications are needed to the earlier formal results.

Next are the STOP rules:

$$\textsc{StopGrasped}_C \;=\; \textbf{if}\;\; t = t_{\text{STOP}} \;\wedge\; \boldsymbol{f}_{\text{HCU}} \geq D \;\;\textbf{then}\;\; \textbf{do}\;\; mode := stop,\, grasped := \text{TRUE} \tag{32}$$

$$\textsc{StopDislodged}_C \;=\; \textbf{if}\;\; t = t_{\text{STOP}} \;\wedge\; \boldsymbol{f}_{\text{HCU}} < D \;\;\textbf{then}\;\; \textbf{do}\;\; mode := stop,\, grasped := \text{FALSE} \tag{33}$$

$$\textsc{StopGrasped}_P \;=\; \textbf{if}\;\; t = t_{\text{STOP}} \;\;\textbf{then do}\;\; \mathsf{skip} \tag{34}$$

$$\textsc{StopDislodged}_P \;=\; \textbf{if}\;\; t = t_{\text{STOP}} \;\;\textbf{then do}\;\; \mathsf{skip} \tag{35}$$

And lastly the final idle rules:

$$\text{F-}\textsc{Idle}_C \;\overset{\text{c}}{=}\; \textbf{if}\;\; mode = stop \;\;\textbf{then do}\;\; \mathsf{skip} \tag{36}$$

$$\text{F-}\textsc{Idle}_P \;\overset{\text{c}}{=}\; \textbf{if}\;\; mode = stop \;\;\textbf{then do}\;\; \mathsf{skip} \tag{37}$$

The preceding shows that the controller synthesis procedure that we have described is as applicable to the continuous extension of ASM as it is to the discrete version. We could now go on to apply the same approach to create a continuous version of the decomposed hand+chopsticks model, but lack of space prevents us from doing this.

7 Conclusion

In this paper we have introduced the controller synthesis problem for ASM systems. The motivation was that from a goal oriented point of view, it is often more convenient to focus on overall system objectives at the outset, and to postpone detailed implementation issues, such as the specific assignment of functionality to controller or to plant, till later.

We showed briefly that controller synthesis, as we have defined it, is undecidable, and we gave a safe approximation. We then illustrated the problem with a case study based on holding food with chopsticks.

We note that the conditions demanded of the controller and of the plant in our conditions for safe controller synthesis in (4), each relate the subsystem in question to the originating system (and only to the originating system). Thus they are completely symmetrical between the controller and plant and do not depend either on there being exactly two subsystems in play. Therefore, the result generalizes to a partition of the originating system into an arbitrary number of subsystems, each built in the same fashion, with some variables to which it has exclusive write access, and a larger set of variables to which it has read access.

The preceding remark is well illustrated by the chopstick case study, since after the initial decomposition into food (plant) and hand plus chopsticks (controller), we were able to repeat the decomposition of the hand plus chopsticks subsystem yielding a further separation into chopsticks (plant) and hand (controller), resulting in a three way partition of the original system.

In practice, the successful satisfaction of the conditions in (4) often demands that a nontrivial domain theory plays a significant role. In effect, this captures the fact that control of a system can be achieved by applying certain signals to it, only because natural laws connect these signals to the behaviour of other system attributes in a predictable way. Our simple chopstick case study illustrated this admirably.

We then considered continuous ASMs, and briefly discussed how the controller synthesis problem could be extended to that formalism, illustrating it with a further elaboration of the chopsticks case study.

Although we have focused on a very simple scenario, the ideas that we have explored have an applicability that is much wider than we have mentioned hitherto, especially in the context of today's hybrid and cyber-physical systems [6, 4, 5, 7]. In these, there is nowadays a strong tendency towards distributed solutions to problems decribable in a global manner. So the initial global conception of the problem needs to be decomposed into a number of subsystems that co-operate to form the global solution. Not only are many of these problems intrinsically control problems anyway, making our approach directly applicable, but the abstract version of the decomposition technique that we have explored, tailored as it is to the details of ASM rule scheduling, acts as a surrogate for a much wider gamut of problems and their solutions.

References

1. Banach, R., Zhu, H., Su, W., Wu, X.: Continuous ASM, and a Pacemaker Sensing Fragment (2011), these proceedings.
2. Börger, E., Stärk, R.: Abstract State Machines. A Method for High Level System Design and Analysis. Springer (2003)
3. Google search: Eating with chopsticks
4. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer (2010)
5. Sztipanovits, J.: Model Integration and Cyber Physical Systems: A Semantics Perspective. In: Butler, Schulte (eds.) Proc. FM-11. Springer, LNCS 6664, p.1, http://sites.lero.ie/download.aspx?f=Sztipanovits-Keynote.pdf (2011), Invited talk, FM 2011, Limerick, Ireland
6. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer (2009)
7. Willems, J.: Open Dynamical Systems: Their Aims and their Origins. Ruberti Lecture, Rome (2007),
http://homes.esat.kuleuven.be/~jwillems/Lectures/2007/Rubertilecture.pdf