Ajax Complexity

Akash K Singh, PhD
IBM Corporation
Sacramento, USA

Abstract—For century, This paper discuss the new era of Internet application and user experience, Ajax is a new technology and this paper address the Software system complexity and Algorithms for better feature and performance.

Keywords- Web Technologies, AJAX, Web2.0

I. INTRODUCTION

Over the last few years, the web is establishing increased importance in society with the rise of social networking sites and the semantic web, facilitated and driven by the popularity of client-side scripting commonly known as AJAX. These allow extended functionality and more interactivity in web applications. Engineering practices dictate that we need to be able to model these applications. However, languages to model web applications have fallen behind, with most existing web modelling languages still solely focused on the hypertext structure of web sites, with little regard for user interaction or common web-specific concepts. This paper provides an overview of technologies in use in today's web applications, along with some concepts we propose are necessary to model these. We present a brief survey of existing web modelling languages including WebML, UWE, W2000 and OOWS, along with a discussion of their capability to describe these new modeling approaches. Finally, we discuss the possibilities of extending an existing language to handle these new concepts. Keywords: web engineering, models, interactivity, AJAX, RIAs, events.

The World Wide Web started out in the early 1990s as an implementation of a globally distributed hypertext system. Primitive pieces of software called web browsers allowed users to render hypertext into visually pleasing representations that could be navigated by keyboard or mouse. These early web sites were generally static pages, and were typically modeled with languages focused on the hypertext structure and navigation of the web site (Garzotto et al. 1993). The full integration of hypertext with relational databases allowed the creation of data-intensive websites, which also necessitated new modelling concepts and languages (Merialdo et al. 2003). Currently, the most popular modelling languages for web applications areWebML (Ceri et al. 2000) and UWE (Koch & Kraus 2002). Both of these languages represent web applications using conceptual models (data structure of the application domain), navigational models, and presentation models. As such, the ability to express the interactivity of the application is generally restricted to the navigational models, which allow designers to visually represent the components, links and pages of the application. These languages are excellent at describing older web applications; however recently the increased use of interactivity, client-side scripting, and web-specific concepts such as cookies and sessions have left existing languages struggling to keep up with these Rich Internet Applications (RIAs: Preciado et al. 2005). In this paper we aim to review these existing languages and identify where they are falling short, and how they could be improved. This paper is organised as follows. Section 2 is an overview of some of the features possible with rich scripting support. To model these new features, we propose in Section 3 some new modelling concepts for interactive web applications. We present a brief survey of the existing modelling languages WebML and UWE in Sections 4 and 5, and discuss their ability to model these new concepts. We briefly mention W2000, OOWS and other potential languages in Section 6; a summary of our language evaluations are presented in Table 2. In the final section, we discuss our findings, provide an overview of related work, and highlight future work of this research project. 2 New Features Arguably, the most important recent feature of the web is the ability to run scripts on the client (generally through Javascript). Combined with the ability to access and modify client-side Document Object Models (DOM:W3C Group 2004) of the browser, and the ability to compose asynchronous background requests to the web, these concepts together are commonly referred to as AJAX (Garrett 2005). AJAX allows applications to provide rich client-side interfaces, and allows the browser to communicate with the web without forcing page refreshes; both fundamental features of RIAs. Technologies like AJAX support thin client applications that can take full advantage of the computer power of the clients. These applications reduce the total cost of ownership (TCO) to organisations as they
are deployed and maintained on directly manageable servers, and aim to be platform-independent on the client side. To achieve this, AJAX has had to overcome limitations of the underlaying HTTP/HTML protocols, such as synchronous and stateless request processing, and the pull model limitation where application state changes are always initiated by the client1. This has resulted in rich applications that use the web browser as a virtual machine. The impact of these technologies has been significant; new services such as Google Docs

(Google Inc. 2006) are implementing collaborative software solutions directly on the web, based on the software as a service philosophy, and to some degree competing with traditional desktop software such as Microsoft Office. RIAs can also be developed in environments such as Flash, which are provided as a plugin to existing web browsers, but can reduce accessibility2. One popular example of AJAX is to provide an auto-compliable destination address text field in an e-mail web application. As the user enters characters into this field, the client contacts the server for addresses containing these characters, displaying a list of suggested addresses. This improves usability, potentially reduces the overall bandwidth of network communication, and improves interactivity and responsiveness. An investigation of some of the most popular AJAX-based websites on the web allows us to identify some of the features that these new technology provides to web applications. This has allowed us to develop a comprehensive selection of use cases for AJAX technologies, which we omit from this paper for brevity. Without going into detail, and removing features that are already addressed in existing modeling languages, new application features that require support include:

1. Storing data on the client and/or server, both volatile and persistent3;
2. Allowing automatic user authentication based on cookies4;
3. Allowing form validation to occur on the server,on the client before submission, or in real-time during form entry;
4. Providing different output formats for resources, including HTML, XML, WML, and Flash, possibly based on the user-agent of the visitor;
5. Providing web services and data feeds, and integration with external services and feeds, both on the server and the client;
6. Preventing the user from corrupting the state of a web application, for example by using browser navigation buttons;
7. Providing more natural user actions such as dragand- drop, keyboard shortcuts, and interactive maps;
8. Describing visual effects of transitions between application states5;
9. Having scheduled events on either the client or the server;
10. Allowing web applications to be used offline6;
11. Distributing functionality between the client and the server, based on client functionality, determined at runtime.

These new features are distributed over both the clients and servers of web applications. Existing languages based solely on replacing the entire client-side DOM on each request are clearly no longer appropriate, as scripting permits modifying the DOM at runtime. We require a more dynamic language, which can be extended to handle these new features.

Recently, many new web trends have appeared under the Web 2.0 umbrella, changing the web significantly, from read-only static pages to dynamic user-created content and rich interaction. Many Web 2.0 sites rely heavily on AJAX (Asynchronous JAVASCRIPT and XML) [8], a prominent enabling technology in which a clever combination of JAVASCRIPT and Document Object Model (DOM) manipulation, along with asynchronous client/server delta communication [16] is used to achieve a high level of user interactivity on the web. With this new change comes a whole set of new challenges, mainly due to the fact that AJAX shatters the metaphor of a web 'page' upon which many classic web technologies are based. One of these challenges is testing such applications [6, 12, 14]. With the ever-increasing demands on the quality of Web 2.0 applications, new techniques and models need to be developed to test this new class of software. How to automate such a testing technique is the question that we address in this paper. In order to detect a fault, a testing method should meet the following conditions [18, 20]: reach the fault-execution, which causes the fault to be executed, trigger the error creation, which causes the fault execution to generate an incorrect intermediate state, and propagate the error, which enables the incorrect intermediate state to propagate to the output and cause a detectable output error. Meeting these reach/trigger/propagate conditions is more difficult for AJAX applications compared to classical web applications. During the past years, the general approach in testing web applications has been to request a response from the server (via a hypertext link) and to analyze the resulting HTML. This testing approach based on the page-sequence paradigm has serious limitations meeting even the first (reach) condition on AJAX sites. Recent tools such as Selenium1 use a capture/replay style for testing AJAX applications. Although such tools are capable of executing the fault, they demand a substantial amount of manual effort on the part of the tester. Static analysis techniques have limitations in revealing faults which are due to the complex run-time behavior of modern rich web applications. It is this dynamic run-time interaction that is believed [10] to make testing such applications a challenging task. On the other hand, when applying dynamic analysis on this new domain of web, the main difficulty lies in detecting the various doorways to different dynamic states and providing proper interface mechanisms for input values. In this paper, we discuss challenges of testing AJAX and propose an automated testing technique for finding faults in AJAX user interfaces. We extend our AJAX crawler, CRAWLJAX (Sections 4–5), to infer a state-flow graph for all (client-side) user interface states. We identify AJAX-specific faults that can occur in such states and generic and application-specific invariants that can serve as oracle to detect such faults (Section 6). From the inferred graph, we automatically generate test cases (Section 7) that cover the paths discovered during the crawling process. In addition, we use our open source tool called ATUSA (Section 8), implementing the testing technique, to conduct a number of case studies (Section 9) to discuss (Section 10) and evaluate the effectiveness of our approach.

A. Interface Model

A web application's interface is most obviously characterized by the variety of UI widgets displayed on each page, which we represent by elements of the set Widgets. Web applications typically distinguish several basic widget classes such as text fields, radio buttons, drop-down list boxes etc.

(Classes := {ctext, cradio, ccheck, cselect1, cselectn}), which we identify through the relation class : Widgets → Classes.

For the purpose of input evaluation, it will be helpful to specify the ranges of values that users can enter/select in widgets. We specify this in the relation range: Widgets →P(S). Depending on the class of the widget w, range(w) will be:

• the generic set S for text fields, which allow any input;
• some fixed subset Sw →S for drop-down list boxes,which allow a 1-of-n selection;
• the power set P(Sw) of some fixed subset Sw →S for multi-select boxes, which allow an m-of-n selection;
• some string sw →S for individual check boxes and radio buttons, which are either undefined or have one particular value.

In applications based on our model, the placement of widgets on web pages (from the set Pages) is governed by a series of hierarchically nested layout containers (Containers) that define visual alignment and semantic cohesion of widgets. The nesting relationships between widgets and containers can be expressed in the relation container: (Widgets→ Containers) → (Containers->Pages) that indicates in which container or page s_→Containers → Pages a widget or container s→Widgets -> Containers is directly contained. To reason about transitive containment, we also define a convenience relation page: (Widgets→Containers) → Pages that identifies which page a widget is placed on by recursive application of the container relation: p = page(s) : → (p → Pages→p = container(s)) →c → Containers : (c = container(s) → p = page(c))

B. Data Model

In our formal model, the variables holding the web application's data are represented by elements of the set Variables. Variables may have different types—in most applications, we find Boolean, integer, floating-point and string values or sets

(Types := {P(B),P(Z),P(R),P(S)},respectively).
We express variables' types by the relationtype : Variables → Types.

To store the entered content, each widget must be bound to a variable in the application's data model. This binding is modeled by the relation binding : Widgets → Variables. Note that several widgets can be bound to the same variable (e.g. a group of check boxes whose combined state is stored as a set of string values).

C. Evaluation Aspects

Input evaluations are characterized by several criteria that together constitute particular behavior rules. In this paper, we will discuss input evaluation for the purpose of deciding validity, visibility, and availability of widgets, i.e. for interface responses such as highlighting violating widgets, hiding invisible widgets, and disabling (e.g. "graying out") unavailable widgets, respectively.

At the core of each rule is an expression e → Expressions that describes the actual evaluation of certain values in order to arrive at a decision for one of the above purposes. Our model allows expressions to consist of arbitrarily nestable terms. These can trivially be literals (out of the universal set L := B → R → S) or variables from the data model, but also comparisons, arithmetic, boolean or string operations, which can be distinguished by their operator op(e), so Expressions → (L → Variables) (for the sake of conciseness, we we will not go into the details of expressions' concrete structure). Ultimately, an expression must resolve to a boolean value indicating the outcome of the decision. Of course, a rule for any purpose must relate to certain subjects on which the respective reaction is effected. These may not only be individual widgets, but also groups of widgets contained directly or transitively in a particular container or page, so we define Subjects := Widgets → Containers → Pages. Note that the subject widgets do not necessarily correspond to the expression's parameters (business requirements might e.g. suggest that only one of several evaluated widgets should be highlighted as invalid if the validation fails). For the purpose of input validation, we must consider several additional characteristics. First, we can distinguish different levels of validation, which we will describe as Levels := {lexist, ltech, ldomain}. The most basic level is checking for the existence of any input in a required field. Next, the technical check concerns whether a particular input can be converted sensibly to the given data type. Finally, performing any domain-specific validation of the input is only sensible if the previous two validation levels were satisfied. In practice, not all validation rules would typically be evaluated at the same time—from our experience from several industrial projects, we rather identified four common validation triggers

(Triggers := {tblurWidget, tleavePage, tsaveData, tcommitData}):

Validation may occur upon a widget's "blurring" (i.e. losing focus) when the cursor is moved to another widget; upon leaving a page in order to jump to the next or previous page of the dialog; upon saving the data entered so far as a draft version, in order to prevent data loss or continue working on the dialog at a later time; and finally upon committing all entered data in order to proceed to the next task in a business process. By staging the validation through assigning rules to appropriate triggers, developers can strike a balance between business requirements and usability considerations, ensuring data integrity while maintaining users' flexibility in working with the application. In a similar vein, experience shows that typically not all rule violations are equally serious: Depending on the business semantics of each rule, developers may choose to assign different severity levels to it. We therefore distinguish

Severities := {sinfo, swarning, serror} (with the natural order sinfo < swarning < serror),

and define different behavior for different severities.

D. Evaluation Rules

Having introduced all aspects characterizing input evaluation, we can now define the constituent elements of the rules for different purposes: Rules determining visibility and availability of widgets are fully described by the deciding expression and the set of affected subjects, while validation rules require all of the aspects described above:

Rvisibility : → Expressions×P(Subjects)
Ravailability : → Expressions×P(Subjects)
Rvalidation: → Expressions×P(Subjects) × Levels × Triggers × Severities

While the visibility and availability rules, as well as the existence and domain validation rules, need to be specified by the application designer, the necessary technical validation checks can be inferred from the interface and data model. To facilitate an integrated display of all validation, we derive the subset of Rvalidation comprising the technical validation rules as
{(λ, w, ltech, tblurWidget, serror) | →w → Widgets},

based on the assumption that type or range violations should be detected as early as possible, and reported as errors. To access particular components of the rules' tuples, our following discussion will assume the existence of the convenience functions expression, subjects, level, trigger, and severity that return the respective components of a rule. Since we will often be interested in all rules pertaining to a certain subject, we also define the abbreviation Rs p to denote all rules for a purpose p that affect a subject s. Summing up, we can describe the static, design-time specification of input evaluation for a web application as a tuple Aspec := (Widgets, class, range, Containers, Pages, container, binding, Variables, type, Rvisibility , Ravailability, Rvalidation).

E. User Interface Behavior

Last but not least, we must define how the user interface reacts to the various conditions that arise from input evaluation; namely validation results, visibility and availability of widgets, and navigation options. These will be covered in the following subsections.

1) Issue Notifications: We suggest that validation issues be displayed in two ways: On top of each page, the interface displays a concise list of human-readable explanations for all violations that were identified on the current and other pages. In case several rules are violated for a particular set of subjects, we display only the most severe notification to reduce clutter, as indicated by the function issueDisp : Rvalidation → B:issueDisp(r) : → r → Issues → _r_ → Issues : (subjects(r_) → subjects(r) → severity(r_) > severity(r))

To further aid the user in identifying the invalid input, we highlight the respective widget in a color corresponding to the severity (e.g. red for errors, orange for warnings etc.). Two relationships influence this coloring scheme: Firstly, if the subject of a rule is not an individual widget, but rather a container, the issue is assumed to apply to all directly and transitively contain widgets, which are all colored accordingly. Secondly, if a subject is affected by several issues (through multiple rules or inclusion in affected containers), it will be colored according to the most severe issue. To indicate this, the partial relation highlight: Subjects →_ Severities indicates which severity (if any) applies to a particular subject: highlight(s) = v: → v = max ({v | v = highlight(container(s))} → {v | →r → Rs validation : (issueDisp(r) → v = severity(r)}))

We assume here that the relation max: P(Severities) → Severities returns the maximum element from a set of severities.

2) Visibility: In the previous section, we have already often relied on an indication of whether a particular interface component is currently visible. For any

given subject, this state depends both on any explicit visibility rules, and on the visibility of the surrounding containers, as the relation isVisible : Subjects → B indicates: isVisible(s) : → (isVisible(container(s)) → s → Pages) → r → Rvisibility(s): isSatisfied(expression(r))

In analogy to validation rules, where just one rule violation suffices to consider an input invalid, we require that all of a widget's applicable visibility rules must be satisfied for it to be visible.

3) Availability: In some use cases, developers may not want to render a widget invisible, thus hiding it from the interface model and removing its input from the data model, but would only like to prevent users from editing the widget's contents, even though it remains part of the interface and data model. This deactivation can be accomplished by "graying out" the widget or otherwise preventing it from gaining the input focus, while still remaining visible. In our model, availability rules are stated and evaluated just like visibility rules, as the relation isAvailable : Subjects → B indicates: isAvailable(s) : → (isAvailable(container(s)) → s → Pages) → r → Ravailability(s): isSatisfied(expression(r))

Note that while visibility affects the data model and is used in quite a few of the above relations, availability is a pure interface reaction that does not affect how data is evaluated or stored.

4) Navigation Opportunities: When considering the availability of widgets, the navigation buttons on each page (typically, for navigating forward and backward in a dialog wizard, saving a draft of the current data, or committing it for further processing) require special treatment: The user should be prevented from saving a draft, let alone committing all input, but possibly even leaving a page, when the model still violates any validation rules. Since the availability of the corresponding buttons does not depend directly on the widget contents, but on the outcome of all validations in the respective scope, this behavior cannot be specified by means of regular availability rules. Instead, our model contains built-in "meta" rules governing navigation opportunities. In the following predicates, we distinguish between validation rules that must be satisfied for saving a draft, and a possibly more restrictive set that must be satisfied for committing the input for further processing: commitEnabled : → r → Issues : (trigger(r) → commitBlocks → severity(r) = serror) saveEnabled : → r → Issues : (trigger(r) → saveBlocks → severity(r) = serror) leaveEnabled(from) : → r → Issues : (trigger(r) → leaveBlocks → severity(r) = serror →s → subjects(r): from = page(s))

F. AJAX Testing Challenges

In AJAX applications, the state of the user interface is determined dynamically, through event-driven changes in the browser's DOM that are only visible after executing the corresponding JAVASCRIPT code. The resulting challenges can be explained through the reach/trigger/propagate conditions as follows. Reach. The event-driven nature of AJAX presents the first serious testing difficulty, as the event model of the browser must be manipulated instead of just constructing and sending appropriate URLs to the server. Thus, simulating user events on AJAX interfaces requires an environment equipped with all the necessary technologies, e.g., JAVASCRIPT, DOM, and the XMLHttpRequest object used for asynchronous communication. One way to reach the fault-execution automatically for AJAX is by adopting a web crawler, capable of detecting and firing events on clickable elements on the web interface. Such a crawler should be able to exercise all user interface events of an AJAX site, crawl through different UI states and infer a model of the navigational paths and states. We proposed such a crawler for AJAX, discussed in our previous work [14], Trigger. Once we are able to derive different dynamic states of an AJAX application, possible faults can be triggered by generating UI events. In addition input values can cause faulty states. Thus, it is important to identify input data entry points, which are primarily comprised of DOM forms. In addition, executing different sequences of events can also trigger an incorrect state. Therefore, we should be able to generate and execute different event sequences. Propagate. In AJAX, any response to a client-side event is injected into the single-page interface and therefore, faults propagate to and are manifested at the DOM level. Hence, access to the dynamic run-time DOM is a necessity to be able to analyze and detect the propagated errors. Automating the process of assessing the correctness of test case output is a challenging task, known as the oracle problem [24]. Ideally a tester acts as an oracle who knows the expected output, in terms of DOM tree, elements and their attributes, after each state change. When the state space is huge, it becomes practically impossible. In practice, a baseline version, also known as the Gold Standard [5], of the application is used to generate the expected behavior. Oracles used in the web testing literature are mainly in the form of HTML comparators [22] and validators [2].

G. Deriving AJAX States

Here, we briefly outline our AJAX crawling technique and tool called CRAWLJAX [14]. CRAWLJAX can exercise client side code, and identify clickable elements that change the state within the browser's dynamically built DOM. From these state changes, we infer a state-flow graph, which captures the states of the user interface, and the possible event-based transitions between them. We define an AJAX UI state change as a change on the DOM tree caused either by server-side state changes propagated to the client, or client-side events handled by the AJAX engine. We model such changes by recording the paths (events) to these DOM changes to be able to navigate between the different states. Inferring the State Machine. The state-flow graph is created incrementally. Initially, it only contains the root state and new states are created and added as the application is crawled and state changes are analyzed. The following components participate in the construction of the graph: CRAWLJAX uses an embedded browser interface (with different implementations: IE, Mozilla) supporting technologies required by AJAX; A robot is used to simulate user input (e.g., click, mouseOver, text input) on the embedded browser; The finite state machine is a data component maintaining the state-flow graph, as well as a pointer to the current state; The controller has access to the browser's DOM and analyzes and detects state changes. It also controls the robot's actions and is responsible for updating the state machine when relevant changes occur on the DOM. Detecting Clickables. CRAWLJAX implements an algorithm which makes use of a set of candidate elements, which are all exposed to an event type (e.g., click, mouseOver). In automatic mode, the candidate clickables are labeled as such based on their HTML tag element name and attribute constraints. For instance, all elements with a tag div, a, and span having attribute class="menuitem" are considered as candidate clickable. For each candidate element, the crawler fires a click on the element (or other event types, e.g., mouseOver), in the embedded browser. Creating States. After firing an event on a candidate clickable, the algorithm compares the resulting DOM tree with the way as it was just before the event fired, in order to determine whether the event results in a state change. If a change is detected according to the Levenshtein edit distance, a new state is created and added to the state-flow graph of the state machine. Furthermore, a new edge is created on the graph between the state before the event and the current state. Processing Document Tree Deltas. After a new state has been detected, the crawling procedure is recursively called to find new possible states in the partial changes made to the DOM tree. CRAWLJAX computes the differences between the previous document tree and the current one, by means of an enhanced Diff algorithm to detect AJAX par-212 trial updates which may be due to a server request call that injects new elements into the DOM. Navigating the States. Upon completion of the recursive call, the browser should be put back into the previous state. A dynamically changed DOM state does not register itself with the browser history engine automatically, so triggering the 'Back' function of the browser is usually insufficient. To deal with this AJAX crawling problem, we save information about the elements and the order in which their execution results in reaching a given state. We then can reload the application and follow and execute the elements from the initial state to the desired state. CRAWLJAX adopts XPath to provide a reliable, and persistent element identification mechanism. For each state changing element, it reverse engineers the XPath expression of that element which returns its exact location on the DOM. This expression is saved in the state machine and used to find the element after a reload. Note that because of side effects of the element execution and server-side state, there is no guarantee that we reach the exact same state when we traverse a path a second time. It is, however, as close as we can get. Data Entry Points in order to provide input values on AJAX web applications, we have adopted a reverse engineering process, similar to [3, 10], to extract all exposed data entry points. To this end, we have extended our crawler with the capability of detecting DOM forms on each newly detected state (this extension is also shown in Algorithm 1). For each new state, we extract all form elements from the DOM tree. For each form, a hashcode is calculated on the attributes (if available) and the HTML structure of the input fields of the form. With this hashcode, custom values are associated and stored in a database, which are used for all forms with the same code. If no custom data fields are available yet, all data, including input fields, their default values, and options are extracted from the DOM form. Since in AJAX forms are usually sent to the server through JAVASCRIPT functions, the action attribute of the form does not always correspond to the server-side entry URL. Also, any element (e.g., A, DIV) could be used to trigger the right JAVASCRIPT function to submit the form. In this case, the crawler tries to identify the element that is responsible for form submission. Note that the tester can always verify the submit element and change it in the database, if necessary. Once all necessary data is gathered, the form is inserted automatically into the database. Every input form provides thus a data entry point and the tester can later alter the database with additional desired input values for each form. If the crawler does find a match in the database, the input values are used to fill the DOM form and submit it. Upon submission, the resulting state is analyzed recursively by the crawler and if a valid state change occurs the state-flow graph is updated accordingly. Testing AJAX States through Invariants with access to different dynamic DOM states we can check the user interface against different constraints. We propose to express those as invariants on the DOM tree, which we thus can check automatically in any state. We distinguish between invariants on the DOM-tree, between DOM-tree states, and application-specific invariants. Each invariant is based on a fault model [5], representing AJAX specific faults that are likely to occur and which can be captured through the given invariant.

II. PROPOSED APPROACH

The goal of the proposed approach is to statically check web application invocations for correctness and detect errors. There are three basic steps to the approach (A) identify generated invocations, (B) compute interfaces and domain constraints, and (C) check that each invocation matches an interface. A. Identify Invocation Related Information The goal of this step is to identify invocation related information in each component of the web application. The information to be identified is: (a) the set of argument names that will be included in the invocation, (b) potential values for each argument, (c) domain information for each argument, and (d) the request method of the invocation. The general process of this step is that the approach computes the possible HTML pages that each component can generate. During this process, domain and value information is identified by tracking the source of each substring in the computed set of pages. Finally, the computed pages and substring source information are combined to identify the invocation information. 1) Compute Possible HTML Pages: The approach analyzes a web application to compute the HTML pages each component can generate. Prior work by the author [4] is extended, to compute these pages in such a way as to preserve domain information about each invocation. The approach computes the fixed point solution to the data-flow equations and at the end of the computation, the fragment associated with the root method of each component contains the set of possible HTML pages that could be generated by executing the component. 2) Identify Domain and Value Information: The approach identifies domain and value information for each argument in an invocation. The key insight for this part of the approach is that the source of the substrings used to define invocations in an HTML page can provide useful information about the domain and possible values of each argument. For example, if a substring used to define the value of an invocation originates from a call to StringBuilder.append(int), this indicates that the argument’s domain is of type integer. To identify this type of information, strings from certain types of sources are identified and annotated using a process similar to static tainting. Then the strings and their corresponding annotations are tracked as the approach computes the fixed point solution to the equations. The mechanism for identifying and tracking string sources starts with the resolve function, which analyzes a node n in an application and computes a conservative approximation of the string values that could be generated at that node. The general intuition is that when the resolve function analyzes a string source that can indicate domain or value information, a special domain and value (DV) function is used to complete the analysis. The DV function returns a finite state automaton (FSA) defined as the quintuple (S, S0, F) whose accepted language is the possible values that could be generated by the expression. In addition, the DV function also defines two domain type, where T is a basic type of character, integer, float, long, double, or string; and V : S that maps each transition to a symbol in or a special symbol that denotes any value. D is used to track the inferred domain of a substring and V is used to track possible values. A DV function is defined for each general type of string source. For the purpose of the description of the DV functions below, e refers to any transition (S) defined by and the function L(e) returns the symbol associated with the transition e. Functions that return a string variable: Substrings originating from these types of functions can have any value and a domain of string. This is represented as V (e) and D(e) string. String constants: The string constant provides a value for the argument and a domain of string. This is represented as V (e) = L(e) and D(e) = string. Member of a collection: For example, a string variable defined by a specific member of a list of strings. More broadly, of the form v = collection hTi[x] where v is the string variable, collection contains objects of type T, and x denotes the index of the collection that defines v. In this case, a domain can be provided based on the type of object contained in the collection. This is represented as D(e) = T, and V (e) = collection[x] if the value is resolvable or V (e) otherwise. Conversion of a basic type to a string: For example, Integer.toString(). More broadly any function convert(X)! S where X is a basic type and S is a string type. This operation implies that the string should be a string representation of type X. This is represented as D(e) = X, and V (e) if X is defined by a variable or V (e) = L(e) otherwise. Append a basic type to a string: For example, a call to StringBuilder.append(int). More broadly, append(S,X) ! S0 where S is a string type, X is a basic type, and S0 is the string representation of the concatenation of the two arguments. In this case, the domain of the substring that was appended to S should be X. This is represented as D(eX) = X. V (eX) if X is defined by a variable or V (eX) = L(eX) otherwise. The subscripts denote the subset of transitions defined by the FSA of the string representation of X.

3) **Combining Information:** The final part of identifying invocation related information is to combine the information identified by computing the HTML pages and the domain and value tracking. The key insight for this step is that substrings of the HTML pages that syntactically define an invocation’s value will also have annotations from the DV functions. To identify this information, a custom parser is used to parse each of the computed HTML pages and recognize HTML tags while maintaining and recording any annotations. Example: Using the equations listed in Figure 3, the Out[exitNode] of servlet OrderStatus is equal to {{2, 5–12, 14–17, 22}, {2, 5–12, 19–22}. The analysis performs resolve on

each of the nodes in each of the sets that comprise Out[exitNode]. Nodes 2, 5, 7–12, 14, 16, 17, 19, 20, and 22 involve constants, so resolve returns the values of the constants and the domain information is any string (*). Nodes 6 and 15 originate from special string sources. The variable oid is defined by a function that returns strings and can be of any value (*), and the variable quant is an append of a basic type, so it is marked as type int. After computing the resolve function for each of the nodes, the final value of fragments[service] is comprised of two web pages, which differ only in that one traverses the true branch at line 13 and therefore includes an argument for quant and a different value for task The approach then parses the HTML to identify invocations. By examining the annotations associated with the substring that defines each argument's value, the value for arguments oid and quant are identified. The <select> tag has three different options that can each supply a different value. So three copies are made of each of the two web form based invocations. Each copy is assigned one of the three possible values for the shipto argument. The final result is the identification of six invocations originating from OrderStatus. Each tuple in the lists -the name, domain type, and values of the identified argument.

A. *Identify Interfaces*

This step of the proposed approach identifies interface information for each component of a web application. The proposed approach extends prior work in interface analysis [5] to also identify the HTTP request method for each interface. The specific mechanism for specifying HTTP request methods depends on the framework. In the Java Enterprise Edition (JEE) framework, the name of the entry method first accessed specifies its expected request method. For example, the doPost or doGet method indicates that the POST or GET request methods, respectively, will be used to decode arguments. The proposed approach builds a call graph of the component and marks all methods that are reachable from the specially named root methods as having the request method of the originating method. Example: ProcessOrder can accept two interfaces due to the branch taken at line 17: (1) {oid, task, shipto, other} and (2) {oid, task, shipto, other, quant}. From the implementation of ProcessOrder it is possible to infer domain information for some of the parameters. From this information, the first interface is determined to have an IDC of int(shipto).(shipto=1_shipto=2).task=”purchase”; and the second interface has an IDC of int(shipto).(shipto=1_shipto=2).task=”modify”.int(quant).

Unless otherwise specified, the domain of a parameter is a string. Lastly, by traversing the call graph of ProcessOrder all parameters (and therefore, all interfaces) are identified as having originated from a method that expects a POST request.

B. *Verify Invocations*

The third step of the approach checks each invocation to ensure that it matches an interface of the invocation's target. An invocation matches an interface if the following three conditions hold: (1) the request method of the invocation is equal to the request method of the interface; (2) the set of the interface's parameter names and the invocation's argument names are equal; and (3) the domains and values of the invocation satisfy an IDC of the interface. For the third condition, domain and value constraints are checked. The domain of an argument is considered to match the domain of a parameter if both are of the same type or if the value of the argument can be successfully converted to the corresponding parameter's domain type. For example, if the parameter domain constraint is Integer and the argument value is “5,” then the constraint would be satisfied. Example: Consider the interfaces identified and the invocations. Each of the six invocations is checked to see if it matches either of the two interfaces. Only invocation 2 represents a correct invocation and the rest will be identified as errors.

C. *Evaluation*

The evaluation measures the precision of the reported results. The proposed approach was implemented as a prototype tool, WAIVE+. The subjects used in the evaluation are four Java Enterprise Edition (JEE) based web applications: Bookstore, Daffodil, Filelister, and JWMA. These applications range in size from 8,600 to 29,000 lines of code. All of the applications are available as open source and are implemented using a mix of static HTML, JavaScript, Java servlets, and regular Java code. To address the research questions, WAIVE+ was run on the four applications. For each application the reported invocation errors were inspected. Table II shows the results of inspecting the reported invocations. Each invocation error was classified as either a confirmed error or a false positive. Invocations in both classifications were also further classified based on whether the error reported was due to a violation of one of the correctness properties, the invocation did not match an interface because of an incorrectly specified request method (R.M.), the argument names did not match the parameter names of any interface of the target (N.), and the value and domain information of an invocation did not match the interface domain constraint (IDC). The table also reports the total number of invocations identified for each application (# Invk.). As the results in Table II show, WAIVE+ identified 69 erroneous invocations and had 20 false positives. Prior approaches can only detect errors related to names, so the comparable total of errors for

WAIVE was 33 erroneous invocations and 19 false positives. These results indicate that the new domain information checks resulted in the discovery of 36 additional errors and 1 false positive. Overall, the results are very encouraging. The approach identified 36 new errors that had been previously undetectable while only producing one additional false positive.

III. CONCURRENT AJAX CRAWLING

The algorithm and its implementation for crawling AJAX, as just described, is sequential, depth-first, and single-threaded. Since we crawl the Web application dynamically, the crawling runtime is determined by the following factors.
(1) The speed at which the Web server responds to HTTP requests.
(2) Network latency.
(3) The crawler's internal processes (e.g., analyzing the DOM, firing events, updating the state machine).
(4) The speed of the browser in handling the events and request/response pairs, modifying the DOM, and rendering the user interface.
We have no influence on the first two factors and already have many optimization heuristics for the third step. Therefore, we focus on the last factor, the browser. Since the algorithm has to wait some considerable amount of time for the browser to finish its tasks after each event, our hypothesis is that we can decrease the total runtime by adopting concurrent crawling through multiple browsers.

A. Multi-threaded, Multi-Browser Crawling

The idea is to maintain a single state machine and split the original controller into a new controller and multiple crawling nodes. The controller is the single main thread monitoring the total crawl procedure. In this new setting, each crawling node is responsible for deriving its corresponding robot and browser instances to crawl a specific path. Compared with Figure 3, the new architecture is capable of having multiple crawler instances, running from a single controller. All the crawlers share the same state machine. The state machine makes sure every crawler can read and update the state machine in a synchronized way. This way, the operation of discovering new states can be executed in parallel.

B. Partition Function

To divide the work over the crawlers in a multi-threaded manner, a partition function must be designed. The performance of a concurrent approach is determined by the quality of its partition function [Garavel et al. 2001]. A partition function can be either static or dynamic. With a static partition function, the division of work is known in advance, before executing the code. When a dynamic partition function is used, the decision of which thread will execute a given node is made at runtime. Our algorithm infers the state-flow graph of an AJAX application dynamically and incrementally. Thus, due to this dynamic nature, we adopt a dynamic partition function. The task of our dynamic partition function is to distribute the work equally over all the participating crawling nodes. While crawling an AJAX application, we define work as bringing the browser back into a given state and exploring the first unexplored candidate state from that state. Our proposed partition function operates as follows. After the discovery of a new state, if there are still unexplored candidate clickables left in the previous state, that state is assigned to another thread for further exploration. The processor chosen will be the one with the least amount of work left. Visualizes our partition function for concurrent crawling of a simple Web application. In the Index state, two candidate clickables are detected that can lead: S 1 and S 11. The initial thread continues with the exploration of the states S 1, S 2, S 3, S 4, and finishes in S 5, in a depth-first manner. Simultaneously, a new thread is branched off to explore state S 11. This new thread (thread #2) first reloads the browser to Index and then goes into S 11. In state S 2 and S 6, this same branching mechanism happens, which results in a total of five threads. Now that the partition function has been introduced, the original sequential crawling algorithm (Algorithm 1) can be changed into a concurrent version.

We consider the following Ajax Complexity field equations defined over an open bounded piece of network and /or feature space $\Omega \subset R^d$. They describe the dynamics of the mean anycast of each of p node populations.

$$\begin{cases} (\frac{d}{dt}+l_i)V_i(t,r) = \sum_{j=1}^{p}\int_{\Omega} J_{ij}(r,\bar{r})S[(V_j(t-\tau_{ij}(r,\bar{r}),\bar{r})-h_{|j})]d\bar{r} \\ \qquad\qquad + I_i^{ext}(r,t), \qquad t \geq 0, 1 \leq i \leq p, \\ V_i(t,r) = \phi_i(t,r) \qquad t \in [-T,0] \end{cases} \tag{1}$$

We give an interpretation of the various parameters and functions that appear in (1), Ω is finite piece of nodes and/or feature space and is represented as an open bounded set of R^d . The vector r and $\bar{r}$ represent points in Ω. The function $S : R \rightarrow (0,1)$ is the normalized sigmoid function:

$$S(z) = \frac{1}{1+e^{-z}} \tag{2}$$

It describes the relation between the input rate v_i of population i as a function of the packets potential, for example, $V_i = v_i = S[\sigma_i(V_i - h_i)]$. We note V the $p-$ dimensional vector $(V_1,...,V_p)$. The p

function $\phi_i, i = 1,..., p,$ represent the initial conditions, see below. We note ϕ the $p-$dimensional vector $(\phi_1,...,\phi_p)$. The p function $I_i^{ext}, i = 1,..., p,$ represent external factors from other network areas. We note I^{ext} the $p-$dimensional vector $(I_1^{ext},...,I_p^{ext})$. The $p \times p$ matrix of functions $J = \{J_{ij}\}_{i,j=1,...,p}$ represents the connectivity between populations i and j, see below. The p real values $h_i, i = 1,..., p,$ determine the threshold of activity for each population, that is, the value of the nodes potential corresponding to 50% of the maximal activity. The p real positive values $\sigma_i, i = 1,..., p,$ determine the slopes of the sigmoids at the origin. Finally the p real positive values $l_i, i = 1,..., p,$ determine the speed at which each anycast node potential decreases exponentially toward its real value. We also introduce the function $S : R^p \to R^p,$ defined by $S(x) = [S(\sigma_1(x_1 - h_1)),..., S(\sigma_p - h_p))],$ and the diagonal $p \times p$ matrix $L_0 = diag(l_1,...,l_p)$. Is the intrinsic dynamics of the population given by the linear response of data transfer. $(\frac{d}{dt} + l_i)$ is replaced by $(\frac{d}{dt} + l_i)^2$ to use the alpha function response. We use $(\frac{d}{dt} + l_i)$ for simplicity although our analysis applies to more general intrinsic dynamics. For the sake, of generality, the propagation delays are not assumed to be identical for all populations, hence they are described by a matrix $\tau(r, \bar{r})$ whose element $\tau_{ij}(r, \bar{r})$ is the propagation delay between population j at $\bar{r}$ and population i at r. The reason for this assumption is that it is still unclear from anycast if propagation delays are independent of the populations. We assume for technical reasons that τ is continuous, that is $\tau \in C^0(\overline{\Omega}^2, R_+^{p \times p})$. Moreover packet data indicate that τ is not a symmetric function i.e., $\tau_{ij}(r, \bar{r}) \neq \tau_{ij}(\bar{r}, r)$, thus no assumption is made about this symmetry unless otherwise stated. In order to compute the righthand side of (1), we need to know the node potential factor V on interval $[-T, 0]$. The value of T is obtained by considering the maximal delay:

$$\tau_m = \max_{i,j(r,\bar{r} \in \overline{\Omega} \times \overline{\Omega})} \tau_{i,j}(r, \bar{r}) \qquad (3)$$

Hence we choose $T = \tau_m$

C. Mathematical Framework

A convenient functional setting for the non-delayed packet field equations is to use the space $F = L^2(\Omega, R^p)$ which is a Hilbert space endowed with the usual inner product:

$$\langle V, U \rangle_F = \sum_{i=1}^{p} \int_{\Omega} V_i(r) U_i(r) dr \qquad (1)$$

To give a meaning to (1), we defined the history space $C = C^0([-\tau_m, 0], F)$ with $\|\phi\| = \sup_{t \in [-\tau_m, 0]} \|\phi(t)\| F,$ which is the Banach phase space associated with equation (3). Using the notation $V_t(\theta) = V(t + \theta), \theta \in [-\tau_m, 0],$ we write (1) as

$$\begin{cases} \dot{V}(t) = -L_0 V(t) + L_1 S(V_t) + I^{ext}(t), \\ V_0 = \phi \in C, \end{cases} \qquad (2)$$

Where

$$\begin{cases} L_1 : C \to F, \\ \phi \to \int_{\Omega} J(., \bar{r}) \phi(\bar{r}, -\tau(., \bar{r})) d\bar{r} \end{cases}$$

Is the linear continuous operator satisfying $\|L_1\| \le \|J\|_{L^2(\Omega^2, R^{p \times p})}$. Notice that most of the papers on this subject assume Ω infinite, hence requiring $\tau_m = \infty$.

Proposition 1.0 If the following assumptions are satisfied.

1. $J \in L^2(\Omega^2, R^{p \times p}),$
2. The external current $I^{ext} \in C^0(R, F),$
3. $\tau \in C^0(\overline{\Omega^2}, R_+^{p \times p}), \sup_{\overline{\Omega^2}} \tau \le \tau_m.$

Then for any $\phi \in C,$ there exists a unique solution $V \in C^1([0, \infty), F) \cap C^0([-\tau_m, \infty, F)$ to (3)

Notice that this result gives existence on R_+, finite-time explosion is impossible for this delayed differential equation. Nevertheless, a particular solution could grow indefinitely, we now prove that this cannot happen.

D. Boundedness of Solutions

A valid model of neural networks should only feature bounded packet node potentials.

Theorem 1.0 All the trajectories are ultimately bounded by the same constant R if $I \equiv \max_{t\in R^+} \left\| I^{ext}(t) \right\|_F < \infty.$

Proof :Let us defined $f : R\times C \to R^+$ as

$$f(t,V_t) \overset{def}{=} \left\langle -L_0 V_t(0) + L_1 S(V_t) + I^{ext}(t), V(t) \right\rangle_F = \frac{1}{2}\frac{d\|V\|_F^2}{dt}$$

We note $l = \min_{i=1,\dots p} l_i$

$$f(t,V_t) \le -l\|V(t)\|_F^2 + (\sqrt{p|\Omega|}\|J\|_F + I)\|V(t)\|_F$$

Thus, if

$$\|V(t)\|_F \ge 2\frac{\sqrt{p|\Omega|}.\|J\|_F + I}{l} \overset{def}{=} R,\ f(t,V_t) \le -\frac{lR^2}{2} \overset{def}{=} -\delta < 0$$

Let us show that the open route of F of center 0 and radius R, B_R, is stable under the dynamics of equation. We know that $V(t)$ is defined for all $t \ge 0s$ and that $f < 0$ on ∂B_R, the boundary of B_R. We consider three cases for the initial condition V_0. If $\|V_0\|_C < R$ and set $T = \sup\{t \mid \forall s \in [0,t], V(s) \in \overline{B_R}\}$. Suppose that $T \in R$, then $V(T)$ is defined and belongs to $\overline{B_R}$, the closure of B_R, because $\overline{B_R}$ is closed, in effect to ∂B_R, we also have $\frac{d}{dt}\|V\|_F^2 |_{t=T} = f(T,V_T) \le -\delta < 0$ because $V(T) \in \partial B_R$. Thus we deduce that for $\varepsilon > 0$ and small enough, $V(T+\varepsilon) \in \overline{B_R}$ which contradicts the definition of T. Thus $T \notin R$ and $\overline{B_R}$ is stable.

Because f<0 on $\partial B_R, V(0) \in \partial B_R$ implies that $\forall t > 0, V(t) \in B_R$. Finally we consider the case $V(0) \in C\overline{B_R}$. Suppose that $\forall t > 0, V(t) \notin \overline{B_R}$, then $\forall t > 0, \frac{d}{dt}\|V\|_F^2 \le -2\delta,$ thus $\|V(t)\|_F$ is monotonically decreasing and reaches the value of R in finite time when $V(t)$ reaches ∂B_R. This contradicts our assumption. Thus $\exists T > 0 \mid V(T) \in B_R$.

Proposition 1.1 : Let s and t be measured simple functions on X. for $E\varepsilon M$, define

$$\phi(E) = \int_E s\,d\mu \qquad (1)$$

Then ϕ is a measure on M .

$$\int_X (s+t)d\mu = \int_X s\,d\mu + \int_X td\mu \qquad (2)$$

Proof : If s and if $E_1, E_2, \dots$ are disjoint members of M whose union is E, the countable additivity of μ shows that

$$\phi(E) = \sum_{i=1}^{n} \alpha_i \mu(A_i \cap E) = \sum_{i=1}^{n} \alpha_i \sum_{r=1}^{\infty} \mu(A_i \cap E_r)$$

$$= \sum_{r=1}^{\infty}\sum_{i=1}^{n} \alpha_i \mu(A_i \cap E_r) = \sum_{r=1}^{\infty} \phi(E_r)$$

Also, $\varphi(\phi) = 0,$ so that φ is not identically ∞.

Next, let s be as before, let $\beta_1, \dots, \beta_m$ be the distinct values of t,and let $B_j = \{x : t(x) = \beta_j\}$ If $E_{ij} = A_i \cap B_j,$ the $\int_{E_{ij}} (s+t)d\mu = (\alpha_i + \beta_j)\mu(E_{ij})$

and $\int_{E_{ij}} sd\mu + \int_{E_{ij}} td\mu = \alpha_i \mu(E_{ij}) + \beta_j \mu(E_{ij})$

Thus (2) holds with E_{ij} in place of X . Since X is the disjoint union of the sets E_{ij} $(1 \le i \le n, 1 \le j \le m),$ the first half of our proposition implies that (2) holds.

Theorem 1.1: If K is a compact set in the plane whose complement is connected, if f is a continuous complex function on K which is holomorphic in the interior of , and if $\varepsilon > 0$, then there exists a polynomial P such that $|f(z) = P(z)| < \varepsilon$ for all $z\varepsilon K$. If the interior of K is empty, then part of the hypothesis is vacuously satisfied, and the conclusion holds for every $f\varepsilon C(K)$. Note that K need to be connected.

Proof: By Tietze's theorem, f can be extended to a continuous function in the plane, with compact support. We fix one such extension and denote it again by f . For any $\delta > 0$, let $\omega(\delta)$ be the supremum of the numbers $|f(z_2) - f(z_1)|$ Where z_1 and z_2 are subject to the condition $|z_2 - z_1| \le \delta$. Since f is uniformly continous, we have

$\lim_{\delta \to 0} \omega(\delta) = 0$ (1) From now on, δ will be fixed. We shall prove that there is a polynomial P such that

$$|f(z) - P(z)| < 10{,}000\ \omega(\delta) \quad (z \varepsilon K) \qquad (2)$$

By (1), this proves the theorem. Our first objective is the construction of a function $\Phi \varepsilon C_c^{'}(R^2)$, such that for all z

$$|f(z) - \Phi(z)| \le \omega(\delta), \qquad (3)$$

$$|(\partial\Phi)(z)| < \frac{2\omega(\delta)}{\delta}, \qquad (4)$$

And

$$\Phi(z) = -\frac{1}{\pi}\iint_X \frac{(\partial\Phi)(\zeta)}{\zeta - z} d\zeta d\eta \qquad (\zeta = \xi + i\eta), \qquad (5)$$

Where X is the set of all points in the support of Φ whose distance from the complement of K does not δ. (Thus X contains no point which is "far within" K.) We construct Φ as the convolution of f with a smoothing function A. Put $a(r) = 0$ if $r > \delta$, put

$$a(r) = \frac{3}{\pi\delta^2}\left(1 - \frac{r^2}{\delta^2}\right)^2 \qquad (0 \le r \le \delta), \qquad (6)$$

And define

$$A(z) = a(|z|) \qquad (7)$$

For all complex z. It is clear that $A \varepsilon C_c^{'}(R^2)$. We claim that

$$\iint_{R^s} A = 1, \qquad (8)$$

$$\iint_{R^2} \partial A = 0, \qquad (9)$$

$$\iint_{R^3} |\partial A| = \frac{24}{15\delta} < \frac{2}{\delta}, \qquad (10)$$

The constants are so adjusted in (6) that (8) holds. (Compute the integral in polar coordinates), (9) holds simply because A has compact support. To compute (10), express ∂A in polar coordinates, and note that

$$\partial A / \partial\theta = 0,$$

$$\partial A / \partial r = -a^{'},$$

Now define

$$\Phi(z) = \iint_{R^2} f(z-\zeta) A d\xi d\eta = \iint_{R^2} A(z-\zeta) f(\zeta) d\xi d\eta \qquad (11)$$

Since f and A have compact support, so does Φ. Since

$$\Phi(z) - f(z) = \iint_{R^2} [f(z-\zeta) - f(z)] A(\xi) d\xi d\eta \quad (12)$$

And $A(\zeta) = 0$ if $|\zeta| > \delta$, (3) follows from (8). The difference quotients of A converge boundedly to the corresponding partial derivatives, since $A \varepsilon C_c^{'}(R^2)$. Hence the last expression in (11) may be differentiated under the integral sign, and we obtain

$$\begin{aligned}(\partial\Phi)(z) &= \iint_{R^2} (\overline{\partial A})(z-\zeta) f(\zeta) d\xi d\eta \\ &= \iint_{R^2} f(z-\zeta)(\partial A)(\zeta) d\xi d\eta \\ &= \iint_{R^2} [f(z-\zeta) - f(z)](\partial A)(\zeta) d\xi d\eta \qquad (13)\end{aligned}$$

The last equality depends on (9). Now (10) and (13) give (4). If we write (13) with Φ_x and Φ_y in place of $\partial\Phi$, we see that Φ has continuous partial derivatives, if we can show that $\partial\Phi = 0$ in G, where G is the set of all $z \varepsilon K$ whose distance from the complement of K exceeds δ. We shall do this by showing that

$$\Phi(z) = f(z) \qquad (z \varepsilon G); \qquad (14)$$

Note that $\partial f = 0$ in G, since f is holomorphic there. Now if $z \varepsilon G$, then $z - \zeta$ is in the interior of K for all ζ with $|\zeta| < \delta$. The mean value property for harmonic functions therefore gives, by the first equation in (11),

$$\begin{aligned}\Phi(z) &= \int_0^\delta a(r) r dr \int_0^{2\pi} f(z - re^{i\theta}) d\theta \\ &= 2\pi f(z) \int_0^\delta a(r) r dr = f(z) \iint_{R^2} A = f(z) \qquad (15)\end{aligned}$$

For all $z\ \varepsilon\ G$, we have now proved (3), (4), and (5) The definition of X shows that X is compact and that X can be covered by finitely many open discs $D_1, \ldots, D_n$, of radius 2δ, whose centers are not in K. Since $S^2 - K$ is connected, the center of each D_j can be joined to ∞ by a polygonal path in $S^2 - K$. It follows that each D_j contains a compact connected set E_j, of diameter at least 2δ, so that $S^2 - E_j$ is connected and so that $K \cap E_j = \phi$.

with $r = 2\delta$. There are functions $g_j \varepsilon H(S^2 - E_j)$ and constants b_j so that the inequalities.

$$|Q_j(\zeta, z)| < \frac{50}{\delta}, \qquad (16)$$

$$\left| Q_j(\zeta, z) - \frac{1}{z-\zeta} \right| < \frac{4,000\delta^2}{|z-\zeta|^2} \qquad (17)$$

Hold for $z \notin E_j$ and $\zeta \in D_j$, if

$$Q_j(\zeta, z) = g_j(z) + (\zeta - b_j) g_j^2(z) \qquad (18)$$

Let Ω be the complement of $E_1 \cup ... \cup E_n$. Then Ω is an open set which contains K. Put $X_1 = X \cap D_1$ and $X_j = (X \cap D_j) - (X_1 \cup ... \cup X_{j-1})$, for $2 \le j \le n$,

Define

$$R(\zeta, z) = Q_j(\zeta, z) \qquad (\zeta \varepsilon X_j, z \varepsilon \Omega) \qquad (19)$$

And

$$F(z) = \frac{1}{\pi} \iint_X (\partial\Phi)(\zeta) R(\zeta, z) d\zeta d\eta \qquad (20)$$

$$(z \ \varepsilon \ \Omega)$$

Since,

$$F(z) = \sum_{j=1} \frac{1}{\pi} \iint_{X_i} (\partial\Phi)(\zeta) Q_j(\zeta, z) d\xi d\eta, \qquad (21)$$

(18) shows that F is a finite linear combination of the functions g_j and g_j^2. Hence $F \varepsilon H(\Omega)$. By (20), (4), and (5) we have

$$|F(z) - \Phi(z)| < \frac{2\omega(\delta)}{\pi\delta} \iint_X |R(\zeta, z)$$

$$- \frac{1}{z-\zeta} | d\xi d\eta \quad (z \ \varepsilon \ \Omega) \quad (22)$$

Observe that the inequalities (16) and (17) are valid with R in place of Q_j if $\zeta \ \varepsilon \ X$ and $z \ \varepsilon \ \Omega$. Now fix $z \ \varepsilon \ \Omega$., put $\zeta = z + \rho e^{i\theta}$, and estimate the integrand in (22) by (16) if $\rho < 4\delta$, by (17) if $4\delta \le \rho$. The integral in (22) is then seen to be less than the sum of

$$2\pi \int_0^{4\delta} \left(\frac{50}{\delta} + \frac{1}{\rho} \right) \rho d\rho = 808\pi\delta \qquad (23)$$

And

$$2\pi \int_{4\delta}^{\infty} \frac{4,000\delta^2}{\rho^2} \rho d\rho = 2,000\pi\delta. \qquad (24)$$

Hence (22) yields

$$|F(z) - \Phi(z)| < 6,000\omega(\delta) \qquad (z \ \varepsilon \ \Omega) \quad (25)$$

Since $F \ \varepsilon \ H(\Omega)$, $K \subset \Omega$, and $S^2 - K$ is connected, Runge's theorem shows that F can be uniformly approximated on K by polynomials. Hence (3) and (25) show that (2) can be satisfied. This completes the proof.

Lemma 1.0 : Suppose $f \varepsilon C_c^{'}(R^2)$, the space of all continuously differentiable functions in the plane, with compact support. Put

$$\partial = \frac{1}{2}\left(\frac{\partial}{\partial x} + i \frac{\partial}{\partial y} \right) \qquad (1)$$

Then the following "Cauchy formula" holds:

$$f(z) = -\frac{1}{\pi} \iint_{R^2} \frac{(\partial f)(\zeta)}{\zeta - z} d\xi d\eta$$

$$(\zeta = \xi + i\eta) \qquad (2)$$

Proof: This may be deduced from Green's theorem. However, here is a simple direct proof:

Put $\varphi(r, \theta) = f(z + re^{i\theta})$, $r > 0$, θ real

If $\zeta = z + re^{i\theta}$, the chain rule gives

$$(\partial f)(\zeta) = \frac{1}{2} e^{i\theta} \left[\frac{\partial}{\partial r} + \frac{i}{r} \frac{\partial}{\partial \theta} \right] \varphi(r, \theta) \qquad (3)$$

The right side of (2) is therefore equal to the limit, as $\varepsilon \to 0$, of

$$-\frac{1}{2} \int_{\varepsilon}^{\infty} \int_0^{2\pi} \left(\frac{\partial\varphi}{\partial r} + \frac{i}{r} \frac{\partial\varphi}{\partial\theta} \right) d\theta dr \qquad (4)$$

For each $r > 0, \varphi$ is periodic in θ, with period 2π. The integral of $\partial\varphi / \partial\theta$ is therefore 0, and (4) becomes

$$-\frac{1}{2\pi} \int_0^{2\pi} d\theta \int_{\varepsilon}^{\infty} \frac{\partial\varphi}{\partial r} dr = \frac{1}{2\pi} \int_0^{2\pi} \varphi(\varepsilon, \theta) d\theta \qquad (5)$$

As $\varepsilon \to 0$, $\varphi(\varepsilon, \theta) \to f(z)$ uniformly. This gives (2)

If $X^{\alpha} \in a$ and $X^{\beta} \in k[X_1, ... X_n]$, then $X^{\alpha} X^{\beta} = X^{\alpha+\beta} \in a$, and so A satisfies the condition $(*)$. Conversely,

$$(\sum_{\alpha\in A} c_\alpha X^\alpha)(\sum_{\beta\in\mathbb{N}^n} d_\beta X^\beta) = \sum_{\alpha,\beta} c_\alpha d_\beta X^{\alpha+\beta} \qquad (\textit{finite sums})$$

and so if A satisfies $(*)$, then the subspace generated by the monomials $X^\alpha, \alpha \in a$, is an ideal. The proposition gives a classification of the monomial ideals in $k[X_1,...X_n]$: they are in one to one correspondence with the subsets A of $\mathbb{N}^n$ satisfying $(*)$. For example, the monomial ideals in $k[X]$ are exactly the ideals (X^n), $n \geq 1$, and the zero ideal (corresponding to the empty set A). We write $\langle X^\alpha \mid \alpha \in A\rangle$ for the ideal corresponding to A (subspace generated by the $X^\alpha, \alpha \in a$).

LEMMA 1.1. Let S be a subset of $\mathbb{N}^n$. The the ideal a generated by $X^\alpha, \alpha \in S$ is the monomial ideal corresponding to

$$A \overset{df}{=} \{\beta \in \mathbb{N}^n \mid \beta - \alpha \in \mathbb{N}^n, \quad some\ \alpha \in S\}$$

Thus, a monomial is in a if and only if it is divisible by one of the $X^\alpha, \alpha \in| S$

PROOF. Clearly A satisfies $(*)$, and $a \subset \langle X^\beta \mid \beta \in A\rangle$. Conversely, if $\beta \in A$, then $\beta - \alpha \in \mathbb{N}^n$ for some $\alpha \in S$, and $X^\beta = X^\alpha X^{\beta-\alpha} \in a$. The last statement follows from the fact that $X^\alpha \mid X^\beta \Leftrightarrow \beta - \alpha \in \mathbb{N}^n$. Let $A \subset \mathbb{N}^n$ satisfy $(*)$. From the geometry of A, it is clear that there is a finite set of elements $S = \{\alpha_1,...\alpha_s\}$ of A such that $A = \{\beta \in \mathbb{N}^n \mid \beta - \alpha_i \in \mathbb{N}^2,\ some\ \alpha_i \in S\}$ (The α_i's are the corners of A) Moreover, $a \overset{df}{=} \langle X^\alpha \mid \alpha \in A\rangle$ is generated by the monomials $X^{\alpha_i}, \alpha_i \in S$.

DEFINITION 1.0. For a nonzero ideal a in $k[X_1,...,X_n]$, we let $(LT(a))$ be the ideal generated by

$\{LT(f) \mid f \in a\}$

LEMMA 1.2 Let a be a nonzero ideal in $k[X_1,...,X_n]$; then $(LT(a))$ is a monomial ideal, and it equals $(LT(g_1),...,LT(g_n))$ for some $g_1,...,g_n \in a$.

PROOF. Since $(LT(a))$ can also be described as the ideal generated by the leading monomials (rather than the leading terms) of elements of a.

THEOREM 1.2. Every *ideal* a in $k[X_1,...,X_n]$ is finitely generated; more precisely, $a = (g_1,...,g_s)$ where $g_1,...,g_s$ are any elements of a whose leading terms generate $LT(a)$

PROOF. Let $f \in a$. On applying the division algorithm, we find $f = a_1 g_1 + ... + a_s g_s + r, \quad a_i, r \in k[X_1,...,X_n]$, where either $r = 0$ or no monomial occurring in it is divisible by any $LT(g_i)$. But $r = f - \sum a_i g_i \in a$, and therefore $LT(r) \in LT(a) = (LT(g_1),...,LT(g_s))$, implies that every monomial occurring in r is divisible by one in $LT(g_i)$. Thus $r = 0$, and $g \in (g_1,...,g_s)$.

DEFINITION 1.1. A finite subset $S = \{g_1, |..., g_s\}$ of an ideal a is a standard ($(Gr\ddot{o}bner)$ bases for a if $(LT(g_1),...,LT(g_s)) = LT(a)$. In other words, S is a standard basis if the leading term of every element of a is divisible by at least one of the leading terms of the g_i.

THEOREM 1.3 *The ring* $k[X_1,...,X_n]$ *is Noetherian i.e., every ideal is finitely generated.*

PROOF. For $n = 1$, $k[X]$ is a principal ideal domain, which means that every ideal is generated by single element. We shall prove the theorem by induction on n. Note that the obvious map $k[X_1,...X_{n-1}][X_n] \to k[X_1,...X_n]$ is an isomorphism – this simply says that every polynomial f in n variables $X_1,...X_n$ can be expressed uniquely as a polynomial in X_n with coefficients in $k[X_1,...,X_n]$:

$$f(X_1,...X_n) = a_0(X_1,...X_{n-1})X_n^r + ... + a_r(X_1,...X_{n-1})$$

Thus the next lemma will complete the proof

LEMMA 1.3. If A is Noetherian, then so also is $A[X]$

PROOF. For a polynomial

$$f(X) = a_0 X^r + a_1 X^{r-1} + \ldots + a_r, \quad a_i \in A, \quad a_0 \neq 0,$$

r is called the degree of f, and a_0 is its leading coefficient. We call 0 the leading coefficient of the polynomial 0. Let a be an ideal in $A[X]$. The leading coefficients of the polynomials in a form an ideal a' in A, and since A is Noetherian, a' will be finitely generated. Let $g_1, \ldots, g_m$ be elements of a whose leading coefficients generate a', and let r be the maximum degree of g_i. Now let $f \in a$, and suppose f has degree $s > r$, say, $f = aX^s + \ldots$ Then $a \in a'$, and so we can write

$$a = \sum b_i a_i, \qquad b_i \in A,$$

$$a_i = leading\ coefficient\ of\ g_i$$

Now

$f - \sum b_i g_i X^{s-r_i}, \quad r_i = \deg(g_i)$, has degree $< \deg(f)$. By continuing in this way, we find that $f \equiv f_t \quad \mathrm{mod}(g_1, \ldots g_m)$ With f_t a polynomial of degree $t < r$. For each $d < r$, let a_d be the subset of A consisting of 0 and the leading coefficients of all polynomials in a of degree d; it is again an ideal in A. Let $g_{d,1}, \ldots, g_{d,m_d}$ be polynomials of degree d whose leading coefficients generate a_d. Then the same argument as above shows that any polynomial f_d in a of degree d can be written $f_d \equiv f_{d-1} \quad \mathrm{mod}(g_{d,1}, \ldots g_{d,m_d})$ With f_{d-1} of degree $\leq d-1$. On applying this remark repeatedly we find that $f_t \in (g_{r-1,1}, \ldots g_{r-1,m_{r-1}}, \ldots g_{0,1}, \ldots g_{0,m_0})$ Hence

$$f_t \in (g_1, \ldots g_m g_{r-1,1}, \ldots g_{r-1,m_{r-1}}, \ldots, g_{0,1}, \ldots, g_{0,m_0})$$

and so the polynomials $g_1, \ldots, g_{0,m_0}$ generate a

One of the great successes of category theory in computer science has been the development of a "unified theory" of the constructions underlying denotational semantics. In the untyped λ-calculus, any term may appear in the function position of an application. This means that a model D of the λ-calculus must have the property that given a term t whose interpretation is $d \in D$, Also, the interpretation of a functional abstraction like $\lambda x . x$ is most conveniently defined as a function from $D\, to\, D$, which must then be regarded as an element of *D*. Let $\psi : [D \to D] \to D$ be the function that picks out elements of *D* to represent elements of $[D \to D]$ and $\phi : D \to [D \to D]$ be the function that maps elements of *D* to functions of *D*. Since $\psi(f)$ is intended to represent the function f as an element of *D*, it makes sense to require that $\phi(\psi(f)) = f$, that is, $\psi\, o\, \psi = id_{[D \to D]}$
Furthermore, we often want to view every element of *D* as representing some function from *D to D* and require that elements representing the same function be equal – that is

$$\psi(\varphi(d)) = d$$

$$or$$

$$\psi\, o\, \phi = id_D$$

The latter condition is called extensionality. These conditions together imply that $\phi\, and\, \psi$ are inverses--- that is, *D* is isomorphic to the space of functions from *D to D* that can be the interpretations of functional abstractions: $D \cong [D \to D]$ .Let us suppose we are working with the untyped $\lambda - calculus$, we need a solution ot the equation $D \cong A + [D \to D]$, where A is some predetermined domain containing interpretations for elements of *C*. Each element of *D* corresponds to either an element of *A* or an element of $[D \to D]$, with a tag. This equation can be solved by finding least fixed points of the function $F(X) = A + [X \to X]$ from domains to domains --- that is, finding domains *X* such that $X \cong A + [X \to X]$, and such that for any domain *Y* also satisfying this equation, there is an embedding of *X* to *Y* --- a pair of maps

$$\begin{tikzcd} X \arrow[r, shift left, "f"] & Y \arrow[l, shift left, "f^R"] \end{tikzcd}$$

Such that

$$f^R\, o\, f = id_X$$

$$f\, o\, f^R \subseteq id_Y$$

Where $f \subseteq g$ means that $f\ approximates\ g$ in some ordering representing their information content. The key shift of perspective from the domain-theoretic to the more general category-theoretic approach lies in considering *F* not as a function on domains, but as a *functor* on a category of domains. Instead of a least fixed point of the function, *F*.

Definition 1.3: Let ***K*** be a category and $F: K \to K$ as a functor. A fixed point of *F* is a pair (A,a), where A is a ***K-object*** and $a: F(A) \to A$ is an isomorphism. A prefixed point of F is a pair (A,a), where A is a ***K-object*** and a is any arrow from F(A) to A

Definition 1.4 : An $\omega - chain$ in a category ***K*** is a diagram of the following form:

$$\Delta = D_o \xrightarrow{f_o} D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots\cdots$$

Recall that a cocone μ of an $\omega - chain$ Δ is a *K*-object X and a collection of K *–arrows* $\{\mu_i : D_i \to X \mid i \geq 0\}$ such that $\mu_i = \mu_{i+1} o\, f_i$ for all $i \geq 0$. We sometimes write $\mu : \Delta \to X$ as a reminder of the arrangement of $\mu 's$ components Similarly, a colimit $\mu : \Delta \to X$ is a cocone with the property that if $\nu : \Delta \to X^{'}$ is also a cocone then there exists a unique mediating arrow $k : X \to X^{'}$ such that for all $i \geq 0,, \nu_i = k\, o\, \mu_i$. Colimits of $\omega - chains$ are sometimes referred to as $\omega - co\lim its$. Dually, an $\omega^{op} - chain$ in ***K*** is a diagram of the following form:

$$\Delta = D_o \xleftarrow{f_o} D_1 \xleftarrow{f_1} D_2 \xleftarrow{f_2} \cdots\cdots$$

A cone $\mu : X \to \Delta$ of an $\omega^{op} - chain$ Δ is a ***K***-object X and a collection of **K**-arrows $\{\mu_i : D_i \mid i \geq 0\}$ such that for all $i \geq 0,\ \mu_i = f_i\, o\, \mu_{i+1}$. An ω^{op} -limit of an $\omega^{op} - chain$ Δ is a cone $\mu : X \to \Delta$ with the property that if $\nu : X^{'} \to \Delta$ is also a cone, then there exists a unique mediating arrow $k : X^{'} \to X$ such that for all $i \geq 0, \mu_i\, o\, k = \nu_i$. We write $\perp_k$ (or just $\perp$) for the distinguish initial object of ***K,*** when it has one, and $\perp \to A$ for the unique arrow from $\perp$ to each ***K***-object A. It is also convenient to write $\Delta^{-} = D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \cdots\cdots$ to denote all of Δ except D_o and f_0. By analogy, μ^{-} is $\{\mu_i \mid i \geq 1\}$. For the images of Δ and μ under ***F*** we write

$$F(\Delta) = F(D_o) \xrightarrow{F(f_o)} F(D_1) \xrightarrow{F(f_1)} F(D_2) \xrightarrow{F(f_2)} \cdots\cdots$$

and $F(\mu) = \{F(\mu_i) \mid i \geq 0\}$

We write F^{i} for the ***i***-fold iterated composition of ***F*** – that is,

$$F^{o}(f) = f, F^{1}(f) = F(f), F^{2}(f) = F(F(f))$$

,etc. With these definitions we can state that every monitonic function on a complete lattice has a least fixed point:

Lemma 1.4. Let ***K*** be a category with initial object $\perp$ and let $F: K \to K$ be a functor. Define the $\omega - chain\, \Delta$ by

$$\Delta = \perp \xrightarrow{!\perp \to F(\perp)} F(\perp) \xrightarrow{F(!\perp \to F(\perp))} F^{2}(\perp) \xrightarrow{F^{2}(!\perp \to F(\perp))} \cdots\cdots\cdots$$

If both $\mu : \Delta \to D$ and $F(\mu) : F(\Delta) \to F(D)$ are colimits, then (D,d) is an intial F-algebra, where $d : F(D) \to D$ is the mediating arrow from $F(\mu)$ to the cocone μ^{-}

Theorem 1.4 Let a DAG G given in which each node is a random variable, and let a discrete conditional probability distribution of each node given values of its parents in G be specified. Then the product of these conditional distributions yields a joint probability distribution P of the variables, and (G,P) satisfies the Markov condition.

Proof. Order the nodes according to an ancestral ordering. Let $X_1, X_2, \ldots\ldots X_n$ be the resultant ordering. Next define.

$$P(x_1, x_2, \ldots. x_n) = P(x_n \mid pa_n) P(x_{n-1} \mid Pa_{n-1}) \ldots$$
$$.. P(x_2 \mid pa_2) P(x_1 \mid pa_1),$$

Where PA_i is the set of parents of X_i of in G and $P(x_i \mid pa_i)$ is the specified conditional probability distribution. First we show this does indeed yield a joint probability distribution. Clearly, $0 \leq P(x_1, x_2, \ldots x_n) \leq 1$ for all values of the variables. Therefore, to show we have a joint distribution, as the variables range through all their possible values, is equal to one. To that end, Specified conditional distributions are the conditional distributions they notationally represent in the joint distribution. Finally, we show the Markov condition is satisfied. To do this, we need show for $1 \leq k \leq n$ that
whenever

$$P(pa_k) \neq 0, if\ \ P(nd_k \mid pa_k) \neq 0$$
$$and\ \ P(x_k \mid pa_k) \neq 0$$
$$then\ P(x_k \mid nd_k, pa_k) = P(x_k \mid pa_k),$$

Where ND_k is the set of nondescendents of X_k of in G. Since $PA_k \subseteq ND_k$, we need only show

$P(x_k \mid nd_k) = P(x_k \mid pa_k)$. First for a given k, order the nodes so that all and only nondescendents of X_k precede X_k in the ordering. Note that this ordering depends on k, whereas the ordering in the first part of the proof does not. Clearly then

$$ND_k = \{X_1, X_2, ...X_{k-1}\}$$

Let

$$D_k = \{X_{k+1}, X_{k+2}, ...X_n\}$$

follows $\sum_{d_k}$

We define the m^{th} *cyclotomic field to be the field* $Q[x]/(\Phi_m(x))$ Where $\Phi_m(x)$ is the m^{th} cyclotomic polynomial. $Q[x]/(\Phi_m(x))$ $\Phi_m(x)$ *has degree* $\varphi(m)$ *over* Q *since* $\Phi_m(x)$ has degree $\varphi(m)$. *The roots of* $\Phi_m(x)$ are just the primitive m^{th} roots of unity, so the complex embeddings of $Q[x]/(\Phi_m(x))$ *are simply the* $\varphi(m)$ *maps*

$$\sigma_k : Q[x]/(\Phi_m(x)) \mapsto C,$$
$$1 \le k \prec m, (k,m) = 1, \quad where$$
$$\sigma_k(x) = \xi_m^k,$$

ξ_m being our fixed choice of primitive m^{th} root of unity. Note that $\xi_m^k \in Q(\xi_m)$ for every k; it follows that $Q(\xi_m) = Q(\xi_m^k)$ for all k relatively prime to m. In particular, the images of the σ_i coincide, so $Q[x]/(\Phi_m(x))$ *is Galois over* Q. *This means that we can write* $Q(\xi_m)$ *for* $Q[x]/(\Phi_m(x))$ *without much fear of ambiguity; we will do so from now on, the identification being* $\xi_m \mapsto x$. *One advantage of this is that one can easily talk about cyclotomic fields being extensions of one another,or intersections or compositums; all of these things take place considering them as subfield of* C. We now investigate some basic properties of cyclotomic fields. The first issue is whether or not they are all distinct; to determine this, we need to know which roots of unity lie in $Q(\xi_m)$ .Note, for example, that if m is odd, then $-\xi_m$ is a $2m^{th}$ root of unity. We will show that this is the only way in which one can obtain any non- m^{th} roots of unity.

LEMMA 1.5 If m divides n, then $Q(\xi_m)$ *is contained in* $Q(\xi_n)$

PROOF. Since $\xi^{n/m} = \xi_m$, *we have* $\xi_m \in Q(\xi_n)$, *so the result is clear*

LEMMA 1.6 If m and n are relatively prime, then

$$Q(\xi_m, \xi_n) = Q(\xi_{nm})$$

and

$$Q(\xi_m) \cap Q(\xi_n) = Q$$

(Recall the $Q(\xi_m, \xi_n)$ is the compositum of $Q(\xi_m)$ *and* $Q(\xi_n)$)

PROOF. One checks easily that $\xi_m \xi_n$ is a primitive mn^{th} root of unity, so that

$$Q(\xi_{mn}) \subseteq Q(\xi_m, \xi_n)$$
$$[Q(\xi_m, \xi_n) : Q] \le [Q(\xi_m) : Q][Q(\xi_n : Q]$$
$$= \varphi(m)\varphi(n) = \varphi(mn);$$

Since $[Q(\xi_{mn}) : Q] = \varphi(mn);$ this implies that $Q(\xi_m, \xi_n) = Q(\xi_{nm})$ We know that $Q(\xi_m, \xi_n)$ has degree $\varphi(mn)$ over Q, so we must have

$$[Q(\xi_m, \xi_n) : Q(\xi_m)] = \varphi(n)$$

and

$$[Q(\xi_m, \xi_n) : Q(\xi_m)] = \varphi(m)$$

$$[Q(\xi_m) : Q(\xi_m) \cap Q(\xi_n)] \ge \varphi(m)$$

And thus that $Q(\xi_m) \cap Q(\xi_n) = Q$

PROPOSITION 1.2 For any m and n

$$Q(\xi_m, \xi_n) = Q(\xi_{[m,n]})$$

And

$$Q(\xi_m) \cap Q(\xi_n) = Q(\xi_{(m,n)});$$

here $[m,n]$ and (m,n) denote the least common multiple and the greatest common divisor of m and n, respectively.

PROOF. Write $m = p_1^{e_1}......p_k^{e_k}$ *and* $p_1^{f_1}....p_k^{f_k}$ where the p_i are distinct primes. (We allow e_i *or* f_i to be zero)

$$Q(\xi_m)=Q(\xi_{p_1^{e_1}})Q(\xi_{p_2^{e_2}})...Q(\xi_{p_k^{e_k}})$$

and

$$Q(\xi_n)=Q(\xi_{p_1^{f_1}})Q(\xi_{p_2^{f_2}})...Q(\xi_{p_k^{f_k}})$$

Thus

$$\begin{aligned}Q(\xi_m,\xi_n)&=Q(\xi_{p_1^{e_1}})........Q(\xi_{p_2^{e_k}})Q(\xi_{p_1^{f_1}})...Q(\xi_{p_k^{f_k}})\\&=Q(\xi_{p_1^{e_1}})Q(\xi_{p_1^{f_1}})...Q(\xi_{p_k^{e_k}})Q(\xi_{p_k^{f_k}})\\&=Q(\xi_{p_1^{\max(e_1,f_1)}}).......Q(\xi_{p_1^{\max(e_k,f_k)}})\\&=Q(\xi_{p_1^{\max(e_1,f_1)}........p_1^{\max(e_k,f_k)}})\\&=Q(\xi_{[m,n]});\end{aligned}$$

An entirely similar computation shows that $Q(\xi_m)\cap Q(\xi_n)=Q(\xi_{(m,n)})$

Mutual information measures the information transferred when x_i is sent and y_i is received, and is defined as

$$I(x_i,y_i)=\log_2\frac{P(x_i/y_i)}{P(x_i)}\ bits \qquad (1)$$

In a noise-free channel, **each** y_i is uniquely connected to the corresponding x_i , and so they constitute an input –output pair (x_i,y_i) for which

$$P(x_i/y_j)=1\ and\ I(x_i,y_j)=\log_2\frac{1}{P(x_i)}\quad \text{bits;}$$

that is, the transferred information is equal to the self-information that corresponds to the input x_i In a very noisy channel, the output y_i and input x_i would be completely uncorrelated, and so $P(x_i/y_j)=P(x_i)$ and also $I(x_i,y_j)=0;$ that is, there is no transference of information. In general, a given channel will operate between these two extremes. The mutual information is defined between the input and the output of a given channel. An average of the calculation of the mutual information for all input-output pairs of a given channel is the average mutual information:

$$I(X,Y)=\sum_{i.j}P(x_i,y_j)I(x_i,y_j)=\sum_{i.j}P(x_i,y_j)\log_2\left[\frac{P(x_i/y_j)}{P(x_i)}\right]$$

bits per symbol . This calculation is done over the input and output alphabets. The average mutual information. The following expressions are useful for modifying the mutual information expression:

$$P(x_i,y_j)=P(x_i/y_j)P(y_j)=P(y_j/x_i)P(x_i)$$

$$P(y_j)=\sum_i P(y_j/x_i)P(x_i)$$

$$P(x_i)=\sum_i P(x_i/y_j)P(y_j)$$

Then

$$I(X,Y)=\sum_{i.j}P(x_i,y_j)$$

$$=\sum_{i.j}P(x_i,y_j)\log_2\left[\frac{1}{P(x_i)}\right]$$

$$-\sum_{i.j}P(x_i,y_j)\log_2\left[\frac{1}{P(x_i/y_j)}\right]$$

$$\sum_{i.j}P(x_i,y_j)\log_2\left[\frac{1}{P(x_i)}\right]$$

$$=\sum_i\left[P(x_i/y_j)P(y_j)\right]\log_2\frac{1}{P(x_i)}$$

$$\sum_i P(x_i)\log_2\frac{1}{P(x_i)}=H(X)$$

$$I(X,Y)=H(X)-H(X/Y)$$

Where $H(X/Y)=\sum_{i,j}P(x_i,y_j)\log_2\frac{1}{P(x_i/y_j)}$

is usually called the equivocation. In a sense, the equivocation can be seen as the information lost in the noisy channel, and is a function of the backward conditional probability. The observation of an output symbol y_j provides $H(X)-H(X/Y)$ bits of information. This difference is the mutual information of the channel. *Mutual Information: Properties* Since

$$P(x_i/y_j)P(y_j)=P(y_j/x_i)P(x_i)$$

The mutual information fits the condition

$$I(X,Y)=I(Y,X)$$

And by interchanging input and output it is also true that

$$I(X,Y)=H(Y)-H(Y/X)$$

Where

$$H(Y)=\sum_j P(y_j)\log_2\frac{1}{P(y_j)}$$

This last entropy is usually called the noise entropy. Thus, the information transferred through the channel is the difference between the output entropy and the noise entropy. Alternatively, it can be said that the channel mutual information is the difference between the number of bits needed for determining a given input symbol before knowing the corresponding output symbol, and the number of bits needed for determining a given input symbol after knowing the corresponding output symbol

$$I(X,Y)=H(X)-H(X/Y)$$

As the channel mutual information expression is a difference between two quantities, it seems that this parameter can adopt negative values. However, and is spite of the fact that for some $y_j, H(X/y_j)$ can be larger than $H(X)$, this is not possible for the average value calculated over all the outputs:

$$\sum_{i,j}P(x_i,y_j)\log_2\frac{P(x_i/y_j)}{P(x_i)}=\sum_{i,j}P(x_i,y_j)\log_2\frac{P(x_i,y_j)}{P(x_i)P(y_j)}$$

Then

$$-I(X,Y)=\sum_{i,j}P(x_i,y_j)\frac{P(x_i)P(y_j)}{P(x_i,y_j)}\leq 0$$

Because this expression is of the form

$$\sum_{i=1}^{M}P_i\log_2(\frac{Q_i}{P_i})\leq 0$$

The above expression can be applied due to the factor $P(x_i)P(y_j)$, which is the product of two probabilities, so that it behaves as the quantity Q_i, which in this expression is a dummy variable that fits the condition $\sum_i Q_i\leq 1$. It can be concluded that the average mutual information is a non-negative number. It can also be equal to zero, when the input and the output are independent of each other. A related entropy called the joint entropy is defined as

$$H(X,Y)=\sum_{i,j}P(x_i,y_j)\log_2\frac{1}{P(x_i,y_j)}$$
$$=\sum_{i,j}P(x_i,y_j)\log_2\frac{P(x_i)P(y_j)}{P(x_i,y_j)}$$
$$+\sum_{i,j}P(x_i,y_j)\log_2\frac{1}{P(x_i)P(y_j)}$$

Theorem 1.5: Entropies of the binary erasure channel (BEC) The BEC is defined with an alphabet of two inputs and three outputs, with symbol probabilities.

$P(x_1)=\alpha$ *and* $P(x_2)=1-\alpha,$ and transition probabilities

$$P(y_3/x_2)=1-p\ \ and\ \ P(y_2/x_1)=0,$$
$$and\ \ P(y_3/x_1)=0$$
$$and\ \ P(y_1/x_2)=p$$
$$and\ \ P(y_3/x_2)=1-p$$

Lemma 1.7. Given an arbitrary restricted time-discrete, amplitude-continuous channel whose restrictions are determined by sets F_n and whose density functions exhibit no dependence on the state s, let n be a fixed positive integer, and $p(x)$ an arbitrary probability density function on Euclidean *n*-space. $p(y\,|\,x)$ for the density $p_n(y_1,...,y_n\,|\,x_1,...x_n)$ and $F\ \ for\ F_n$. For any real number a, let

$$A=\left\{(x,y):\log\frac{p(y\,|\,x)}{p(y)}>a\right\}\qquad(1)$$

Then for each positive integer u, there is a code (u,n,λ) such that

$$\lambda\leq ue^{-a}+P\{(X,Y)\notin A\}+P\{X\notin F\}\qquad(2)$$

Where

$$P\{(X,Y)\in A\}=\int_A...\int p(x,y)dxdy,\qquad p(x,y)=p(x)p(y\,|\,x)$$

and

$$P\{X\in F\}=\int_F...\int p(x)dx$$

Proof: A sequence $x^{(1)}\in F$ *such that*

$$P\{Y\in A_{x^1}\,|\,X=x^{(1)}\}\geq 1-\varepsilon$$

$$where\ A_x=\{y:(x,y)\varepsilon A\};$$

Choose the decoding set B_1 to be $A_{x^{(1)}}$. Having chosen $x^{(1)},........,x^{(k-1)}$ and $B_1,...,B_{k-1}$, select $x^k\in F$ such that

$$P\left\{Y\in A_{x^{(k)}}-\bigcup_{i=1}^{k-1}B_i\,|\,X=x^{(k)}\right\}\geq 1-\varepsilon;$$

Set $B_k=A_{x^{(k)}}-\bigcup_{i=1}^{k-1}B_i$, If the process does not terminate in a finite number of steps, then the sequences $x^{(i)}$ and decoding sets $B_i,\ i=1,2,...,u,$ form the desired code. Thus assume that the process terminates after t steps. (Conceivably $t=0$). We will show $t\geq u$ by showing that $\varepsilon\leq te^{-a}+P\{(X,Y)\notin A\}+P\{X\notin F\}$. We proceed as follows.

Let

$B = \bigcup_{j=1}^{t} B_j.$ *(If* $t = 0,$ *take* $B = \phi$). *Then*

$$P\{(X,Y) \in A\} = \int_{(x,y)\in A} p(x,y)dx\,dy$$

$$= \int_x p(x) \int_{y\in A_x} p(y\,|\,x)dy\,dx$$

$$= \int_x p(x) \int_{y\in B\cap A_x} p(y\,|\,x)dy\,dx + \int_x p(x)$$

E. Algorithms

Ideals. Let A be a ring. Recall that an *ideal a* in A is a subset such that a is subgroup of A regarded as a group under addition;

$a \in a, r \in A \Rightarrow ra \in A$

The ideal generated by a subset S of A is the intersection of all ideals A containing a ----- it is easy to verify that this is in fact an ideal, and that it consist of all finite sums of the form $\sum r_i s_i$ with $r_i \in A, s_i \in S$. When $S = \{s_1,.....,s_m\}$, we shall write $(s_1,.....,s_m)$ for the ideal it generates.

Let a and b be ideals in A. The set $\{a+b\,|\,a \in a, b \in b\}$ is an ideal, denoted by $a+b$. The ideal generated by $\{ab\,|\,a \in a, b \in b\}$ is denoted by ab. Note that $ab \subset a \cap b$. Clearly ab consists of all finite sums $\sum a_i b_i$ with $a_i \in a$ and $b_i \in b$, and if $a = (a_1,...,a_m)$ and $b = (b_1,...,b_n)$, then $ab = (a_1b_1,...,a_ib_j,...,a_mb_n)$ .Let a be an ideal of A. The set of cosets of a in A forms a ring A/a, and $a \mapsto a+a$ is a homomorphism $\phi : A \mapsto A/a$. The map $b \mapsto \phi^{-1}(b)$ is a one to one correspondence between the ideals of A/a and the ideals of A containing a An ideal p if *prime* if $p \neq A$ and $ab \in p \Rightarrow a \in p$ or $b \in p$. Thus p is prime if and only if A/p is nonzero and has the property that $ab = 0, \quad b \neq 0 \Rightarrow a = 0,$ i.e., A/p is an integral domain. An ideal m is *maximal* if $m \neq\!| A$ and there does not exist an ideal n contained strictly between m and A. Thus m is maximal if and only if A/m has no proper nonzero ideals, and so is a field. Note that m maximal $\Rightarrow$ m prime. The ideals of $A \times B$ are all of the form $a \times b$, with a and b ideals in A and B. To see this, note that if c is an ideal in $A \times B$ and $(a,b) \in c$, then $(a,0) = (a,b)(1,0) \in c$ and $(0,b) = (a,b)(0,1) \in c$. This shows that $c = a \times b$ with

$a = \{a\,|\,(a,b) \in c \ \ some \ \ b \in b\}$

and

$b = \{b\,|\,(a,b) \in c \ \ some \ \ a \in a\}$

Let A be a ring. An A-algebra is a ring B together with a homomorphism $i_B : A \to B$. A *homomorphism of* A -algebra $B \to C$ is a homomorphism of rings $\varphi : B \to C$ such that $\varphi(i_B(a)) = i_C(a)$ for all $a \in A$. An A-algebra B is said to be *finitely generated* (or of *finite-type* over A) if there exist elements $x_1,...,x_n \in B$ such that every element of B can be expressed as a polynomial in the x_i with coefficients in $i(A)$, i.e., such that the homomorphism $A[X_1,...,X_n] \to B$ sending X_i to x_i is surjective. A ring homomorphism $A \to B$ is *finite,* and B is finitely generated as an A-module. Let k be a field, and let A be a k -algebra. If $1 \neq 0$ in A, then the map $k \to A$ is injective, we can identify k with its image, i.e., we can regard k as a subring of A. If 1=0 in a ring R, the R is the zero ring, i.e., $R = \{0\}$.

Polynomial rings. Let k be a field. A *monomial* in $X_1,...,X_n$ is an expression of the form $X_1^{a_1}...X_n^{a_n},\quad a_j \in N$. The *total degree* of the monomial is $\sum a_i$. We sometimes abbreviate it by $X^\alpha,\ \alpha = (a_1,...,a_n) \in \Box^n$. The elements of the polynomial ring $k[X_1,...,X_n]$ are finite sums $\sum c_{a_1....a_n} X_1^{a_1}...X_n^{a_n}, \quad c_{a_1....a_n} \in k, \quad a_j \in \Box$

With the obvious notions of equality, addition and multiplication. Thus the monomials from basis for $k[X_1,...,X_n]$ as a k -vector space. The ring $k[X_1,...,X_n]$ is an integral domain, and the only units in it are the nonzero constant polynomials. A polynomial $f(X_1,...,X_n)$ is *irreducible* if it is nonconstant and has only the obvious factorizations, i.e., $f = gh \Rightarrow g$ or h is constant. **Division in** $k[X]$. The division algorithm allows us to divide a nonzero polynomial into another: let f and g be polynomials in $k[X]$ with $g \neq 0$; then there exist unique polynomials $q, r \in k[X]$ such that

$f = qg + r$ with either $r = 0$ or $\deg r < \deg g$. Moreover, there is an algorithm for deciding whether $f \in (g)$, namely, find r and check whether it is zero. Moreover, the Euclidean algorithm allows to pass from finite set of generators for an ideal in $k[X]$ to a single generator by successively replacing each pair of generators with their greatest common divisor.

(Pure) ***lexicographic*** *ordering (lex).* Here monomials are ordered by lexicographic(dictionary) order. More precisely, let $\alpha = (a_1,...a_n)$ and $\beta = (b_1,...b_n)$ be two elements of $\mathbb{N}^n$; then $\alpha > \beta$ *and* $X^\alpha > X^\beta$ (lexicographic ordering) if, in the vector difference $\alpha - \beta \in \mathbb{Z}$, the left most nonzero entry is positive. For example,

$XY^2 > Y^3Z^4; \quad X^3Y^2Z^4 > X^3Y^2Z$. Note that this isn't quite how the dictionary would order them: it would put *XXXYYZZZZ* after *XXXYYZ*. *Graded reverse lexicographic order (grevlex).* Here monomials are ordered by total degree, with ties broken by reverse lexicographic ordering. Thus, $\alpha > \beta$ if $\sum a_i > \sum b_i$, or $\sum a_i = \sum b_i$ and in $\alpha - \beta$ the right most nonzero entry is negative. For example:

$X^4Y^4Z^7 > X^5Y^5Z^4$ *(total degree greater)*

$XY^5Z^2 > X^4YZ^3, \quad X^5YZ > X^4YZ^2$.

Orderings on $k[X_1,...X_n]$ **.** Fix an ordering on the monomials in $k[X_1,...X_n]$. Then we can write an element f of $k[X_1,...X_n]$ in a canonical fashion, by re-ordering its elements in decreasing order. For example, we would write

$$f = 4XY^2Z + 4Z^2 - 5X^3 + 7X^2Z^2$$

as

$$f = -5X^3 + 7X^2Z^2 + 4XY^2Z + 4Z^2 \quad (lex)$$

or

$$f = 4XY^2Z + 7X^2Z^2 - 5X^3 + 4Z^2 \quad (grevlex)$$

Let $\sum a_\alpha X^\alpha \in k[X_1,...,X_n]$, in decreasing order:

$$f = a_{\alpha_0} X^{\alpha_0} +_{\alpha_1} X^{\alpha_1} + ..., \qquad \alpha_0 > \alpha_1 > ..., \quad \alpha_0 \neq 0$$

Then we define.

- The *multidegree* of f to be multdeg(f)= α_0;
- The *leading coefficient of* f to be *LC(* f *)=* a_{α_0};
- The *leading monomial of* f to be *LM(* f *)* = X^{α_0};
- The *leading term of* f to be *LT(* f *)* = $a_{\alpha_0} X^{\alpha_0}$

For the polynomial $f = 4XY^2Z + ...,$ the multidegree is (1,2,1), the leading coefficient is 4, the leading monomial is XY^2Z, and the leading term is $4XY^2Z$. **The division algorithm in** $k[X_1,...X_n]$**.** Fix a monomial ordering in $\mathbb{N}^2$. Suppose given a polynomial f and an ordered set $(g_1,...g_s)$ of polynomials; the division algorithm then constructs polynomials $a_1,...a_s$ and r such that $f = a_1g_1 + ... + a_sg_s + r$ Where either $r = 0$ or no monomial in r is divisible by any of $LT(g_1),...,LT(g_s)$ **Step 1:** If $LT(g_1) | LT(f)$, divide g_1 into f to get

$$f = a_1g_1 + h, \qquad a_1 = \frac{LT(f)}{LT(g_1)} \in k[X_1,...,X_n]$$

If $LT(g_1) | LT(h)$, repeat the process until $f = a_1g_1 + f_1$ (different a_1) with $LT(f_1)$ not divisible by $LT(g_1)$. Now divide g_2 into f_1, and so on, until $f = a_1g_1 + ... + a_sg_s + r_1$ With $LT(r_1)$ not divisible by any $LT(g_1),...LT(g_s)$ **Step 2:** Rewrite $r_1 = LT(r_1) + r_2$, and repeat Step 1 with r_2 for f:

$f = a_1g_1 + ... + a_sg_s + LT(r_1) + r_3$ (different a_i's) **Monomial ideals.** In general, an ideal a will contain a polynomial without containing the individual terms of the polynomial; for example, the ideal $a = (Y^2 - X^3)$ contains $Y^2 - X^3$ but not Y^2 or X^3.

DEFINITION 1.5. An ideal a is *monomial* if $\sum c_\alpha X^\alpha \in a \Rightarrow X^\alpha \in a$ all α with $c_\alpha \neq 0$.

PROPOSITION 1.3. Let a be a *monomial ideal*, and let $A = \{\alpha \mid X^\alpha \in a\}$. Then A satisfies the condition $\alpha \in A, \ \beta \in \mathbb{N}^n \Rightarrow \alpha + \beta \in$ (*)

And a is the k -subspace of $k\left[X_1,\dots,X_n\right]$ generated by the $X^{\alpha}, \alpha \in A$. Conversely, of A is a subset of $\square^n$ satisfying $(*)$, then the k-subspace a of $k\left[X_1,\dots,X_n\right]$ generated by $\left\{X^{\alpha} \mid \alpha \in A\right\}$ is a monomial ideal.

PROOF. It is clear from its definition that a monomial ideal a is the k -subspace of $k\left[X_1,\dots,X_n\right]$
generated by the set of monomials it contains. If $X^{\alpha} \in a$ and $X^{\beta} \in k\left[X_1,\dots,X_n\right]$.

If a permutation is chosen uniformly and at random from the $n!$ possible permutations in S_n, then the counts $C_j^{(n)}$ of cycles of length j are dependent random variables. The joint distribution of $C^{(n)} = (C_1^{(n)},\dots,C_n^{(n)})$ follows from Cauchy's formula, and is given by

$$P[C^{(n)} = c] = \frac{1}{n!}N(n,c) = 1\left\{\sum_{j=1}^{n} jc_j = n\right\}\prod_{j=1}^{n}\left(\frac{1}{j}\right)^{c_j}\frac{1}{c_j!},$$

for $c \in \square_+^n$.

Lemma1.7 For nonnegative integers $m_{1,\dots,}m_n$,

$$E\left(\prod_{j=1}^{n}(C_j^{(n)})^{[m_j]}\right) = \left(\prod_{j=1}^{n}\left(\frac{1}{j}\right)^{m_j}\right)1\left\{\sum_{j=1}^{n} jm_j \le n\right\} \qquad (1.4)$$

Proof. This can be established directly by exploiting cancellation of the form $c_j^{[m_j]}/c_j^{!} = 1/(c_j - m_j)!$ when $c_j \ge m_j$, which occurs between the ingredients in Cauchy's formula and the falling factorials in the moments. Write $m = \sum jm_j$. Then, with the first sum indexed by $c = (c_1,\dots c_n) \in \square_+^n$ and the last sum indexed by $d = (d_1,\dots,d_n) \in \square_+^n$ via the correspondence $d_j = c_j - m_j$, we have

$$E\left(\prod_{j=1}^{n}(C_j^{(n)})^{[m_j]}\right) = \sum_{c} P[C^{(n)} = c]\prod_{j=1}^{n}(c_j)^{[m_j]}$$

$$= \sum_{c:c_j \ge m_j \ \ for\ all\ \ j} 1\left\{\sum_{j=1}^{n} jc_j = n\right\}\prod_{j=1}^{n}\frac{(c_j)^{[m_j]}}{j^{c_j}c_j!}$$

$$= \prod_{j=1}^{n}\frac{1}{j^{m_j}}\sum_{d}1\left\{\sum_{j=1}^{n} jd_j = n-m\right\}\prod_{j=1}^{n}\frac{1}{j^{d_j}(d_j)!}$$

This last sum simplifies to the indicator $1(m \le n)$, corresponding to the fact that if $n - m \ge 0$, then $d_j = 0$ for $j > n-m$, and a random permutation in S_{n-m} must have some cycle structure $(d_1,\dots,d_{n-m})$. The moments of $C_j^{(n)}$ follow immediately as

$$E(C_j^{(n)})^{[r]} = j^{-r}1\{jr \le n\} \qquad (1.2)$$

We note for future reference that (1.4) can also be written in the form

$$E\left(\prod_{j=1}^{n}(C_j^{(n)})^{[m_j]}\right) = E\left(\prod_{j=1}^{n}Z_j^{[m_j]}\right)1\left\{\sum_{j=1}^{n} jm_j \le n\right\}, \qquad (1.3)$$

(1.1)

Where the Z_j are independent Poisson-distribution random variables that satisfy $E(Z_j) = 1/j$

The marginal distribution of cycle counts provides a formula for the joint distribution of the cycle counts C_j^n, we find the distribution of C_j^n using a combinatorial approach combined with the inclusion-exclusion formula.

Lemma 1.8. For $1 \le j \le n$,

$$P[C_j^{(n)} = k] = \frac{j^{-k}}{k!}\sum_{l=0}^{[n/j]-k}(-1)^l\frac{j^{-l}}{l!} \qquad (1.1)$$

Proof. Consider the set I of all possible cycles of length j, formed with elements chosen from $\{1,2,\dots n\}$, so that $|I| = n^{[j]/j}$. For each $\alpha \in I$, consider the "property" G_{α} of having α; that is, G_{α} is the set of permutations $\pi \in S_n$ such that α is one of the cycles of π. We then have $|G_{\alpha}| = (n-j)!$, since the elements of $\{1,2,\dots,n\}$ not in α must be permuted among themselves. To use the inclusion-exclusion formula we need to calculate the term S_r, which is the sum of the probabilities of the r -fold intersection of properties, summing over all sets of r distinct properties. There are two cases to consider. If the r properties are indexed by r cycles having no elements in common, then the intersection specifies how rj elements are moved by the permutation, and there are $(n-rj)!1(rj \le n)$ permutations in the intersection. There are $n^{[rj]}/(j^r r!)$ such intersections. For the other case, some two distinct properties name some element in common, so no permutation can have both these properties, and the r-fold intersection is empty. Thus

$$S_r = (n-rj)!1(rj \le n)$$
$$\times \frac{n^{[rj]}}{j^r r!}\frac{1}{n!} = 1(rj \le n)\frac{1}{j^r r!}$$

Finally, the inclusion-exclusion series for the number of permutations having exactly k properties is

$$\sum_{l \ge 0}(-1)^l \binom{k+l}{l} S_{k+l},$$

Which simplifies to (1.1) Returning to the original hat-check problem, we substitute j=1 in (1.1) to obtain the distribution of the number of fixed points of a random permutation. For $k = 0,1,\ldots,n,$

$$P[C_1^{(n)} = k] = \frac{1}{k!}\sum_{l=0}^{n-k}(-1)^l \frac{1}{l!}, \qquad (1.2)$$

and the moments of $C_1^{(n)}$ follow from (1.2) with $j=1.$ In particular, for $n \ge 2,$ the mean and variance of $C_1^{(n)}$ are both equal to 1. The joint distribution of $(C_1^{(n)},\ldots,C_b^{(n)})$ for any $1 \le b \le n$ has an expression similar to (1.7); this too can be derived by inclusion-exclusion. For any $c = (c_1,\ldots,c_b) \in \mathbb{N}_+^b$ with $m = \sum ic_i,$

$$P[(C_1^{(n)},\ldots,C_b^{(n)}) = c]$$
$$= \left\{\prod_{i=1}^{b}\left(\frac{1}{i}\right)^{c_i}\frac{1}{c_i!}\right\}\sum_{\substack{l \ge 0 \text{ with} \\ \sum il_i \le n-m}}(-1)^{l_1+\ldots+l_b}\prod_{i=1}^{b}\left(\frac{1}{i}\right)^{l_i}\frac{1}{l_i!}$$

The joint moments of the first b counts $C_1^{(n)},\ldots,C_b^{(n)}$ can be obtained directly from (1.2) and (1.3) by setting $m_{b+1} = \ldots = m_n = 0$

The limit distribution of cycle counts

It follows immediately from Lemma 1.2 that for each fixed $j,$ as $n \to \infty,$

$$P[C_j^{(n)} = k] \to \frac{j^{-k}}{k!}e^{-1/j}, \quad k = 0,1,2,\ldots,$$

So that $C_j^{(n)}$ converges in distribution to a random variable Z_j having a Poisson distribution with mean $1/j;$ we use the notation $C_j^{(n)} \to_d Z_j$ where $Z_j \sim P_o(1/j)$ to describe this. Infact, the limit random variables are independent.

Theorem 1.6 The process of cycle counts converges in distribution to a Poisson process of $\mathbb{N}$ with intensity j^{-1}. That is, as $n \to \infty,$

$$(C_1^{(n)}, C_2^{(n)},\ldots) \to_d (Z_1, Z_2,\ldots) \qquad (1.1)$$

Where the $Z_j, j = 1,2,\ldots,$ are independent Poisson-distributed random variables with $E(Z_j) = \frac{1}{j}$

Proof. To establish the converges in distribution one shows that for each fixed $b \ge 1,$ as $n \to \infty,$

$$P[(C_1^{(n)},\ldots,C_b^{(n)}) = c] \to P[(Z_1,\ldots,Z_b) = c]$$

Error rates

The proof of Theorem says nothing about the rate of convergence. Elementary analysis can be used to estimate this rate when $b = 1$. Using properties of alternating series with decreasing terms, for $k = 0,1,\ldots,n,$

$$\frac{1}{k!}\left(\frac{1}{(n-k+1)!} - \frac{1}{(n-k+2)!}\right) \le \left|P[C_1^{(n)} = k] - P[Z_1 = k]\right|$$
$$\le \frac{1}{k!(n-k+1)!}$$

It follows that

$$\frac{2^{n+1}}{(n+1)!}\frac{n}{n+2} \le \sum_{k=0}^{n}\left|P[C_1^{(n)} = k] - P[Z_1 = k]\right| \le \frac{2^{n+1}-1}{(n+1)!} \qquad (1.11)$$

Since

(1.3)
$$P[Z_1 > n] = \frac{e^{-1}}{(n+1)!}\left(1 + \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} + \ldots\right) < \frac{1}{(n+1)!},$$

We see from (1.11) that the total variation distance between the distribution $L(C_1^{(n)})$ of $C_1^{(n)}$ and the distribution $L(Z_1)$ of Z_1

Establish the asymptotics of $\mathrm{P}\left[A_n(C^{(n)})\right]$ under conditions (A_0) and $(B_{01}),$ where

$$A_n(C^{(n)}) = \bigcap_{1 \le i \le n}\ \bigcap_{r_i'+1 \le j \le r_i}\left\{C_{ij}^{(n)} = 0\right\},$$

and $\zeta_i = (r_i'/r_{id}) - 1 = O(i^{-g'})$ as $i \to \infty,$ for some $g' > 0.$ We start with the expression

$$P[A_n(C^{(n)})] = \frac{P[T_{0m}(Z') = n]}{P[T_{0m}(Z) = n]}$$

$$\prod_{\substack{1\le i\le n \\ r_i'+1\le j\le r_i}} \left\{1 - \frac{\theta}{ir_i}(1+E_{i0})\right\} \qquad (1.1)$$

$$P[T_{0n}(Z') = n]$$

$$= \frac{\theta d}{n}\exp\left\{\sum_{i\ge1}[\log(1+i^{-1}\theta d) - i^{-1}\theta d]\right\}$$

$$\left\{1 + O(n^{-1}\varphi'_{\{1,2,7\}}(n))\right\} \qquad (1.2)$$

and

$$P[T_{0n}(Z') = n]$$

$$= \frac{\theta d}{n}\exp\left\{\sum_{i\ge1}[\log(1+i^{-1}\theta d) - i^{-1}\theta d]\right\}$$

$$\left\{1 + O(n^{-1}\varphi_{\{1,2,7\}}(n))\right\} \qquad (1.3)$$

Where $\varphi'_{\{1,2,7\}}(n)$ refers to the quantity derived from Z'. It thus follows that $P[A_n(C^{(n)})] \sim Kn^{-\theta(1-d)}$ for a constant K, depending on Z and the r_i' and computable explicitly from (1.1) – (1.3), if Conditions (A_0) and (B_{01}) are satisfied and if $\zeta_i^* = O(i^{-g'})$ from some $g' > 0$, since, under these circumstances, both $n^{-1}\varphi'_{\{1,2,7\}}(n)$ and $n^{-1}\varphi_{\{1,2,7\}}(n)$ tend to zero as $n \to \infty$. In particular, for polynomials and square free polynomials, the relative error in this asymptotic approximation is of order n^{-1} if $g' > 1$.

For $0 \le b \le n/8$ and $n \ge n_0$, with n_0

$$d_{TV}(L(C[1,b]), L(Z[1,b]))$$

$$\le d_{TV}(L(\tilde{C}[1,b]), L(\tilde{Z}[1,b]))$$

$$\le \varepsilon_{\{7,7\}}(n,b),$$

Where $\varepsilon_{\{7,7\}}(n,b) = O(b/n)$ under Conditions $(A_0),(D_1)$ and (B_{11}) Since, by the Conditioning Relation,

$$L(\tilde{C}[1,b] \mid T_{0b}(C) = l) = L(\tilde{Z}[1,b] \mid T_{0b}(Z) = l),$$

It follows by direct calculation that

$$d_{TV}(L(\tilde{C}[1,b]), L(\tilde{Z}[1,b]))$$

$$= d_{TV}(L(T_{0b}(C)), L(T_{0b}(Z)))$$

$$= \max_A \sum_{r\in A} P[T_{0b}(Z) = r]$$

$$\left\{1 - \frac{P[T_{bn}(Z) = n-r]}{P[T_{0n}(Z) = n]}\right\} \qquad (1.4)$$

Suppressing the argument Z from now on, we thus obtain

$$d_{TV}(L(\tilde{C}[1,b]), L(\tilde{Z}[1,b]))$$

$$= \sum_{r\ge0} P[T_{0b} = r]\left\{1 - \frac{P[T_{bn} = n-r]}{P[T_{0n} = n]}\right\}_+$$

$$\le \sum_{r>n/2} P[T_{0b} = r] + \sum_{r=0}^{[n/2]} \frac{P[T_{0b} = r]}{P[T_{0b} = n]}$$

$$\times \left\{\sum_{s=0}^{n} P[T_{0b} = s](P[T_{bn} = n-s] - P[T_{bn} = n-r]\right\}_+$$

$$\le \sum_{r>n/2} P[T_{0b} = r] + \sum_{r=0}^{[n/2]} P[T_{0b} = r]$$

$$\times \sum_{s=0}^{[n/2]} P[T_{0b} = s]\frac{\{P[T_{bn} = n-s] - P[T_{bn} = n-r]\}}{P[T_{0n} = n]}$$

$$+ \sum_{s=0}^{[n/2]} P[T_{0b} = r] \sum_{s=[n/2]+1}^{n} P[T = s]P[T_{bn} = n-s] / P[T_{0n} = n]$$

The first sum is at most $2n^{-1}ET_{0b}$; the third is bound by

$$(\max_{n/2<s\le n} P[T_{0b} = s]) / P[T_{0n} = n]$$

$$\le \frac{2\varepsilon_{\{10.5(1)\}}(n/2,b)}{n}\frac{3n}{\theta P_\theta[0,1]},$$

$$\frac{3n}{\theta P_\theta[0,1]} 4n^{-2}\phi^*_{\{10.8\}}(n) \sum_{r=0}^{[n/2]} P[T_{0b} = r] \sum_{s=0}^{[n/2]} P[T_{0b} = s]\frac{1}{2}|r-s|$$

$$\le \frac{12\phi^*_{\{10.8\}}(n)}{\theta P_\theta[0,1]}\frac{ET_{0b}}{n}$$

Hence we may take

$$\varepsilon_{\{7,7\}}(n,b) = 2n^{-1}ET_{0b}(Z)\left\{1 + \frac{6\phi^*_{\{10.8\}}(n)}{\theta P_\theta[0,1]}\right\}P$$

$$+ \frac{6}{\theta P_\theta[0,1]}\varepsilon_{\{10.5(1)\}}(n/2,b) \qquad (1.5)$$

Required order under Conditions $(A_0),(D_1)$ and (B_{11}), if $S(\infty) < \infty$. If not, $\phi^*_{\{10.8\}}(n)$ can be replaced by $\phi^{*}_{\{10.11\}}(n)$ in the above, which has the required order, without the restriction on the r_i implied by $S(\infty)<\infty$. Examining the Conditions $(A_0),(D_1)$ and (B_{11}), it is perhaps surprising to find that (B_{11}) is required instead of just (B_{01}); that is, that we should need $\sum_{l\geq 2} l\varepsilon_{il} = O(i^{-a_1})$ to hold for some $a_1>1$. A first observation is that a similar problem arises with the rate of decay of ε_{i1} as well.

For this reason, n_1 is replaced by $\tilde{n}_1$. This makes it possible to replace condition (A_1) by the weaker pair of conditions (A_0) and (D_1) in the eventual assumptions needed for $\varepsilon_{\{7,7\}}(n,b)$ to be of order $O(b/n)$; the decay rate requirement of order $i^{-1-\gamma}$ is shifted from ε_{i1} itself to its first difference. This is needed to obtain the right approximation error for the random mappings example. However, since all the classical applications make far more stringent assumptions about the $\varepsilon_{i1}, l\geq 2$, than are made in (B_{11}). The critical point of the proof is seen where the initial estimate of the difference $P[T^{(m)}_{bn}=s]-P[T^{(m)}_{bn}=s+1]$. The factor $\varepsilon_{\{10.10\}}(n)$, which should be small, contains a far tail element from $\tilde{n}_1$ of the form $\phi^{\theta}_1(n)+u^{*}_1(n)$, which is only small if $a_1>1$, being otherwise of order $O(n^{1-a_1+\delta})$ for any $\delta>0$, since $a_2>1$ is in any case assumed. For $s\geq n/2$, this gives rise to a contribution of order $O(n^{-1-a_1+\delta})$ in the estimate of the difference $P[T_{bn}=s]-P[T_{bn}=s+1]$, which, in the remainder of the proof, is translated into a contribution of order $O(tn^{-1-a_1+\delta})$ for differences of the form $P[T_{bn}=s]-P[T_{bn}=s+1]$, finally leading to a contribution of order $bn^{-a_1+\delta}$ for any $\delta>0$ in $\varepsilon_{\{7.7\}}(n,b)$. Some improvement would seem to be possible, defining the function g by $g(w)=1_{\{w=s\}}-1_{\{w=s+t\}}$, differences that are of the form $P[T_{bn}=s]-P[T_{bn}=s+t]$ can be directly estimated, at a cost of only a single contribution of the form $\phi^{\theta}_1(n)+u^{*}_1(n)$. Then, iterating the cycle, in which one estimate of a difference in point probabilities is improved to an estimate of smaller order, a bound of the form

$$\left|P[T_{bn}=s]-P[T_{bn}=s+t]\right| = O(n^{-2}t+n^{-1-a_1+\delta})$$

for any $\delta>0$ could perhaps be attained, leading to a final error estimate in order $O(bn^{-1}+n^{-a_1+\delta})$ for any $\delta>0$, to replace $\varepsilon_{\{7.7\}}(n,b)$. This would be of the ideal order $O(b/n)$ for large enough b, but would still be coarser for small b.

With b and n as in the previous section, we wish to show that

$$\left|d_{TV}(L(C[1,b]),L(Z[1,b]))-\frac{1}{2}(n+1)^{-1}\left|1-\theta\right|E\left|T_{0b}-ET_{0b}\right|\right|$$
$$\leq \varepsilon_{\{7,8\}}(n,b),$$

Where $\varepsilon_{\{7.8\}}(n,b)=O(n^{-1}b[n^{-1}b+n^{-\beta_{12}+\delta}])$ for any $\delta>0$ under Conditions $(A_0),(D_1)$ and (B_{12}), with β_{12}. The proof uses sharper estimates. As before, we begin with the formula

$$d_{TV}(L(\tilde{C}[1,b]),L(\tilde{Z}[1,b]))$$
$$=\sum_{r\geq 0}P[T_{0b}=r]\left\{1-\frac{P[T_{bn}=n-r]}{P[T_{0n}=n]}\right\}_{+}$$

Now we observe that

$$\left|\sum_{r\geq 0}P[T_{0b}=r]\left\{1-\frac{P[T_{bn}=n-r]}{P[T_{0n}=n]}\right\}_{+}-\sum_{r=0}^{[n/2]}\frac{P[T_{0b}=r]}{P[T_{0n}=n]}\right|$$
$$\times\left|\sum_{s=[n/2]+1}^{n}P[T_{0b}=s](P[T_{bn}=n-s]-P[T_{bn}=n-r])\right|$$
$$\leq 4n^{-2}ET^{2}_{0b}+(\max_{n/2<s\leq n}P[T_{0b}=s])/P[T_{0n}=n]$$
$$+P[T_{0b}>n/2]$$
$$\leq 8n^{-2}ET^{2}_{0b}+\frac{3\varepsilon_{\{10.5(2)\}}(n/2,b)}{\theta P_{\theta}[0,1]}, \qquad (1.1)$$

We have

$$\Big| \sum_{r=0}^{[n/2]} \frac{P[T_{0b}=r]}{P[T_{0n}=n]}$$

$$\times \Big(\Big\{\sum_{s=0}^{[n/2]} P[T_{0b}=s](P[T_{bn}=n-s]-P[T_{bn}=n-r]\Big\}_+$$

$$-\Big\{\sum_{s=0}^{[n/2]} P[T_{0b}=s]\frac{(s-r)(1-\theta)}{n+1}P[T_{0n}=n]\Big\}_+\Big)\ \Big|$$

$$\le \frac{1}{n^2 P[T_{0n}=n]}\sum_{r\ge 0}P[T_{0b}=r]\sum_{s\ge 0}P[T_{0b}=s]|s-r|$$

$$\times\Big\{\varepsilon_{\{10.14\}}(n,b)+2(r\vee s)|1-\theta|n^{-1}\big\{K_0\theta+4\phi^*_{\{10.8\}}(n)\big\}\Big\}$$

$$\le \frac{6}{\theta n P_\theta[0,1]}ET_{0b}\varepsilon_{\{10.14\}}(n,b)$$

$$+4|1-\theta|n^{-2}ET_{0b}^2\big\{K_0\theta+4\phi^*_{\{10.8\}}(n)\big\}$$

$$\Big(\frac{3}{\theta n P_\theta[0,1]}\Big)\ \Big\}, \qquad (1.2)$$

The approximation in (1.2) is further simplified by noting that

$$\sum_{r=0}^{[n/2]}P[T_{0b}=r]\Big|\Big\{\sum_{s=0}^{[n/2]}P[T_{0b}=s]\frac{(s-r)(1-\theta)}{n+1}\Big\}_+$$

$$-\Big\{\sum_{s=0}P[T_{0b}=s]\frac{(s-r)(1-\theta)}{n+1}\Big\}_+\ \Big|$$

$$\le \sum_{r=0}^{[n/2]}P[T_{0b}=r]\sum_{s>[n/2]}P[T_{0b}=s]\frac{(s-r)|1-\theta|}{n+1}$$

$$\le |1-\theta|n^{-1}E(T_{0b}1\{T_{0b}>n/2\})\le 2|1-\theta|n^{-2}ET_{0b}^2, \qquad (1.3)$$

and then by observing that

$$\sum_{r>[n/2]}P[T_{0b}=r]\Big\{\sum_{s\ge 0}P[T_{0b}=s]\frac{(s-r)(1-\theta)}{n+1}\Big\}$$

$$\le n^{-1}|1-\theta|(ET_{0b}P[T_{0b}>n/2]+E(T_{0b}1\{T_{0b}>n/2\}))$$

$$\le 4|1-\theta|n^{-2}ET_{0b}^2 \qquad (1.4)$$

Combining the contributions of (1.2) –(1.3), we thus find tha

$$\Big|\ d_{TV}(L(\tilde{C}[1,b]),L(\tilde{Z}[1,b]))$$

$$-(n+1)^{-1}\sum_{r\ge 0}P[T_{0b}=r]\Big\{\sum_{s\ge 0}P[T_{0b}=s](s-r)(1-\theta)\Big\}_+\ \Big|$$

$$\le \varepsilon_{\{7.8\}}(n,b)$$

$$=\frac{3}{\theta P_\theta[0,1]}\big\{\varepsilon_{\{10.5(2)\}}(n/2,b)+2n^{-1}ET_{0b}\varepsilon_{\{10.14\}}(n,b)\big\}$$

$$+2n^{-2}ET_{0b}^2\Big\{4+3|1-\theta|+\frac{24|1-\theta|\phi^*_{\{10.8\}}(n)}{\theta P_\theta[0,1]}\Big\} \qquad (1.5)$$

The quantity $\varepsilon_{\{7.8\}}(n,b)$ is seen to be of the order claimed under Conditions $(A_0),(D_1)$ and (B_{12}), provided that $S(\infty)<\infty$; this supplementary condition can be removed if $\phi^*_{\{10.8\}}(n)$ is replaced by $\phi^*_{\{10.11\}}(n)$ in the definition of $\varepsilon_{\{7.8\}}(n,b)$, has the required order without the restriction on the r_i implied by assuming that $S(\infty)<\infty$. Finally, a direct calculation now shows that

$$\sum_{r\ge 0}P[T_{0b}=r]\Big\{\sum_{s\ge 0}P[T_{0b}=s](s-r)(1-\theta)\Big\}_+$$

$$=\frac{1}{2}|1-\theta|E|T_{0b}-ET_{0b}|$$

Example 1.0. Consider the point $O=(0,...,0)\in\mathbb{R}^n$. For an arbitrary vector r, the coordinates of the point $x=O+r$ are equal to the respective coordinates of the vector $r: x=(x^1,...x^n)$ and $r=(x^1,...,x^n)$. The vector r such as in the example is called the position vector or the radius vector of the point x. (Or, in greater detail: r is the radius-vector of x w.r.t an origin O). Points are frequently specified by their radius-vectors. This presupposes the choice of O as the “standard origin”. Let us summarize. We have considered $\mathbb{R}^n$ and interpreted its elements in two ways: as points and as vectors. Hence we may say that we leading with the two copies of $\mathbb{R}^n$: $\mathbb{R}^n$ = {points}, $\mathbb{R}^n$ = {vectors}
Operations with vectors: multiplication by a number, addition. Operations with points and vectors: adding a vector to a point (giving a point), subtracting two points (giving a vector). $\mathbb{R}^n$ treated in this way is called an *n-dimensional affine space*. (An “abstract” affine space is a pair of sets , the set of points and the set of vectors so that the operations as above are defined axiomatically). Notice that vectors in an affine space are also known as “free vectors”. Intuitively, they are not fixed at points and “float

freely" in space. From $\mathbb{R}^n$ considered as an affine space we can precede in two opposite directions: $\mathbb{R}^n$ as an Euclidean space $\Leftarrow$ $\mathbb{R}^n$ as an affine space $\Rightarrow$ $\mathbb{R}^n$ as a manifold.Going to the left means introducing some extra structure which will make the geometry richer. Going to the right means forgetting about part of the affine structure; going further in this direction will lead us to the so-called "smooth (or differentiable) manifolds". The theory of differential forms does not require any extra geometry. So our natural direction is to the right. The Euclidean structure, however, is useful for examples and applications. So let us say a few words about it:

Remark 1.0. *Euclidean geometry.* In $\mathbb{R}^n$ considered as an affine space we can already do a good deal of geometry. For example, we can consider lines and planes, and quadric surfaces like an ellipsoid. However, we cannot discuss such things as "lengths", "angles" or "areas" and "volumes". To be able to do so, we have to introduce some more definitions, making $\mathbb{R}^n$ a Euclidean space. Namely, we define the length of a vector $a = (a^1, ..., a^n)$ to be

$$|a| := \sqrt{(a^1)^2 + ... + (a^n)^2} \qquad (1)$$

After that we can also define distances between points as follows:

$$d(A, B) := |\overrightarrow{AB}| \qquad (2)$$

One can check that the distance so defined possesses natural properties that we expect: is it always non-negative and equals zero only for coinciding points; the distance from A to B is the same as that from B to A (symmetry); also, for three points, A, B and C, we have $d(A,B) \le d(A,C) + d(C,B)$ (the "triangle inequality"). To define angles, we first introduce the scalar product of two vectors

$$(a,b) := a^1b^1 + ... + a^nb^n \qquad (3)$$

Thus $|a| = \sqrt{(a,a)}$. The scalar product is also denote by dot: $a.b = (a,b)$, and hence is often referred to as the "dot product" . Now, for nonzero vectors, we define the angle between them by the equality

$$\cos\alpha := \frac{(a,b)}{|a||b|} \qquad (4)$$

The angle itself is defined up to an integral multiple of 2π . For this definition to be consistent we have to ensure that the r.h.s. of (4) does not exceed 1 by the absolute value. This follows from the inequality

$$(a,b)^2 \le |a|^2 |b|^2 \qquad (5)$$

known as the Cauchy–Bunyakovsky–Schwarz inequality (various combinations of these three names are applied in different books). One of the ways of proving (5) is to consider the scalar square of the linear combination $a + tb,$ where $t \in R$. As $(a + tb, a + tb) \ge 0$ is a quadratic polynomial in t which is never negative, its discriminant must be less or equal zero. Writing this explicitly yields (5). The triangle inequality for distances also follows from the inequality (5).

Example 1.1. Consider the function $f(x) = x^i$ (the i-th coordinate). The linear function dx^i (the differential of x^i) applied to an arbitrary vector h is simply h^i .From these examples follows that we can rewrite df as

$$df = \frac{\partial f}{\partial x^1} dx^1 + ... + \frac{\partial f}{\partial x^n} dx^n, \qquad (1)$$

which is the standard form. Once again: the partial derivatives in (1) are just the coefficients (depending on x); $dx^1, dx^2, ...$ are linear functions giving on an arbitrary vector h its coordinates $h^1, h^2, ...,$ respectively. Hence

$$df(x)(h) = \partial_{hf(x)} = \frac{\partial f}{\partial x^1} h^1 + ... + \frac{\partial f}{\partial x^n} h^n, \qquad (2)$$

Theorem 1.7. Suppose we have a parametrized curve $t \mapsto x(t)$ passing through $x_0 \in \mathbb{R}^n$ at $t = t_0$ and with the velocity vector $x(t_0) = \upsilon$ Then

$$\frac{df(x(t))}{dt}(t_0) = \partial_\upsilon f(x_0) = df(x_0)(\upsilon) \qquad (1)$$

Proof. Indeed, consider a small increment of the parameter $t : t_0 \mapsto t_0 + \Delta t$, Where $\Delta t \mapsto 0$. On the other hand, we have $f(x_0 + h) - f(x_0) = df(x_0)(h) + \beta(h)|h|$ for an arbitrary vector h , where $\beta(h) \to 0$ when $h \to 0$. Combining it together, for the increment of $f(x(t))$ we obtain

$$f(x(t_0+\Delta t)-f(x_0)$$
$$=df(x_0)(\upsilon.\Delta t+\alpha(\Delta t)\Delta t)$$
$$+\beta(\upsilon.\Delta t+\alpha(\Delta t)\Delta t).\left|\upsilon\Delta t+\alpha(\Delta t)\Delta t\right|$$
$$=df(x_0)(\upsilon).\Delta t+\gamma(\Delta t)\Delta t$$

For a certain $\gamma(\Delta t)$ such that $\gamma(\Delta t)\to 0$ when $\Delta t\to 0$ (we used the linearity of $df(x_0)$). By the definition, this means that the derivative of $f(x(t))$ at $t=t_0$ is exactly $df(x_0)(\upsilon)$. The statement of the theorem can be expressed by a simple formula:

$$\frac{df(x(t))}{dt}=\frac{\partial f}{\partial x^1}x^1+\ldots+\frac{\partial f}{\partial x^n}x^n \qquad (2)$$

To calculate the value Of df at a point x_0 on a given vector υ one can take an arbitrary curve passing Through x_0 at t_0 with υ as the velocity vector at t_0 and calculate the usual derivative of $f(x(t))$ at $t=t_0$.

Theorem 1.8. For functions $f,g:U\to\mathbb{R}$, $U\subset\mathbb{R}^n$,

$$d(f+g)=df+dg \qquad (1)$$
$$d(fg)=df.g+f.dg \qquad (2)$$

Proof. Consider an arbitrary point x_0 and an arbitrary vector υ stretching from it. Let a curve $x(t)$ be such that $x(t_0)=x_0$ and $x(t_0)=\upsilon$. Hence

$$d(f+g)(x_0)(\upsilon)=\frac{d}{dt}(f(x(t))+g(x(t)))$$

at $t=t_0$ and

$$d(fg)(x_0)(\upsilon)=\frac{d}{dt}(f(x(t))g(x(t)))$$

at $t=t_0$ Formulae (1) and (2) then immediately follow from the corresponding formulae for the usual derivative Now, almost without change the theory generalizes to functions taking values in $\mathbb{R}^m$ instead of $\mathbb{R}$. The only difference is that now the differential of a map $F:U\to\mathbb{R}^m$ at a point x will be a linear function taking vectors in $\mathbb{R}^n$ to vectors in $\mathbb{R}^m$ (instead of $\mathbb{R}$). For an arbitrary vector $h\in|\mathbb{R}^n$,

$$F(x+h)=F(x)+dF(x)(h)$$
$$+\beta(h)\left|h\right| \qquad (3)$$

Where $\beta(h)\to 0$ when $h\to 0$. We have $dF=(dF^1,\ldots,dF^m)$ and

$$dF=\frac{\partial F}{\partial x^1}dx^1+\ldots+\frac{\partial F}{\partial x^n}dx^n$$
$$=\begin{pmatrix}\frac{\partial F^1}{\partial x^1}\cdots\frac{\partial F^1}{\partial x^n}\\ \ldots\quad\ldots\quad\ldots\\ \frac{\partial F^m}{\partial x^1}\cdots\frac{\partial F^m}{\partial x^n}\end{pmatrix}\begin{pmatrix}dx^1\\ \ldots\\ dx^n\end{pmatrix} \qquad (4)$$

In this matrix notation we have to write vectors as vector-columns.

Theorem 1.9. For an arbitrary parametrized curve $x(t)$ in $\mathbb{R}^n$, the differential of a map $F:U\to\mathbb{R}^m$ (where $U\subset\mathbb{R}^n$) maps the velocity vector $x(t)$ to the velocity vector of the curve $F(x(t))$ in $\mathbb{R}^m$:

$$\frac{dF(x(t))}{dt}=dF(x(t))(\dot{x}(t)) \qquad (1)$$

Proof. By the definition of the velocity vector,

$$x(t+\Delta t)=x(t)+\dot{x}(t).\Delta t+\alpha(\Delta t)\Delta t \qquad (2)$$

Where $\alpha(\Delta t)\to 0$ when $\Delta t\to 0$. By the definition of the differential,

$$F(x+h)=F(x)+dF(x)(h)+\beta(h)\left|h\right. \qquad (3)|$$

Where $\beta(h)\to 0$ when $h\to 0$. we obtain

$$F(x(t+\Delta t))=F(x+\underbrace{\dot{x}(t).\Delta t+\alpha(\Delta t)\Delta t)}_{h}$$
$$=F(x)+dF(x)(\dot{x}(t)\Delta t+\alpha(\Delta t)\Delta t)+$$
$$\beta(\dot{x}(t)\Delta t+\alpha(\Delta t)\Delta t).\left|\dot{x}(t)\Delta t+\alpha(\Delta t)\Delta t\right|$$
$$=F(x)+dF(x)(\dot{x}(t)\Delta t+\gamma(\Delta t)\Delta t$$

For some $\gamma(\Delta t)\to 0$ when $\Delta t\to 0$. This precisely means that $dF(x)\dot{x}(t)$ is the velocity vector of $F(x)$. As every vector attached to a point can be viewed as the velocity vector of some curve passing through this point, this theorem gives a clear geometric picture of dF as a linear map on vectors.

Theorem 1.10 Suppose we have two maps $F:U\to V$ and $G:V\to W,$ where $U\subset\mathbb{R}^n, V\subset\mathbb{R}^m, W\subset\mathbb{R}^p$ (open domains). Let $F:x\mapsto y=F(x)$. Then the differential of the composite map $GoF:U\to W$ is the composition of the differentials of F and G:

$$d(GoF)(x)=dG(y)odF(x) \qquad (4)$$

Proof. We can use the description of the differential .Consider a curve $x(t)$ in $\mathbb{R}^n$ with the velocity vector $\dot{x}$. Basically, we need to know to which vector in $\mathbb{R}^p$ it is taken by $d(GoF)$. the curve $(GoF)(x(t)=G(F(x(t))$. By the same theorem, it equals the image under dG of the Anycast Flow vector to the curve $F(x(t))$ in $\mathbb{R}^m$. Applying the theorem once again, we see that the velocity vector to the curve $F(x(t))$ is the image under dF of the vector $\dot{x}(t)$. Hence $d(GoF)(\dot{x})=dG(dF(\dot{x}))$ for an arbitrary vector $\dot{x}$.

Corollary 1.0. If we denote coordinates in $\mathbb{R}^n$ by $(x^1,\dots,x^n)$ and in $\mathbb{R}^m$ by $(y^1,\dots,y^m)$, and write

$$dF=\frac{\partial F}{\partial x^1}dx^1+\dots+\frac{\partial F}{\partial x^n}dx^n \qquad (1)$$

$$dG=\frac{\partial G}{\partial y^1}dy^1+\dots+\frac{\partial G}{\partial y^n}dy^n, \qquad (2)$$

Then the chain rule can be expressed as follows:

$$d(GoF)=\frac{\partial G}{\partial y^1}dF^1+\dots+\frac{\partial G}{\partial y^m}dF^m, \qquad (3)$$

Where dF^i are taken from (1). In other words, to get $d(GoF)$ we have to substitute into (2) the expression for $dy^i=dF^i$ from (3). This can also be expressed by the following matrix formula:

$$d(GoF)=\begin{pmatrix}\frac{\partial G^1}{\partial y^1}\cdots\frac{\partial G^1}{\partial y^m}\\ \dots\ \dots\ \dots\\ \frac{\partial G^p}{\partial y^1}\cdots\frac{\partial G^p}{\partial y^m}\end{pmatrix}\begin{pmatrix}\frac{\partial F^1}{\partial x^1}\cdots\frac{\partial F^1}{\partial x^n}\\ \dots\ \dots\ \dots\\ \frac{\partial F^m}{\partial x^1}\cdots\frac{\partial F^m}{\partial x^n}\end{pmatrix}\begin{pmatrix}dx^1\\ \dots\\ dx^n\end{pmatrix} \qquad (4)$$

i.e., if dG and dF are expressed by matrices of partial derivatives, then $d(GoF)$ is expressed by the product of these matrices. This is often written as

$$\begin{pmatrix}\frac{\partial z^1}{\partial x^1}\cdots\frac{\partial z^1}{\partial x^n}\\ \dots\ \dots\ \dots\\ \frac{\partial z^p}{\partial x^1}\cdots\frac{\partial z^p}{\partial x^n}\end{pmatrix}=\begin{pmatrix}\frac{\partial z^1}{\partial y^1}\cdots\frac{\partial z^1}{\partial y^m}\\ \dots\ \dots\ \dots\\ \frac{\partial z^p}{\partial y^1}\cdots\frac{\partial z^p}{\partial y^m}\end{pmatrix}\begin{pmatrix}\frac{\partial y^1}{\partial x^1}\cdots\frac{\partial y^1}{\partial x^n}\\ \dots\ \dots\ \dots\\ \frac{\partial y^m}{\partial x^1}\cdots\frac{\partial y^m}{\partial x^n}\end{pmatrix}, \qquad (5)$$

Or

$$\frac{\partial z^\mu}{\partial x^a}=\sum_{i=1}^{m}\frac{\partial z^\mu}{\partial y^i}\frac{\partial y^i}{\partial x^a}, \qquad (6)$$

Where it is assumed that the dependence of $y\in\mathbb{R}^m$ on $x\in\mathbb{R}^n$ is given by the map F, the dependence of $z\in\mathbb{R}^p$ on $y\in\mathbb{R}^m$ is given by the map $G,$ and the dependence of $z\in\mathbb{R}^p$ on $x\in\mathbb{R}^n$ is given by the composition GoF.

Definition 1.6. Consider an open domain $U\subset\mathbb{R}^n$. Consider also another copy of $\mathbb{R}^n$, denoted for distinction $\mathbb{R}^n_y$, with the standard coordinates $(y^1\dots y^n)$. A system of coordinates in the open domain U is given by a map $F:V\to U,$ where $V\subset\mathbb{R}^n_y$ is an open domain of $\mathbb{R}^n_y$, such that the following three conditions are satisfied :

(1) F is smooth;

(2) F is invertible;

(3) $F^{-1}:U\to V$ is also smooth

The coordinates of a point $x\in U$ in this system are the standard coordinates of $F^{-1}(x)\in\mathbb{R}^n_y$

In other words,

$$F:(y^1\dots,y^n)\mapsto x=x(y^1\dots,y^n) \qquad (1)$$

Here the variables $(y^1\dots,y^n)$ are the “new” coordinates of the point x

Example 1.2. Consider a curve in $\mathbb{R}^2$ specified in polar coordinates as

$$x(t):r=r(t),\varphi=\varphi(t) \qquad (1)$$

We can simply use the chain rule. The map $t\mapsto x(t)$ can be considered as the composition of the maps $t \mapsto (r(t), \varphi(t)), (r,\varphi) \mapsto x(r,\varphi)$. Then, by the chain rule, we have

$$\dot{x} = \frac{dx}{dt} = \frac{\partial x}{\partial r}\frac{dr}{dt} + \frac{\partial x}{\partial \varphi}\frac{d\varphi}{dt} = \frac{\partial x}{\partial r}\dot{r} + \frac{\partial x}{\partial \varphi}\dot{\varphi} \qquad (2)$$

Here $\dot{r}$ and $\dot{\varphi}$ are scalar coefficients depending on t, whence the partial derivatives $\partial x/\partial r, \partial x/\partial \varphi$ are vectors depending on point in $\mathbb{R}^2$. We can compare this with the formula in the “standard” coordinates: $\dot{x} = e_1\dot{x} + e_2\dot{y}$. Consider the vectors $\partial x/\partial r, \partial x/\partial \varphi$. Explicitly we have

$$\frac{\partial x}{\partial r} = (\cos\varphi, \sin\varphi) \qquad (3)$$

$$\frac{\partial x}{\partial \varphi} = (-r\sin\varphi, r\cos\varphi) \qquad (4)$$

From where it follows that these vectors make a basis at all points except for the origin (where $r = 0$). It is instructive to sketch a picture, drawing vectors corresponding to a point as starting from that point. Notice that $\partial x/\partial r, \partial x/\partial \varphi$ are, respectively, the velocity vectors for the curves $r \mapsto x(r,\varphi)$ $(\varphi = \varphi_0 \ fixed)$ and $\varphi \mapsto x(r,\varphi)$ $(r = r_0 \ fixed)$. We can conclude that for an arbitrary curve given in polar coordinates the velocity vector will have components $(\dot{r}, \dot{\varphi})$ if as a basis we take $e_r := \partial x/\partial r, e_\varphi := \partial x/\partial \varphi$:

$$\dot{x} = e_r\dot{r} + e_\varphi\dot{\varphi} \qquad (5)$$

A characteristic feature of the basis e_r, e_φ is that it is not “constant” but depends on point. Vectors “stuck to points” when we consider curvilinear coordinates.

Proposition 1.3. The velocity vector has the same appearance in all coordinate systems.

Proof. Follows directly from the chain rule and the transformation law for the basis e_i. In particular, the elements of the basis $e_i = \partial x/\partial x^i$ (originally, a formal notation) can be understood directly as the velocity vectors of the coordinate lines $x^i \mapsto x(x^1,\ldots,x^n)$ (all coordinates but x^i are fixed). Since we now know how to handle velocities in arbitrary coordinates, the best way to treat the differential of a map $F : \mathbb{R}^n \to \mathbb{R}^m$ is by its action on the velocity vectors. By definition, we set

$$dF(x_0) : \frac{dx(t)}{dt}(t_0) \mapsto \frac{dF(x(t))}{dt}(t_0) \qquad (1)$$

Now $dF(x_0)$ is a linear map that takes vectors attached to a point $x_0 \in \mathbb{R}^n$ to vectors attached to the point $F(x) \in \mathbb{R}^m$

$$dF = \frac{\partial F}{\partial x^1}dx^1 + \ldots + \frac{\partial F}{\partial x^n}dx^n$$

$$(e_1,\ldots,e_m)\begin{pmatrix} \frac{\partial F^1}{\partial x^1} \cdots \frac{\partial F^1}{\partial x^n} \\ \cdots \ \cdots \ \cdots \\ \frac{\partial F^m}{\partial x^1} \cdots \frac{\partial F^m}{\partial x^n} \end{pmatrix}\begin{pmatrix} dx^1 \\ \cdots \\ dx^n \end{pmatrix}, \qquad (2)$$

In particular, for the differential of a function we always have

$$df = \frac{\partial f}{\partial x^1}dx^1 + \ldots + \frac{\partial f}{\partial x^n}dx^n, \qquad (3)$$

Where x^i are arbitrary coordinates. The form of the differential does not change when we perform a change of coordinates.

Example 1.3 Consider a 1-form in $\mathbb{R}^2$ given in the standard coordinates:

$A = -ydx + xdy$ In the polar coordinates we will have $x = r\cos\varphi, y = r\sin\varphi$, hence

$$dx = \cos\varphi dr - r\sin\varphi d\varphi$$

$$dy = \sin\varphi dr + r\cos\varphi d\varphi$$

Substituting into A, we get

$$A = -r\sin\varphi(\cos\varphi dr - r\sin\varphi d\varphi)$$
$$+r\cos\varphi(\sin\varphi dr + r\cos\varphi d\varphi)$$
$$= r^2(\sin^2\varphi + \cos^2\varphi)d\varphi = r^2 d\varphi$$

Hence $A = r^2 d\varphi$ is the formula for A in the polar coordinates. In particular, we see that this is again a 1-form, a linear combination of the differentials of coordinates with functions as coefficients. Secondly, in a more conceptual way, we can define a 1-form in a domain U as a linear function on vectors at every point of U:

$$\omega(\upsilon) = \omega_1\upsilon^1 + \ldots + \omega_n\upsilon^n, \qquad (1)$$

If $\upsilon = \sum e_i\upsilon^i$, where $e_i = \partial x/\partial x^i$. Recall that the differentials of functions were defined as linear functions on vectors (at every point), and

$$dx^i(e_j) = dx^i\left(\frac{\partial x}{\partial x^j}\right) = \delta^i_j \qquad (2)$$

at every point x.

Theorem 1.9. For arbitrary 1-form ω and path γ, the integral $\int_\gamma \omega$ does not change if we change parametrization of γ provide the orientation remains the same.

Proof: Consider $\left\langle \omega(x(t)), \frac{dx}{dt^{'}} \right\rangle$ and $\left\langle \omega(x(t(t^{'}))), \frac{dx}{dt^{'}} \right\rangle$ As $\left\langle \omega(x(t(t^{'}))), \frac{dx}{dt^{'}} \right\rangle = \left| \left\langle \omega(x(t(t^{'}))), \frac{dx}{dt^{'}} \right\rangle . \frac{dt}{dt^{'}} \right.$,

Let p be a rational prime and let $K = \square(\zeta_p)$. We write ζ for ζ_p or this section. Recall that K has degree $\varphi(p) = p-1$ over $\square$. We wish to show that $O_K = \square[\zeta]$. Note that ζ is a root of $x^p - 1$, and thus is an algebraic integer; since O_K is a ring we have that $\square[\zeta] \subseteq O_K$. We give a proof without assuming unique factorization of ideals. We begin with some norm and trace computations. Let j be an integer. If j is not divisible by p, then ζ^j is a primitive p^{th} root of unity, and thus its conjugates are $\zeta, \zeta^2, ..., \zeta^{p-1}$. Therefore

$$Tr_{K/\square}(\zeta^j) = \zeta + \zeta^2 + ... + \zeta^{p-1} = \Phi_p(\zeta) - 1 = -1$$

If p does divide j, then $\zeta^j = 1$, so it has only the one conjugate 1, and $Tr_{K/\square}(\zeta^j) = p-1$ By linearity of the trace, we find that

$$Tr_{K/\square}(1-\zeta) = Tr_{K/\square}(1-\zeta^2) = ...$$

$$= Tr_{K/\square}(1-\zeta^{p-1}) = p$$

We also need to compute the norm of $1-\zeta$. For this, we use the factorization

$$x^{p-1} + x^{p-2} + ... + 1 = \Phi_p(x)$$

$$= (x-\zeta)(x-\zeta^2)...(x-\zeta^{p-1});$$

Plugging in $x = 1$ shows that

$$p = (1-\zeta)(1-\zeta^2)...(1-\zeta^{p-1})$$

Since the $(1-\zeta^j)$ are the conjugates of $(1-\zeta)$, this shows that $N_{K/\square}(1-\zeta) = p$ The key result for determining the ring of integers O_K is the following.

LEMMA 1.9

$$(1-\zeta)O_K \cap \square = p\square$$

Proof. We saw above that p is a multiple of $(1-\zeta)$ in O_K, so the inclusion $(1-\zeta)O_K \cap \square \supseteq p\square$ is immediate. Suppose now that the inclusion is strict. Since $(1-\zeta)O_K \cap \square$ is an ideal of $\square$ containing $p\square$ and $p\square$ is a maximal ideal of $\square$, we must have $(1-\zeta)O_K \cap \square = \square$ Thus we can write $1 = \alpha(1-\zeta)$ For some $\alpha \in O_K$. That is, $1-\zeta$ is a unit in O_K.

COROLLARY 1.1 For any $\alpha \in O_K$, $Tr_{K/\square}((1-\zeta)\alpha) \in p\square$

PROOF. We have

$$Tr_{K/\square}((1-\zeta)\alpha) = \sigma_1((1-\zeta)\alpha) + ... + \sigma_{p-1}((1-\zeta)\alpha)$$
$$= \sigma_1(1-\zeta)\sigma_1(\alpha) + ... + \sigma_{p-1}(1-\zeta)\sigma_{p-1}(\alpha)$$
$$= (1-\zeta)\sigma_1(\alpha) + ... + (1-\zeta^{p-1})\sigma_{p-1}(\alpha)$$

Where the σ_i are the complex embeddings of K (which we are really viewing as automorphisms of K) with the usual ordering. Furthermore, $1-\zeta^j$ is a multiple of $1-\zeta$ in O_K for every $j \neq 0$. Thus $Tr_{K/\square}(\alpha(1-\zeta)) \in (1-\zeta)O_K$ Since the trace is also a rational integer.

PROPOSITION 1.4 Let p be a prime number and let $K = |\square(\zeta_p)$ be the p^{th} cyclotomic field. Then $O_K = \square[\zeta_p] \cong \square[x]/(\Phi_p(x))$; Thus $1, \zeta_p, ..., \zeta_p^{p-2}$ is an integral basis for O_K.

PROOF. Let $\alpha \in O_K$ and write $\alpha = a_0 + a_1\zeta + ... + a_{p-2}\zeta^{p-2}$ With $a_i \in \square$. Then

$$\alpha(1-\zeta) = a_0(1-\zeta) + a_1(\zeta - \zeta^2) + ...$$
$$+ a_{p-2}(\zeta^{p-2} - \zeta^{p-1})$$

By the linearity of the trace and our above calculations we find that $Tr_{K/\square}(\alpha(1-\zeta)) = pa_0$ We also have $Tr_{K/\square}(\alpha(1-\zeta)) \in p\square$, so $a_0 \in \square$ Next consider the algebraic integer

$(\alpha - a_0)\zeta^{-1} = a_1 + a_2\zeta + ... + a_{p-2}\zeta^{p-3}$; This is an algebraic integer since $\zeta^{-1} = \zeta^{p-1}$ is. The same argument as above shows that $a_1 \in \mathbb{Z}$, and continuing in this way we find that all of the a_i are in $\mathbb{Z}$. This completes the proof.

Example 1.4 Let $K = \mathbb{Q}$, then the local ring $\mathbb{Z}_{(p)}$ is simply the subring of $\mathbb{Q}$ of rational numbers with denominator relatively prime to p . Note that this ring $\mathbb{Z}_{(p)}$ is not the ring $\mathbb{Z}_p$ of p -adic integers; to get $\mathbb{Z}_p$ one must complete $\mathbb{Z}_{(p)}$. The usefulness of $O_{K,p}$ comes from the fact that it has a particularly simple ideal structure. Let a be any proper ideal of $O_{K,p}$ and consider the ideal $a \cap O_K$ of O_K. We claim that $a = (a \cap O_K)O_{K,p}$; That is, that a is generated by the elements of a in $a \cap O_K$. It is clear from the definition of an ideal that $a \supseteq (a \cap O_K)O_{K,p}$. To prove the other inclusion, let α be any element of a . Then we can write $\alpha = \beta / \gamma$ where $\beta \in O_K$ and $\gamma \notin p.$ In particular, $\beta \in a$ (since $\beta / \gamma \in a$ and a is an ideal), so $\beta \in O_K$ and $\gamma \notin p.$ so $\beta \in a \cap O_K$. Since $1/\gamma \in O_{K,p}$, this implies that $\alpha = \beta / \gamma \in (a \cap O_K)O_{K,p}$, as claimed.We can use this fact to determine all of the ideals of $O_{K,p}$. Let a be any ideal of $O_{K,p}$ and consider the ideal factorization of $a \cap O_K$ in O_K. write it as $a \cap O_K = p^n b$ For some n and some ideal b, relatively prime to $p.$ we claim first that $bO_{K,p} = O_{K,p}$. We now find that

$a = (a \cap O_K)O_{K,p} = p^n bO_{K,p} = p^n O_{K,p}$ Since $bO_{K,p}$. Thus every ideal of $O_{K,p}$ has the form $p^n O_{K,p}$ for some n; it follows immediately that $O_{K,p}$ is noetherian. It is also now clear that $p^n O_{K,p}$ is the unique non-zero prime ideal in $O_{K,p}$. Furthermore, the inclusion $O_K \mapsto O_{K,p} / pO_{K,p}$ Since $pO_{K,p} \cap O_K = p$, this map is also surjection, since the residue class of $\alpha / \beta \in O_{K,p}$ (with $\alpha \in O_K$ and $\beta \notin p$) is the image of $\alpha\beta^{-1}$ in $O_{K/p}$, which makes sense since β is invertible in $O_{K/p}$. Thus the map is an isomorphism. In particular, it is now abundantly clear that every non-zero prime ideal of $O_{K,p}$ is maximal. To show that $O_{K,p}$ is a Dedekind domain, it remains to show that it is integrally closed in K. So let $\gamma \in K$ be a root of a polynomial with coefficients in $O_{K,p}$; write this polynomial as $x^m + \frac{\alpha_{m-1}}{\beta_{m-1}} x^{m-1} + ... + \frac{\alpha_0}{\beta_0}$ With $\alpha_i \in O_K$ and $\beta_i \in O_{K-p}$. Set $\beta = \beta_0\beta_1 ... \beta_{m-1}$. Multiplying by β^m we find that $\beta\gamma$ is the root of a monic polynomial with coefficients in O_K. Thus $\beta\gamma \in O_K$; since $\beta \notin p$, we have $\beta\gamma / \beta = \gamma \in O_{K,p}$. Thus $O_{K,p}$ is integrally close in K.

COROLLARY 1.2. Let K be a number field of degree n and let α be in O_K then $N'_{K/\mathbb{Q}}(\alpha O_K) = |N_{K/\mathbb{Q}}(\alpha)|$

PROOF. We assume a bit more Galois theory than usual for this proof. Assume first that $K/\mathbb{Q}$ is Galois. Let σ be an element of $Gal(K/\mathbb{Q})$. It is clear that $\sigma(O_K)/\sigma(\alpha) \cong O_{K/\alpha}$; since $\sigma(O_K) = O_K$, this shows that $N'_{K/\mathbb{Q}}(\sigma(\alpha)O_K) = N'_{K/\mathbb{Q}}(\alpha O_K)$. Taking the product over all $\sigma \in Gal(K/\mathbb{Q})$, we have $N'_{K/\mathbb{Q}}(N_{K/\mathbb{Q}}(\alpha)O_K) = N'_{K/\mathbb{Q}}(\alpha O_K)^n$ Since $N_{K/\mathbb{Q}}(\alpha)$ is a rational integer and O_K is a free $\mathbb{Z}$ -module of rank n,

$O_K / N_{K/\mathbb{Q}}(\alpha)O_K$ Will have order $N_{K/\mathbb{Q}}(\alpha)^n$; therefore

$$N'_{K/\mathbb{Q}}(N_{K/\mathbb{Q}}(\alpha)O_K) = N_{K/\mathbb{Q}}(\alpha O_K)^n$$

This completes the proof. In the general case, let L be the Galois closure of K and set $[L:K] = m.$

F. Concurrent Crawling Algorithm

The concurrent crawling approach

Global State-flow Graph. The first change is the separation of the state-flow graph from the state machine. The graph is defined in a global scope, so that it can be centralized and used by all concurrent nodes. Upon the start of the crawling process, an initial crawling node is created and its RUN procedure is called.

Browser Pool. The robot and state machine are created for each crawling node. Thus, they are placed in the local scope of the RUN procedure. Generally, each node needs to acquire a browser instance, and after the process is finished, the browser is killed. Creating new browser instances is a process intensive and time-consuming operation. To optimize, a new structure is introduced: the BrowserPool, which creates and maintains browsers in a pool of browsers to be reused by the crawling nodes. This reduces start-up and shut-down costs. The BrowserPool can be queried for a browser instance, and when a node is finished working, the browser used is released back to the pool. In addition, the algorithm now takes the desired number of browsers as input. Increasing the number of browsers used can decrease the crawling runtime, but it also comes with some limitations and tradeoffs.

Forward-Tracking. In the sequential algorithm, after finishing a crawl path, we need to bring the crawler to the previous (relevant) state. In the concurrent algorithm, however, we create a new crawling node for each path to be examined. Thus, instead of bringing the crawler back to the desired state (backtracking), we must take the new node forward to the desired state, hence, forward-tracking. This is done after the browser is pointed to the URL. The first time the RUN procedure is executed, no forward-tracking is taking place, since the event-path (i.e., the list of clickable items resulting to the desired state) is empty, so the initial crawler starts from the Index state. However, if the event path is not empty, the clickables are used to take the browser forward to the desired state. At that point, the CRAWL procedure is called.

Crawling Procedure. The first part of the CRAWL procedure is unchanged. To enable concurrent nodes accessing the candidate clickables in a thread-safe manner, the body of the for loop is synchronized around the candidate element to be examined. To avoid examining a candidate element multiple times bymultiple nodes, each node first checks the examined state of the candidate element. If the element has not been examined previously, the robot executes an event on the element in the browser and sets its state as examined. If the state is changed, before going into the recursive CRAWL call, the PARTITION procedure is called.

Partition Procedure. The partition procedure, called on a particular state cs, creates a new crawling node for every unexamined candidate clickable in cs. The new crawlers are initialized with two parameters, namely, (1) the current state cs, and (2) the execution path from the initial Index state to this state. Every new node is distributed to the work queue participating in the concurrent crawling. When a crawling node is chosen from the work queue, its corresponding RUN procedure is called in order to spawn a new crawling thread.

G. Applying Crawljax

The results of applying CRAWLJAX to C1–C6 are displayed. The key characteristics of the sites under study, such as the average DOM size and the total number of candidate clickables. Furthermore, it lists the key configuration parameters set, most notably the tags used to identify candidate clickables and the maximum crawling depth.

H. Accuracy

Experimental Setup. Assessing the correctness of the crawling process is challenging for two reasons. First, there is no strict notion of “correctness” with respect to state equivalence. The state comparison operator part of our algorithm can be implemented in different ways: the more states it considers equal, the smaller and the more abstract the resulting state-flow graph is. The desirable level of abstraction depends on the intended use of the crawler (regression testing, program comprehension, security testing, to name a few) and the characteristics of the system being crawled. Second, no other crawlers for AJAX are available, making it impossible to compare our results to a “gold standard.” Consequently, an assessment in terms of precision (percentage of correct states) and recall (percentage of states recovered) is impossible to give. To address these concerns, we proceed as follows. For the cases in which we have full control—C1 and C2—we inject specific clickable elements.

—For C1, 16 elements were injected, out of which 10 were on the top-level index page. Furthermore, to evaluate the state comparison procedure, we intentionally introduced a number of identical (clone) states.

—For C2, we focused on two product categories, CATS and DOGS, from the five available categories. We annotated 36 elements (product items) by modifying the JAVASCRIPT method, which turns the items retrieved from the server into clickables on the interface.

Subsequently, we manually create a referencemodel, to which we compare the derived state-flow graph. To assess the four external sites C3–C6, we inspect a selection of the states. For each site, we randomly select ten clickables in advance, by noting their tag names, attributes, and XPath expressions. After crawling of each site, we check the presence of these ten elements among the list of detected clickables. In order to do the manual inspection of the results, we run CRAWLJAX with the Mirror plugin enabled. This post-crawling plugin creates a static mirror, based on the derived state-flow graph, by writing all DOM states to file and replacing edges with appropriate hyperlinks.

I. Scalability

Experimental Setup. In order to obtain an understanding of the scalability of our approach, we measure the time needed to crawl, as well as a number of site characteristics that will affect the time needed. We expect the crawling performance to be directly proportional to the input size, which is composed of (1) the average DOM string size, (2) number of candidate elements, and (3) number of detected clickables and states, which are the characteristics that we measure for the six cases. To test the capability of our method in crawling real sites and coping with unknown environments, we run CRAWLJAX on four external cases, C3–C6. We run CRAWLJAX with depth level 2 on C3 and C5, each having a huge state space to examine the scalability of our approach in analyzing tens of thousands of candidate clickables and finding clickables.

J. Findings.

Concerning the time needed to crawl the internal sites, we see that it takes CRAWLJAX 14 and 26 seconds to crawl C1 and C2, respectively. The average DOM size in C2 is five times bigger, and the number of candidate elements is three times higher. In addition to this increase in DOM size and in the number of candidate elements, the C2 site does not support the browser's built-in Back method. Thus, as discussed in Section 3.6, for every state change on the browser, CRAWLJAX has to reload the application and click through to the previous state to go further. This reloading and clicking through naturally has a negative effect on the performance. Note that the performance is also dependent on the CPU and memory of the machine CRAWLJAX is running on, as well as the speed of the server and network properties of the case site. C6, for instance, is slow in reloading and retrieving updates from its server, which increases the performance measurement numbers in our experiment. CRAWLJAX was able to run smoothly on the external sites. Except a few minor adjustments, we did not witness any difficulties. C3 with depth level 2 was crawled successfully in 83 minutes, resulting in 19,247 examined candidate elements, 1,101 detected clickables, and 1,071 detected states. For C5, CRAWLJAX was able to finish the crawl process in 107 minutes on 32,365 candidate elements, resulting in 1,554 detected clickables, and 1,234 states. As expected, in both cases, increasing the depth level from 1 to 2 greatly expands the state space.

K. Concurrent Crawling

In our final experiment, the main goal is to assess the influence of the concurrent crawling algorithm on the crawling runtime.

Experimental Object. Our experimental object for this study is Google ADSENSE11, an AJAX application developed by Google, which empowers online publishers to earn revenue by displaying relevant ads on their Web content. The ADSENSE interface is built using GWT (Google Web Toolkit) components and is written in Java. The index page of ADSENSE. On the top, there are four main tabs (Home, My ads, Allow & block ads, Performance reports). On the top left side, there is a box holding the anchors for the current selected tab. Underneath the left-menu box, there is a box holding links to help-related pages. On the right of the left-menu we can see the main contents,which are loaded by AJAX calls.

L. Applications of Crawljax

As mentioned in the introduction, we believe that the crawling and generating capabilities of our approach have many applications for modern Web applications. We believe that the crawling techniques that are part of our solution can serve as a starting point and be adopted by general search engines to expose the hidden-web content induced by JAVASCRIPT, in general, and AJAX, in particular. In their proposal for making AJAX applications crawlable,15 Google proposes using URLs containing a special hash fragment, that is, #!, for identifying dynamic content. Google then uses this hash fragment to send a request to the server. The server has to treat this request in a special way and send an HTML snapshot of the dynamic content, which is then processed by Google's crawler. In the same proposal, they suggest using CRAWLJAX for creating a static snapshot for this purpose. Web developers can use the model inferred by CRAWLJAX to automatically generate a static HTML snapshot of their dynamic content, which then can be served to Google for indexing. The ability to automatically detect and exercise the executable elements of an AJAX site and navigate between the various dynamic states gives us a powerful Web-analysis and test-automation mechanism. In the recent past, we have applied CRAWLJAX in the following Web-testing domains.
(1) Invariant-based testing of AJAX user interfaces [Mesbah and van Deursen 2009],
(2) Spotting security violations in Web widget interactions [Bezemer et al. 2009] (3) Regression testing of dynamic and nondeterministic Web interfaces [Roest et al. 2010],
(4) Automated cross-browser compatibility testing [Mesbah and Prasad 2011].

M. HTTP Request Origin Identification

The main challenge of detecting the origin widget of a request is to couple the request to the DOM element from which it originated. This is not a trivial task, since HTTP requests do not carry information about the element that triggered the request. To be able to analyze HTTP requests, all requests must be intercepted. For this purpose, we pro- pose to place an HTTP proxy between the client browser and the server, which bu_ers all outgoing HTTP requests. The only way to attach information about DOM elements to an HTTP request, without a_ecting the behavior of the web server handling the request, is by adding data to the re- quest query string (e.g., ?wid=w23&requestForProxyId=123). This data should be selected carefully, to ensure it does not interfere with other parameters being sent to the server. If the request parameters contain the value of a unique at- tribute, such as the element's ID, it can be extracted and used to identify the element in the DOM. Enforcing all HTTP requests to contain a value with which the origin widget can be detected requires having mechanisms for the enforcement of a unique attribute in each DOM element, and the attachment of the unique attribute of the originat- ing element to outgoing requests. First we need to consider ways HTTP requests can be triggered in Ajax-based web applications. Static Elements. HTTP requests triggered by the src attribute of an static element, for instance in a SCRIPT or IMG element in the source code of the HTML page, are sent immediately when the browser parses them. This leaves us no time to dynamically annotate a unique value on these elements, as the requests are sent before we can access the DOM. The solution we propose is to use the proxy for inter- cepting responses as well. The responses can be adjusted by the proxy to ensure that each element with a src attribute is given a unique identifying attribute. Note that the attribute is annotated twice: in the URL so that it reaches the proxy, and as an attribute for easy identication on the DOM tree using XPath when the violation validation process is carried out.

Dynamic Elements. The src attribute of an element that is dynamically created on the client through JavaScript and added to the DOM tree, can also trigger an HTTP request. Annotating attributes through the proxy has limitations for this type of request, since elements that are added dynamically on the client-side are missed. During dynamic annotation these elements are missed as well, because the request is triggered before the element can be annotated. Because we assume every element has a unique attribute in our approach, requests triggered from dynamically generated elements can be detected easily as they do not contain a unique attribute. We believe dynamically generated elements with a src attribute are rare in modern web applications, and since this attribute should point to, for instance, a JavaScript or image, the HTTP request they trigger should be easy to verify manually by a tester. Therefore, all requests made from elements which are not annotated, should be aged as suspicious and inspected by the tester.

Ajax Calls. HTTP requests sent through an Ajax call, via the XMLHttpRequest object, are the most essential form of sending HTTP requests in modern single-page web appli- cations [2]. These requests are often triggered by an event, e.g., click, mouseover, on an element with the corresponding event listener. Note that this type of elements could also be created dynamically, and therefore proxy annotation is not desirable. Hence, we propose to dynamically annotate such elements. To that end, we annotate a unique attribute on the element right before an event is red. Note that this annotation is easiest to implement by means of aspects, as explained in Section 6. After the annotation, the attribute (and its value) must be appended to all HTTP requests that the event triggers. To that end, we take advantage of a technique known as Prototype Hijacking[17], in which the Ajax call responsible for client/server communication can be subverted using a wrapper function around the XMLHttpRequest object. Dur- ing the subversion, we can use the annotated attribute of the element, on which the event initiating the call was _red, to add a parameter to the query string of the Ajax HTTP call. It is possible that the annotated origin element is removed from the DOM by the time the request is validated. To avoid this problem, we keep track of the DOM history. After an event is red, and a DOM change is occurred, the state is saved in the history list. Assuming the history size is large enough, a request can always be coupled to its origin element, and the state from which it was triggered, bysearching the DOM history.

N. Trusted Requests

After detecting the origin widget of a request, the request must be validated to verify whether the widget was allowed to send this request. To this end, a method must be denied for specifying which requests a widget is allowed to make. Our approach uses an idea often applied in Firewall technology, in which each application has an allowed list of URLs[10]. For each widget, we can automatically create a list of allowed URLs by crawling it in an isolated environment. This way, every request intercepted by the proxy can be assigned to that specific widget. At the end of the crawling process, the proxy buyer contains all the requests the widget has triggered. This list can be saved, edited by the tester, and retrieved during the validation phase of a request. In addition, it is possible for a tester to manually ag URLs in the list as suspicious. If during the validation process a request URL does not exist in the allowed URL list of its origin widget, or if the

URL is aged as suspicious, we assume the widget does not have permission to trigger the request and thus an HTTP request violation has occurred. Assuming a request contains the annotated attribute of the origin element, Algorithm can be used to automatically detect the origin widget of the request and report HTTP request violations. Note that this approach also works for requests that do not originate from a widget, but from a non-widget element instead. By crawling the framework with only an empty widget, an allowed URL list can be created for the frame- work. A request which originates from an element that does not have a widget boundary will be validated against the allowed URL list of the overall framework.

O. Framework and Language Contributions

FORWARD facilitates the development of Ajax pages by treating them as rendered views. The pages consist of a page data tree, which captures the data of the page state at a logical level, and a visual layer, where a page unit tree maps to the page data tree and renders its data into an html page, typically including JavaScript and Ajax components also. The page data tree is populated with data from an SQL statement, called the page query. SQL has been minimally extended with (a) SELECT clause nesting and (b) variability of schemas in SQL's CASE statements so that it creates nested heterogeneous tables that the programmer easily maps to the page unit tree. A user request from the context of a unit leads to the invocation of a server-side program, which updates the server state. In this paper, which is focused on the report part of data-driven pages and applications, we assume that the server state is captured by the state of an SQL database and therefore the server state update is fully captured by respective updates of the tables of the database, which are expressed in SQL. Conceptually, the updates indirectly lead to a new page data tree, which is the result of the page query on the new server state, and consequently to a new rendered page. FORWARD makes the following contributions towards rapid, declarative programming of Ajax pages:

A minimal SQL extension that is used to create the page data tree, and a page unit tree that renders the page data tree. The combination enables the developer to avoid multiple language programming (JavaScript, SQL, Java) in order to implement Ajax pages. Instead the developer declaratively describes the reported data and their rendering into Ajax pages.

We chose SQL over XQuery/XML because (a) SQL has a much larger programmer audience and installed base (b) SQL has a smaller feature set, omitting operators such as // and * that have created challenges for efficient query processing and view maintenance and do not appear to be necessary for our problem, and (c) existing database research and technology provide a great leverage for implementation and optimization, which enables focus on the truly novel research issues without having to re-express already solved problems in XML/X- Query or having to re-implement database server functionality. Our experience in creating commercial level applications and prior academic work in the area indicate that if the application does not interface with external systems then SQL's expressive power is typically sufficient.

A FORWARD developer avoids the hassle of programming JavaScript and Ajax components for partial updates. Instead he specifies the unit state using the page data tree, which is a declarative function expressed in the SQL ex- tension over the state of the database. For example, a map unit (which is a wrapper around a Google Maps component) is used by specifying the points that should be shown on the map, without bothering to specify which points are new, which ones are updated, what methods the component covers for modifications, etc. Roadmap we present the framework in with a running example. A naive implementation of the FORWARD's simple programming model would exhibit the crippling performance and interface quality problems of pure server-side applications. Instead FORWARD achieves the performance and interface quality of Ajax pages by solving performance optimization problems that would otherwise need to be hand-coded by the developer. In particular:

Instead of literally creating the new page data tree, unit tree and html/JavaScript page from scratch in each step, FORWARD incrementally computes them using their prior versions. Since the page data tree is typically fueled by our extended SQL queries, FORWARD leverages prior database research on incremental view maintenance, essentially treating the page data tree as a view. We extend prior work on incremental view maintenance to capture (a) nesting, (b) variability of the output tuples and (c) ordering, which has been neglected by prior work focusing on homogeneous sets of tuples.

FORWARD provides an architecture that enables the use of massive JavaScript/Ajax component libraries (such as Dojo [30]) as page units into FORWARD's framework. The basic data tree incremental maintenance algorithm is modified to account for the fact that a component may not over methods to implement each possible data tree change. Rather a best-effort approach is enabled for wrap- ping data tree changes into component method calls. The net effect is that FORWARD's ease-of-development is accomplished at an acceptable performance penalty over hand-crafted programs. As a data point, revising an existing review and re-rendering the page takes 42 ms in FORWARD, which compares favorably to

WAN network latency (50-100 ms and above), and the average human reaction time of 200 ms.

IV. CHARACTERIZING COMPLEXITY

Our analysis of our measurement dataset is two-pronged. First, in this section, we analyze web pages with respect to various complexity metrics. Next, we analyze the impact of these metrics on performance. Note that our focus is on capturing the complexity of web pages as visible to browsers on client devices; we do not intend to capture the complexity of server-side infrastructure of websites [43]. We consider two high-level notions of web page complexity. Content complexity metrics capture the number and size of objects fetched to load the web page and also the different MIME types (e.g., image, javascript, CSS, text) across which these objects are spread. Now, loading www.foo.com may require fetching content not only from other internal servers such as images.foo.com and news.foo.com, but also involve third-party services such as CDNs (e.g., Akamai), analytics providers (e.g., Google analytics), and social network plugins (e.g., Facebook). Service complexity metrics capture the number and contributions of the various servers and administrative origins involved in loading a web page. We begin with the content-level metrics before moving on to service-level metrics. In each case, we present a breakdown of the metrics across different popularity rank ranges (e.g., top 1–1000 vs. 10000–20000) and across different categories of websites (e.g., Shopping vs. News). Here, we only show results for one of the vantage points as the results are (expectedly) similar across vantage points.

A. Content Complexity

Number of objects: We begin by looking, at the total number of object requests required, i.e., number of HTTP GETs issued, to load a web page. Across all the rank ranges, loading the base web page requires more than 40 objects to be fetched in the median case. We also see that a non-trivial fraction (20%) of websites request more than 100–125 objects on their landing web page, across the rank ranges. While the top 1– 400 sites load more objects, the distributions for the different rank ranges are qualitatively and quantitatively similar; even the lower rank websites have a large number of requests. Next, we divide the sites by their categories. For clarity, we only focus on the top-two-level categories. To ensure that our results are statistically meaningful, Median number of requests for objects of different MIME-types across different rank ranges. The categories that have at least 50 websites in our dataset. The breakdown across the categories shows a pronounced difference between categories; the median number of objects requested on News sites is nearly 3× the median for Business sites. We suspect that this is an artifact of News sites tending to cram in more content on their landing pages compared to other sites to give readers quick snippets of information across different news topics. Types of objects: Having considered the total number of object requests, we next consider their breakdown by content MIME types. For brevity, only the median number of requests for the four most popular content types across websites of different rank ranges. The first order observation again is that the different rank ranges are qualitatively similar in their distribution, with higher ranked websites having only slightly more objects of each type. However, we find several interesting patterns in the prevalence of different types of content. While it should not come as a surprise that many websites use these different content types, the magnitude of these fractions is surprising. For example, we see that, across all rank ranges, more than 50% of sites fetch at least 6 Javascript ob- jects. Similarly, more than 50% of the sites have at least 2 CSS objects. The median value for Flash is small; many websites keep their landing pages simple and avoid rich Flash content. These results are roughly consistent with recent independent measurements [31]. The corresponding breakdown for the number of objects requested of various content types across different categories of websites. Again, we see the News category being dominant across different content types. News sites load a larger number of objects overall compared to other site categories. Hence, a natural follow-up question is whether News sites issue requests for a proportionately higher number of objects across all content types. Therefore, for each website, we normalize the number of objects of each content type by the total number of objects for that site. The distribution of the median values of the normalized fraction of objects of various content types (not shown) presents a slightly different picture than that seen with absolute counts. Most categories have a very similar normalized contribution from all content types in terms of the median value. The only significant difference we observe is in the case of Flash objects. Kids and Teens sites have a significantly greater fraction of Flash objects than sites in other categories.

Bytes downloaded: The above results show the number of objects requested across different content types, but do not tell us the contribution of these content types to the total number of bytes downloaded. Again, for brevity, we summarize the full distribution with the median values for different website categories. Surprisingly, we find that Javascript objects contribute a sizeable fraction of the total number of bytes downloaded (the median fraction of bytes is over 25% across all categories). Less surprising is that images contribute a similar fraction as well. For websites in the Kids and Teens category, like in the case of number of objects, the contribution of Flash is significantly greater than in other categories. As in the case of the number of objects, we see no significant difference across different rank ranges. Fraction of objects accounted for by Flash objects, normalized per category.

B. *Service Complexity*

Anecdotal evidence suggests that the seemingly simple task of loading a webpage today requires the client-side browser to connect to multiple servers distributed across several administrative domains. However, there is no systematic understanding of how many different services are involved and what they contribute to the overall task. To this end, we introduce several service complexity metrics. Number of distinct servers: the distribution across websites of the number of distinct webservers that a client contacts to render the base web page of each website. We identify a server by its fully qualified domain name, e.g., bar.foo.com. Across all five rank ranges, close to 25–55% of the websites require a client to contact at least 10 distinct servers. Thus, even loading simple content like the base page of websites requires a client to open multiple HTTP/TCP connections to many distinct servers. News sites have the most number of distinct servers as well. Number of non-origin services: Not all the servers contacted in loading a web page may be under the web page provider's control. For example, a typical website today uses content distribution networks (e.g., Akamai, Limelight) to distribute static content, analytics services (e.g., google-analytics) to track user activity, and advertisement services (e.g., doubleclick) to monetize visits. Identifying non-origins, however, is slightly tricky. The subtle issue at hand is that some providers use multiple origins to serve content. For example, yahoo.com also owns yimg.com and uses both domains to serve content. Even though their top-level domains are different, we do not want to count yimg.com as a non-origin for yahoo.com because they are owned by the same entity. To this end, we use the following heuristic. We start by using the two level domain identifier to identify an origin; e.g., x.foo.com and y.foo.com are clustered to the same logical origin foo.com. Next, we consider all two-level domains involved in loading the base page of www.foo.com, and identify all potential non-origin domains (i.e., two-level domain not equal to foo.com). We then do an additional check and mark domains as belonging to different origins only if the authoritative name servers of the two domains do not match [33]. Because yimg.com and yahoo.com share the same authoritative name servers, we avoid classifying yimg.com as having a different origin from yahoo.com.

C. *Authors and Affiliations*

Dr Akash Singh is working with IBM Corporation as an IT Architect and has been designing Mission Critical System and Service Solutions; He has published papers in IEEE and other International Conferences and Journals.

He joined IBM in Jul 2003 as a IT Architect which conducts research and design of High Performance Smart Grid Services and Systems and design mission critical architecture for High Performance Computing Platform and Computational Intelligence and High Speed Communication systems. He is a member of IEEE (Institute for Electrical and Electronics Engineers), the AAAI (Association for the Advancement of Artificial Intelligence) and the AACR (American Association for Cancer Research). He is the recipient of numerous awards from World Congress in Computer Science, Computer Engineering and Applied Computing 2010, 2011, and IP Multimedia System 2008 and Billing and Roaming 2008. He is active research in the field of Artificial Intelligence and advancement in Medical Systems. He is in Industry for 18 Years where he performed various role to provide the Leadership in Information Technology and Cutting edge Technology.

V. References

[1] Dynamics and Control of Large Electric Power Systems. Ilic, M. and Zaborszky, J. John Wiley & Sons, Inc. © 2000, p. 756.

[2] Modeling and Evaluation of Intrusion Tolerant Systems Based on Dynamic Diversity Backups. Meng, K. et al. Proceedings of the 2009 International Symposium on Information Processing (ISIP'09). Huangshan, P. R. China, August 21-23, 2009, pp. 101–104

[3] Characterizing Intrusion Tolerant Systems Using A State Transition Model. Gong, F. et al., April 24, 2010.

[4] Energy Assurance Daily, September 27, 2007. U.S. Department of Energy, Office of Electricity Delivery and Energy Reliability, Infrastructure Security and Energy Restoration Division. April 25, 2010.

[5] CENTIBOTS Large Scale Robot Teams. Konoledge, Kurt et al. Artificial Intelligence Center, SRI International, Menlo Park, CA 2003.

[6] Handling Communication Restrictions and Team Formation in Congestion Games, Agogino, A. and Tumer, K. Journal of Autonomous Agents and Multi Agent Systems, 13(1):97–115, 2006.

[7] Robotics and Autonomous Systems Research, School of Mechanical, Industrial and Manufacturing Engineering, College of Engineering, Oregon State University

[8] D. Dietrich, D. Bruckner, G. Zucker, and P. Palensky, "Communication and computation in buildings: A short introduction and overview," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3577–3584, Nov. 2010.

[9] V. C. Gungor and F. C. Lambert, "A survey on communication networks for electric system automation," *Comput. Networks*, vol. 50, pp. 877–897, May 2006.

[10] S. Paudyal, C. Canizares, and K. Bhattacharya, "Optimal operation of distribution feeders in smart grids," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4495–4503, Oct. 2011.

[11] D. M. Laverty, D. J. Morrow, R. Best, and P. A. Crossley, "Telecommunications for smart grid: Backhaul solutions for the distribution network," in *Proc. IEEE Power and Energy Society General Meeting*, Jul. 25–29, 2010, pp. 1–6.

[12] L. Wenpeng, D. Sharp, and S. Lancashire, "Smart grid communication network capacity planning for power utilities," in *Proc. IEEE PES, Transmission Distrib. Conf. Expo.*, Apr. 19–22, 2010, pp. 1–4.

[13] Y. Peizhong, A. Iwayemi, and C. Zhou, "Developing ZigBee deployment
guideline under WiFi interference for smart grid applications," *IEEE Trans. Smart Grid*, vol. 2, no. 1, pp. 110–120, Mar. 2011.

[14] C. Gezer and C. Buratti, “A ZigBee smart energy implementation for energy efficient buildings,” in *Proc. IEEE 73rd Veh. Technol. Conf. (VTC Spring)*, May 15–18, 2011, pp. 1–5.
[15] R. P. Lewis, P. Igic, and Z. Zhongfu, “Assessment of communication methods for smart electricity metering in the U.K.,” in *Proc. IEEE PES/IAS Conf. Sustainable Alternative Energy (SAE)*, Sep. 2009, pp. 1–4.
[16] A. Yarali, “Wireless mesh networking technology for commercial and industrial customers,” in *Proc. Elect. Comput. Eng., CCECE*,May 1–4, 2008, pp. 000047–000052.
[17] M. Y. Zhai, “Transmission characteristics of low-voltage distribution networks in China under the smart grids environment,” *IEEE Trans. Power Delivery*, vol. 26, no. 1, pp. 173–180, Jan. 2011.
[18] V. Paruchuri, A. Durresi, and M. Ramesh, “Securing powerline communications,” in *Proc. IEEE Int. Symp. Power Line Commun. Appl., (ISPLC)*, Apr. 2–4, 2008, pp. 64–69.
[19] Q.Yang, J. A. Barria, and T. C. Green, “Communication infrastructures for distributed control of power distribution networks,” *IEEE Trans. Ind. Inform.*, vol. 7, no. 2, pp. 316–327, May 2011.
[20] T. Sauter and M. Lobashov, “End-to-end communication architecture for smart grids,” *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1218–1228, Apr. 2011.
[21] K. Moslehi and R. Kumar, “Smart grid—A reliability perspective,” *Innovative Smart Grid Technologies (ISGT)*, pp. 1–8, Jan. 19–21, 2010.
[22] Southern Company Services, Inc., “Comments request for information on smart grid communications requirements,” Jul. 2010
[23] R. Bo and F. Li, “Probabilistic LMP forecasting considering load uncertainty,” *IEEE Trans. Power Syst.*, vol. 24, pp. 1279–1289, Aug. 2009.
[24] *Power Line Communications*, H. Ferreira, L. Lampe, J. Newbury, and T. Swart (Editors), Eds. New York: Wiley, 2010.
[25] G. Bumiller, “Single frequency network technology for fast ad hoc communication networks over power lines,” WiKu-Wissenschaftsverlag Dr. Stein 2010.
[31] G. Bumiller, L. Lampe, and H. Hrasnica, “Power line communications for large-scale control and automation systems,” *IEEE Commun. Mag.*, vol. 48, no. 4, pp. 106–113, Apr. 2010.
[32] M. Biagi and L. Lampe, “Location assisted routing techniques for power line communication in smart grids,” in *Proc. IEEE Int. Conf. Smart Grid Commun.*, 2010, pp. 274–278.
[33] J. Sanchez, P. Ruiz, and R. Marin-Perez, “Beacon-less geographic routing made partical: Challenges, design guidelines and protocols,” *IEEE Commun. Mag.*, vol. 47, no. 8, pp. 85–91, Aug. 2009.
[34] N. Bressan, L. Bazzaco, N. Bui, P. Casari, L. Vangelista, and M. Zorzi, “The deployment of a smart monitoring system using wireless sensors and actuators networks,” in *Proc. IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, 2010, pp. 49–54.
[35] S. Dawson-Haggerty, A. Tavakoli, and D. Culler, “Hydro: A hybrid routing protocol for low-power and lossy networks,” in *Proc. IEEE Int. Conf. Smart Grid Commun. (SmartGridComm)*, 2010, pp. 268–273.
[36] S. Goldfisher and S. J. Tanabe, “IEEE 1901 access system: An overview of its uniqueness and motivation,” *IEEE Commun. Mag.*, vol. 48, no. 10, pp. 150–157, Oct. 2010.
[37] V. C. Gungor, D. Sahin, T. Kocak, and S. Ergüt, “Smart grid communications and networking,” Türk Telekom, Tech. Rep. 11316-01, Apr 2011.