Codifying Medicare Using Computable Contracts for Improved Understandings of Medical Insurance

Butch Nasser
Department of Computer Science
Stanford University
ban@stanford.edu

Michael Genesereth
Department of Computer Science
Stanford University
genesereth@stanford.edu

***Abstract*—The modern health insurance industry is fundamentally one of contracts: insurance policies exist as natural language texts that both define the conditions upon which an insuree can receive payments for certain services and also outline what amount of payment can be expected. Because the nature of health insurance contracts is "all-encompassing," arriving at a complete understanding of the conditions under which a person is covered by such a contract is a task that proves difficult for everyday individuals who lack expertise in understanding legal texts. Computable contracts, or codified representations of contracts that match the logical representation of a contract's terms and conditions under the eye of the law, present the everyday individual with an opportunity to access a more comprehensive understanding of an insurance contract's coverage. To make insurance contracts more accessible and understandable, we need to consider a codified representation of insurance contracts into computable contracts that represents the logical components behind the written natural language. This paper outlines the codification process of the Ambulance Services Chapter of Medicare to reflect on the scalability of computable contracts and the feasibility of the encoding process in impacting the health insurance industry. We propose guidelines for an open approach to encoding publicly available contracts such as Medicare. We also evaluate possible and necessary next steps for encoding computable insurance contracts.**

***Index Terms*—computable contract, Logic Programming, Epilog, health insurance, Computational Law**

I. Introduction

Health insurance is confusing. The modern health insurance industry has locked itself into a complicated state of archaic documents and inefficient processes. Attempts to improve this status to one of digital cohesion have been in the works for decades, [**?**][1] but these attempts to create digital representations of these contracts have encountered a prominent bottleneck: the wording and enforceability of health insurance policies.

Fundamentally, a health insurance policy is a contract written in natural language (such as English) with definitions stating the conditions, exclusions, and exemptions upon which coverage may or may not be provided. These policy documents can easily exceed hundreds of pages in length, and often get grouped with a "summary of benefits," providing an overview of the contract but lacking the specific conditions necessary for someone to determine the potential coverage status of services without filing an insurance claim. General confusion and widespread distribution of relevant information leads not only to a slowdown of claims processing for insurance agencies, but also a reliance on summaries of benefits that lack accurate detail of coverage. Likewise, this practice results in confusing and time-consuming labor for the insuree.

To move the insurance industry towards a more understandable future, insurance agencies must consider the transition of insurance policies into *computable contracts*: an approach of rewriting the natural language of insurance policies into logical programming languages. Through such rewriting, we can establish a definition of the contract using code, such that a computer can understand and evaluate claims, whilst also giving immediate, accurate feedback to users on the factors that led to the evaluated outcome.

Computable contracts benefit insurance agencies and the insuree alike: they automate the process of following the contract's structure to determine whether a claim is obligated to receive payment (a process known as *automated claims analysis*). Such progression would both accelerate the process by which insurance claims are handled by insurance companies, as well as provide insurees a distinctive understanding of what circumstances are covered before the occurrence of such circumstances—or possibly, even before choosing to purchase a selected insurance policy altogether. Computable contracts in general also benefit from both streamlined comparison processes to determine if policies are subsets of another (known as *Containment Testing*), as well as group compilation methods to quickly determine coverages when considering multiple types of policies acting in coalition (known as *Portfolio Analysis*)[3].

The Computable Contracts approach applies well to encoding insurance policies in general, and particularly well for health insurance contracts: compared to other sectors of contract law, health insurance contracts specifically benefit from the legal doctrine of *contra proferentem*[9], under which rulings concerning ambiguous contract wordings typically find against the contract drafter. In other words, health insurance contracts are written primarily for an attorney: to guarantee a drafter legal protection under their contractual obligations, rather than to ensure general understanding. This consideration has historically led the health industry to tend towards the long, all-encompassing contracts that exist today. While this history acts as a consideration for why health insurance documents require codification with more urgency compared to other contracts, it also situates health insurance policy documents as optimal for codification, as such policies lack the legal obscurity or undefined jargon that may be more prominent in other legal documents and contracts.

Furthermore, most insurance policies exist as a collection of benefits, which for the most part can act independently of one another: people need not concern themselves with coverage for medication while they are looking into the coverage of surgeries, for example. This nature of health insurance policies allows for the modularization of contracts for codification such that an insurance policy can be broken down into components representing a single type of service (for example, just surgeries) and then codified in this state. This process permits the accelerated verification of a policy's encoding and contributes towards improved scalability of contract codification, as certain services will match a similar encoding structure across different policies.

Given these considerations, we chose Medicare[1] as the policy of encoding interest for four primary reasons: first, the sheer scale of Medicare would result in the most significant number of individuals benefiting from a single policy codification; second, as a federal policy, the majority of services are standardized across states or include specific notification of state-specific distinction, leading to a surprisingly generalizable encoding requiring minimal modifications to maximize coverage area; third, as a federal health insurance program, access to the complete wording beyond the summary of benefits is easily attainable, which is not the case for all health insurance policies; lastly, barring certain cost-proportions for coverage that change annually, many chapters of Medicare witness minimal changes, posing them as strong candidates for encodings that can serve users without constant need for encoding updates.

Additionally, we specifically chose Medicare's subchapter concerning coverage of Ambulance Services as the area of interest due to its relevance for individuals with Medicare, and its specific inclusion of "revenue codes," or detailed cost expectations dependent on the distance and locale of the designated service. Because of this, such a codification of coverage will be highly beneficial to insurees, and would represent an example of a complex policy for future reference. While this paper takes a binary approach to determining coverage (whether *any* coverage/payment would be present based on claim conditions), full cost determination poses an excellent, relevant extension upon this paper's work within reach, which would also push the limits of the current status of computable medical insurance contracts.

This paper quickly overviews the syntax of Epilog, the chosen language for encoding. It then discusses the specific constructive elements of insurance contracts, and how these elements prime insurance contracts for codification into a logical programming language. Afterward, we spend the bulk of the paper demonstrating specific portions of an encoding of Medicare's Ambulance Services coverage. This demonstration will be accompanied by reflection on the process of this codification, including the difficulties encountered, to act as guidance for future work in computable contracts for health insurance and beyond. The paper concludes with an analysis of related work in the field, and with relevant considerations and guidelines for encoders undergoing the task of computable contract encoding.

[1]For the purposes of this paper, Medicare refers to Medicare Parts A/B (also known as Original Medicare), which details the mandatory coverages for all Medicare-affiliated service centers as standardized across the United States. Medicare Parts C (Medicare Advantage) and D, which include optional, supplier-specific services, are worthy to encode additionally, but not within the intended scope of this paper.

II. Background

In order to understand the encodings and methods provided in this paper, we first provide an overview of Logic Programming and the encoding language, Epilog. We review first key elements of a contract and second how specific contract verbiage translates well to logical programming languages. These two subsections are tailored to provide a detailed understanding for those already familiar with software development. Lastly, we outline the structure of Medicare as a contract, and make specific note of the subsections of the Ambulance Services chapter encoded.

A. Logic Programming and Epilog

Epilog is a declarative programming language based on Dynamic Logic Programming, which allows for a higher level of logical abstraction and elimination of lower-level specifications to reach a desired logical outcome compared to imperative languages such as Python or other Object-Oriented programming languages. Epilog obeys the rules of declarative statements that notably have two key types of data to manage: *facts*, which represent innate properties of individual objects or relations between two objects; and *rules*, which consist of conditions that arrive at a conclusion, but have the ability to take in variable objects.

For certain design tasks, Epilog allows for increased parallelization of relational comparisons implicitly without mandating cognizant definitions of algorithms or methodologies, ultimately providing very efficient evaluations at runtime. Additionally, Epilog allows for the indexing of symbols, such that an evaluation will only ever reference segments of code relevant to a certain query's inputs, ignoring all other elements. These advantages contribute to further efficient code evaluation.

For an example of syntax, an Epilog representation of the statement: "Person X is the father of Person Y if they are the Parent of Person Y and they are male." In Epilog, this could be written as shown in Fig. 1.

```
father(X, Y):
    parent(X, Y) &
    male(X)
```

Fig. 1. Simple Epilog Syntax Example for "Person X is the father of Person Y if they are the Parent of Person Y and they are male."

```
f(X,Y):
    p(X, Y) &
    m(X)
```

Fig. 2. Alternate symbolization of Fig. 1, dependent on metadata.

When provided two inputs, father(X, Y) would either be true or false, depending on the outputs. One thing to note is that the symbols and words used have no semantic meaning until defined by metadata. For example, this could easily be rewritten as shown in Fig. 2.

Another note is that variables are typically denoted as starting with a capital letter, using camel case

```
VariableName
```

whereas facts or objects are typically denoted in lowercase, using snake case.

```
is_fact
```

Creation and maintenance of the set of existing facts (world data) and a description of possible attribute relations of variables (metadata) is relevant to producing usable encodings using Epilog or any Logic Programming Languages. With that said, Epilog presents itself as a language that is relatively simple for somebody to gain proficiency in, even if they otherwise lack programming experience.

B. Contract Verbiage

Before discussing the specific policy undertaken, we present an explanation for the different components of an insurance contract that make them amenable to Logic Programming and Epilog encodings. Notably, while there are many ways to define an insurance contract, a common definition, and the one we prefer for forming rules, consider an insurance contract's nature as divisible into the following components:

- General Conditions
- Exclusions
- Exemptions (from Exclusions)

General conditions are considered the conditions that must be true for coverage to be provided. For example, some general conditions of the relatively simple Capital One Visa Signature Card Benefits insurance policy are outlined below (specifically, for their *Lost Luggage Reimbursement*[8]).

> *You are eligible if You charge a covered trip to Your valid, Visa card issued in the United States. Only Your Checked Luggage or Carry-on Baggage is covered.*

In this example, some conditions must be true to receive coverage, namely that

1) You charged the trip to your card.
2) Your card is valid.
3) Your card is a Visa card.
4) Your card was issued in the United States.
5) Your claim is for Checked Luggage or Carry-on Baggage.

From the first 4 bullets, many conditions were contingent upon an object (Your card) and a selection of its attributes (its usage to pay for the trip, its validity, its status as a Visa card, and its issuance country). We can rewrite this as the shown in Fig. 3.

```
eligible(Claim, Card) :-
    trip_charged_to(Claim, Card)     &
    valid(Card)                      &
    is_visa(Card)                    &
    issued_in_united_states(Card)    &
    claim_for_checked_or_carryon(Claim)
```

Fig. 3. General Eligibility Determination for Visa Signature Card, Lost Luggage Reimbursement

In summary, General Conditions are mainly written as a set of related logical intersections, wherein all of the facts must be true in order for the condition to be met.

Exclusions act to automatically eliminate an item from being covered by the insurance contract, even if all other general conditions are met. For example, the above policy has the following exclusions listed under "What items or losses are not covered?:" *Business Items, cellular telephones,* or *art objects*. One way to write this exclusion, which may be intuitive to programmers experienced in imperative languages, is the shown in Fig. 4.

```
excluded(Item) :-
    business_item(Item)          ||
    cellular_telephone(Item)     ||
    art_object(Item)
```

Fig. 4. Exclusions for Visa Signature Card, Lost Luggage Reimbursement

However, the lines within a relation must be analyzed in order, meaning that if the Item is an art object, one must first check that it was not a business item or cellular telephone before the

```
excluded(Item)
```

rule can be recognized as true. Instead, creating multiple truthful definitions of the rule preserves faster evaluation of the relation due to Epilog's use of Indexing [7], and is the encouraged approach to such conditions (Fig. 5).

```
excluded(Item) :-
    business_item(Item)

excluded(Item) :-
    cellular_telephone(Item)

excluded(Item) :-
    art_object(Item)
```

Fig. 5. Adjusted Exclusions Definition for Visa Signature Card, Lost Luggage Reimbursement

If we assume that the Item being requested is an attribute of the claim, adding this exclusion to the other general conditions arrives at the encoding shown in Fig. 6.

```
eligible(Claim, Card) :-
    trip_charged_to(Claim, Card)         &
    valid(Card)                          &
    is_visa(Card)                        &
    issued_in_united_states(Card)        &
    claim_for_checked_or_carryon(Claim)  &
    claim.item(Claim, Item)              &
    !excluded(Item)
```

Fig. 6. Expanded determination for Visa Signature Card, Lost Luggage Reimbursement

That is, you are eligible to receive reimbursement if all of the general conditions are met, and no eliminating exclusions are met. Finally, there are *exemptions*, which are conditions that may allow a general exclusion to be permissible. For example, if the Visa policy permitted excluded items to be eligible if and only if you had a valid police report filed for the item, another alternate definition of the *eligible* claim may be as shown in Fig. 7.

```
eligible(Claim, Card) :-
    trip_charged_to(Claim, Card)         &
    valid(Card)                          &
    is_visa(Card)                        &
    issued_in_united_states(Card)        &
    claim_for_checked_or_carryon(Claim)  &
    claim.item(Claim, Item)              &
    excluded(Item)                       &
    valid_police_report_filed(Item)
```

Fig. 7. Additional Extension determination for Visa Signature Card, Lost Luggage Reimbursement

Note that the presence of *both* Fig. 6. and Fig. 7 above under a policy encoding would be necessary to account for all possible mixes of general conditions, exclusions, and exemptions that would be eligible for a reimbursement under the policy. Also note that often, policies present a general exclusion in a format highly similar to a general condition: in those cases, exclusions can be inverted to be presented as a general condition. An example of this is shown in Methodology, section ii. Fig. 3-7 and 8 showcase how we can reformulate an insurance policy into a collection of rules and facts that take in inputs and determine whether or not those specific inputs are covered under the chosen policy.

Furthermore, insurance contracts are often internally divided into separate categories or chapters of covered items or services, which allows for clear modularization of coverage relations without needing significant unnecessary information: if someone is filing a claim to cover a medical surgery, they need not concern themself with whether or not they have coverage for prescription medications, even if both are covered under the same insurance policy. A claim analyzer for the insurance policy would restrict themself only to the relevant services under the policy for that specific claim, so we can encode entire policies as a collection of services, requesting similar input to that of a paper form, while providing more clarity on why certain claims are covered or not based on the user's input.

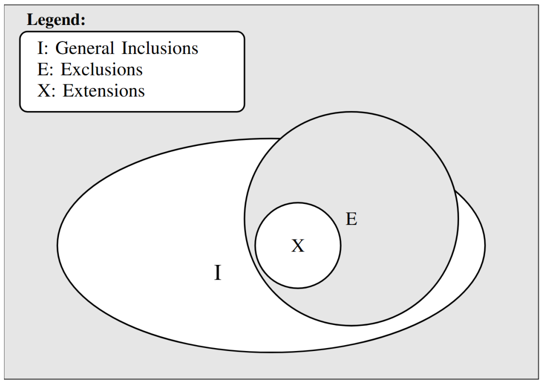


Fig. 8. Coverage Chart: White area shows the space where coverage applies, and the gray area shows where a claim would not be covered.

Such claims analysis is performed using the policy's complete documents, which can easily exceed hundreds of pages in length, and typically get grouped into a summary of general conditions and exclusions without specific details or definitions for clarification. The mismatch between the level of detail in a schedule and the actual policy is a large source for misunderstanding one's insurance coverage; encoding a policy as a computable contract allows for the condensing of all related facts into a single unified location. This unification allows a user to better understand how the intricate details of a policy beyond that of the summary affect their claim in particular.

C. Overview of Medicare Policy and Ambulance Services Chapter

With the considerations for the structure of insurance contracts in general, Medicare yields specific benefits and affordances that position it as a worthy policy to encode. Notably, the complete policy wordings and claims processing manuals are publicly available, which is not the case for most private medical insurance policies. Medicare matches the structure of other medical insurance policies, consisting of over 17 chapters of coverage categories across over 900 pages of policy specifications. The chapters vary from 7 to 301 pages in length; Chapter 10: Ambulance Services, occupies 34 pages, placing it at approximately the median length of the 17 chapters[6].

A schedule of this chapter can be found at cms.gov[5]. This short summary of coverage information includes some core general conditions, exclusions, and exemptions, but lacks the detail necessary to adequately determine coverage in all cases. The full conditions of coverage are contained under

Chapter 10 of the Medicare policy documentation, with 5 major subcategories of general coverage conditions.

Section 10.1 details the Vehicle and Crew Requirement for the responding ambulance, which contains coverage details for use by Medicare Administrative Contractors, form information received either by manual evaluation or statements of certifying requirements provided by ambulance suppliers. We propose computable contracts as beneficial to accelerate the process of claims analysis for both insurers and claim analysis contractors, so choose to encode these subsections for fullness.

Section 10.2 details Necessity and Reasonableness, with definitions that are properly fulfilled by the insurance codes filled out by ambulance staff during provision of service. However, with a computable contracts system, insuree users can supply information related to such conditions to self-determine coverage.

Section 10.3 specifies validity of The Destination, which is dependent on their locality and their Point of Pickup. This section contains both general exclusions and exemptions upon these exclusions, and also relies on spreadsheet data to determine coverage.

Section 10.4 includes Air Ambulance Services, a section of the general conditions with more exceptions, and more considerations for cost. Since this paper takes a binary approach to coverage, this section is encoded to allow for verification of the presence of general conditions, but with the concept of future cost considerations noted, without implementation.

Section 10.5 shortly describes Joint Responses, in which multiple Ambulances respond and provide transportation service to the insuree. This section acts as an exemption from the general exclusion that only a single ambulance that provides transportation from the Point of Pickup to the Destination can be covered.

Each section contains up to 10 subsections further specifying general conditions, exclusions, and exemptions. As will be discussed in the next section, care was taken to note the translations of the encoding to the policy wording and structure, such that any revisions to the policy can be quickly adjusted in an encoding.[2] However, during the encoding process, many conditions are found to exist as subsets of other conditions, or possess conditional relevance upon other sections outside of the order written in the policy document. As a result, the ordering of the evaluated logical representation differs from the policy document, but specifications of their original section and subsections have been included.

Throughout the chapter, the individual receiving care is labeled as the "Beneficiary " of the policy, or as the "patient" for select medical services. We shall solely refer to such an individual as a Beneficiary, but note that the person filing a claim need not be the Beneficiary of the claim's coverage.

[2]Note that this is done due to the current nature of insurance policies as written in natural language. In the future, we envision the logical representation to be the initial representation which is then translated to natural language.

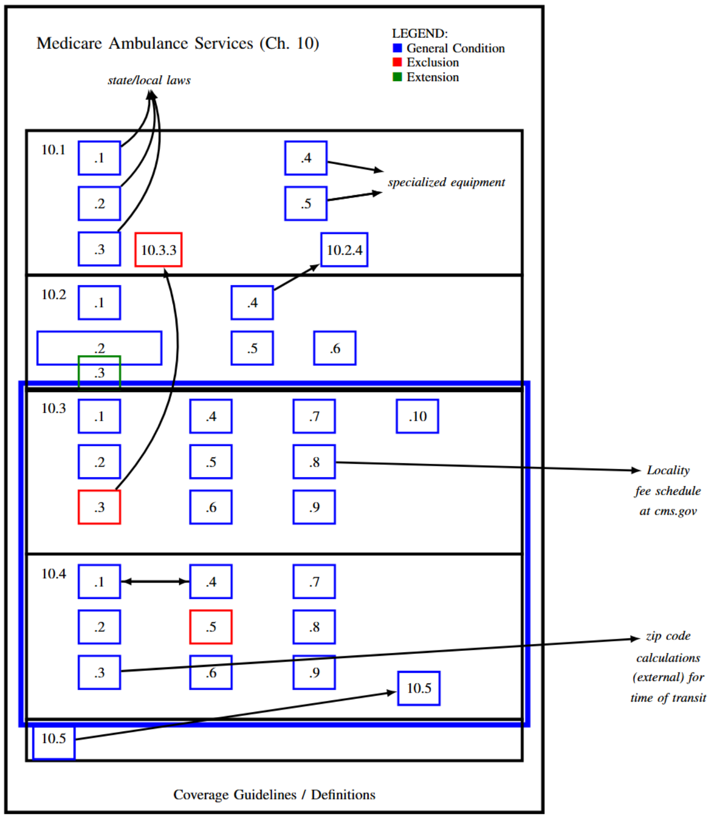


Fig. 9. Diagram representing the sections of Medicare Chapter 10, including references to external material outside of the chapter. Many sections influence or affect the conditions of other sections in the form of exclusions or extensions, and certain subsections can be written as subsets of others. It is not to scale with respect to individual sizes of each section, but acts to give a general idea of the interwoven nature of these contracts, as well as the wide distribution of necessary information necessary to understand and encode them.

III. Methodology of Codifying Medicare Ambulance Coverage

This section details the process of translating a contract into a computable contract such that encoders and programmers can replicate our methods and develop computable contracts themselves. For general readers, the conclusion includes a summary of this methodology with reduced technical emphasis.

A. *Considerations/Assumptions*

The encoding of the Ambulance Services subsection of the Medicare policy was done with the following considerations:

1) The encoding followed a *covered(Claim)* relation that will only ever be implicitly applied for Ambulance claims under the Medicare Policy (i.e. no other services nor other policies will be accounted for, and no explicit error checking is implemented in the encoding to verify inputs obey this assumption)
   a) Likewise, a user checking the requirements for this policy is assumed to be acting upon claims with a

valid Beneficiary during times that the policy is in effect (i.e. the policy is fully paid for the claim period, with no gaps of coverage).

2) The encoding was designed to be compliant with the upload requirements of the CodeX Insurance Analyst for usage by users who are not experts at reading legal texts.
    a) Note that error checking from the prior bullet is thus accounted for since a user would have to deliberately choose the policy and service for which they wish to file a claim.
3) The encoding shall properly represent the policy for all of its coverage requirements, including those beyond the knowledge of an insuree.
    a) This consideration verifies the usability of the policy encoding for claims analysis on behalf of insurance companies and their contractors.
4) Above all, accuracy of the policy took precedence. Any uncertainty or ambiguity in the encoding or policy document should be noted to allow for future verification of such conditions.

B. Initial Approach/Analysis

Since our chosen subsection or Medicare (Ambulance Services) is Chapter 10 of the Medicare Benefit Policy Manual, certain abbreviations used are not defined, as they are defined in prior chapters. For cases such as these, whenever an abbreviation was used without a definition within the policy, we searched the cms.gov web page for the abbreviation, and verified that any present definitions aligned with the context of this chapter. As terms appear within the chapter, the tail-end of the chapter, Section 30.1, includes definitions for terms used within the chapter in approximately the same relative order that they appear under the policy text in Section 10. We began approaching the encoding by gathering a larger understanding of how the chapter's subsections interacted with one another, and verifying understanding of any terms, definitions, or necessary outside materials in order to achieve a complete representation of the policy.

This approach resulted in adequate preparation and allowed for avoiding redundant encoding of certain rules. For example, sections 10.2.6 and 10.4.9 both include clauses that alter the coverage specifications dependent on the time of Beneficiary death, with different preconditions for arrival dependent on if the responding ambulance was a land/water ambulance versus an air ambulance (Fig. 10).[3] As such, some unification of related rules and facts is possible. Steps were taken to unify rules whenever there were no significant differences in the considerations for the cases.

Beyond such similarities, encoding was generally done with consideration for each of the 5 major subsections of the chapter, in the order that sections and subsections were presented. If, at a later point, an approach to streamline the rules became evident, the encoding was adjusted accordingly.

[3]* in Fig. 10 denotes conditions under which *some* coverage would be granted. In reality, the amount of the covered payments would differ, but the scope of this paper only seeks to present a determination if any coverage is attainable.

```
(1) death_consideration(C) :-
    claim.beneficiary_died_during_claim_timeframe(C, no)

(2) death_consideration(C) :-
    claim.beneficiary_died_during_claim_timeframe(C, yes)   &
    claim.ambulance_type(land_or_water_ambulance)           &
    death_accounted_for(C)

(3) death_consideration(C) :-
    claim.beneficiary_died_during_claim_timeframe(C, yes)   &
    claim.ambulance_type(air_ambulance)                     &
    death_accounted_for_air(C)

death_accounted_for(C) :-
    claim.beneficiary_death_condition(C, before_loaded)*

death_accounted_for(C) :-
    claim.beneficiary_death_condition(C, before_arrival)*

death_accounted_for_air(C) :-
    claim.beneficiary_death_condition(C, before_loaded)     &
    ~claim.was_there_reasonable_opportunity_to_abort(C, yes)&
    claim.air_ambulance_taken_off(C, yes)

death_accounted_for_air(C) :-
    claim.beneficiary_death_condition(C, before_arrival)*
```

Fig. 10. Sections 10.2.6 and Section 10.4.9 cover considerations for whether or not a Beneficiary dies before arrival at a hospital. Acting as mutually exclusive but fully covered cases, the related rules can be streamlined between the two, regardless of the proximity within the Chapter. For terms of coverage, (1) "In general, if the beneficiary dies before being transported, then no Medicare payment may be made." Instead of writing this as an exception, the inversion of such was written to present a general condition: "If the beneficiary does not die before being transported, then Medicare payment may be made (if no other exclusions are met)." (2) handles the case of a beneficiary dying in the case of a land or water ambulance. (3) handles the case of a beneficiary dying in the case of an air ambulance. These conditions were mapped from a table of scenarios that would be applied in a flowchart fashion.

C. General Process

After analysis, we considered a structure for the encoding. Notably, there would be one covered relation that took in a claim, and all necessary information for the claim would be described as an attribute of the claim (such as claim.arranger). In the case of our encoding, this is presented as in Fig. 11:

```
covered(C) :-
    meets_minimum_ambulance_service_requirements(C) &
    meets_ambulance_service_requirements(C)
```

Fig. 11. Main covered relation: The claim must meet the minimum requirements, and then meet the more detailed, subsection requirements.

Generally, certain techniques were derived from experience encoding insurance policies. This includes the modularization of each section (10.X) into a general rule, and then including each subsection of that section (10.X.1-10.X.Y) as another rule that must be true for the initial rule to evaluate to true. This was generally followed, with additions in places where a subsection logically acts independently of its parent section.

Another technique utilized was for the encoding of tables/flowchart structures in Logic Programming: using the

```
meets_ambulance_service_requirements(C) :-
  claim.beneficiary_onboard_during_all_claim_times(C, yes) &
  % 10.2.5 overall exception
  meets_vehicle_crew_requirement(C)               & % 10.1
  meets_arrangement_requirements(C)               & % 10.1.4
  meets_necessity_reasonableness_requirement(C) & % 10.2
  death_consideration(C)                          & % 10.2.6/4.9
  meets_destination_requirements(C)               & % 10.3
  valid_ambulance_service(C)                      % 10.4
```

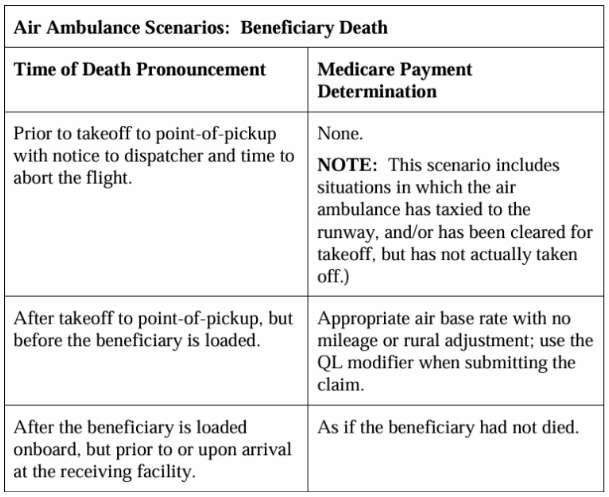
Fig. 12. The overall requirements were written as a logical AND of all of the conditions from the key subsections 10.1-10.4 (the absence of 10.5 shall be explained later). Additionally, any subsection that included general exclusions not logically related to the rest of the subsection has been extracted, such that their independent impact as an exclusion is more present.

example from Fig. 5, considering Air Transport as described in Section 10.4.9 (Fig. 13).

```
death_consideration(C) :-
  claim.beneficiary_died_during_claim_timeframe(C, yes) &
  claim.ambulance_type(air_ambulance)                   &
  death_accounted_for_air(C)

death_accounted_for_air(C) :-
  claim.beneficiary_death_condition(C, before_loaded)   &
  ~claim.was_there_reasonable_opportunity_to_abort(C, yes)* &
  claim.air_ambulance_taken_off(C, yes)

death_accounted_for_air(C) :-
  claim.beneficiary_death_condition(C, before_arrival)
```



Air Ambulance Scenarios: Beneficiary Death	
Time of Death Pronouncement	**Medicare Payment Determination**
Prior to takeoff to point-of-pickup with notice to dispatcher and time to abort the flight.	None. **NOTE:** This scenario includes situations in which the air ambulance has taxied to the runway, and/or has been cleared for takeoff, but has not actually taken off.)
After takeoff to point-of-pickup, but before the beneficiary is loaded.	Appropriate air base rate with no mileage or rural adjustment; use the QL modifier when submitting the claim.
After the beneficiary is loaded onboard, but prior to or upon arrival at the receiving facility.	As if the beneficiary had not died.

Fig. 13. Section 10.4.9 Table For this table, coverage determination would depend on a flowchart. Logically, flowcharts are followed as a combination of mutually exclusive cases, so we encode each case separately and ensure that the case is only entered if the mutually exclusive facts have occurred. This allows flowcharts to be followed with fewer data dependencies.

Generally, for all sections and subsections, the natural language included in the document was encoded using the following approach:

1) Identify the key input variables that might be needed to evaluate in this section.
2) Determine the possible range of values for each input variable, and determine which of these values matches the policy wording for coverage or exclusion.
3) Consider paradigms for certain presentations of the information. For example, tables as mutually exclusive definitions of the same rule, as described above.
4) Cite any uncertain facts that may impact a rule. Such cases require either further lookup, investigation, or an added attribute to our claim that allows for proceeding with analysis, noting knowledge that certain components ”assumed to be compliant” may be found to not be compliant.
5) Review the related Conditions and Review Actions outlined in the Coverage Guidelines, for additional information that may clarify considerations
6) Review the Claims Processing Manual for the chapter for similar reasons as 5.
7) Cite the location in the document that the encoded rules correspond to (using comments).

Steps 1-3 were followed to allow for proper logical translations of the natural language. Step 4 exists due to the uncertainty present in most contracts—even if minimal for the health insurance industry, considerations that, for example, depend on state or locality would require necessary encodings for each of these areas; until each location's determinations/requirements are added to the encoding, an attribute that encompasses this fact's truthfulness should be added as a placeholder.

Steps 5-6 include a sanity-check verification of an encoding using information for Ambulance Service Claims Processors, in a checklist (flowchart) format. This section provides no conflicting or additional rules but includes some clarification of ambiguity, including but not limited to (non-exhaustive) examples of some qualifications for certain conditions.

Step 7 was added due to the ever-evolving nature of widely applied insurance policies. Although the Ambulance Services chapter of Medicare has not had an issued adjustment since 2018,[4] such adjustments typically are done to certain subsections. Such commenting facilitates the determination of impacts that changes have upon the entire encoding, and modularizing rules to approximately match these subsections allows for minimal necessary adjustment of existing encodings if policy changes occur.

D. Metadata and Verification

During the encoding process, the modularization of each subsection allows for easier verification of encodings in a compartmentalized fashion. Each subsection was verified using Sierra, a browser-based Integrated Development Environment (IDE) for Epilog. Sierra permits the creation of mock rulesets and datasets that can be traced through, returning the evaluated output of a query of a certain ruleset on a specific dataset. Each rule was tested upon completion, and then testing of the complete dataset was also performed for further verification. This process is recommended while still in the debugging process.

Verification using Sierra can be done with objects and attributes alone, but the presentation of claims in a more accessible format under the CodeX Insurance Analyst requires the creation of metadata for the encoding. As discussed

[4]excluding annual changes to localities and fee schedules, which are imported in as spreadsheet data, and does not affect the logical structure of the encoding

earlier, the attributes and relation do not garner meaning until metadata defines them. During the encoding process, after the creation of a new attribute that would be necessary as user input, metadata for that attribute was added. Generally, a medical insurance policy has a few core relations that would be necessary (Fig. 14).

```
% Core Relations
type(claim, class)
type(medicare_ambulance_service_coverage, class)
superclass(medicare_ambulance_service_coverage, claim)

% Attribute Relations
attribute(medicare_ambulance_service_coverage, claim.arranger)
type(claim.arranger, attributerelation)
domain(claim.arranger, claim)
codomain(claim.arranger, arranger)
unique(claim.arranger, yes)
total(claim.arranger, yes)
changestyle(claim.arranger, selector)
... (repeat for every attribute)
% Standard types
type(boolean,class)
type(number,class)
% Claim-specific types
type(arranger, class)
type(participating_hospital, arranger)
type(skilled_nursing_facility, arranger)
type(home_health_agency, arranger)
type(other, arranger)
```

Fig. 14. Metadata Relations: `medicare_ambulance_service_coverage` is the denomination assigned to the specific claim being analyzed. Then, we assign attribute relations for every attribute/input field requested from a user. The final attribute, `changestyle`, is only necessary for the input fields for testing on the CodeX Insurance Analyst and does not in itself affect the functionality of the encoding using Sierra

During development, we often adjusted the name of an attribute to better reflect its semantic meaning or to avoid confusion between similarly named attributes; likewise, we would occasionally remove an attribute to combine the attribute's class with that of an already-existing input field. To simplify the creation of metadata, we created a Python script that would parse our rules for any attribute relations and would generate the attribute relations for those present. This simplified the process of code modification and development.

E. Final Encoding Results

This methodology resulted in the construction of an encoding consisting of 56 rules, 76 necessary attributes, 100 facts for the world, and over 600 lines of metadata necessary for describing the situations in which Ambulance Services would be at least partially covered under Medicare. The encodings are deployed at the live version of the CodeX Insurance Analyst (insurance.stanford.edu). There are certain components (such as state or regional-specific laws) that take presumptive compliance in terms of a boolean, which are noted in the final encoding.

Versions of the encoding exist that solely include information available to an insuree/Beneficiary, and which include all the information necessary to evaluate a claim for Medicare Administrative Contractors, resulting in a tool that assists both parties: it simplifies claim understanding for insurees and accelerates the claims analysis process for claims adjudicators. This culminates as a product where users can instantly see the causes for a claim to not be covered, or get a concrete, justified explanation of why conditions of coverage are fully met.

Our modularized approach allowed the for proper codification of a subchapter of the gargantuan policy that is Medicare; from this, it is not unreasonable to consider all of Medicare to be codifiable in similar fashion, paving the way for the simplification of one of the highest-impact, health insurance plans in the United States.

IV. Difficulties Encountered During Development

As we only set out to check the feasibility of the computable contracts approach, we encountered difficulties during our development and modified our approach as we learned from them. We provide an overview of our largest difficulties and offer general guidance on how to avoid similar difficulties when choosing to encode insurance policies using the computable contracts approach.

A. Overhauling of Attribute and Objects Schema

Initially, the encoding took in a claim, and its attributes were objects such as an *ambulance, beneficiary*, etc. These objects would themselves have attributes: an ambulance would have a *point of pickup* and a *destination*, and a *destination* would have attributes of its *locality* and *type* (such as a hospital or beneficiary's home). Conceptually, this decision was made early on to allow for the storing of objects relevant to a user within the CodeX Insurance Analyst system, allowing a user to reference previously used information if relevant to multiple claims. However, during the development process, we recognized that the additional layers of objects and the resulting differences in rule arity caused unnecessary confusion and were prone to syntax errors if encoding approaches changed. We also realized that the necessary input fields for each encoding were 1) high in quantity for certain objects and 2) not guaranteed to have much reusability across different encodings. As such, modifications were made to make all rule relations unary, taking in solely the claim, and all attributes became attributes of the claim; any attribute that was a sub-attribute of an object retains an implied attribute of another object, with no functional impact.

For future encoders, we encourage the use of unary relations related to the claim, with all attributes being unique attributes of a claim. This structure not only more accurately represents how a claim would be filed, but simplifies the analysis process without introducing unnecessary complication. Our encountered difficulties acted as the inciting incident for the creation of the Python script to adjust metadata, to automatically reflect rule changes, since a majority of the 76 attribute relations had to be restructured.

B. Necessity of Outside Information in Informing Policy Knowledge

As expressed in Fig. 9, a significant quantity of information relevant to encoding this chapter was not available in the policy document, and was instead available elsewhere online, requiring encoders to track it down. This presented a major delay in the encoding process, as finding documents is sometimes tedious, and then verifying the validity of found documents (such as having the most up-to-date ones) can also present a difficult task. Additionally, for a federal policy such as Medicare, some documents were retired during the encoding process, without links to updated information; in these cases, investigative attempts proved extremely difficult. However, once the relevant documents were located, encoding was allowed to proceed more smoothly.

We encourage future encoders to locate all relevant exterior documents prior to starting an encoding. These documents influenced the schema of our encoding, and also verified the ability of using external data sources (such as APIs or spreadsheets) instead of manually encoding such information. Having a complete picture of the encoding allows the most-educated approach in implementing it. The widespread locations of necessary information is a core problem with the insurance industry today, and a major argument for the collection of such information in a single, solidified location within a logic program representation.

Additionally, there exists a conflict between the information necessary to evaluate a claim, and the information an insuree is reasonably able to discern for themselves. To tackle this difficulty, we considered any information relevant solely to a medical professional (doctor, ambulance staff, etc.) to be outside of the reasonable knowledge of an insuree. This results in an encoding that lists the mandatory requirements for a Claims Processor, but grants an insuree an option to ”assume” such standards are met in evaluating their claim, with the note that if such an assumption turns out to be false, coverage would not be provided. This is still an active difficulty with encoding health insurance policies, and we hope to alleviate this difficulty in future work.

V. Conclusion

To modernize the insurance industry, we must transition from the lawyer-centered natural language representations of insurance contracts in favor of logical, easily understandable computable contracts. Computable contracts possess the ability to overcome the hardships related to the layman's understanding of the large, archaic documents that are health insurance policies. Through a process of modularization and relevant attribute definition, a health insurance policy can be procedurally minimized such that each necessary condition, exclusion, or extension can be represented through Logic Programming and then exported to a user interface that simplifies the process of both understanding and evaluating health insurance claims. This process can create a representation of a health insurance policy amenable to people lacking expertise in understanding legal texts.

This paper discussed the foundations of Logic Programming, contract representation, and Medicare's structure before detailing the specific structure and codification of Medicare Ambulance Services policy. In covering what is one of the most extensive and encompassing insurance contracts within the United States, we showcase that the computable contracts approach can properly represent the legal definition of such a contract, and propose the widespread transition to adopt computable contracts as the primary representation of medical insurance contracts, with the natural language of the contract coming afterward. With the successes of the manual processes described in this paper, our next steps include encoding the remaining 15 chapters of Medicare. We also plan to add full cost determination functionality to our encodings, instead of the binary coverage simplification of our current methodology.

While this paper covered a novel approach to encoding large-scale medical insurance contracts, the general approach of creating and deploying computable contracts is not a new or small movement. Researchers in other countries share the vision and have developed domain-specific Natural Language extensions upon Logic Programming languages to simplify the understanding of encodings of their own insurance contracts[2]. Other researchers at CodeX are striving to complete encodings for Stanford University-specific policies, such as Cardinal Care and NorCal Kaiser HMO policies. Notably, researchers are investigating the use of breakthrough Large Language Models (LLMs) to automatically develop Logic Programming encodings from natural language texts, with verification from human engineers.

The process of modularization outlined in this paper allows for the simple distribution of the separate components of a policy to provide a natural scalability of encodings due to minimal interacting components and reusability of developed ontologies. With this in mind, if a policy as intimidating as Medicare can be fully encoded, there is no reason to believe that there exists any health insurance policy that cannot follow suit; with the relevant insurance company partnerships, we can increase the accessibility of even private policies' details to the extent of the federally regulated Medicare. Encoders and insurers alike should recognize the unique opportunity that computable contracts create to redefine, simplify, and streamline the insurance industry as we know it. A representation of a contract purely in logical programming language would relieve us of the archaic standard of hundred-page documents requiring a scattered collection of external information. By transforming medical insurance policies into logical, computable contracts, we can create a foundation for an insurance industry where clarity and accessibility become standard; the most logical path to healthcare comprehension is through a logical representation.

Acknowledgment

Special thanks to Preston Carlson for advising the early stages of this paper and work. Also, thanks to all members of Stanford's Computational Law Research Group for lending active assistance for the creation and development of this research.

References

urlb1 J. Cummins and C. D. Clack, “Transforming commercial contracts through computable contracting,” *Journal of Strategic*

Contracting and Negotiation, vol. 6, no. 1, pp. 3–25, Feb. 2022. doi:10.1177/20555636211072560. b2 Cummins, John and Dávila Quintero, Jacinto and Kowalski, Robert and Ovenden, David. (2025). Computable Contracts for Insurance: Establishing an Insurance-Specific Controlled Natural Language - InsurLE. 1-31. b3 P. Carlson and M. Genesereth, "Insurance portfolio analysis as containment testing," *Frontiers in Artificial Intelligence and Applications*, Dec. 2023. doi:10.3233/faia230957. b4 O. R. Goodenough and P. J. Carlson, "Words or code first? Is the legacy document or a code statement the better starting point for complexity-reducing legal automation?," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 382, no. 2270, Feb. 2024. doi:10.1098/rsta.2023.0160. b5 "Ambulance services," Medicare, Available: https://www.medicare.gov/coverage/ambulance-services. [Accessed Mar. 19, 2025]. b6 "100-04 Medicare Claims Processing Manual," CMS.gov, Available: https://www.cms.gov/regulations-and-guidance/guidance/manuals/internet-only-manuals-ioms-items/cms018912. [Accessed Mar. 19, 2025]. b7 M. Genesereth, *Logic Programming*. Available: http://logicprogramming.stanford.edu/stanford/index.php. [Accessed Mar. 19, 2025]. b8 "Visa signature card," Available: https://ecm.capitalone.com/WCM/card/visa-signature-guide-to-benefits-revised.pdf. [Accessed Mar. 20, 2025]. b9 C. Cole and R. Henry, "An analysis of interpretation of insurance contracts: Common law versus strict *Contra proferentem*," *Journal of Insurance Regulation*, 2023. doi:10.52227/23499.2017.