Formalizing and Reasoning about Supply Chain Contracts between Agents*

Dylan Flynn[1], Chasity Nadeau[1], Jeannine Shantz[1], Marcello Balduccini[1], Tran Cao Son[2], and Edward R. Griffor[3]

[1] Saint Joseph's University, Philadelphia, PA, USA
{df752850,cnadeau,js486075,mbalducc}sju.edu
[2] Department of Computer Science, New Mexico State University, Las Cruces, NM, USA
stran@nmsu.edu
[3] National Institute of Standards and Technologies, MD, USA
edward.griffor@nist.gov

Abstract. Inspired by the recent problems in supply chains, we propose an approach to declarative modeling of contracts between agents that will eventually support reasoning about resilience of and about ways to improve supply chains. Specifically, we present a high-level language for specifying and reasoning about contracts over action domains of agents. We assume that the behavior of the agents can be formally expressed through action theories and view a contract as a collection of constraints. Each constraint specifies the responsibility of an agent to achieve a certain result by a deadline. Each agent also has a mapping between constraints and the agent's *concerns*, i.e. issues that the agent is concerned about, which are modeled in accordance with the CPS Framework proposed by the National Institute of Standards and Technology. We discuss how common questions related to the fulfillment of a contract or the concerns of the agents can be answered and computed via Answer Set Programming.

Keywords: Specifying and reasoning about contracts · Supply chain · Cyber-Physical System Framework · Answer Set Programming .

1 Introduction

Supply chains have historically been optimized with respect to costs and other specific attributes, including the provisioning of materials, manufacturing processes, and distribution logistics. This high degree of optimization makes supply chains inherently brittle, in that their optimized network of exchanges is sensitive to sudden or extreme changes in demand. Alarming demonstrations of this brittleness have been experienced during the COVID-19 pandemic, i.e. with the supply chain's inability to respond to the surge in demand for masks and ventilators. As these recent events demonstrated, supply chains nowadays constitute a widely distributed and critical infrastructure with legs into

* Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreement 70NANB21H167. Son Tran was also partially supported by the NSF grants 1812628 and 1914635.

economy and public welfare. Finding effective methods of curbing their brittleness in response to sudden changes and surges in demand is thus paramount.

In this paper, we report on our progress in an investigation into methodologies that are ultimately aimed at making supply chains more resilient, and specifically for the identification of critical dependencies and for the evaluation, verification and restoration of properties of the supply chain.

As a first step, this paper proposes an approach to declarative modeling of supply chains that views a supply chain as a collection of contracts between agents, where a contract is a set of constraints. Each constraint specifies the responsibility of an agent to achieve a certain result by a deadline. The approach aims at describing, and reasoning about, the evolution of the state of the supply chain over time in response to events. Thus, we assume that the behavior of the agents can be formalized declaratively through action theories.

Typically, a supply chain involves a multitude of stakeholders (e.g., C-suite positions but also people involved in different level of supply, production, inventory, etc.) with substantially different types of expertise and goals. For this reason, a critical challenge with designing and managing resilient supply chains is that one needs not only to clearly identify all the interdependencies among the relevant elements, but also to formalize them in such a way that their relevance and ramifications are understandable by multiple stakeholders regardless of their different views.

To overcome this challenge, we build upon the notion of *concern* from the CPS Framework proposed by the National Institute of Standards and Technology (NIST) [3]. Our approach provides each agent with a mapping between constraints and the agent's concerns, i.e. issues that the agent is concerned about. While resilience may have many faces, we hypothesize that these faces can be captured within the concerns provided by the CPS Framework. An important reason that led us to rely on the CPS Framework is that it is designed to enable meaningful and grounded discussion among stakeholders from different backgrounds and with different objectives – supported in particular by a rich hierarchy of broadly-applicable concerns. This ensures broad applicability of our approach to a variety of types of supply chains, problems, and mix of stakeholders. All in all, our approach consists in viewing a supply chain as a large and complex CPS, and in capturing the supply chain's interdependencies by means of the elements of the NIST CPS Framework.

Because resilience is a broad and multi-faceted topic, as a start in this paper we focus on the three reasoning problems, which we view as fundamental stepping stones towards reasoning about resilience: *Contract feasibility*: given an initial state of the world, can a given contract be successfully executed? (If that is not the case, then the contract is set up for failure.) *Clause satisfaction check*: assuming that the contract has been in execution for some time, has any agent violated any clause of the contract? If so, then how can the problem be mitigated? *Concern satisfaction check*: assuming that the contract has been in execution for some time, which (and whose) concerns are not satisfied? What can be done to mitigate the problem? Besides the formalization of the problem and of the reasoning mechanisms, we also discuss the implementation of our approach, which is based on the declarative knowledge representation formalism of Answer Set Programming (ASP) [10,12].

2 Preliminaries

2.1 The NIST CPS Framework

An important challenge with supply chains is that stakeholders of varying backgrounds may use different terminology when discussing a supply chain and likely have different, possibly even conflicting, goals, which can lead to challenges under normal circumstances. When the unexpected occurs, disruptions in communication and conflicts among objectives may be magnified, making it more difficult to ensure resilience. Establishing a structure incorporating primitives that promote a common vocabulary and meaningful, grounded discussion among stakeholders may mitigate risks. To create a "common foundation", we propose viewing the supply chain as a large, complex Cyber Physical System (CPS) and leveraging the NIST CPS Framework as a lens through which we can look at a supply chain. With this framework the processes surrounding developing, verifying and delivering products can be formalized and more easily understood by a diverse group of stakeholders [3]. By design, the scope of the CPS Framework is very broad so that it may be adopted by a broad range of applications.

The NIST Framework for Cyber-Physical Systems, referred to as "NIST CPS Framework" or simply "Framework" below, comprises a set of concerns and facets related to the system under design or study. This section briefly clarifies the intent and purpose of the framework. The interested reader is directed to SP 1500-201, SP 1500-202 and SP 1500-203, available on the NIST website.

The CPS Framework provides the taxonomy and methodology for designing, building, and assuring CPS that meet the expectations and concerns of system stakeholders, including engineers, users, and the community that benefits from the system's functions. The concerns of the Framework are represented in a forest, where branching corresponds to the decomposition of concerns. We refer to each tree as a *concern tree* of the CPS Framework. The concerns at the roots of this forest are called *aspects*. For instance, the sub-concerns of the Trustworthiness aspect are Privacy, Reliability, Resilience, Safety, and Security. In turn, the Security concern has sub-concerns Cybersecurity and Physical Security, and the Cybersecurity concern has sub-concerns Confidentiality, Integrity and Availability. The Framework comprises nine aspects. In this paper, we will mainly focus on Business, Functional, and Trustworthiness. A concern about a given system reflects consensus thinking about method or practice, involved in addressing the concern, and in some cases consensus-based standards describing that method or practice. Associated with each concern is a set of *requirements* that address the concern in question. For example, in a CPS that stores personally identifiable information, the system's designers may agree that the requirement to use encrypted memory addresses the Confidentiality concern. Because the Confidentiality concern is a descendant of the Trustworthiness aspect, this requirement, together with other relevant ones, addresses the CPS's Trustworthiness aspect as well. The dependencies among concerns and between requirements and concerns can be formally represented by means of an ontology. Leveraging the ontology, tasks related to reasoning about the satisfaction of concerns can be reduced to: (a) identifying which requirements are satisfied in the current state of the system and which ones are not, and (b) propagating this information up the concern forest, ultimately determining the satisfaction of the aspects. For details on

this approach, we refer the interested reader to [11]. For the purpose of this paper, it is sufficient to mention the existence of algorithms for determining whether a requirement or concern γ is satisfied given the ontology $\mathcal{O}$ and a current state s of the CPS. Below, we will write $\mathcal{O} \cup s \models \gamma$ to denote that γ is satisfied under s.

2.2 Action Language $\mathcal{B}$

An action domain in the action language $\mathcal{B}$ [6] is defined over two disjoint sets, a set of actions **A** and a set of fluents **F**. A *fluent literal* is either a fluent $f \in \mathbf{F}$ or its negation $\neg f$. A *fluent formula* is a propositional formula constructed from fluent literals. An action domain is a set of laws of the following form:

$$\textit{Executability condition}: \quad \textbf{executable } a \textbf{ if } \varphi \tag{1}$$

$$\textit{Dynamic law}: \quad a \textbf{ causes } \psi \textbf{ if } \varphi \tag{2}$$

$$\textit{Static Causal Law}: \quad \psi \textbf{ if } \varphi \tag{3}$$

where ψ and φ are fluent formulas and a is an action. Intuitively, an executability condition of the form (1) states that a can only be executed if φ holds. (2), referred to as a *dynamic causal law*, states that ψ is caused to be true after the execution of a in any state of the world where φ is true. (3) represents a *static causal law*, i.e., a relationship between fluents. It conveys that whenever the fluent formula φ holds then so is ψ. For an action domain D, we denote the set of laws of the form (3) by K .

Let D be a domain. A set of fluent literals is said to be *consistent* if it does not contain f and $\neg f$ for some fluent f. An *interpretation* I of the fluents in D is a maximal consistent set of fluent literals of D. A fluent f is said to be true (resp. false) in I iff $f \in I$ (resp. $\neg f \in I$). The truth value of a fluent formula φ in I is defined recursively over the propositional connectives in the usual way. $I \models \varphi$ indicates that φ is true in I.

Let u be a consistent set of fluent literals and K a set of static causal laws. We say that u is closed under K if for every static causal law "ψ **if** φ" in K, if $u \models \varphi$ then $u \models \psi$. By $Cl_K(u)$ we denote the least consistent set of literals from D that contains u and is also closed under K. It is worth noting that $Cl_K(u)$ might be undefined. For instance, if u contains both f and $\neg f$ for some fluent f, then $Cl_K(u)$ cannot contain u and be consistent; another example is that if $u = \{f, g\}$ and K contains both "f **if** h" and "$\neg h$ **if** f, g" then $Cl_K(u)$ does not exist because it has to contain both h and $\neg h$, which means that it is inconsistent. For a formula η, $Cl_K(\eta)$ denotes the set $\{Cl_K(u) \mid u \text{ is a set of fluent literals such that } u \models \eta\}$. Formally, a *state* of D is an interpretation of the fluents in **F** that is closed under the set of static causal laws K of D.

An action a is *executable* in a state s if there exists an executability proposition **executable** a **if** φ in D such that $s \models \varphi$. The *direct effect of an action a* in a state s is the set $e(a, s) = \bigwedge_a$ **causes** ψ **if** $_{\varphi \in D, s \models \varphi}\, \psi$. For a domain D, $\Phi_D(a, s)$, the set of states that may be reached by executing a in s, is defined as follows: (*i*) if a is executable in s, then $\Phi_D(a, s) = \{s' \mid s' \text{ is a state and } s' \in Cl_K\left(e(a, s) \wedge \bigwedge_{l \in s \cap s'} l\right)\}$; and (*ii*) if a is not executable in s, then $\Phi_D(a, s) = \emptyset$. Φ_D is unique for each domain D and is called the *transition function of* D.

Given a domain D, an alternate sequence of states and actions $\alpha = s_0 a_0 s_1 \ldots a_{n-1} s_n$, where s_i's are states and a_i's are actions, is a *trajectory* over the domain D if $s_{i+1} \in$

$\Phi_D(a_i, s_i)$ for every $i = 0, \ldots, n-1$. We say that n is the length of α and s_0 is the starting state of α. Furthermore, α satisfies a fluent formula φ over the set of fluents in D_A, denoted by $\alpha \models \varphi$, if s_n satisfies φ.

3 A Motivating Scenario

Let us consider a scenario involving two agents: XYZ Homes and Lumber Yard A. XYZ Homes aims at building a certain number of homes and contracts with Lumber Yard A to provide suitable lumber.

Example 1 (A Contract Between XYZ Homes and Lumber Yard A). XYZ Homes builds eight to nine 2,000 square foot homes each month for new home buyers. Each new home requires 16,000 board feet of Number 2 Common grade lumber. In order to complete eight to nine homes, XYZ Homes must purchase 144,000 board feet of Number 2 Common grade lumber each month. Lumber Yard A is the preferred supplier of this lumber.

In the first part of our scenario, we look at the agreement XYZ Homes contracts with Lumber Yard A for the required lumber. The agreement specifies the responsibilities of each agent. It is formalized as a set of constraints on how the work is to be conducted. These constraints can be viewed as requirements (in the sense of the CPS Framework), and each requirement is mapped to one or more of an agent's concerns. A sample of these constraints and concerns includes:

1. Lumber Yard A will produce a total of 144,000 board feet of lumber for XYZ Homes. This constraint addresses the functionality concern of XYZ Homes.
2. Lumber Yard A guarantees to schedule the transport and delivery of 14-16 tractor trailers worth of lumber in one month to XYZ Homes. This constraint addresses the time to market concern.
3. The lumber delivered to XYZ Homes will be at or above Number 2 Common grade. This constraint addresses several concerns including physical, reliability, quality and trustworthiness. For example, if Lumber Yard A were to provide lumber that is of a lesser quality than Number 2 Common grade, then from XYZ Home's perspective, Lumber Yard A would no longer be trustworthy.
4. The agreed upon cost of lumber is at $122,000 for 144,000 board feet and the transport and delivery cost will be at or below $500,000 for 144,000 board feet. This constraint addresses the cost concern.

In the context of a contract, agents will normally have to execute actions to fulfill their commitments. For instance, Lumber Yard A has to *produce* 144,000 board feet and *deliver* them to XYZ Homes. On the other hand, XYZ Homes has to pay for the board, etc. We therefore proposed to formalize contacts between agents where each agent is associated with an action domain. Intuitively, the agent's domain describes the actions that the agent can execute, when can an action be executed, and what are the effects of an action. We believe that the action language $\mathcal{B}$ is sufficiently expressive enough for us to represent domains in supply chains.

In using $\mathcal{B}$, we can easily encode *functional fluents* that frequently occurred in action domains of agents involved in supply chain. For example, the number of board feet of

lumber that a company possessed is a functional fluent whose domain is between 0 and 5000.

For later use, we encode a simple set of actions for Lumber Yard A and XYZ Homes below. Lumber Yard A has an abstract action for producing lumber and it is represented as follows:

$$produce(X, Z) \textbf{ causes } \exists Y.[Y \leq X : board(Y, Z)]$$

Here, $board(Y, Z)$ is a functional fluent denoting that Y board feet of lumber of quality Z is available for Lumber Yard A.

Lumber Yard A also has the following action:

$$deliver(Y, Z) \textbf{ causes } delivered(Y, Z)] \wedge board(X - Y, Z) \textbf{ if } \quad board(X, Z), Y \leq X$$

This law states that if Lumber Yard A delivers Y board feet of lumber of quality Z, then they are delivered ($delivered(Y, Z)$) if at least Y board feet of lumber are available ($board(X, Z) \wedge Y \leq X$). As a result of the action, there will be $X - Y$ board feet available after the delivery. Observe that the action domain can be refined to include, for example, the specific detail on shipping such as the need for tractor trailers, the capacity of the trailers, etc.

XYZ Homes needs to receive the board and pay. These actions are represented below

$$receive(X, Z) \textbf{ causes } available\_board(X, Z) \textbf{ if } \quad delivered(X, Z)$$

which says that the company would have X board feet of lumber of quality Z if it executes the action $receive(X, Z)$. This action can only be executed if the said amount of board feet of lumber is delivered. In addition, XYZ Homes also has the action

$$pay(X, C) \textbf{ causes } payment(X, C) \wedge available\_funds(Y - X) \textbf{ if } \quad available\_funds(Y), X \geq Y$$

where C is either $board$ or $shipping$. The law says that XYZ Homes pays an amount X for the category C and the available fund will be reduced by X. This effect is achieved only if sufficient funds are available to XYZ Hones.

The above representation encodes the actions and their effects under normal circumstances. We will assume that for each action a in our discussion, the executability condition of a is of the form

$$\textbf{executable } a \textbf{ if } \neg ab(a), \varphi$$

which, intuitive, says that a can be executed whenever φ is true and $ab(a)$ is false where $ab(a)$ denotes that some abnormal condition under which a cannot be executed. For example, for the action $produce(X, Y)$, we have

$$\textbf{executable } produce(X, Z) \textbf{ if } \quad ab(produce(X, Z))$$

In the following, we denote by D_L and D_H the action domains of Lumber Yard A and XYZ Homes, respectively.

4 Formalizing a Contract

We now introduce $\mathcal{L}_c$, a high-level language for the specification of contracts between agents with focus on supply chain management. The language is built on a set of agents and the action domains associated with these agents. In formalizing a contract, we assume that each agent is aware of the state of the world and can observe the changes within its environment that it is interested in. To each agent, the contract has two facets, the public part encodes the agreement between the agent and another agent, while the private part details its concerns. For example, the contract between XYZ Homes and Lumber Yard A contains:

- the statement "Lumber Yard A will produce a total of 144,000 board feet of lumber for XYZ Homes". This is a public part of the contract that is known to both parties;
- the statement that the above is a constraint addressing the functionality concern is a private part of the contract (that both sides happen to agree upon);
- the statement "the lumber delivered to XYZ Homes will be at or above Number 2 Common grade" is related to the trustworthiness concern of XYZ Homes; it is not necessarily related to a concern of Lumber Yard A.

Observe that each of the above clauses specifies a goal, the agent responsible for the achievement of the goal, and the deadline. Motivated by this observation, we develop $\mathcal{L}_c$ as follows. From now on until the end of this section, we assume two fixed agents A and B, whose action domains are D_A and D_B, respectively.

4.1 Syntax of $\mathcal{L}_c$

We formalize the public part of a contract between A and B over D_a and D_B using clauses of the form:

$$\textit{ref\_id}: \quad agent \quad \textbf{responsible\_for} \quad goal \quad \textbf{when} \quad time\_expression \tag{4}$$

Intuitively, a clause of the form (4) is associated with a reference identifier *ref_id* and says that $agent \in \{A, B\}$ is responsible for achieving $goal$ within the time constraint specified by $time\_expression$, where:

- $goal$ is a fluent formula constructed over fluents appearing in $D_A \cup D_B$; and
- a time constraint is a simple temporal expression of one the following forms:

$$always \mid eventually \mid per\_unit[n \ldots m] \mid by\_unit\ n \tag{5}$$

  where $unit$ can be any time unit such as day, week, etc., n and m are integers, $n \leq m$, and $[n \ldots m]$ denotes the range $[n, n+1, \ldots, m]$.

Given the public part of a contract C between A and B, the private part of C for either agent A or B is represented by statements of the form

$$\textit{ref\_id}: \quad \rho \tag{6}$$

where *ref_id* is a reference identifier C and ρ is a requirement. We assume that ontology $\mathcal{O}$ associates each requirement with one or more concerns (of the agent) from the concern forest defined in the CPS Framework, or any customized concern forest specific to the agent. Formally, a contract between two agents[4] is defined as follows:

Definition 1. *A contract $\mathcal{C}$ between two agents A and B constructed over two actions domains D_A and D_B is a triple (C, P_A, P_B) where*

- *C is a set of clauses of the form* (4)*; and*
- *P_A (resp. P_B) is a set of statements of the form* (6) *for A (resp. B).*

Intuitively, (C, P_A) or (C, P_B) is the contract under A or B's perspective, respectively, and it will be used by A or B to evaluate the progress of the contract. Observe that A (resp., B) does not necessarily know about P_B (resp., P_A).

Example 2. Let L and H denote Lumber Yard A and XYZ Homes, respectively. The public part of the contract in Example 1 is encoded by the following clauses:

$$C1 : L \textbf{ responsible\_for } board(144K, Q) \wedge 1 \leq Q \textbf{ when } by\_week\ 4 \tag{7}$$

$$C2 : L \textbf{ responsible\_for } delivered(144K, Q) \wedge 2 \leq Q \textbf{ when } by\_week\ 4 \tag{8}$$

$$C3 : H \textbf{ responsible\_for } payment(122K, board) \textbf{ when } by\_week\ 4 \tag{9}$$

$$C4 : H \textbf{ responsible\_for } \exists X \leq 500K.[payment(X, shipping)] \textbf{ when } by\_week\ 4 \tag{10}$$

The first clause $C1$ states that L is responsible for producing 144K board feet of lumber by the end of the project (week 4). $C2$ says that L needs to deliver, i.e., responsible for the shipping of, 144K board feet of lumber of quality 2 or greater (also by week 4). $C3$ and $C4$ indicate that H must pay \$122K for the board feet of lumber and for shipping, respectively, and the cost of shipping must be no greater than \$500K[5].

The links between clauses and each agent's requirements are encoded by the following sets of statements:[6]

- $P_L = \{C2 :$ *match-customer-expected-grade*$, C4 :$ *receive-due-payment*$\}$ where requirement *match-customer-expected-grade* addresses the Quality concern from the CPS Framework's Business aspect, and requirement $receive-due-payment$ addresses the Reliability concern under the Trustworthiness aspect and the Cost concern from the Business aspect.

[4] Throughout the paper, we only discuss contracts between two agents but the formalization is easily adapted for contracts among multiple agents.

[5] Observe that we have simplified the contract slightly as there is no mention about the tractor trailers. This can be easily encoded if we extend the action domains of H and L to consider the shipping company.

[6] A thorough discussion on requirements is beyond the scope of this paper. Thus, for compactness, we use short requirement names, although in practice a requirement would be spelled out in more details, e.g. the requirement that here we call *match-customer-expected-grade* would likely be expressed by a statement "lumber shall be produced in a grade matching the customer's expectations."

- $P_H = \{C1$: *sufficient-material-for-building*, $C2$: *material-safe-for-building*, $C2$: *material-sufficiently-durable*, $C4$: *promptly-send-payment*, $C3$: *acceptable-shipping-cost*$\}$ *sufficient-material-for-building* addresses the TimeToMarket concern (Business aspect), *material-safe-for-building* addresses Safety and Reliability (both under the Trustworthiness aspect), *material-sufficiently-durable* addresses Performance (Functional aspect), *promptly-send-payment* addresses Policy (Business aspect), and *acceptable-shipping-cost* addresses Cost (also under the Business aspect).

Observe that concerns are not symmetrical between agents.

4.2 Semantics of $\mathcal{L}_c$

Given a contract $\mathcal{C} = (C, P_A, P_B)$ between two agents A and B. The semantics of $\mathcal{L}_c$ is defined over pairs of trajectories over D_A and D_B of the form (H_A, H_B). For simplicity of the presentation, we assume that the action $wait$, which can always be executed and has no effect, belongs to every domain. Therefore, whenever we refer to two trajectories H_A and H_B over D_A and D_B, respectively, without the loss of generality, we will assume that H_A and H_B have the same length.

Given D_A and D_B, a *joint state s over D_A and D_B* (or, *joint state*, for short) is an interpretation over the set of fluents in $D_A \cup D_B$ that is closed with respect to the set of static causal laws in $D_A \cup D_B$. For a joint state s over $D_A \cup D_B$, by s_A (or s_B) we denote the restriction of s over the fluents in D_A (or D_B), respectively. Obviously, s_A (s_B) is a state in D_A (D_B). The truth value of a formula φ over the language of $D_A \cup D_B$ in a joint state s is defined as usual. We will next define the satisfaction of a contract given (H_A, H_B). To do so, we need the following notion.

Definition 2. *Given two agents A and B whose action domains are D_A and D_B, respectively, and two trajectories $H_A = s_0^A a_0^A \dots s_{n-1}^A a_{n-1}^A s_n^A$ over D_A and $H_B = s_0^B a_0^B \dots s_{n-1}^B a_{n-1}^B s_n^B$ over D_B, we say that H_A and H_B are* compatible *if $s_i^A \cup s_i^B$ is a joint state for every $i = 0, \dots, n$.*

The satisfaction of a clause is defined next.

Definition 3. *Given two compatible trajectories $H_A = s_0^A a_0^A \dots s_{n-1}^A a_{n-1}^A s_n^A$ in D_A and $H_B = s_0^B a_0^B \dots s_{n-1}^B a_{n-1}^B s_n^B$ in D_B, a clause*

$$\textit{ref\_id} : x \ \textbf{\textit{responsible\_for}}\ \varphi\ \textbf{\textit{when}}\ \ time\_exp$$

is satisfied *by (H_A, H_B), denoted by $(H_A, H_B) \models$ ref_id, if*

- *φ is true in every $s_i = s_i^A \cup s_i^B$ for $i = 0, \dots, n$ when $time\_exp$ is $always$; or*
- *φ is true in $s_i = s_i^A \cup s_i^B$ for some $i = 0, \dots, n$ when $time\_exp$ is $eventual$; or*
- *φ is true in $s_i = s_i^A \cup s_i^B$ for $i = u, \dots, l$ when $time\_exp$ is per_unit $[u \dots l]$; or*
- *φ is true in $s_k = s_k^A \cup s_k^B$ when $time\_exp$ is by_unit k.*

We say that ref_id is violated *by (H_A, H_B) if $(H_A, H_B) \not\models$ ref_id.*

Building on the above definition, the satisfaction of a contract is defined as follows.

Definition 4. *Given two compatible trajectories* $H_A = s_0^A a_0^A \dots s_{n-1}^A a_{n-1}^A s_n^A$ *in* D_A *and* $H_B = s_0^B a_0^B \dots s_{n-1}^B a_{n-1}^B s_n^B$ *in* D_B *and a contract* (C, P_A, P_B) *between* A *and* B*, we say that* C *is satisfied by* (H_A, H_B) *if every clause in* C *is satisfied by* (H_A, H_B)*.*

Definition 4 allows for the reasoning about the satisfaction of the public part of a contract. The satisfaction of the private part of a contract with respect an ontology $\mathcal{O}$ is defined next.

Definition 5. *Given two compatible trajectories* $H_A = s_0^A a_0^A \dots s_{n-1}^A a_{n-1}^A s_n^A$ *in* D_A *and* $H_B = s_0^B a_0^B \dots s_{n-1}^B a_{n-1}^B s_n^B$ *in* D_B *and a contract* (C, P_A, P_B) *between* A *and* B*. Let* $X \in \{A, B\}$ *and* $s(P_X) = \{reg \mid ref\_id : reg \in P_X\}$*. We say that a concern* c *of agent* X *is satisfied by* (H_A, H_B) *if* $\mathcal{O} \cup s(P_X) \models c$*.*

4.3 Reasoning about Contracts

Let $\mathcal{C} = (C, D_A, D_B)$ be a contract between A and B. We will next discuss how the semantics of $\mathcal{L}_c$ can be employed in evaluating $\mathcal{C}$ from the perspective of the agents. Naturally, a contract can be evaluated at different time points such as at the time the contract is signed or after some actions have been taken by the agents. Let us briefly discuss the questions that would be of interest to the agents.

Q1 *Contract feasibility*: given an initial state of the world, can $\mathcal{C}$ be successfully fulfilled? A different perspective of this question is whether there exists any state of the world in which $\mathcal{C}$ is satisfied. If there is none, it is clear that the contract is set up for failure!
Q2 *Clause satisfaction check*: assuming that $\mathcal{C}$ has been in execution for some time, has any agent violated a clause in $\mathcal{C}$? If so, then how can the problem be mitigated?
Q3 *Concern satisfaction check*: assuming that $\mathcal{C}$ has been in execution for some time, which/whose concerns are not satisfied? What can be done to mitigate the problem?

The above definitions in the previous section allow us to answer questions **Q1**–**Q3** by

Q1 computing two compatible trajectories H_A and H_B that start from the given initial joint state and satisfy the contract (or determining a joint state such that two compatible trajectories, that start from this state and satisfy the contract, can be identified);
Q2 identifying the clauses of the contract that are violated; and, to mitigate the problem caused by a violation of a clause (by an agent), new compatible trajectories, which start from the joint state at the end of the given trajectories (e.g., $s_n^A \cup s_n^B$ given the two trajectories in Def. 3) and satisfy the contract, need to be computed.
Q3 determining the concerns that are satisfied (or not satisfied). To mitigate the problem, similar approach as described in **Q2** needs to be adopted.

We illustrate the aforementioned idea using the running example.

Example 3 (Illustration). Let us consider the situation where Lumber Yard A (L) has not produced any board feet and XYZ Homes (H) has $1M on its account. Furthermore, all factories of Lumber Yard A are in good operational order. The initial state

for L and H can be represented by the set $s_0^L = \{board(0,Y) \mid Y = 1,2,3\} \cup \{\neg ab(production)\} \cup \{delivered(0,Y) \mid Y = 1,2,3\}$ and $s_0^H = \{available\_board(0,Y) \mid Y = 1,2,3\} \cup \{payment(0,C) \mid C = board, shipping\} \cup \{available\_funds(1M)\}$, respectively.

In this case, we can check that the two compatible trajectories satisfying the clauses (C1)–(C3) in the contract specified in Example 2 are:

$$H_L = s_0^L\ produce(144K,2)\ s_1^L\ deliver(144K,2)\ s_2^L\ wait\ s_3^L\ wait\ s_4^L$$
$$H_H = s_0^H\ wait\ s_1^H\ wait\ s_2^H\ receive(144K,2)\ s_3^H\ pay(122K, board)\ s_4^H$$

In the above trajectories, we have that $s_2^L = s_3^L = s_4^L$ and $s_0^H = s_1^H = s_2^H$ and $board(144K,2)$ is true in s_1^L, $board(0,2)$ is true in s_2^L, $available\_board(144K,2)$ is true in s_3^H, $available\_fund(878K)$ is true in s_4^H, etc. Assuming that the unit on the trajectories is week, it is easy to check that these trajectories satisfy the three clauses (C1)-(C3) of the contract in Example 2. One can also see that they can be extended to satisfy clause (C4), provided that the action domain of Lumber Yard A is extended with proper actions for billing the cost for shipping, organizing the delivery, and receiving money. We omit this discussion for brevity.

Consider a situation where XYZ Homes does not have any money in the initial state, i.e., $available\_funds(0)$ belongs to s_0^H, but it can borrow any amount from the bank. In this case, replacing one of the *wait* actions in H_H with *borrow(122K)*, which represents the action of borrowing 122K from the bank by XYZ Homes, will result in H'_H that is compatible with H_L and (H_L, H'_H) satisfies (C1)-(C3).

Let us consider yet another situation in which COVID-19 forces Lumber Yard A to close all of its factories right after the signing of the contract. It means that the initial state for L changes to $u_0^L = s_0^L \setminus \{\neg ab(production)\} \cup \{ab(production)\}$. In this case, H_L is no longer a valid trajectory for L. It is easy to see that there exists no pair of compatible trajectories for H and L that can satisfy clauses (C1)-(C2) of the contract. If Lumber Yard A could, for example, purchase the lumber from some other companies, then alternative trajectories could be identified and the contract can be fulfilled. Observe also that the action domain allows XYZ Homes to pay regardless of whether it receives the board feet, clause (C3) can still be satisfied!

5 Reasoning about Contracts using Answer Set Programming

Answer Set Programming. [10,12] is a declarative programming paradigm based on logic programming under the answer set semantics. A logic program Π is a set of rules of the form: $c_0 \vee \ldots \vee c_k \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n$ where c_i's, a_i's, and b_i's are atoms of a propositional language[7] and not represents (default) negation. $c_0 \vee \ldots \vee c_k$ can be absent.

The semantics of a logic program Π is defined via a special class of models called *answer sets* [4]. A program Π can have several answer sets, one answer set, or no answer set. Π is said to be consistent if it has at least one answer set; it is inconsistent otherwise. Several extensions (e.g., *choice atoms*, *aggregates*, etc.) have been introduced to simplify the use of ASP. We will use and explain them when needed.

[7] For convenience, we often use first order logic literals under the assumption that they represent all suitable ground instantiations.

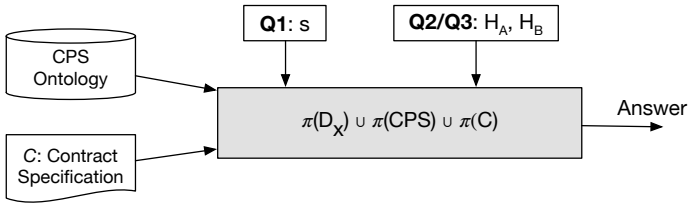

Fig. 1. Overview of Components used in Answering Queries **Q1-Q3** for Agent $X \in \{A, B\}$

5.1 Answer Set Programming for Reasoning about Contracts

This section presents an ASP encoding given a contract between two agents for reasoning about contracts and concerns of the agents, building on the work on planning in ASP and on formalizing CPS (e.g., [5,1,11]). Throughout this section, we assume that $\mathcal{C} = (C, P_A, P_B)$, where A and B are two agents with action domains D_A and D_B, respectively, is given. Figure 1 gives an overview of the components used by agent X in answering Queries **Q1-Q3** (gray box). Depending on the queries, the input is given to them might be an initial joint state (**Q1**) or two trajectories (**Q2-Q3**) that implicitly specify a joint state as well.

We assume that n is a constant in the unit of time used in $\mathcal{C}$ and denotes the maximal length of the trajectories considered by the two agents. Let s_0 be a joint state. For each agent X, we create a programs $\pi(D_X)$, $\pi(CPS)$, and $\pi(\mathcal{C})$ as follows.

- $\pi(D_X)$ is the program for reasoning about actions and changes (e.g., [5]) over predicates $h(f,t)$ (fluent f is true at time step t) and $occ(a,t)$ (action a occurs at step t) and consists of the following rules[8]:
  - declaration of steps $step(0..n)$;
  - for each fluent f, the declaration $fluent(f)$;
  - for each action a, the declaration $action(a)$;
  - for each fluent f that is true in s_0 then $h(f,0)$ belongs to $\pi(D_X)$ and the rule
    $$\neg h(f,0) \leftarrow not\ h(f,0)$$
  - for each executability law **executable** a **if** φ, the constraint that prevents a to be executed when its precondition is not satisfied: $\leftarrow occ(a,T), not\ h(\varphi,T)$
  - for each dynamic law a **causes** ψ **if** φ, the rule encoding the effect of a
    $$h(\psi, T+1) \leftarrow occ(a,T), h(\varphi,T)$$
  - for each static law ψ **if** φ, a rule stating that ψ must be true whenever φ is true
    $$h(\psi,T) \leftarrow h(\varphi,T)$$
  - for each fluent f, the inertial rules
    $$h(f,T+1) \leftarrow h(f,T), not\ \neg h(f,T+1) \text{ and}$$
    $$\neg h(f,T+1) \leftarrow \neg h(f,T), not\ h(f,T+1).$$
  - rules for reasoning about truth value of formulas where complex formulas are encoded using a set of atoms using fresh constants and membership functions, e.g., $f \wedge g$ is encoded by a fresh constant fg and the set of atoms $\{conjunction(fg), member(f,fg), member(g,fg)\}$ and rules such as

[8] In each rule with T as variable, we omit $step(T)$ from the right hand side.

$h(conjunction(F), T) \leftarrow conjunction(F),$
$N = \#count\{1, X : member(X, F)\},$
$C = \#count\{1, G : member(G, F), h(G, T)\}, N == C.$

Note that this rule uses the $\#count$ aggregate which counts the number of atoms satisfying a condition (e.g., the number of members of a formula). We omit other rules for brevity.

- $\pi(CPS)$ contains atoms encoding concerns, requirements, and relationships among all of them (e.g., $requirement(R)$, $concern(C)$ or $subCo(X, Y)$ indicating that R is a requirement, that C is a concern, and that Y is a subconcern of X) that can be obtained from a translation of the CPS ontology to ASP facts (see, e.g., [11]) and the rules for reasoning about the satisfaction of requirements and concerns. For example, given that a formula F over requirements addresses a concern C (i.e., contributes to its satisfaction), encoded by $addrBy(C, F)$, the following rules determine whether the concern C is satisfied at step T.

  $\neg h(sat(C), T) \leftarrow concern(C), addrBy(C, F), not\ h(F, T).$
  $\neg h(sat(X), T) \leftarrow subCo(X, Y), not\ h(sat(Y), T), concern(X), concern(Y).$
  $\neg h(sat(X), T) \leftarrow subCo(X, Y), \neg h(sat(Y), T), concern(X), concern(Y).$
  $h(sat(C), T) \leftarrow not\ \neg(sat(C), T), concern(C).$
- $\pi(\mathcal{C})$ encodes C and P_X in $\mathcal{C}$ and is constructed as follows.
  - for each clause of the form (4), the program contains the atom $clause($*ref_id*$)$, the set of atoms encoding v_g, where v_g is the name of the formula $goal$, and the following rules:

$time\_expression$	$\pi(\mathcal{C})$ contains
$always$	$\neg h(sat($*ref_id*$), n) \leftarrow not\ h(v_g, T).$
	$h(sat($*ref_id*$), n) \leftarrow not\ \neg h(sat($*ref_id*$), n)$
$eventually$	$h(sat($*ref_id*$), n) \leftarrow h(v_g, T).$
$per\_unit[u \dots l]$	$\neg h(sat($*ref_id*$), n) \leftarrow not\ h(v_g, T), u \leq T, T \leq l.$
	$h(sat($*ref_id*$), n) \leftarrow not\ \neg h(sat($*ref_id*$), n)$
$by\_unit[k]$	$h(sat($*ref_id*$), n) \leftarrow h(v_g, k).$

  where $h(sat($*ref_id*$), n)$ says that *ref_id* is satisfied at step n.
  - for each requirement r that occurs in P_X,
    - if there is only one element of the form *ref_id* : ρ in P_X that contains r, then $\pi(\mathcal{C})$ contains the atom $addrBy(r,$ *ref_id*$)$;
    - if there is more than one element of the form *ref_id* : ρ in P_X, then $\pi(\mathcal{C})$ contains the atoms $addrBy(cg,$ *ref_id*$)$ and $conjunction(cg)$ and the set $\{member(c, cg) \mid$ *ref_id* $: r \in P_X\}$ where cg is a fresh constant.

  This set of atoms helps propagating the satisfaction of clauses of contracts to the satisfaction of concerns of the agent.

Let $H_X = s_0^X a_0^X \dots s_{n-1}^X a_{n-1}^X s_n^X$ be an alternate sequence of states and actions in D_X. Define $e(H_X) = \{occ(a_i, i) \mid i = 0, \dots, n-1\} \cup \{h(f, i) \mid f$ is true in $s_i\}$. It holds that

Proposition 1.
- *$\pi(D_X) \cup e(H_X)$ has a unique answer set if H_X is a trajectory over D_X and a fluent formula φ is true in s_i iff $\pi(D_X) \cup e(H_X) \models h(\varphi, i)$.*
- *if H_X is a trajectory over D_X then H_X satisfies a clause ref_id in $\mathcal{C}$ iff $\pi(D_X) \cup \pi(\mathcal{C}) \cup e(H_X)$ has a unique answer set S that contains $h(sat($ref_id$), n)$.*

– *if H_X is a trajectory over D_X and ref_id : c in P_X then concern c is satisfied at the end of the trajectory iff $\pi(D_X) \cup \pi(CPS) \cup \pi(\mathcal{C}) \cup e(H_X)$ has a unique answer set S that contains $h(sat(c), n)$.*

Intuitively, the first property ensures that $\pi(D_X)$ correctly encodes the transition function in D_X. The second property shows that checking whether a clause is satisfied by H_X can be reduced to computing an answer set of $\pi(D_X) \cup \pi(\mathcal{C}) \cup e(H_X)$. Similarly, the third property shows that determining whether a concern is satisfied by a trajectory can be reduced to computing an answer set of $\pi(D_X) \cup \pi(CPS) \cup \pi(\mathcal{C}) \cup e(H_X)$. These properties help us answer questions related to the satisfaction of a contract or concerns given the compatible trajectories H_A and H_B. Observe that the only requirement for an agent X to reason about the satisfaction of the contract or a concern is the compatibility of H_A and H_B. X does not need to know the actions that the other agent executes. This is important when finding plans to satisfy a contract of an agent, which we discuss next.

To compute trajectories satisfying a contract, we observe that agents do usually have to plan by themselves. Furthermore, observe that in the context of this paper, the action domain of an agent (e.g., D_H of XYZ Homes in Example 2) might contain fluents which cannot be changed by XYZ Homes, such as $delivered(b, q)$. Therefore, any trajectory created by an agent will need to assume that certain properties that it cannot affect by its actions must be established by the other agent. Formally, we say that fluent f is *exogenous* in D_X if there exists no state s and action a such that f is true/false in s and false/true in some state belonging to $\Phi(a, s)$. Given a formula φ, let $\pi(D_X, \varphi)$ be the program $\pi(D_X)$ extended with the following rules

– $1\{h(f, 0); \neg h(f, 0)\}1$ for each exogenous fluent f;
– $1\{occ(A, T) : action(A)\}1 \leftarrow step(T)$ (exactly one action occurs at one step);
– the goal constraint $\leftarrow not\ h(\varphi, n)$ (φ must be true at step n);
– $\leftarrow not\ h(sat(\textit{ref\_id}), n)$ for each clause of the form *ref_id* : X in C, i.e., X is responsible for the satisfaction of the clause *ref_id* in C

It can be shown that there is a 1-to-1 correspondence between answer sets of $\pi(D_X, \varphi)$ and trajectories satisfying φ and all the clauses for which X is responsible for. Furthermore, for each answer set S, the set of assumptions made by the agent is indicated by the set $\{h(f, 0) \mid f$ is an exogenous fluent and $h(f, 0) \in S\}$. Additional constructs can be added to minimize the set of assumptions (e.g., if the assumption is made then it must be utilized in at least one state; for example, XYZ Homes can assume that Lumber Yard A will make fluent $delivered(140K, 2)$ true and plan to receive the board and pay; on the other hand, XYZ Homes does not need to assume $board(140K, 2)$ is true since its actions do not refer to this fluent). It is important to point out that when an agent needs to replan, it might not need to take into consideration *all* clauses that belong to its responsibilities; for example, if Lumber Yard A has already produced 140K board feet of lumber Number 2 Common grade (i.e., (7) is already satisfied) but has not been able to deliver the boards then the replanning process should only consider the other clauses. In this sense, the proposed framework supports the resilience of the contract execution. Similar construction can be done so that the obtained trajectory satisfies all the concerns of agent X. We omit it due to the space limitation.

6 Conclusions and Related Work

In this paper, we discussed an approach for the modeling, and reasoning about, a supply chain as a collection of contracts. We view this as a stepping stone towards enabling reasoning about supply chains' resilience and about ways to improve it. We focused on the development of a framework that supports various reasoning tasks in contract realization such as the feasibility of a contract, the satisfaction of clauses, requirements or concerns of agents of a contract, and the possible plans for mitigating an unsatisfied clause or concern. The framework assumes that each agent operates in accordance to its own action domain and has full observability of its environment. It formalizes a contract as a set of public clauses specifying the responsibility of each agent and, for each agent, a set of statements relating the public clauses with the agent's private concerns. By exploiting knowledge representation and reasoning techniques, we show that all of the above reasoning tasks can be reduced to the task of computing answer sets of suitable programs encoding the action domains and the contract. To the best of our knowledge, this is the first attempt at combining the modeling of supply chain contracts via action languages and ASP, and at linking the contracts with the relevant concerns through the CPS Framework. Having said that, there has been a large amount of research on representing and reasoning about contracts and related topics. Due to space constraints, we provide here limited highlights.

Within the ASP community, [13] addressed the problem of traceability in the supply chain. However, their work did not extend to modeling of contracts and stakeholder concerns, nor reasoning about their satisfaction. Others employed ASP to formalize negotiation – e.g., [17] focuses on establishing a contract (an exchange) between agents that can be satisfied by both parties. In other words, the goal of negotiation is different from that of reasoning about contracts. From the implementation perspective, the ASP-based system for multi-agent planning described in [16] could be useful in computing compatible trajectories for different agents. A different logic-based approach is used in [14], which discusses the representation and reasoning about contracts through deontic-logic based language $\mathcal{CL}$, with a focus on preserving many of the natural properties and concepts relevant to legal contracts.

In a different area of the spectrum of supply chain research, [9] discusses resilience through an equifinality lens to demonstrate that there are different pathways to supply resilience, with a focus on studying combinations of low and high redundancy scenarios.

A related direction of research is around service level agreements (e.g., [2]), which are typically focused on models and protocols for managing the negotiations surrounding access to resources in a distributed system and their use. Researchers have also focused on representing agreements via commitments rather than messaging protocols, see e.g. [18]. Finally, another line of research has been focused on standardized representations of contracts, as in [7,8]. Those approaches are focused on a rich representation of the relationships among contracts, but do not address the challenges posed by the multiplicity and diversity of stakeholders, and do not reason about the evolution of the state of the system over time.

In the future, we plan to investigate the relationship between our approach and smart contracts, whose formalization has recently gained attention (see, e.g., [15]).

References

1. Balduccini, M., Griffor, E., Huth, M., Vishik, C., Burns, M., Wollman, D.A.: Ontology-based reasoning about the trustworthiness of cyber-physical systems. ArXiv **abs/1803.07438** (2018)
2. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In: Job Scheduling Strategies for Parallel Processing (JSSPP 2002). pp. 153–183 (2002)
3. Edward, Greer, C., Wollman, D.A., Burns, M.J.: Framework for cyber-physical systems: volume 1, overview (2017)
4. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Warren, D., Szeredi, P. (eds.) Logic Programming: Proceedings of the Seventh International Conference. pp. 579–597 (1990)
5. Gelfond, M., Lifschitz, V.: Representing actions and change by logic programs. Journal of Logic Programming **17**(2,3,4), 301–323 (1993)
6. Gelfond, M., Lifschitz, V.: Action Languages. Electronic Transactions on Artificial Intelligence **3**(6), 193–210 (1998)
7. Governatori, G.: Representing Business Contracts in RuleML. International Journal of Cooperative Information Systems **14**(2–3), 181–216 (2005)
8. Governatori, G., Rotolo, A., Sartor, G.: Normative Autonomy and Normative Co-ordination: Declarative Power, Representation, and Mandate. Artificial Intelligence and Law **12**(1–2), 53–81 (2004)
9. Jahre, M., Selviaridis, K., Li, Q., Dube, N.: One Crisis, Different Paths to Supply Resilience: the Case of Ventilator Procurement for the COVID-19 Pandemic. Journal of Purchasing and Supply Management **28**(5) (2022)
10. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-year Perspective. pp. 375–398 (1999)
11. Nguyen, T.H., Bundas, M., Son, T.C., Balduccini, M., Garwood, K.C., Griffor, E.R.: Specifying and reasoning about CPS through the lens of the NIST CPS framework. TPLP (2022)
12. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3,4), 241–273 (1999)
13. Nogueira, M., Greis, N.P.: Supply Chain Tracing of Multiple Products under Uncertainty and Incomplete Information - An Application of Answer Set Programming. In: Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD 2013). pp. 399–406 (2013)
14. Prisacariu, C., Schneider, G.: CL: An Action-Based Logic for Reasoning about Contracts. In: Proceedings of Logic, Language, Information and Computation (WoLLIC 2009). pp. 335–349 (2009)
15. Singh, A., Parizi, R.M., Zhang, Q., Choo, K.K.R., Dehghantanha, A.: Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. Computers Security **88** (2020)
16. Son, T.C., Pontelli, E., Nguyen, N.H.: Planning for multiagent using asp-prolog. In: Dix, J., Fisher, M., Novák, P. (eds.) Computational Logic in Multi-Agent Systems - 10th International Workshop, CLIMA X, Hamburg, Germany, September 9-10, 2009, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 6214, pp. 1–21. Springer (2010)
17. Son, T.C., Pontelli, E., Nguyen, N., Sakama, C.: Formalizing negotiations using logic programming. ACM Trans. Comput. Log. **15**(2), 12 (2014)
18. Wan, F., Singh, M.P.: Formalizing and Achieving Multiparty Agreements via Commitments. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems (AAMAS '05). pp. 770–777 (2005)