CTR-S: A Logic for Specifying Contracts in Semantic Web Services*

Hasan Davulcu
Department of CSE
Arizona State University
Box 875406,
Tempe, AZ 85287-5406
hdavulcu@asu.edu

Michael Kifer
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
kifer@cs.stonybrook.edu

I.V. Ramakrishnan
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
ram@cs.stonybrook.edu

ABSTRACT

A requirements analysis in the emerging field of *Semantic Web Services* (SWS) (see http://daml.org/services/swsl/requirements/) has identified four major areas of research: intelligent *service discovery*, automated *contracting of services*, *process modeling*, and *service enactment*. This paper deals with the intersection of two of these areas: process modeling as it pertains to automated contracting. Specifically, we propose a logic, called $\mathcal{CTR}$-S, which captures the *dynamic aspects* of contracting for services. Since $\mathcal{CTR}$-S is an extension of the classical first-order logic, it is well-suited to model the static aspects of contracting as well. A distinctive feature of contracting is that it involves two or more parties in a potentially adversarial situation. $\mathcal{CTR}$-S is designed to model this adversarial situation through its novel model theory, which incorporates certain game-theoretic concepts. In addition to the model theory, we develop a proof theory for $\mathcal{CTR}$-S and demonstrate the use of the logic for modeling and reasoning about Web service contracts.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous; I.1.3 [**Computing Methodologies**]: Symbolic and algebraic manipulation—*Languages and Systems*

General Terms

Algorithms, Languages, Verification

Keywords

Web Services, Services Composition, Contracts

1. INTRODUCTION

A Web service is a process that interacts with the client and other services to achieve a certain goal. A requirements analysis in the emerging field of *Semantic Web Services* (SWS)[1] has identified four major areas of research: intelligent *service discovery*, automated *contracting of services*, *process modeling*, and *service enactment*. It is generally agreed that Semantic Web Services should be based on a formalism with a well-defined model-theoretic semantics, *i.e.*, on some sort of a logic. In this paper we propose a logic, called $\mathcal{CTR}$-S, which captures the *dynamics* of contracting for services and thus is in the intersection of the areas of contracting and process modeling. Since $\mathcal{CTR}$-S is an extension of the classical first-order logic, it is well-suited for modeling of the static aspects of contracting as well. If object-oriented representation is desired, F-logic [14] (and an adaptation of $\mathcal{CTR}$-S) can be used instead.

The idea of using a logic to model processes is not new [12, 2, 23, 8, 21]. These methodologies are commonly based on the classical first-order logic, temporal logic [9], and Concurrent Transaction Logic [5]. A distinctive aspect of contracting in Web services, which is not captured by these formalisms, is that contracting involves multi-party processes, which can be adversarial in nature. One approach to deal with this situation could be to try and extend a multi-modal logic of knowledge [10]. However, we found it more expedient to extend Concurrent Transaction Logic [5] (or $\mathcal{CTR}$), which has been proven a valuable tool for modeling and reasoning about processes [8, 4, 17]. The extension is called $\mathcal{CTR}$-S and is designed to model the adversarial situation that arises in service contracting. This is achieved by extending the model theory of $\mathcal{CTR}$ with certain concepts borrowed from the Game Theory [18, 13, 19]. In this paper we also develop a proof theory for $\mathcal{CTR}$-S and illustrate the use of this logic for modeling and reasoning about Web service contracts.

A typical situation in contracting where different parties may sometimes have conflicting goals is when a buyer interacts with a seller and a delivery service. The buyer needs to be assured that the goods will either be delivered (using a delivery service) or money will be returned. The seller might need assurance that if the buyer breaks the contract then part of the down-payment can be kept as compensation. We thus see that services can be adversarial to an extent. Reasoning about such services is an unexplored research area and is the topic of this paper.

Overview and summary of results. We introduce game-theoretic aspects into $\mathcal{CTR}$ using a new connective, the *opponent's conjunction*. This connective represents the choice of action that can be made by a party other than the reasoner. The $\boldsymbol{reasoner}$ here can be the client of a Web service who needs to verify that her goals are met or a service that needs to make sure that its business rules are satisfied no matter what the other parties (the clients and other services) do. Actors other than the reasoner are collectively referred to as the $\boldsymbol{opponent}$. We then develop a model theory for $\mathcal{CTR}$-S and show how this new logic can be used to specify executions of services that may be non-cooperating and have potentially conflicting goals. We also discuss reasoning about a fairly large class of temporal and causality constraints.

In $\mathcal{CTR}$-S, a contract is modeled as a *workflow* that represents the various possibilities for the service and the outside actors (or the

*M. Kifer and I.V. Ramakrishnan were supported in part by the NSF grants CCR-0311512 and IIS-0072927.

[1]See http://daml.org/services/swsl/requirements/

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

client and other services). The $\mathcal{CTR}$-S model theory characterizes all possible *outcomes* of a formula $\mathcal{W}$ that represents such a workflow. A *constraint*, Φ, represents executions of the contract with certain desirable properties. For instance, from the client's point of view, a desirable property might be that either the good is delivered or the payment is refunded. The formula $\mathcal{W} \wedge \Phi$ characterizes those executions of the contract that satisfy the constraint Φ. If $\mathcal{W} \wedge \Phi$ is satisfiable, *i.e.*, there is at least one execution in its model, then we say that the constraint Φ is *enforcible* in the workflow formula $\mathcal{W}$.

We describe a synthesis algorithm that converts declarative specifications, such as $\mathcal{W} \wedge \Phi$, into equivalent $\mathcal{CTR}$-S formulas that can be executed more efficiently and without backtracking. The transformation also detects unsatisfiable specifications, which are contracts that the reasoner cannot force to execute with desirable outcomes. In game-theoretic terms, the result of such a transformation can be thought of as a concise representation of all *winning strategies*, *i.e.*, all the ways for the reasoner to achieve the desired outcome, regardless of what the rest of the system does, if all the parties obey the terms of the contract.

Finally, since $\mathcal{CTR}$-S is a natural generalization of $\mathcal{CTR}$, a pleasing aspect of this work is that our earlier results in [8] become special cases of the new results.

The rest of the paper is organized as follows. In Section 2, we introduce $\mathcal{CTR}$-S and discuss its use for modeling workflows and contracts. In Section 3, we introduce the model theory of $\mathcal{CTR}$-S. Section 4 discusses the proof theory. Section 5 introduces causal and temporal workflow constraints that can be used to specify goals of the participants in a contract. In Section 6, we present an algorithm for solving these constraints and discuss its complexity. Section 7 concludes with a discussion of related formalisms.

2. $\mathcal{CTR}$-S AND THE DYNAMICS OF SERVICE CONTRACTS

Familiarity with $\mathcal{CTR}$ [5] can help understanding of $\mathcal{CTR}$-S and its relationship to workflows and contracts. However, this paper is self-contained and includes all the necessary definitions. We first describe the syntax of $\mathcal{CTR}$-S and then supply intuition to help the reader make sense out of the formal definitions that follow.

2.1 Syntax

The $\boldsymbol{atomic\ formulas}$ of $\mathcal{CTR}$-S are expressions of the form $p(t_1, ..., t_n)$, where *p* is a predicate symbol and t_i are terms, *i.e.*, they are the same as in classical logic. Complex formulas are built with the help of connectives and quantifiers: if ϕ and ψ are $\mathcal{CTR}$-S formulas, then so are $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \otimes \psi$, $\phi \mid \psi$, $\neg\phi$, $\phi \sqcap \psi$, $(\exists X)\phi$, and $(\forall X)\phi$, where X is a variable. Intuitively, the formula $\phi \otimes \psi$ means: execute ϕ and then execute ψ. The connective $\otimes$ is called $\boldsymbol{serial\ conjunction}$. The formula $\phi \mid \psi$ denotes an interleaved execution of two games ϕ and ψ. The connective $\mid$ is called $\boldsymbol{concurrent\ conjunction}$. The formula $\phi \sqcap \psi$ means that the opponent chooses whether to execute ϕ or ψ, and therefore $\sqcap$ is called $\boldsymbol{opponent's\ conjunction}$. The meaning of $\phi \vee \psi$ is similar, except the reasoner makes the decision. In $\mathcal{CTR}$ this connective is called $\boldsymbol{classical\ disjunction}$ but because of its interpretation as reasoner's choice we will also refer to it as $\boldsymbol{reasoner's\ disjunction}$. Finally, the formula $\phi \wedge \psi$ denotes execution of ϕ *constrained* by ψ (or ψ constraint by ϕ). It is called $\boldsymbol{classical\ conjunction}$.[2]

As in classical logic, we introduce $\phi \leftarrow \psi$ as an abbreviation for $\phi \vee \neg\psi$. The usual dualities $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$ and $\exists\, \phi \equiv \neg\forall\neg\phi$ also hold. The opponent's conjunction has a dual connective, $\sqcup$, but we will not discuss it in this paper.

As mentioned in the introduction, we model the dynamics of service contracts using the abstraction of a 2-party workflow, where the first party is the reasoner and the other represents the rest of the players involved in the contract. In general, if several parties need to be able to reason about the same contract, the contract can be represented as several 2-party workflows, each representing the contract from the point of view of a different reasoner.

Definition 1. (**Workflows**) A $\boldsymbol{\mathcal{CTR}}$-S $\boldsymbol{goal}$ is recursively defined as either an atomic formula or an expression of the form $\phi \otimes \psi$, $\phi \mid \psi$, $\phi \vee \psi$, or $\phi \sqcap \psi$, where ϕ and ψ are $\mathcal{CTR}$-S goals. A $\boldsymbol{rule}$ is of the form $head \leftarrow body$, where *head* is an atomic formula and *body* a $\mathcal{CTR}$-S goal. A $\boldsymbol{workflow\ control\ specification}$ consists of a $\mathcal{CTR}$-S $goal$ and a (possibly empty) set of $rules$.

Note that the connective $\wedge$ is not allowed in workflow control specifications, but is used to specify constraints.

2.2 Modeling Contract Dynamics in $\mathcal{CTR}$-S

Example 1. (**Procurement Contract**) Consider a procurement application that consists of a $buyer$ interacting with three services, $sell$, $finance$, and $deliver$. We assume that the buyer is the reasoner in this example.

Services are modeled in terms of their $\boldsymbol{significant\ events}$. For instance, the buy service begins when the significant event pay_escrow occurs. When *pay_escrow* is finished, a concurrent execution of the $sell$ and $finance$ services begins.

Thus, at a high level, the buy service can be represented as:

$$pay\_escrow \otimes (finance \mid sell)$$

The connective $\otimes$ represents succession of events or actions: when the above expression "executes," the underlying database state is first changed by the execution of the formula pay_escrow and then by the execution of $finance \mid sell$. The connective $\mid$ represents concurrent, *interleaved* execution of the two sequences of actions. Intuitively, this means that a legal execution of $(finance \mid sell)$ is a sequence of database states where the initial subsequence corresponds, say, to a partial execution of the subformula $finance$; the next subsequence of states corresponds to an execution of $sell$; the following subsequence is a continuation of the execution of $finance$; and so on. The overall execution sequence of $(finance \mid sell)$ is a merge of an execution sequence for the left subformula and an execution sequence for the right subformula.

Execution has precise meaning in the model and proof theories of $\mathcal{CTR}$. Truth of $\mathcal{CTR}$ formulas is established *not* over database states, as in classical logic, but over sequences of states; it is interpreted as an execution of that formula in which the initial database state of the sequence is successively changed to the second, third, etc., state. The database ends up in the final state of the sequence when the execution terminates.[3]

Workflow formulas can be modularized with the help of rules. The intuitive meaning of a rule, $head \leftarrow body$, where *head* is an atomic formula and *body* is a $\mathcal{CTR}$-S goal, is that *head* is an invocation interface to *body*, where *body* is viewed as a subroutine. This is because according to the semantics described in Section 3,

[2]The meaning of $\wedge$ is all but classical. However, its semantic definition looks very much like that of a conjunction in predicate calculus. This similarity is the main reason for the name.

[3]Space limitation does not permit us to compare $\mathcal{CTR}$ to other logics that on the surface might appear to address the same issues (*e.g.*, temporal logics, process and dynamic logics, the situation calculus, etc.). We refer the reader to the extensive comparisons provided in [5, 6].

such a rule is true if every legal execution of *body* must also be a legal execution of *head*. Combined with the minimal model semantics this gives the desired effect [6]. With this in mind, we can now express the above procurement workflow as follows:

$$buy \longleftarrow pay\_escrow \otimes (sell \mid finance)$$

Next, we search for matching services for the $sell$ service using a service directory to discover the following rules.

$$sell \longleftarrow reserve\_item \otimes (deliver \vee keep\_escrow)$$
$$deliver \longleftarrow insured \vee uninsured$$

The $\vee$ connective in the definition of $sell$ represents alternative executions. For instance, a legal execution of $insured \vee uninsured$ is either a legal execution of $insured$ or of $uninsured$. Similarly, a legal execution of $sell$ involves the execution of $reserve\_item$ and then, an execution of either $deliver$ or $recv\_escrow$.

The above definition of $sell$ also requires compliance with the following contract requirements between the buyer and the seller:

- if *buyer* cancels, then *seller* keeps the escrow
- if *buyer* pays, then *seller* must deliver

Thus, the connective $\vee$ represents a *choice*. The question is *whose* choice is it: the reasoner's or that of the opponent? In an environment where workflow activities might not always cooperate, we need a way to distinguish these two kinds of choices. For instance, the contract may say that the outcomes of the actions of the delivery agent are that the goods might be $delivered$ or $lost$. This alternative is clearly not under the control of the buyer, who is the reasoner here. On the other hand, the choice of whether to use $insured$ or $uninsured$ delivery is made by the buyer, *i.e.*, the reasoner. With this understanding, the $insured$ and $uninsured$ services can be defined as follows:

$$insured \longleftarrow (delivered \otimes satisfied) \sqcap (lost \otimes satisfied)$$
$$uninsured \longleftarrow (delivered \otimes satisfied) \sqcap lost$$

The connective $\sqcap$ here represents the choice made by the actors other than the reasoner (the buyer). If the buyer uses insured delivery then she is guaranteed satisfaction if the item is $delivered$ or $lost$ (in the latter case the buyer presumably gets the money back). If the buyer uses $uninsured$ delivery then she can get satisfaction only if the item is $delivered$. Whether the item is $delivered$ or $lost$ is outside of the control of the buyer.

Next, we identify the following matching service for $finance$:

$$finance \longleftarrow (approve \otimes (make\_payment \vee cancel)) \sqcap (reject \otimes cancel)$$

Note that approval or rejection of the financing request is an opponent's (the servicing agent's) choice. However, if financing is approved the choice of whether to proceed and $make\_payment$ or to $cancel$ depends on the reasoner (the buyer). In addition, the financing agent might require the following clause in the contract:

- if *financing* is approved and *buyer* does not $cancel$ then *delivery* should *satisfy*

Details of how to express the above contract requirement in $\mathcal{CTR}$-S will be given in Section 5.

The buyer and the services involved might have specific goals with respect to the above contract. For instance, the buyer wants that if financing is approved then she has a strategy to ensure that she is satisfied (either by receiving the goods or by getting the money back). The seller might want to have the peace of mind knowing that if the buyer cancels the contract after receiving financing then the seller can keep the escrow. In Section 6 we will show how such goals can be represented and that they can be enforced under this contract even in adversarial situations.

We shall see that a large class of temporal and causality constraints can be represented as $\mathcal{CTR}$-S formulas. If Φ represents such a formula for the above example, then finding a strategy to enforce the constraints under the rules of the contract is tantamount to checking whether $buy \wedge \Phi$ is satisfiable in $\mathcal{CTR}$-S.

Before going on, we should clear up one possible doubt: why is there only one opponent? The answer is that this is sufficient for a vast majority of practical cases, especially those that arise in Web services. Indeed, even when multiple independent actors are involved, we can view each one of them (or any group that decides to cooperate) as the *reasoner* and all the rest as the *opponent*. Any such actor or a group can then use $\mathcal{CTR}$-S to verify that its goals (specified as a condition Φ) are indeed enforcible.

3. MODEL THEORY

In this section we define a model theory for our logic. The importance of a model theory is that it provides the exact semantics for the behavioral aspects of service contracts and thus serves as a yardstick of correctness for the algorithms in Section 6.

3.1 Sets of Multipaths

A $\boldsymbol{path}$ is a sequence of database states, $d_1 ... d_n$. Informally, a $\boldsymbol{database\ state}$ can be a collection of facts or even more complex formulas, but for this paper we can think of them as simply symbolic identifiers for various collections of facts.

In $\mathcal{CTR}$ [5], which allows concurrent, interleaved execution, the semantics is based on *sequences* of paths, $\pi = \langle p_1, ..., p_m \rangle$, where each p_i is a path. Such a sequence is called a $\boldsymbol{multipath}$, or an $\boldsymbol{m\text{-}path}$ [5]. For example, $\langle d_1 d_2,\ d_3 d_4 d_5 \rangle$ is an m-path that consists of two paths: one having two database states and the other three (note that a comma separates paths, not states in a path). As explained in Example 1, multipaths capture the idea of an execution of a transaction that interleaves with executions of other transactions. Thus, an m-path can be viewed as an execution that is broken into segments, such that other transactions could execute in-between the segments.

$\mathcal{CTR}$-S further extends this idea by recognizing that in the presence of other parties, the reasoner cannot be certain which execution (or "play") will actually take place, due to the lack of information about the actual moves that the opponent will make. However, the reasoner can have a *strategy* to ensure that regardless of what the opponent does the resulting execution will be contained within a certain set of plays. If every play in the set satisfies the properties that the reasoner wants, then the strategy will achieve the reasoner's objectives. Such a set of plays is called an $\boldsymbol{outcome\ of\ the\ game}$. Thus, while truth values of formulas in $\mathcal{CTR}$ are determined on m-paths, $\mathcal{CTR}$-S formulas get their truth values on $\boldsymbol{sets\ of\ m\text{-}paths}$. Each such set, S, is interpreted as an outcome of the game in the above sense, and saying that a $\mathcal{CTR}$-S formula, ϕ, is true on S is tantamount to saying that S is an outcome of ϕ. In particular, two games are considered to be equivalent if and only if they have the same sets of outcomes.

3.2 Satisfaction on Sets of Multipaths

The following definitions make the above discussion precise.

Definition 2. (**m-Path Structure** [5]) An $\boldsymbol{m\text{-}path\ structure}$ is a triple of the form $\langle U, I_{\mathcal{F}}, I_{path} \rangle$, where U is the domain, $I_{\mathcal{F}}$ is an interpretation function for constants and function symbols (exactly like in classical logic), and I_{path} is a

mapping such that if π is an m-path, then $I_{path}(\pi)$ is a first-order semantic structure (as commonly used in classical predicate logic).

For a $\mathcal{CTR}$ formula, ϕ, and an m-path, π, the truth of ϕ on π with respect to an m-path structure is determined by the truth values of the components of ϕ on the appropriate sub-m-paths of π. In a well-defined sense, establishing the truth of a formula, ϕ, over an m-path, $\pi = \langle p_1, ..., p_n \rangle$, corresponds to the possibility of executing ϕ along π where the gaps between p_1, ..., p_n are filled with executions of other formulas [5].

The present paper extends this notion to $\mathcal{CTR}$-S by defining truth of a formula ϕ over *sets* of m-paths, where each such set represents a possible outcome of the game corresponding to ϕ. The new definition reduces to $\mathcal{CTR}$'s for formulas that have no $\sqcap$'s.

Definition 3. (**Satisfaction**) Let $\mathbf{I} = \langle U, I_{\mathcal{F}}, I_{path} \rangle$ be an m-path structure, π be an arbitrary m-path. Let S, T, S_1, S_2, etc., denote *non-empty* sets of m-path, and let ν be a *variable assignment*, which assigns an element of U to each variable. We define the notion of $\boldsymbol{satisfaction}$ of a formula, ϕ, in $\mathbf{I}$ on S by structural induction on ϕ:

1. **Base Case:** $\mathbf{I}, \{\pi\} \models_\nu p(t_1, \ldots, t_n)$ iff $I_{path}(\pi) \models_\nu^{classic} p(t_1, \ldots, t_n)$. Here $\{\pi\}$ is a set of m-paths that contains only one m-path, π, and $p(t_1, \ldots, t_n)$ is an atomic formula. Recall that $I_{path}(\pi)$ is a usual first-order semantic structure, so $\models^{classic}$ here denotes the usual, classical first-order entailment.

   Typically, $p(t_1, ..., t_n)$ is either defined via rules (as in Example 1) or is a "built-in," such as $insert(q(a, b))$, with a fixed meaning. For instance, in case of $insert(q(a, b))$ the meaning would be that $\mathbf{I}, \{\pi\} \models_\nu insert(q(a, b))$ iff π is an m-path of the form $\langle d_1\, d_2 \rangle$, which consists of a single path, and $d_2 = d_1 \cup \{q(a, b)\}$.[4] These built-ins are called $\boldsymbol{elementary\ updates}$ and constitute the basic building blocks from which more complex actions, such as those at the end of Example 1, are constructed.

2. **Negation:** $\mathbf{I}, S \models_\nu \neg\phi$ iff it is *not* the case that $\mathbf{I}, S \models_\nu \phi$.

3. **Reasoner's Disjunction:** $\mathbf{I}, S \models_\nu \phi \vee \psi$ iff $\mathbf{I}, S \models_\nu \phi$ or $\mathbf{I}, S \models_\nu \psi$. We define $\phi \wedge \psi$ as a shorthand for $\neg(\neg\phi \vee \neg\psi)$.

4. **Opponent's Conjunction:** $\mathbf{I}, S \models_\nu \phi \sqcap \psi$ iff $S = S_1 \cup S_2$, for some pair of m-path sets, such that $\mathbf{I}, S_1 \models_\nu \phi$, and $\mathbf{I}, S_2 \models_\nu \psi$. The dual connective, $\sqcup$, also exists, but is not used in this paper.

5. **Serial Conjunction:** $\mathbf{I}, S \models_\nu \phi \otimes \psi$ iff there is a set R of m-paths, such that S can be represented as $\bigcup_{\pi \in R} \pi \circ T_\pi$, where each T_π is a set of m-paths, $\mathbf{I}, R \models_\nu \phi$, and for each T_π, $\mathbf{I}, T_\pi \models_\nu \psi$.
   Here $\pi \circ T = \{\pi \circ \delta \mid \delta \in T\}$, where $\pi \circ \delta$ is an m-path obtained by appending the m-path δ to the end of the m-path π. (For instance, if $\pi = \langle d_1 d_2,\ d_3 d_4 \rangle$ and $\delta = \langle d_5 d_6,\ d_7 d_8 d_9 \rangle$ then $\pi \circ \delta = \langle d_1 d_2,\ d_3 d_4,\ d_5 d_6,\ d_7 d_8 d_9 \rangle$.) In other words, R is a set of prefixes of the m-paths in S.

6. **Concurrent Conjunction:** $\mathbf{I}, S \models_\nu \phi \mid \psi$ iff there is a set R of m-paths, such that S can be represented as $\bigcup_{\pi \in R} \pi \| T_\pi$, where each T_π is a set of m-paths, and

   - either $\mathbf{I}, R \models_\nu \phi$ and for all T_π, $\mathbf{I}, T_\pi \models_\nu \psi$;
   - or $\mathbf{I}, R \models_\nu \psi$ and for all T_π, $\mathbf{I}, T_\pi \models_\nu \phi$

   Here $\pi \| T_\pi$ denotes the set of *all* m-paths that are obtained by interleaving π with some m-path in T_π. For instance, if $\pi = \langle d_1 d_2,\ d_3 d_4 \rangle$ and $\langle d_5 d_6,\ d_7 d_8 d_9 \rangle \in T_\pi$ then one interleaving is $\langle d_1 d_2,\ d_5 d_6,\ d_3 d_4,\ d_7 d_8 d_9 \rangle$, another is $\langle d_1 d_2,\ d_5 d_6,\ d_7 d_8 d_9,\ d_3 d_4 \rangle$, etc.

7. **Universal Quantification:** $\mathbf{I}, \pi \models_\nu \forall X.\phi$ if and only if $\mathbf{I}, \pi \models_\mu \phi$ for *every* variable assignment μ that assigns the same value as ν to all variables except X. Existential quantification, $\exists X.\phi$, is a shorthand for $\neg\forall X \neg\phi$.

Example 2. (**Database Transactions**) Consider the following formula, where *st* means "start," *ab* means "abort," *cm* is "commit," *cp* means "compensate," and *no* stands for a noop. Further assume that each elementary update *em* in the following formula denotes an $insert(em)$ operation which satisfies $\{\langle d \quad d \cup \{em\} \rangle\} \models insert(em)$ where d is a set of ground atomic formulas.

$$\phi \;=\; st \otimes (ab \sqcap cm) \otimes (cp \vee no)$$

Then the possible outcomes for ϕ can be computed from the outcomes of its components as follows:

1. By (3) in Definition 3: $\{\langle \emptyset \ \{cp\} \rangle\} \models (cp \vee no)$, and $\{\langle \emptyset \ \{no\} \rangle\} \models (cp \vee no)$
2. By (5) in Definition 3: $\{\langle \emptyset \ \{ab\} \rangle,\ \langle \emptyset \ \{cm\} \rangle\} \models (ab \sqcap cm)$
3. By (6) in Definition 3: $\{\langle \emptyset\ \{st\}\ \{st, ab\} \rangle,\ \langle \emptyset\ \{st\}\ \{st, cm\} \rangle\} \models st \otimes (ab \sqcap cm)$
4. By (6) in Definition 3: Hence there are four possible outcomes for ϕ;
   $\{\langle \emptyset\{st\}\{st, ab\}\{st, ab, cp\} \rangle,\ \langle \emptyset\{st\}\{st, cm\}\{st, cm, cp\} \rangle\}$
   $\{\langle \emptyset\{st\}\{st, ab\}\{st, ab, cp\} \rangle,\ \langle \emptyset\{st\}\{st, cm\}\{st, cm, no\} \rangle\}$
   $\{\langle \emptyset\{st\}\{st, ab\}\{st, ab, no\} \rangle,\ \langle \emptyset\{st\}\{st, cm\}\{st, cm, cp\} \rangle\}$
   $\{\langle \emptyset\{st\}\{st, ab\}\{st, ab, no\} \rangle,\ \langle \emptyset\{st\}\{st, cm\}\{st, cm, no\} \rangle\}$

Definition 4. (**Playset**) As in classical logic, $\phi \vee \neg\phi$ is a tautology for any ϕ, *i.e.*, it is true on every set of m-paths. We denote this tautology with a special proposition `Playset`, an analogue of *true* in classical logic.

By definition, $\mathbf{I}, S \models$ `Playset` for any m-path structure and any set of m-paths. Therefore, $\neg$`Playset` is unsatisfiable. Intuitively, `Playset` is the game in which all outcomes are possible, while $\neg$`Playset` is a game with no outcomes.

4. PROOF THEORY

4.1 Execution as Entailment

We now define *executional entailment*, a concept that connects model theory with the execution of a certain strategy of the reasoner.

Definition 5. (**Executional Entailment**) Let ϕ be a $\mathcal{CTR}$-S goal and $\mathcal{W}$ a set of rules that define services (see Definition 1). Let $\mathbf{D}_0, ..., \mathbf{D}_n$ be a sequence of database states. A $\boldsymbol{path\ tree}$ with a $\boldsymbol{shared\ prefix}$ $\mathbf{D}_0, ..., \mathbf{D}_n$ is a set of paths where each begins with this sequence of states. We define

$$\mathbf{D}_0, ..., \mathbf{D}_n \text{ --- } \models (\exists)\phi \tag{4.1}$$

to mean that for every model $\mathbf{M}$ of ϕ there is a path tree T such that $\mathbf{M}, T \models (\exists)\phi$ and $\langle \mathbf{D}_0, ..., \mathbf{D}_n \rangle$ is a shared prefix of T. Here $(\exists)$ indicates that all variables in ϕ are quantified existentially.

[4] Formally, the semantics of such built-ins is defined using the notion of the $\boldsymbol{transition\ oracle}$ [6].

Intuitively, (4.1) means that the reasoner playing the game ϕ can ensure that the execution will begin with the database state $\mathbf{D}_0$ and continue with $\mathbf{D}_1$, ..., $\mathbf{D}_n$.

Observe that executional entailment is defined over path trees, not over arbitrary outcomes. Hence, when we talk about execution of a game we are only interested in enforcible outcomes that can reduce to a path tree. We call these ***executable outcomes***. We are interested in these outcomes because ultimately we want to obtain strategires that contain *complete* plays—plays that represent movements of all the players involved. Such a play must be represented by a path, not m-path, because m-paths are incomplete plays—they contain gaps, which must be filled by external players.

The plays in an outcome that represents a strategy must form a path tree because all the plays in an outcome of a real game start with the same initial state $\mathbf{D}_0$. Thus, finding out if a winning strategy exists in state $\mathbf{D}_0$ is tantamount to proving $\mathcal{W}, \mathbf{D}_0 \text{ --- } \models (\exists)\phi$.

4.2 Inference Rules

We now develop an inference system for proving statements of the form $\mathcal{W}, D_0 \text{ --- } \models (\exists)\phi$, where $\mathcal{W}$ is a set of rules and ϕ is a $\mathcal{CTR}$-S goal. The system manipulates expressions of the form $\mathcal{W}, D_0 \text{ --- } \vdash \phi$, called ***sequents***.

First we need the notion of the ***hot component*** of a formula; it is a generalization of a similar notion from [5]: $hot(\phi)$ is a set of subformulas of ϕ, defined by induction on the structure of ϕ as follows:

1. $hot(\phi) = \{\phi\}$, if ϕ is an atomic formula
2. $hot(\phi \otimes \psi) = hot(\phi)$
3. $hot(\phi \mid \psi) = hot(\phi) \cup hot(\psi)$
4. $hot(\phi \vee \psi) = \{\phi \vee \psi\}$
5. $hot(\phi \sqcap \psi) = \{\phi \sqcap \psi\}$.

Note that in cases of $\vee$ and $\sqcap$, the hot component is a singleton set that contains the very formula that is used as an argument to hot. Here are some examples of hot components:

$a \otimes b \otimes c$	$\{a\}$
$(a \otimes b) \mid (c \otimes d)$	$\{a, c\}$
$(a \sqcap b) \otimes c$	$\{a \sqcap b\}$
$(a \sqcap b) \mid (c \vee d)$	$\{a \sqcap b,\ c \vee d\}$
$((a \sqcap b) \otimes c) \vee ((f \mid g) \otimes h)$	$\{((a \sqcap b) \otimes c) \vee ((f \mid g) \otimes h)\}$

Note that a hot component represents a *particular occurrence* of a subformula in a bigger formula. For instance, $hot(a \otimes b \otimes a)$ is $\{a\}$, where a corresponds to the *first* occurrence of this subformula in $a \otimes b \otimes a$ and not the second one. This point is important because in the inference rules, below, we will sometime say that ψ' is obtained from ψ by deleting a hot occurrence of a (or some other subformula). Thus, in the above, deleting the hot component a leaves us with $b \otimes a$, *not* $a \otimes b$.

The inference rules are as follows:

Axiom: $\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\,)$, for any $\mathbf{D}$.
Here $(\,)$ is the empty $\mathcal{CTR}$-S goal; it represents a game that starts and stops in the same state without making any moves. The axiom says that such a game can be played in any state.

Inference Rules: In Rules 1–4 below, σ is a substitution, ψ and ψ' are concurrent serial conjunctions, and a is a formula in $hot(\psi)$.

1. *Applying transaction definitions:* Let $b \leftarrow \beta$ be a rule in $\mathbf{P}$, and assume that its variables have been renamed so that none are shared with ψ. If a and b unify with mgu (most general unifier) σ then

$$\frac{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi'\, \sigma}{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi}$$ where ψ' is ψ where a hot occurrence of a is replaced by β.

   For instance, if $\psi = (c \sqcap e) \mid (a \otimes f) \mid (d \vee h)$ and the hot component in question is a in the middle subformula, then $\psi' = (c \sqcap e) \mid (\beta \otimes f) \mid (d \vee h)$.

2. *Querying the database:* If $\mathcal{O}^d(\mathbf{D}) \models^c (\exists) a\sigma$; $a\sigma$ and $\psi'\sigma$ share no variables; then

$$\frac{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi'\, \sigma}{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi}$$ where ψ' is obtained from ψ by deleting a hot occurrence of a.

   For instance, if $\psi = (c \sqcap e) \mid (a \otimes f) \mid (d \vee h)$ and the hot component is a in the middle subformula, then $\psi' = (c \sqcap e) \mid f \mid (d \vee h)$.

3. *Executing elementary updates:* If $\mathcal{O}^t(\mathbf{D}_1, \mathbf{D}_2) \models^c (\exists) a\sigma$; $a\sigma$ and $\psi'\sigma$ share no variables, then

$$\frac{\mathbf{P}, \mathbf{D}_2 \text{ --- } \vdash (\exists)\, \psi'\, \sigma}{\mathbf{P}, \mathbf{D}_1 \text{ --- } \vdash (\exists)\, \psi}$$ where ψ' is obtained from ψ by deleting a hot occurrence of a.

   For instance, if $\psi = (c \sqcap e) \mid (a \otimes f) \mid (d \vee h)$ and the hot component is a in the middle subformula, then $\psi' = (c \sqcap e) \mid f \mid (d \vee h)$.
   Note that in this rule the current state changes from $\mathbf{D}_2$ to $\mathbf{D}_1$.

4. *Reasoner's move:* Let ψ be a formula with a hot component η of the form $\alpha \vee \beta$. Then we have the following pair of inference rules, which can lead to two *independent* possible derivations.

$$\frac{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi'}{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi} \qquad \frac{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi''}{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi}$$

   Here ψ' is obtained from ψ by replacing the hot component η with α and ψ'' is obtained from ψ by replacing η with β.
   For instance, if $\psi = (c \sqcap e) \mid (a \otimes f) \mid (d \vee h)$ and the hot component η is $d \vee h$, then $\psi' = (c \sqcap e) \mid (a \otimes f) \mid d$ and $\psi'' = (c \sqcap e) \mid (a \otimes f) \mid h$.

5. *Opponent's move:* Let ψ be a formula with the hot component, τ, of the form $\gamma \sqcap \delta$. Then we have the following inference rule:

$$\frac{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi' \quad \text{and} \quad \mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi''}{\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi}$$

   where ψ' is obtained from ψ by replacing the hot component τ with γ and ψ'' is obtained from ψ by replacing τ with δ.
   For instance, if $\psi = (c \sqcap e) \mid (a \otimes f) \mid (d \vee h)$ and the hot component τ is $c \sqcap e$, then $\psi' = c \mid (a \otimes f) \mid (d \vee h)$ and $\psi'' = e \mid (a \otimes f) \mid (d \vee h)$.
   Note that unlike in the reasoner's case when we have two inference rules, the opponent's case is a *single* inference rule. It says that in order to prove $\mathbf{P}, \mathbf{D} \text{ --- } \vdash (\exists)\, \psi$ (*i.e.*, to execute ψ on a set S of

paths emanating from $\mathbf{D}$) we need to be able to execute ψ' on a set S_1 of paths emanating from $\mathbf{D}$ and ψ'' on another (possibly the same) set S_2 of paths emanating from $\mathbf{D}$ such that $S = S_1 \cup S_2$.

THEOREM 1 (SOUNDNESS OF INFERENCE SYSTEM). *Under the Horn conditions, the entailment* $\mathbf{P}, \mathbf{D} \dashv\models (\exists)\,\phi$ *holds if there is a deduction of the sequent* $\mathbf{P}, \mathbf{D} \dashv\vdash \phi$ *in the above inference system.*

We conjecture that the above proof theory is also complete for workflow control specifications. The importance of the proof theory in $\mathcal{CTR}$-S is that it can be used to execute workflow specifications. When these specifications represent a service contract, the inference system will be able to execute the contract.

5. CONSTRAINTS ON CONTRACT EXECUTION

In [8], we have shown how a large class of constraints on workflow execution can be expressed in $\mathcal{CTR}$. In $\mathcal{CTR}$-S we are interested in finding a similar class of constraints, which could be used to denote the desirable properties of a contract, as explained at the end of Sections 1 and 2.2. In this context, verification of a constraint against a contract means that the reasoner has a way of executing the contract so that the constraint will hold no matter what the other parties do (for instance, that the goods are delivered or the payment is refunded regardless). Our verification algorithm requires that behavioral specifications of contracts have no loops in them and that they have the *unique event property* defined below.[5] The no-loops requirement is captured by the restriction that the workflow rules are *non-recursive* (so having rules is just a matter of convenience, which does not increase the expressive power).

We assume that there is a subset of propositions, $\mathcal{EVENT}$, which represents the "interesting" events that occur in the course of workflow execution. These events are the building blocks of both workflows and constraints. In terms of Definition 3, these propositions would be defined as built-in elementary updates.

Definition 6. (**Unique-event property**) A $\mathcal{CTR}$-S workflow $\mathcal{W}$ has the $\boldsymbol{unique\ event\ property}$ if and only if every proposition in $\mathcal{EVENT}$ can execute at most once in any execution of $\mathcal{W}$. Formally, this can be defined both model-theoretically and syntactically. The syntactic definition is that for every proposition $e \in \mathcal{EVENT}$:

> If $\mathcal{W}$ is $\mathcal{W}_1 \otimes \mathcal{W}_2$ or $\mathcal{W}_1 \mid \mathcal{W}_2$ and e occurs in $\mathcal{W}_1$ then it cannot occur in $\mathcal{W}_2$, and vice versa.

For workflows with no loops, we can always rename different occurrences of the same type of event to satisfy the above property. We shall call such workflows $\boldsymbol{unique\ event\ workflows}$.

Definition 7. (**Constraints**) Let ϕ be a $\sqcap$-free formula. Then $*\phi$ denotes a formula that is true on a set of m-paths, S, if and only if ϕ is true on *every* m-path in S. The operator $*$ can be expressed using the basic machinery of $\mathcal{CTR}$-S.

Our constraints on workflow execution, defined below, will all be of the form $*\phi$ because, intuitively, the most common thing that a reasoner wants is to make sure that *every* execution in the outcome has certain desirable properties. Items 1–3 define *primitive* constraints, denoted $\mathcal{PRIMITIVE}$. Item 4 defines the set $\mathcal{CONSTR}$ of all constraints.

1. **Elementary primitive constraints**: If $e \in \mathcal{EVENT}$ is an event, then $*e$ and $*(\neg e)$ are primitive constraints. Informally, the constraint $*e$ is true on a set S of m-paths in an m-path structure $\mathbf{I} = (U, I_{\mathcal{F}}, I_{path})$ iff e occurs on every m-path in S. Similarly, $*(\neg e)$ is true on S iff e does *not* occur on any m-path in S.

   Formally, $*e$ says that every execution of the contract, *i.e.*, every m-path $\langle p_1, ..., p_i, ..., p_n\rangle \in S$, includes a path, p_i, of the form $d_1 ... d_k d_{k+1} ... d_m$, such that for some pair of adjacent states, d_k and d_{k+1}, the event e occurs at d_k and causes a state transition to d_{k+1}, *i.e.*, $I_{path}(\langle d_k d_{k+1}\rangle) \models^{classic} e$ (see Definition 3). The constraint $*(\neg e)$ means that e does not occur on any m-path in S.

2. **Disjunctive and Conjunctive Primitive constraints**: Any $\vee$, $\wedge$, or $\neg$ combination of propositions from $\mathcal{EVENT}$ is allowed under the scope of $*$. For instance, $*(e_1 \vee e_2)$ and $*(e_1 \vee \neg e_2)$ are allowed. The former is true on a set of m-paths, S, if either e_1 or e_2 occurs on every m-path in S. The latter is true if, for every m-path in S, the occurrence of e_1 on the m-path implies that e_2 occurs on the same m-path. For $*(e_1 \vee \neg e_2)$, we will use the abbreviation $*(e_2 \rightarrow e_1)$.

3. **Serial primitive constraints**: If $e_1, ..., e_n \in \mathcal{EVENT}$ then $*(e_1 \otimes ... \otimes e_n)$ is a primitive constraint. It is true on a set of m-paths S iff e_1 occurs before e_2 before ... before e_n on every path in S.

4. **Complex constraints**: The set of all constraints, $\mathcal{CONSTR}$, consists of all Boolean combinations of primitive constraints (*i.e.*, constraints defined by Items 1–3) using the connectives $\vee$ and $\wedge$: $\phi \wedge \psi$ (resp. $\phi \vee \psi$) is satisfied by a set of m-paths S iff S satisfies ϕ and (resp. or) ψ.

It can be shown that under the unique event assumption any serial primitive constraint can be decomposed into a conjunction of binary serial constraints. For instance, $*(e_1 \otimes e_2 \otimes e_3)$ is equivalent to $*(e_1 \otimes e_2) \wedge *(e_2 \otimes e_3)$. Here are some typical constraints in $\mathcal{CONSTR}$ and their real-world meaning:

$*e$ — event e should always eventually happen;
$*e \wedge *f$ — events e and f must always both occur (in some order);
$*(e \vee f)$ — always either event e or event f or both must occur;
$*e \vee *f$ — either always event e occurs or always event f occurs;
$*(\neg e \vee \neg f)$ — it is not possible for e and f to happen together;
$*(e \rightarrow f)$ — if event e occurs, then f must also occur.

Example 3. (**Contract Goals**) The actors in the procurement workflow of Example 1 may want to ensure that they have a way to reach their goals within the scope of the contract. We express these goals using the following set of constraints:

$*(approve \wedge \neg cancel \rightarrow satisfied)$
if financing is *approve*d and buyer does not *cancel* then she wants to be *satisfied*
$*(cancel \rightarrow keep\_escrow)$
if buyer *cancel*s, then seller *keep*s the escrow
$*(make\_payment \rightarrow deliver)$
if buyer *pay*s, then seller must *deliver*

[5]This assumption is made by virtually all formal approaches to workflow modeling (*e.g.*, [2, 22]) and even such specification languages as WSFL — IBM's proposal for a Web service specification language that was one of the inputs to BPEL4WS [15].

6. ENFORCEMENT OF EXECUTION CONSTRAINTS

Given a contract represented as a $\mathcal{CTR}$-S workflow, $\mathcal{W}$, and a reasoner's goal specified as a constraint, $\Phi \in \mathcal{CONSTR}$, we would like to construct another workflow, $\mathcal{W}_\Phi$, that represents a class of strategies that *enforce* Φ within the rules of the contract. "Enforcement" here means that as long as the opponent plays by the rules of the contract (*i.e.*, chooses only the alternatives specified by the $\sqcap$-connective), the reasoner can still ensure that all the plays belong to an outcome that satisfies the constraints. In $\mathcal{CTR}$-S this amounts to computing $\mathcal{W} \wedge \Phi$ — the formula that represents the collection of all reasoner's strategies for finding the outcomes of $\mathcal{W}$ that satisfy the constraint Φ.

We must clarify what we mean by "computing" $\mathcal{W} \wedge \Phi$. After all, $\mathcal{W} \wedge \Phi$ already *is* a representation of all the winning strategies for the reasoner and all outcomes of $\mathcal{W} \wedge \Phi$ satisfy Φ. However, this formula is not an explicit set of instructions to the workflow scheduler. In fact, it is not even a workflow specification in the sense of Definition 1. In contrast, the formula $\mathcal{W}_\Phi$ that we are after *is* a workflow specification. It does not contain the $\wedge$-connective, and it can be directly executed by the $\mathcal{CTR}$-S proof theory. This means that the proof theory can be used to execute the contract $\mathcal{W}$ in such a way that the reasoner's objectives Φ will be satisfied. The precise relationship between $\mathcal{W} \wedge \Phi$ and $\mathcal{W}_\Phi$ will be established in Definition 8. Informally speaking, the two formulas are equivalent in the sense that they have exactly the same executions modulo certain synchronization acts.

We now develop an algorithm for computing $\mathcal{W}_\Phi$ through a series of transformations.

Enforcing complex constraints. Let $*C_1, *C_2 \in \mathcal{CONSTR}$, $\mathcal{W}$ be a $\mathcal{CTR}$-S workflow, then

$$(*C_1 \vee *C_2) \ \wedge \ \mathcal{W} \equiv (*C_1 \ \wedge \ \mathcal{W}) \ \vee \ (*C_2 \ \wedge \ \mathcal{W})$$
$$(*C_1 \wedge *C_2) \ \wedge \ \mathcal{W} \equiv (*C_1 \ \wedge \ (*C_2 \ \wedge \ \mathcal{W}))$$

Thus, we can compute $\mathcal{W}_{(*C_1 \vee *C_2)}$ as $\mathcal{W}_{*C_1} \ \vee \ \mathcal{W}_{*C_2}$ and $\mathcal{W}_{(*C_1 \wedge *C_2)}$ as $\mathcal{W}_{*C_1} \ \wedge \ \mathcal{W}_{*C_2}$.

Enforcing elementary constraints. The following transformation takes an elementary primitive constraint Φ of the form $*\alpha$ or $*\neg\alpha$ and a $\mathcal{CTR}$-S unique-event workflow $\mathcal{W}$, and returns a $\mathcal{CTR}$-S workflow that is equivalent to $\mathcal{W} \wedge \Phi$. Let $\alpha, \beta \in \mathcal{EVENT}$ and $\mathcal{W}_1, \mathcal{W}_2$ be $\mathcal{CTR}$-S workflows. Then:

$$*\alpha \ \wedge \ \alpha \ \equiv \ \alpha \qquad *\neg\alpha \ \wedge \ \alpha \ \equiv \ \neg\texttt{Playset}$$
$$*\alpha \ \wedge \ \beta \ \equiv \ \neg\texttt{Playset} \qquad *\neg\alpha \ \wedge \ \beta \ \equiv \ \beta \ \text{ if } \alpha \neq \beta$$

$$*\alpha \wedge (\mathcal{W}_1 \otimes \mathcal{W}_2) \equiv (*\alpha \wedge \mathcal{W}_1) \otimes \mathcal{W}_2 \vee \mathcal{W}_1 \otimes (*\alpha \wedge \mathcal{W}_2)$$
$$*\neg\alpha \ \wedge \ (\mathcal{W}_1 \otimes \mathcal{W}_2) \equiv (*\neg\alpha \ \wedge \ \mathcal{W}_1) \otimes (*\neg\alpha \ \wedge \ \mathcal{W}_2)$$
$$*\alpha \wedge (\mathcal{W}_1 \mid \mathcal{W}_2) \equiv (*\alpha \wedge \mathcal{W}_1) \mid \mathcal{W}_2 \vee \mathcal{W}_1 \mid (*\alpha \wedge \mathcal{W}_2)$$
$$*\neg\alpha \ \wedge \ (\mathcal{W}_1 \mid \mathcal{W}_2) \equiv (*\neg\alpha \ \wedge \ \mathcal{W}_1) \mid (*\neg\alpha \ \wedge \ \mathcal{W}_2)$$

(Recall that `Playset` was introduced in Definition 4.) These logical equivalences are identical to those used for workflows of cooperating tasks in [8]. The first equivalence below is specific to $\mathcal{CTR}$-S. Here we use Φ to denote $*\alpha$ or $*\neg\alpha$:

$$\Phi \ \wedge \ (\mathcal{W}_1 \sqcap \mathcal{W}_2) \equiv (\Phi \ \wedge \ \mathcal{W}_1) \ \sqcap \ (\Phi \ \wedge \ \mathcal{W}_2)$$
$$\Phi \ \wedge \ (\mathcal{W}_1 \vee \mathcal{W}_2) \equiv (\Phi \ \wedge \ \mathcal{W}_1) \vee (\Phi \ \wedge \ \mathcal{W}_2)$$

For example, if $\mathcal{W}$ is $abort \sqcap prepare \otimes (abort \vee commit)$, then:

$$*abort \ \wedge \ \mathcal{W} \equiv abort \ \sqcap \ (prepare \otimes abort)$$
$$*\neg abort \ \wedge \ \mathcal{W} \equiv \neg\texttt{Playset}$$

The above equivalences enable us to decompose the problem of computing $\mathcal{W}_\Phi$ into simpler problems. For instance, $(\alpha)_{*\alpha} \ = \ \alpha$, and to compute $(\mathcal{W}_1 \sqcap \mathcal{W}_2)_\Phi$ it suffices to compute $(\mathcal{W}_1)_\Phi \ \sqcap \ (\mathcal{W}_2)_\Phi$.

Enforcing serial constraints. Next we deal with constraints of the form $*(\alpha \otimes \beta)$. Let $\alpha, \beta \in \mathcal{EVENT}$ and let $\mathcal{W}$ be a $\mathcal{CTR}$-S workflow. We define $\mathcal{W}_{*(\alpha\otimes\beta)}$ as

$$\texttt{sync}(\alpha < \beta, \ (*\alpha \ \wedge \ (*\beta \ \wedge \ \mathcal{W})))$$

where `sync` is a transformation that synchronizes events in the right order. It uses elementary updates $send(\xi)$ and $receive(\xi)$ and is defined as follows: $\texttt{sync}(\alpha < \beta, \mathcal{W}) \ = \ \mathcal{W}'$, where $\mathcal{W}'$ is like $\mathcal{W}$, except that every occurrence of event α is replaced with $\alpha \otimes send(\xi)$ and every occurrence of β with $receive(\xi) \otimes \beta$, where ξ is a new constant.

The primitives *send* and *receive* can be defined as $insert(channel(\xi))$ and $channel(\xi) \otimes delete(channel(\xi))$, respectively, where ξ is a new proposition symbol. The point here is that these two primitives can be used to provide synchronization: $receive(\xi)$ can become true if and only if $send(\xi)$ has been executed previously. In this way, β cannot start before α is done. The following examples illustrate the definition of $\mathcal{W}_{*(\alpha\otimes\beta)}$:

$$(\beta \otimes \alpha)_{*(\alpha\otimes\beta)} \ = \ receive(\xi) \otimes \beta \otimes \alpha \otimes send(\xi)$$
$$(\alpha \mid \beta \mid \rho_1 \mid ... \mid \rho_n)_{*(\alpha\otimes\beta)} \ =$$
$$(\alpha \otimes send(\xi)) \ \mid (receive(\xi) \otimes \beta) \ \mid \ \rho_1 \ \mid \ ... \ \mid \ \rho_n)$$

Note that $\mathcal{W}_{*(\alpha\otimes\beta)}$ is not logically equivalent to $\mathcal{W} \wedge *(\alpha \otimes \beta)$, but these two formulas are *behaviorally equivalent* as defined next.

Definition 8. (**Behavioral Equivalence**) A $\mathcal{CTR}$-S formula ϕ ***behaviorally entails*** ψ iff for every m-path structure $\mathbf{I} = (U, I_\mathcal{F}, I_{path})$ and a set on m-paths S, if $\mathbf{I}, S \models \phi$ then there is another set S' such that $\mathbf{I}, S' \models \psi$, where S' and S are *congruent modulo synchronization*; namely, these sets of m-paths reduce to the same set of m-paths under the following operations:

- Remove all synchronization messages inserted by the *send* and *receive* primitives (*i.e.*, the atoms of the form $channel(\xi)$) from all database states in S' and S.
- Eliminate adjacent duplicate states from all m-paths (*i.e.*, if π is an m-path of the form $\langle ..., \ d_1 ... d_k d_{k+1} ..., \ ... \rangle$ and d_k is the same state as d_{k+1} then delete d_{k+1} from π. (After the previous step, such adjacent duplicate states normally correspond to state transitions that occur due to the execution of *send* and *receive*.)

ϕ and ψ are ***behaviorally equivalent*** if each behaviorally entails the other.

Proposition 1. (**Enforcing Elementary and Serial Constraints**) The above transformations compute a $\mathcal{CTR}$-S workflow that is behaviorally equivalent to $\mathcal{W} \wedge \Phi$, where Φ is an elementary or serial constraint and $\mathcal{W}$ is a unique event workflow.

Enforcing conjunctive primitive constraints. To enforce a primitive constraint of the form $*(\Phi_1 \wedge ... \wedge \Phi_m)$, where all Φ_i are elementary, we utilize the logical equivalence $*(\Phi_1 \wedge ... \wedge \Phi_m) \ \equiv \ * \Phi_1 \wedge ... \wedge *\Phi_m$ (and the earlier equivalences for enforcing complex constraints).

Enforcing disjunctive primitive constraints. These constraints have the form $*(\Phi_1 \vee ... \vee \Phi_n)$, where all Φ_i are elementary constraints. Enforcement of such constraints relies on the following lemma.

Lemma 1. (**Disjunctive Primitive Constraints**) Let Φ_i be elementary constraints. Then

$$*(\Phi_1 \vee ... \vee \Phi_n) \equiv (*\neg\Phi_2 \wedge ... \wedge *\neg\Phi_n \rightarrow *\Phi_1) \sqcap ...$$
$$\sqcap (*\neg\Phi_1 \wedge ... \wedge *\neg\Phi_{i-1} \wedge *\neg\Phi_{i+1} \wedge ... \wedge *\neg\Phi_n \rightarrow *\Phi_i) \sqcap ...$$
$$\sqcap (*\neg\Phi_1 \wedge ... \wedge *\neg\Phi_{n-1} \rightarrow *\Phi_n)$$

This equivalence allows us to decompose the set of all plays in an outcome into subsets that satisfy the different implications shown in the lemma. Unfortunately, enforcing such implications is still not easy. Unlike the other constraints in this section, enforcement of the implicational constraints cannot be described by a series of simple equivalences. Instead, we have to resort to equivalence transformations defined with the help of parse trees of $\mathcal{CTR}$-S formulas that represent unique event workflows.

Definition 9. (**Maximal guarantee for an event**) Let $*\Phi$ be an elementary constraint (*i.e.*, Φ is e or $\neg e$), $\mathcal{W}$ be a formula for a unique event workflow, and φ be a subformula of $\mathcal{W}$. Then φ is said to be a $\boldsymbol{maximal\ guarantee}$ for $*\Phi$ iff

1. $(\mathcal{W} \wedge (\varphi \mid \texttt{Playset})) \models *\Phi$
2. φ is a maximal subformula of $\mathcal{W}$ that satisfies (1)

Intuitively, a maximal guarantee for $*e$ is any maximal subformula, ϕ, of $\mathcal{W}$ such that e occurs in *every* execution of ϕ. A maximal guarantee for $*\neg e$ is such a maximal subformula, ϕ, of $\mathcal{W}$ that e does not occur in *any* execution of ϕ. The set of all maximal guarantees for an elementary event $*\Phi$ is denoted by $GS_{*\Phi}(\mathcal{W})$.

Definition 10. (**Co-executing sub-formulas**) Let $\mathcal{W}$ be a formula for a unique event workflow and ψ, φ be a pair of subformulas of $\mathcal{W}$. We say that ψ $\boldsymbol{coexecutes}$ with φ in $\mathcal{W}$, denoted $\psi \in coExec(\mathcal{W}, \varphi)$, iff

1. $(\mathcal{W} \wedge (\varphi \mid \texttt{Playset})) \models (\psi \mid \texttt{Playset})$,
2. ϕ and ψ are disjoint subformulas in $\mathcal{W}$, and
3. ψ is a maximal subformula in $\mathcal{W}$ satisfying (1) and (2)

Intuitively members of $coExec(\mathcal{W}, \varphi)$ correspond to maximal sub-formulas of $\mathcal{W}$ that *must* be executed whenever φ executes as part of the workflow $\mathcal{W}$.

Proposition 2. (**Computing $GS_{*\Phi}(\mathcal{W})$ and $coExec(\mathcal{W}, \varphi)$**) The following procedures compute $GS_{*\Phi}(\mathcal{W})$ and $coExec(\mathcal{W}, \varphi)$. They operate on the parse tree of $\mathcal{W}$, which is defined as usual: the inner nodes correspond to composite subformulas and the leaves to atomic subformulas. Thus, the leaves are labeled with their corresponding atomic subformulas, while the inner nodes are labeled with the main connective of the corresponding composite subformula.

$GS_{*e}(\mathcal{W})$**:** The set of subformulas that correspond to the nodes in the parse tree of $\mathcal{W}$ that are closest to the root of the tree and can be reached by the following marking procedure: (i) mark all the leaves labeled with e; (ii) mark any node corresponding to a subformula of $\mathcal{W}$ of the form $(\varphi \otimes \psi)$ or $(\varphi \mid \psi)$ if either the node for φ or for ψ is marked; (iii) mark any node corresponding to a subformula of the form $(\varphi \vee \psi)$ or $(\varphi \sqcap \psi)$ if both the node for φ *and* the node for ψ are marked.

$coExec(\mathcal{W}, \varphi)$**:** Consider a subformula of $\mathcal{W}$ of the form $\psi_1 \mid \psi_2$, $\psi_2 \mid \psi_1$, $\psi_1 \otimes \psi_2$, or $\psi_2 \otimes \psi_1$, where ϕ is a subformula of ψ_1. Suppose that ψ_2 is a maximal subformula with such a property, *i.e.*, $\mathcal{W}$ has no subformula of the form $\psi_1' \mid \psi_2'$, $\psi_2' \mid \psi_1'$, $\psi_1' \otimes \psi_2'$, or $\psi_2' \otimes \psi_1'$, respectively, such that ϕ is a subformula of ψ_1' and ψ_2 is a subformula of ψ_2' ($\psi_2' \neq \psi_2$). Then, $\psi_2 \in coExec(\mathcal{W}, \varphi)$.

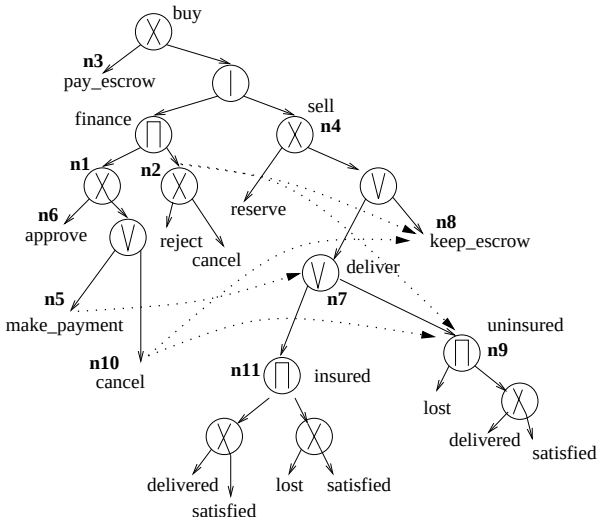

Figure 1: Workflow parse tree and workflow graph for Example 3

$GS_{*\neg e}(\mathcal{W})$**:** Let T be the set of nodes in the parse tree of $\mathcal{W}$ that belong to any of the subformulas $\varphi \in GS_{*e}(\mathcal{W})$ or $\psi \in coExec(\mathcal{W}, \varphi)$. Then, $\eta \in GS_{*\neg e}(\mathcal{W})$ iff it is a subformula of $\mathcal{W}$ such that its subtree contains no nodes in T and η is a maximal subformula with such a property.

Example 4. (**Computation of maximal guarantees and co-execution**) The workflow parse tree for Example 3 is shown in Figure 1. For the event $approve$, the highest node that can be reached by the aforesaid marking procedure is $n1$ and there are no other such nodes. Therefore, $GS_{*approve}(\mathcal{W})$ is $\{approve \otimes (make\_payment \vee cancel)\}$. The set of co-executing subformulas for $n1$, $coExec(\mathcal{W}, n1)$, consists of two formulas that correspond to the nodes $n3$ and $n4$ in the figure. The only maximal guarantee for $\neg approve$ is the subformula $reject \otimes cancel$, which corresponds to node $n2$.

procedure Compute $\mathcal{W}_{(*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow *\Phi)}$
1. **if** $\mathcal{W} \models *\Phi_1 \wedge ... \wedge *\Phi_n$ **then** Compute $\mathcal{W}_{*\Phi}$
2. **else** $Guard(g) := \emptyset$ for all $g \in subformulas(\mathcal{W})$
3. **for** each i such that $\mathcal{W} \not\models *\Phi_i$ **do**
4. **for** each $f \in GS_{*\neg\Phi_i}(\mathcal{W})$ **do**
5. **if** $\exists h \in coExec(\mathcal{W}, f)$ and $(h \wedge *\Phi)$ is satisfiable
6. **then** Rewrite f to $send(\xi) \otimes f$ and
7. for every $g \in GS_{*\neg\Phi}(h)$ set
8. $Guard(g) := Guard(g) \cup \{receive(\xi)\}$
9. **else** Compute $sibling(f)_{(*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow *\Phi)}$
10. **for** each $g \in subformulas(\mathcal{W})$ **do**
11. **if** $Guard(g) \neq \emptyset$ **then**
12. rewrite g to $\left(\bigvee_{receive(\xi) \in Guard(g)} receive(\xi)\right) \otimes g$

Figure 2: Computation of $\mathcal{W}_{(*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow *\Phi)}$

Recall that, according to Lemma 1, in order to enforce a disjunctive constraint we need to learn how to enforce implicational constraints of the form $*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow *\Phi$, where Φ and the Φ_is are elementary. This is done using the algorithm in Figure 2, which computes a workflow that is equivalent to $(*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow$

$*\Phi) \wedge \mathcal{W}$. If the antecedent of the constraint is true during an execution, then (in line 1) $*\Phi$ must be enforced on $\mathcal{W}$. Otherwise, for every $*\Phi_i$ that is not true everywhere, we identify the subformulas $f \in GS_{*\neg\Phi_i}$ (lines 2-3). Note that, whenever subformulas in $GS_{*\neg\Phi_i}$ are executed the constraint $*\Phi_1 \wedge ... \wedge *\Phi_n \rightarrow *\Phi$ is vacuously true. In lines 4-6, we identify the subformulas h of $\mathcal{W}$ that co-execute with the formulas $f \in GS_{*\neg\Phi_i}$. If $*\Phi$ is enforcible in any of these subformulas h, *i.e.,* $h \wedge *\Phi$ is satisfied (there can be at most one such subformula h per f, due to the unique event property, Definition 6), then we enforce the above constraint by *delaying* executions of those subformulas in h that violate $*\Phi$ (these are exactly the g's in line 7) *until* it is guaranteed that the execution moves into $f \in GS_{*\neg\Phi_i}$, because once f is executed our constraint becomes satisfied. This delay is achieved by synchronizing the executions of f to occur before the executions of g by rewriting f into $send(\xi) \otimes f$ (in line 6) and by adding $receive(\xi)$ to the guard for g (in line 8). Otherwise, if no such h exists, in line 9, we explicitly enforce the constraint on the sibling nodes (in the parse tree of $\mathcal{W}$) of the formulas $f \in GS_{*\neg\Phi_i}$ (because an outcome that satisfies $*\Phi_i$ might exist in a sibling). Finally, in lines 10-12, we make sure that the execution of every g that has a non-empty guard is conditioned on receiving of a message from at least one f with which g is synchronized.

Example 5. (**Procurement Workflow, contd.**) The algorithm in Figure 2 creates the following workflow by applying the constraints in Example 3 to the procurement workflow in Example 1. Refer to the parse tree for that workflow in Figure 1.

- To enforce $(*cancel \rightarrow *keep\_escrow)$ we first compute $GS_{*\neg cancel}(buy) = \{n5\}$, and $coExec(buy, n5) = \{n3, n4, n6\}$. Of these, $n4$ (substituted for h) satisfies the conditions on line 5 of the algorithm in Figure 2. Since $GS_{*\neg keep\_escrow}(n4) = \{n7\}$, we insert a synchronization from node $n5$ to $n7$ shown in Figure 1 as a dotted line. This ensures that if the buyer cancels the procurement workflow, the seller collects the escrow.

- To enforce $(*approve \wedge * \neg cancel \rightarrow *satisfied)$, we compute $GS_{*\neg approve}(buy) = \{n2\}$ and notice that $n4 \in coExec(buy, n2)\}$ satisfies the conditions in line 5 of the algorithm in Figure 2. Since $GS_{*\neg satisfied}(n4) = \{n8, n9\}$, we insert a synchronization from node $n2$ to $n8$ and $n9$ which yields the dotted edges in Figure 1. We also compute $GS_{*cancel}(buy) = \{n10, n2\}$ and notice that $n4 \in coExec(buy, n2)\}$, $n4 \in coExec(buy, n10)\}$, and $n4$ satisfies the conditions in line 5 of the algorithm. Since $GS_{*\neg satisfied}(n4) = \{n8, n9\}$, we insert a synchronization from the nodes $n10$ and $n2$ to $n8$ and $n9$, which yields the dotted edges in Figure 1. This synchronization ensures that if buyer's financing is approved and he chooses to make the payment and buy the item then delivery must use the insured method. Also, once the constraint $(*make\_payment \rightarrow *deliver)$ is enforced too, the seller can no longer pocket the escrow. The resulting strategy is:

$$
\begin{aligned}
&buy \longleftarrow pay\_escrow \otimes (finance \mid sell)\\
&finance \longleftarrow (approve \otimes ((send(\xi_1) \otimes make\_payment) \vee\\
&\quad ((send(\xi_3) \otimes cancel))) \sqcap (send(\xi_2) \otimes (reject \otimes cancel))\\
&sell \longleftarrow reserve\_item \otimes ((receive(\xi_1) \otimes deliver)\\
&\quad \vee ((receive(\xi_2) \vee receive(\xi_3)) \otimes keep\_escrow))\\
&deliver \longleftarrow insured \vee ((receive(\xi_2) \vee receive(\xi_3)) \otimes uninsured)\\
&insured \longleftarrow (delivered \otimes satisfied) \sqcap (lost \otimes satisfied)\\
&uninsured \longleftarrow (delivered \otimes satisfied) \sqcap lost
\end{aligned}
$$

Proposition 3. (**Enforcing disjunctive primitive constraints**) The above algorithm for enforcing disjunctive primitive constraints computes a $\mathcal{CTR}$-S workflow $\mathcal{W}_\Phi$ that is behaviorally equivalent to $\mathcal{W} \wedge *(\Phi_1 \vee ... \vee \Phi_n)$ where Φ_i are *elementary constraints*.

THEOREM 2 (DISJUNCTIVE CONSTRAINTS). *Let $\mathcal{W}$ be a control flow graph and $*\Phi \in \mathcal{PRIMITIVE}$ be a disjunctive primitive constraint. Let $|\mathcal{W}|$ denote the size of $\mathcal{W}$, and d be the number of elementary disjuncts in $*\Phi$. Then the worst-case size of $\mathcal{W}_\Phi$ is $O(d \times |\mathcal{W}|)$, and the time complexity is $O(d \times |\mathcal{W}|^2)$.*

Enforcement of arbitrary constraints. If $\Phi = \bigwedge_N(\vee_j Prim)$ where $Prim \in \mathcal{PRIMITIVE}$, we compute $\mathcal{W}_\Phi$ by applying the above transformations for complex and elementary constraints. Each transformation is either a logical equivalence or a behavioral equivalence. Therefore, $\mathcal{W}_\Phi$ is behaviorally equivalent to $\mathcal{W} \wedge \Phi$.

THEOREM 3 (ARBITRARY CONSTRAINTS). *Let $\mathcal{W}$ be a control flow graph $\mathcal{W}$ and $\Phi \subset \mathcal{CONSTR}$ be a set of global constraints in the conjunctive normal form $\bigwedge_N(\vee_j Prim)$ where $Prim \in \mathcal{PRIMITIVE}$. Let $|\mathcal{W}|$ denote the size of $\mathcal{W}$, N be the number of constraints in Φ, and d be the largest number of disjuncts in a primitive constraint in Φ. Then the worst-case size of $\mathcal{W}_\Phi$ is $O(d^N \times |\mathcal{W}|)$, and the time complexity is $O(d^N \times |\mathcal{W}|^2)$.*

Cycle detection and removal. We can still improve the above transformation by eliminating certain "useless" parts of $\mathcal{W}_\Phi$—the parts that will never be executed. The problem is that even though $\mathcal{W}_\Phi$ is an executable workflow specification, $\mathcal{W}_\Phi$ may have subformulas where the *send/receive* primitives cause a cyclic wait. This means that those parts of $\mathcal{W}_\Phi$ can never be involved in an execution. Fortunately, we can show that all cyclic waits can be removed from $\mathcal{W}$ in time $O(|\mathcal{W}|^3)$.

Example 6. (**Cyclic Wait Removal**) Let $\mathcal{W}$ be $(a \vee b) \otimes (c \sqcap d)$ and Φ be $(*c \rightarrow *a)$. Our algorithm transforms $\mathcal{W} \wedge \Phi$ into $(a \vee receive(\xi) \otimes b) \otimes (c \sqcap (send(\xi) \otimes d))$. Now, if the reasoner chooses b, a deadlock occurs. However, we can rewrite this formula into a behaviorally equivalent formula $a \otimes (c \sqcap d)$ and avoid the problem.

7. CONCLUSION AND RELATED WORK

We presented a novel formalism, $\mathcal{CTR}$-S, for modeling the dynamics of service contracts. $\mathcal{CTR}$-S is a logic in which service contracts are represented as formulas that specify the various choices that are allowed for the parties to the contracts. The logic permits the reasoner to state the desired outcomes of the contract execution and verify that a desired outcome can be achived no matter what the other parties do as long as they obey the rules of the contract.

There is a body of preliminary work trying to formalize the representation of Web service contracts [20, 11], but none deals with the dynamics of such contracts, which is the main subject of this paper. Technically, the works closest to ours come from the fields of model checking and game logics.

Process algebras and alternating temporal logic [7, 1] have been used for modeling open systems with game semantics. Model checking is a standard mechanism for verifying temporal properties of such systems and deriving automata for scheduling. In [16], the complexity and size of computing the winning strategies for infinite games played on finite graphs are explored. A result analogous to ours is obtained for infinite games: assuming the size of the graph is Q and the size of the winning condition is W, the complexity of computing winning strategies is exponential in the size of W and polynomial in the size of the set Q.

The use of $\mathcal{CTR}$-S has enabled us to find a more efficient verification algorithm than what one would get using model checking. Indeed, standard model checking techniques [7, 1] are worst-case exponential *in the size of the entire formula* and the corresponding scheduling automata are also exponential. This is often referred to as the *state-explosion problem*. In contrast, the size of our solver is *linear* in the size of the original workflow specification and exponential only in the size of the constraint set (Theorem 3), which is a much smaller object. In a sense, our solver can be viewed as a specialized and more efficient model checker for the problem at hand. It accepts high level specifications of workflows and yields strategies and schedulers in the same high level language.

Logic games have been proposed before in other contexts [13, 19]. As in $\mathcal{CTR}$-S, validity of a statement in such a logic means that the reasoner has a winning strategy against the opponent. In $\mathcal{CTR}$-S however, games, winning conditions, and strategies are *themselves* logical formulas (rather than modal operators). Logical equivalence in $\mathcal{CTR}$-S is a basis for constructive algorithms for solving games and synthesizing strategies, which are in turn executable by the proof theory of $\mathcal{CTR}$-S. Related game logic formalisms, such as [13, 19], only deal with assertions about games and their winning strategies. In these logics, games are modalities rather than executable specifications, so they can only be used for *reasoning* about Web service contracts, but *not* for *modeling* and *executing* them.

Related work in planning, where goals are expressed as temporal formulas, includes [3]. In [3], plans are generated using a forward chaining engine that generates finite linear sequences of actions. As these linear sequences are generated, the paths are incrementally checked against the temporal goals. This approach is sound and complete. However, in the worst case it performs an exhaustive search of the model similar to the model checking approaches.

For the future work, we are planning to extend our results to allow contracts that include iterative behaviour. Such contracts can already be specified in $\mathcal{CTR}$-S. However, iteration requires new verification algorithms to enable reasoning about the desired outcomes of such contracts.

8. REFERENCES

[1] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Intl. Conference on Foundations of Computer Science*, pages 100–109, 1997.

[2] P.C. Attie, M.P. Singh, E.A. Emerson, A.P. Sheth, and M. Rusinkiewicz. Scheduling workflows by enforcing intertask dependencies. *Distributed Systems Engineering Journal*, 3(4):222–238, December 1996.

[3] F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.

[4] A.J. Bonner. Workflow, transactions, and datalog. In *ACM Symposium on Principles of Database Systems*, Philadelphia, PA, May/June 1999.

[5] A.J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 142–156, Bonn, Germany, September 1996. MIT Press.

[6] A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.

[7] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, pages 244–263, 1986.

[8] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*, pages 25–33, Seattle, Washington, June 1998.

[9] E.A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, pages 997–1072. Elsevier and MIT Press, 1990.

[10] R. Fagin, J. Y. Halpern, Y. Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1994.

[11] B.N. Grosof and T.C. Poon. Sweetdeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the twelfth international conference on World Wide Web*, pages 340–349, May 2003.

[12] R. Gunthor. Extended transaction processing based on dependency rules. In *Proceedings of the RIDE-IMS Workshop*, 1993.

[13] J. Hintikka. *Logic, Language Games, and Information*. Oxford Univ. Press, Clarendon, Oxford, 1973.

[14] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, pages 741–843, July 1995.

[15] F. Leymann. Web services flow language (wsfl 1.0). Technical report, IBM, 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[16] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[17] S. Mukherjee, H. Davulcu, M. Kifer, G. Yang, and P. Senkul. Survey of logic based approaches to workflow modeling. In J. Chomicki, R. van der Meyden, and G. Saake Springer, editors, *Logics for Emerging Applications of Databases*, LNCS. Springer-Verlag, October 2003.

[18] M. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1998.

[19] Rohit Parikh. Logic of games and its applications. In *Annals of Discrete Mathematics*, volume 24, pages 111–140. Elsevier Science Publishers, March 1985.

[20] D.M. Reeves, M.P. Wellman, and B.N. Grosof. Automated negotiation from declarative contract descriptions. In J.P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 51–58, Montreal, Canada, 2001. ACM Press.

[21] C. Schlenoff, M. Gruninger, M. Ciocoiu, and J. Lee. The essence of the process specification language. *Transactions of the Society for Computer Simulation International*, 16(4):204–216, 2000.

[22] M.P. Singh. Semantical considerations on workflows: An algebra for intertask dependencies. In *Proceedings of the International Workshop on Database Programming Languages*, Gubbio, Umbria, Italy, September 6–8 1995.

[23] M.P. Singh. Synthesizing distributed constrained events from transactional workflow specifications. In *Proceedings of 12-th IEEE Intl. Conference on Data Engineering*, pages 616–623, New Orleans, LA, February 1996.