Modeling Change Without Breaking Promises

Alva L. Couch, Hengky Susanto, and Marc Chiarini

Tufts University, Medford, Massachusetts, USA
alva.couch@cs.tufts.edu, hsusan0a@cs.tufts.edu, mchiar01@cs.tufts.edu

Abstract. Promise theory defines a method by which static service bindings are made in a network, but little work has been done on handling the dynamic case in which bindings must change over time due to both contingencies and changes in policy. We define two new kinds of promises that provide temporal scope for a conditional promise. We show that simple temporally-scoped promises can describe common network behaviors such as leasing and failover, and allow an agent to completely control the sequence of sets of promises to which it commits with another agent, over time. This allows agents to adapt to changing conditions by making short-term bilateral agreements rather than the long-term unilateral agreements provided by previous promise constructions.

1 Introduction

Promise theory[1–4] provides a mechanism by which one can model intelligent autonomous service binding between clients and services. A *promise* has three parts:

1. A *sender* s that is committing to a particular behavior.
2. A *receiver* r that is receiving that commitment.
3. A *body* b of information describing the commitment.

We can describe the promise between nodes s and r as a labeled edge $\langle s, r, b\rangle$ and the sum total of all promises among a set of nodes as a *promise graph* $G = \langle V, E\rangle$, where V represents all nodes (agents), and each edge $e \in E$ is a labeled directed edge of the form $e = \langle s, r, b\rangle$, with source s, destination r, and label b.[1]

Promise theory allows one to easily characterize the function of network services in terms of promises between entities. For example, we can describe DNS, file service, and web service in terms of kinds of promises[5]. For ease of notation, we utilize wildcards to denote promises to everyone: $\langle s, *, b\rangle$ means "s promises b to every node", while $\langle *, r, b\rangle$ means that "r is promised b by every node."

Prior work on promise theory identified three kinds of promises: "regular", "use", and "coordination". A *regular promise* is a commitment to provide some

[1] We find it instructive to utilize the notation of traditional graph theory, instead of the labeled arrow notation introduced by Burgess et al. This notation allows us to construct derived graphs with a simple notation.

service (e.g. b), while a *use promise* is a commitment to use some service (e.g., $U(b)$). A "coordination promise" obligates one agent to follow the "instructions" given by another agent (e.g., $C(b)$). In addition, a "commitment promise"[2] is a special kind of promise body b representing the commitment of a *non-returnable investment*. In this paper, we do not consider commitment promises, and use the term "commitment" to refer more generically to any promise made.

Recently, *conditional promises* were introduced in order to encode simple interactions between promises[3, 6–8]. A conditional promise is a promise that is contingent upon the validity of other specified promises.

Definition 1. *A promise is* primitive *if it consists of a single promise body, transmitted from a single originator to a single recipient, with no conditions stated.*

A conditional promise is a promise that is held by an agent, but whose validity is contingent upon the validity of other stated promises[1]:

Definition 2. *A* conditional promise *has the form* $(p|q_1, q_2, \ldots, q_k)$ *where* p *is a* consequent *primitive promise and* $q_1, q_2, \ldots, q_k$ *are* antecedent *primitive promises.*

A primitive promise p can be thought of as a conditional promise $(p|)$ with an empty set of antecedents. In the above definition, each of p and $q_1, \ldots, q_k$ are promises with sender, receiver, and body, e.g., $q_i = \langle s_{qi}, r_{qi}, b_{qi} \rangle$. At this point, the values of s_{qi}, r_{qi}, and b_{qi} are unconstrained, though we will discuss later what it means for such a promise to be meaningful in the context of a particular agent.

The purpose of a conditional promise $(p|q_1, \ldots, q_k)$ is to state conditions under which a consequent promise p is considered to be valid. To make discussion easier, we will refer to a promise that is valid in a particular context as *operative* in that context, and a promise that is not valid as *inoperative*. A promise that is not valid but could potentially become valid in the future is *latent*.

Axiom 1. *All primitive promises in a set* C *are operative with respect to* C*.*

Axiom 2. *A conditional promise* $c = (p|q_1, \ldots, q_k)$ *is operative with respect to a set of conditional promises* C *precisely when each of* $q_1, \ldots, q_k$ *is operative with respect to* C*.*

Definition 3. *For a conditional promise* $c = (p|q_1, \ldots, q_k)$*, we say* p *is operative (with respect to* C*) whenever* c *is operative (with respect to* C*).*

A conditional promise describes *one way* that a primitive promise can become operative. It can also become operative, e.g., by being promised explicitly and unconditionally, or by being a consequent of some other conditional promise whose antecedents are operative. Thus the conditional construction is *sufficient but not necessary;* it is quite possible that the consequent of a conditional becomes operative by means other than that particular conditional.

One simple extension of basic conditional promises allows conditional promises to express arbitrary zeroth-order logical statements inside conditions as *sets* of simple conditional promises.

Definition 4. *If p is a promise, then the promise $\neg p$ is the assertion that p is not operative, which is operative exactly when p is not operative.*

Theorem 1. *A conditional promise involving any of the logical operators $\rightarrow$ (implication), $\leftrightarrow$ (equivalence), $\wedge$ (conjunction), $\vee$ (disjunction) or $\oplus$ (exclusive-or) in the condition can be represented as a* set *of conditional promises of the above form.*

Proof. Note first that conditional promises are conjunctions: $(p|q_1, \ldots, q_k)$ means $(p|q_1 \wedge \cdots \wedge q_k)$. Disjunctions are represented by sets of promises: The single conditional $(p|q_1 \vee q_k)$ is equivalent to the set of conditionals $\{(p|q_1), \ldots, (p|q_k)\}$. Exclusive-or is represented as a set of alternatives: $(p|q \oplus r)$ means $\{(p|q \wedge \neg r), (p|\neg q \wedge r)\}$. Implication $(p|q \rightarrow r)$ means $p|\neg q \vee r$, which expands to $\{(p|\neg q), (p|r)\}$. Equivalence is similar. □

In this paper, we intentionally avoid first-order logical constructions such as quantifiers and variables. One reason for this is that we seek to extend the capabilities of the configuration management tool CFEngine[9–13], which contains only the ability to interpret zeroth-order logical expressions[14]. Another reason for excluding quantifiers is that a condition that uses quantification over a finite, known set is logically equivalent to a set of zeroth-order conditionals: if $S = \{s_1, \ldots, s_n\}$, then the set $\{(p|\forall(x \in S)x)\}$ is equivalent to the set $\{(p|s_1, \ldots, s_n)\}$ (which we can notate as $(p|S)$ without ambiguity), while the set $\{p|\exists(x \in S)x\}$ is equivalent to the set $\{(p|s_1), (p|s_2), \ldots, (p|s_n)\}$. Thus, we can express a first-order condition involving quantification over finite sets of promises as a *finite set of conditional promises* in our notation.

2 Related work

Burgess et al. continue to refine promise theory and its domains of applicability. The basic framework above was described in [1], including conditional promises. We differ from this view in one important respect: Burgess suggests that temporal logic cannot be used due to lack of knowledge of prerequisites. We show in this paper that a simple concept of sequencing, based upon mutual observability, is sufficient to allow simple kinds of temporal constructions.

[2] explores the use of voluntarily collaborating agents to perform system administration tasks in an uncertain environment with minimal trust. We agree with these conclusions, but also believe that more trust is possible with the addition of new kinds of promises, described here.

[4] contrasts promise theory with traditional control theory and introduces the notion of promise *reliability* (probability that a promise will be kept) to reduce promise graphs and enable spectral analysis of relationships. This is one approach to understanding evolution of services; our model is complementary to this and involves hard bindings that change over time.

In [7], Bergstra and Burgess apply promise theory to modeling trust relationships, reputation, and expectation. Their discussion of how bundled promises

(made in parallel) should affect trust includes an XOR scenario in which mutually exclusive conditional promises act like a switch. This inspires our temporal operators, which are used in a similar manner.

In [3], Burgess and Fagernes introduce the Common Currency Equivalent Graph and extract eigenvectors from its matrix representation to determine the sustainability of network policy. The paper remarks on chains of conditional promises requiring an external mediating component to act as a broker. This informs our own ideas regarding knowledge binding.

Note that while we add operators that allow a new kind of dynamic scoping, at any particular time, our promise network is reducible to one in which the scoping operators do not appear. Thus all prior results apply, except that the "hard bindings" in our networks can change over time.

There has also been much work on how promises can be reasoned about within an agent. This requires a policy that is (regardless of representation) equivalent to a set of first-order rules[15]. While this is an important topic, we avoid specifically describing policies here, and concentrate instead on what a particular set of promises means, and how that meaning can evolve over time, regardless of "why" specific promises were made or what policies caused them to be promised.

3 Temporal scoping

One limit of prior promise theory is that although one can prove that a set of promises creates a functional network, there is no mechanism by which promises can change over time except by "breaking promises". A "broken promise" occurs when an agent promises something contradictory to a prior promise. The receiving agent may consider this as evidence of untrustability in its trust model[7]. Thus promise-based networks evolve toward a state of stability that is immutable unless agents break their promises.

Then how do promise networks react to changes in policy and needs? At present, the only way an agent can be ready for change (without explicitly breaking promises) is to fail to commit to a particular course of action. For example, if an agent X is offered the same service from servers A, B, and C, and X commits to use one exclusively, then it is implicitly promising – forever – not to use the others, until it breaks that promise. Thus the agent must maintain relatively weak bindings with others in order to remain ready to change bindings without breaking promises. This means, in turn, that the potential servers A, B, C can never know client X's full intent; even if X intends to use A exclusively, it cannot make that intent clear unless it also – in the future – potentially issues a contradictory promise. Agents unable to promise to use services represent points of instability in a network, rather than points of stability. A new notion is needed, that allows an agent to be predictable (and commit to a particular course of action) *for some period of time* and then make choices about future action.

3.1 α and τ

In this paper, we suggest two mechanisms for "scoping" promises in time by means of two new promise bodies. The promise body $\tau(t)$ is operative from the time that it is received to the time t time units in the future. The promise body $\alpha(p)$ is operative from the time it is received to the time that primitive promise p becomes operative (Figure 1). If p is already operative then $\alpha(p)$ never becomes operative.

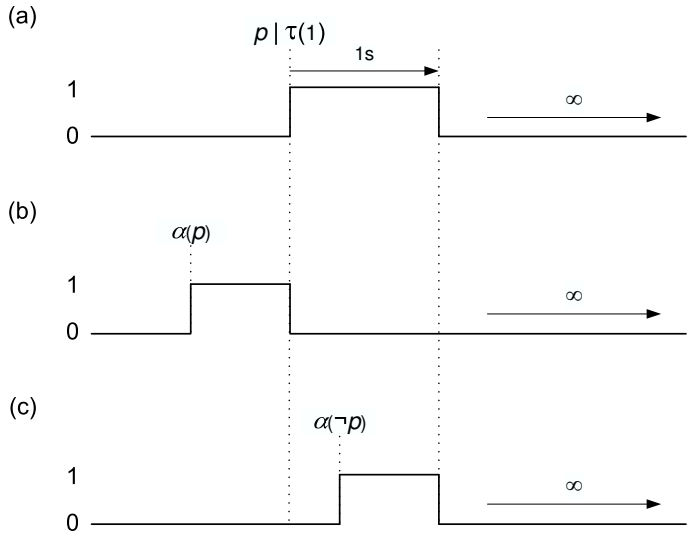

Fig. 1. Temporal scoping. (a) The τ promise is given such that p is operative for one second. (b) The α promise is operative until p becomes operative. (c) The α of a negated promise p is operative until p becomes non-operative.

To simplify notation, we often omit details that can be inferred from context. Since α and τ only make sense in conditional promises, it is implicit that their initiator is the sender in the consequent and that their receiver is the receiver in the consequent. Thus we will write $(\langle s, r, b\rangle|\alpha(p))$ to mean $(\langle s, r, b\rangle|\langle s, r, \alpha(p)\rangle)$ and $(\langle s, r, b\rangle|\tau(t))$ instead of $(\langle s, r, b\rangle|\langle s, r, \tau(t)\rangle)$. Thus we will refer to the *promises* $\alpha(p)$ and $\tau(t)$ without confusion.

τ and α allow one to make conditional promises whose consequents are operative for a limited time. The promise $(p|\tau(t))$ means that p is operative for time t, and then inoperative thereafter. Likewise, the promise $(p|\alpha(q))$ means that p is operative until replaced by a "presumably better" promise q.

This means, in particular, that any promise containing an (unnegated) α or τ condition may be *permanently deleted* from an agent's knowledge base after the state transition from operative to inoperative has been accomplished. In other

words, $(p|\alpha(q))$ and $(t|\tau(t))$ (and any conditional promises containing them) have a *limited lifetime* that is the temporal scope implied by their conditionals. Once α and τ become inoperative, they never become operative again, and any promise they condition can be forgotten forever.

Conversely, any promise containing the *negation* of an α or τ construction cannot become operative *until* the clause is fulfilled. $(p|\neg\alpha(q))$ means that p becomes operative only when q has become operative. Likewise, $(p|\neg\tau(t))$ means that p becomes operative after time t. This means that after any negation of α or τ becomes operative, it can be *omitted from the conditional expression that contains it* from then on.

Combining negative and positive temporal operators allows one to specify any scope whatever in time. For example, $(p|\neg\tau(t_1), \tau(t_2))$ means that p becomes operative from the time t_1 units in the future, until the time t_2 units in the future. Likewise, $(p|\neg\alpha(q), \alpha(r))$ says that p becomes operative from when q becomes operative, to when r becomes operative.

3.2 Leasing

Several simple examples can illustrate the utility of α and τ in modeling common service binding behaviors. A lease is a time-limited promise. The use of a lease is – in turn – similarly limited in time. Let us model a typical DHCP lease in promise theory:

1. The client requests a lease. Requests are not promises.
2. $\langle s, c, b\rangle|\tau(t), \langle c, s, U(b)\rangle$: each server responds with a time-limited offer.
3. $\langle c, s, U(b)|\tau(t)$: the client responds to one server with an acceptance.

(See Figure 2a).

3.3 Gating

Another common situation is that a client wants to commit to a service until it decides to do otherwise. We can accomplish this via an "abstract" promise. An abstract promise has a body that has no behavioral effect nor any effect upon commitment except to gate a condition. Let us rewrite the leasing example to be gated instead:

1. The client requests a lease.
2. $\langle s, c, b\rangle|\langle c, s, U(b)\rangle$: each server responds with a conditional offer.
3. $\langle c, s, U(b)\rangle|\alpha(\langle c, s, \text{not}\rangle)$: the client responds with a gated acceptance, where the abstract body "not" represents the promise that ends the commitment.
4. $\langle c, s, \text{not}\rangle|\tau(0)$: the client, when it wishes to disconnect, issues a "gate promise".

The last is a promise that becomes operative and non-operative at a single time, thus nullifying α in the previous promise (and, in doing so, nullifying the entire promise) but becoming inoperative itself immediately after (Figure 2b). Note

that gating can be coded in either direction. One could, e.g., allow the server to renege instead of the client, via:

$$\langle c, s, U(b)\rangle | \alpha(\langle s, c, \text{not}\rangle)$$

or even allow either to unilaterally back out of the agreement, via:

$$\langle c, s, U(b)\rangle | \alpha(\langle c, s, \text{not}\rangle), \alpha(\langle s, c, \text{not}\rangle)$$

A gated promise avoids "breaking promises" by declaring in advance which promises *will* become inoperative, and the events that will make them inoperative.

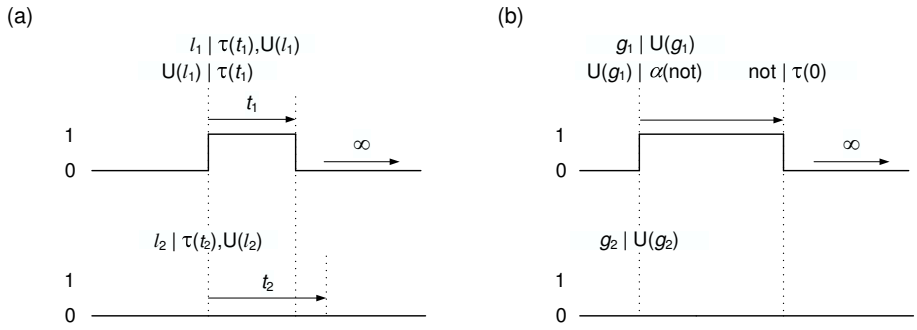

Fig. 2. (a) Lease (promise) l_1 is operative, conditioned on a use promise and the time being less than the current time plus t_1. Lease l_2 never becomes operative. (b) Gating. Promise g_1 is operative until such time as its recipient decides otherwise. g_2 never becomes operative.

4 Observability and knowledge

Before we can utilize timing and gating in more complex situations, we need to grapple with exactly when α and τ conditions make sense. A key element of promise theory is that an agent only has access to the promises that it makes and receives. In the above examples, it is clear what α and τ mean, because they involve only two servers with some concept of shared knowledge, but in more complex constructions, one must be careful not to write conditional expressions with no reasonable meaning. The constructions that are meaningful are those in which the agents involved in a promise transaction *can mutually observe the outcome* of the transaction. This is an extension of the principles of observability discussed in [16].

Consider, first, the case of one agent X making a promise conditioned by $\tau(t)$ to another agent Y. The outcome of this promise is trivially mutually observable,

because both agents can start a clock and make the promise inoperative after time t. No further communication is necessary in order to observe the effect.

In this paper, we assume that there is "reliable communication" between agents. Thus, if an agent sends a promise to another, that promise is guaranteed to be received. We are aware that this assumption is *not* made in most of current promise theory. But it is a valid assumption in many of the networks that promise theory attempts to model.

4.1 Observing α

Now consider the more complex case of $\alpha(p)$. This promise is not mutually observable unless it is made between a sender and receiver that match those in $p = \langle s, r, b\rangle$. $(\langle s, r, b_2\rangle|\alpha(\langle s, r, b_1\rangle))$ and $(\langle r, s, b_2\rangle|\alpha(\langle r, s, b_1\rangle))$ make sense, but mentioning promises in α involving any external agent not equal to s or r does not make sense.

In general, it makes little sense for an agent to promise something conditionally unless it already holds a guarantee sufficient for it to believe that it can observe the conditions itself and thus know whether the promise is binding or not. Likewise, it makes little sense for the receiving agent to agree to react to a change in state unless it can observe the change whenever it occurs.

Definition 5. *A promise $\langle s, r, b\rangle$ is* observable *by an external agent x not equal to r or s if x holds sufficient promises to be able to determine with certainty whether $\langle s, r, b\rangle$ is operative in r (and s).*

Obviously, $\alpha(p)$ is observable exactly when p is mutually observable between sender and receiver.

4.2 Knowledge and κ

But how can an agent x learn of a promise between two other agents s and r? By definition, this is a communication between s and r with no intermediary. There is no standard mechanism other than to embed the promise in a body of another promise between some agent and x.

Definition 6. *Given a promise $p = \langle s, r, b\rangle$, the promise body $\kappa(p)$ means that "I will provide knowledge of whether the promise p is operative."*

It is reasonable for either s or r to communicate this information to a third party x; e.g., the promises $\langle s, x, \kappa(\langle s, r, b\rangle)\rangle$ and $\langle r, x, \kappa(\langle s, r, b\rangle)\rangle$ are both reasonable[2]. κ represents an *offer* to communicate this information, while $U(\kappa(\langle s, r, b\rangle))$ represents a commitment to *interpret* this information. Further, once promises $\kappa(p)$ and $U(\kappa(p))$ have been exchanged by a server agent x and a client agent y, it makes sense for conditional expressions sent from x to y to contain $\alpha(p)$, even though p may not involve x and y.

[2] Note that this information is similar to a "reification" in a resource description framework, and is of the form "s claims that $\langle s, r, b\rangle$ holds."

The addition of κ makes it possible to scope promises based upon knowledge of those made between other agents, in particular, $\alpha(p)$ makes sense as a condition even if p is not a promise to the recipient. It is important to remember, however, that without a commitment to report ongoing state of an agent s to an agent x, x cannot be sure that it is completely aware of the relevant state of s. Likewise, an agent x receiving such information, unless it agrees to *use* it, cannot be expected to react to it.

Note that unlike $\alpha(p)$, which is abstract, $\kappa(p)$ and $U(\kappa(p))$ are promises that make sense as consequents, and do not concern the state of p, but rather an intent to *communicate* that state from sender to receiver. It thus makes sense to consider "foreign" conditions (that do not share sender and receiver with the consequent promise) to be inoperative unless a κ-binding is in effect.

These new promise bodies – together with conditional promises – give us a variety of ways to express complex temporal relationships and state changes.

4.3 Failover

Normally, in a non-failover situation, we have a situation of offer and use promises: $\langle s, c, b\rangle$ and $\langle c, s, U(b)\rangle$. The server agent promises until its death, and the client agent promises until its death, respectively.

Failover is represented as a conditional promise network in which the failure of one service triggers a response from another server. We need several components to make this work.

1. $\langle s_1, *, b\rangle$: The original server s_1 promises to provide the service to everyone.
2. $\langle s_2, *, b\rangle$: the backup server s_2 promises to provide the service to everyone.
3. $\langle c, s_1, U(b)\rangle$: the client promises to use the original server until it or the client fails.
4. $\langle c, s_2, U(b)\rangle | \alpha(\langle s_1, c, b\rangle)$: in a failover situation, the client promises to use the failover server if the original server fails, until the original server comes alive again.

When the first server fails, the client is *not* breaking its promise to use the failed server; *but it must receive a new promise from the original server in order to continue*. Note that the backup server is only utilized until the client receives a valid promise from the original server.

But the above picture is not complete, because the last promise above *cannot be directly observed.* Without further information, that promise is nonsense. One way to fix this is to create a knowledge binding from server s_1 to server s_2: $\langle s_1, s_2, \kappa(\langle s_1, c, b\rangle)$, as well as a usage-of-knowledge binding in return: $\langle s_2, s_1, U(\kappa(\langle s_1, c, b\rangle))$.

4.4 Complex time-varying behavior

Suppose we wish to utilize the temporal calculus to describe a state machine for the state of a particular agent, where states will change over time due to external events or timeouts. The temporal operators are powerful enough to describe any such state machine:

Theorem 2. *Suppose agent A wishes to program a time progression on agent B in which promises $p_1, p_2, \ldots, p_k$ become exclusively operative in sequence. This can be accomplished with conditional promises, α, and negation.*

Proof. Let $\{p_i\}$ be a set of promises to transition between, and let $\{s_i \mid i = 1, \ldots, k+1\}$ be a set of "abstract" promises whose assertion by any means "gates" the promises $\{p_i\}$ by accomplishing state changes. Let the promise set C contain:

$$\begin{aligned} p_1 &\mid \neg\alpha(s_1), \alpha(s_2) \\ p_2 &\mid \neg\alpha(s_2), \alpha(s_3) \\ p_3 &\mid \neg\alpha(s_3), \alpha(s_4) \\ &\ldots \\ p_i &\mid \neg\alpha(s_i), \alpha(s_{i+1}) \\ &\ldots \\ p_k &\mid \neg\alpha(s_k), \alpha(s_{k+1}) \end{aligned}$$

Now we can accomplish state changes via the gate promises $s_i|\tau(0)$. If these are asserted in order, then promise $s_i|\tau(0)$ makes promise p_i operative and promise p_{i-1} non-operative. In this way, the target agent transitions in sequence between states p_1 to p_k. □

The same construction can be utilized to cycle through any number of sets of promises, over time or in reaction to events.

Note that these transitions can only happen *once* because of the self-canceling behavior of α. Once α has done its work, it becomes permanently inoperative and its rule effectively disappears. Thus:

Corollary 1. *An agent can achieve a cyclic behavior in a set of promises sent to another agent only by re-promising clauses that have become permanently non-operative.*

Note that any such control structure, once promised, can be eradicated as well:

Corollary 2. *The state machine in the previous proof can be erased from the target Y by the agent X, as needed.*

Proof. The state machine consists of one-time transitions based upon asserting certain events. When these transitions are used, they become permanently inoperative. To erase the state machine, one must take it through its transitions, after which each one becomes inoperative.

This is complicated unless one remembers that one can assert *any* state for a short time, or even for no time at all. To erase a rule, one makes its un-negated antecedents operative for a short time, then makes them inoperative again. □

Thus conditional promises provide a way both to accomplish state changes and to erase the mechanism that enables them, as needed.

4.5 Calculating operative promises

Temporal promises add no major complexity to the calculation of which promises are operative or not.

Theorem 3. *At a particular time t, on a particular agent X, the calculation of whether a particular promise is operative can be done in two phases: first eliminating scoping rules, then interpreting pure conditionals.*

Proof. First suppose there is an abstract promise called "true" that is operative at all times. Create a pure conditional network C' from the temporo-conditional network C as follows:

1. Remove the whole promise for any τ or α conditions that have expired. τ expires when its time is up, while α expires when the event for which it is watching has been observed.
2. Replace all operative τ and α with "true".

Claim: the operative promises in the resulting network are equivalent with the operative promises in the original network. First, by definition of conditional promises and the temporal operators, all temporal operators expire at the end of their events. This means we can safely discard them, as they cannot become operative *again*. Second, when a condition is true in a conditional, the operative effect is that it is simply operative, and does not depend upon its temporal qualities. If for example we have $p|q_1, q_2, \ldots, q_k$, and we know q_i is operative, then $p|q_1, \ldots, q_{i-1}, \text{true}, q_{i+1}, \ldots, q_k$ has the same effect as the previous one on p. The negation of "true", if it appears, behaves properly and one can delete the conditional promise containing it from the set of promises to be considered. □

As a corollary, any promise whose temporal conditions aren't operative can be discarded as obsolete. This helps us "tidy up" promise space.

5 Conclusions

We have demonstrated that extending conditional promises with temporal promises α and τ (as well as negation) allows one to synthesize common network behaviors such as leasing and event-driven reaction. This mechanism allows an agent to almost completely control other agents' views of its commitments, over time. At any particular time, however, the commitments binding upon agents are analyzable via normal promise theory. Since temporally scoped promises become permanently inoperative after their conditions become inoperative, the promises an agent holds can be "tidied" by removing permanently inoperative conditionals.

Many questions remain to be answered. What is the most efficient way to compute the operative promises? Are there operators more efficient than the proposed operators? What depictions of conditional and temporal state are useful? There are more questions than answers.

One thing is certain, however: the ability for two agents to agree to forget a promise after a specific time is both useful and necessary in modeling contemporary networks.

References

1. Burgess, M.: An approach to understanding policy based on autonomy and voluntary cooperation. In: DSOM. (2005) 97–108
2. Burgess, M., Begnum, K.: Voluntary cooperation in pervasive computing services. In: LISA. (2005) 143–154
3. Burgess, M., Fagernes, S.: Pervasive computer management: A model of network policy with local autonomy. IEEE Transactions on Networking (2006) (submitted)
4. Burgess, M., Fagernes, S.: Promise theory - a model of autonomous objects for pervasive computing and swarms. In: ICNS. (2006) 118
5. Burgess, M., Couch, A.: Modeling next generation configuration management tools. In: LISA. (2006) 131–147
6. Aredo, D., Burgess, M., Hagen, S.: A promise theory view on the policies of object orientation and the service oriented architecture. (preprint) (November 2006) (submitted)
7. Bergstra, J., Burgess, M.: Local and global trust based on the concept of promises. Technical Report PRG0606, University of Amsterdam (September 2006)
8. Burgess, M., Couch, A.: Autonomic computing approximated by fixed-point promises. In: Proceedings of First IEEE International Workshop on Modeling Autonomic Communication Environments (MACE). (2006) 197–222
9. Burgess, M.: A site configuration engine. Computing systems (MIT Press: Cambridge MA) **8** (1995) 309
10. Burgess, M., Ralston, R.: Distributed resource administration using cfengine. Software practice and experience **27** (1997) 1083
11. Burgess, M.: Automated system administration with feedback regulation. Software practice and experience **28** (1998) 1519
12. Burgess, M.: Cfengine as a component of computer immune-systems. Proceedings of the Norwegian conference on Informatics (1998)
13. Burgess, M.: Configurable immunity for evolving human-computer systems. Science of Computer Programming **51** (2004) 197
14. Couch, A., Gilfix, M.: It's elementary, dear watson: Applying logic programming to convergent system management processes. Proceedings of the Thirteenth Systems Administration Conference (LISA XIII) (USENIX Association: Berkeley, CA) (1999) 123
15. Bergstra, J., Bethke, I., Burgess, M.: A process algebra based framework for promise theory. Technical Report PRG0701, University of Amsterdam (March 2007)
16. Couch, A., Sun, Y.: On observed reproducibility in network configuration management. Science of Computer Programming **53** (2004) 215–253