Universal Concurrent Constraint Programing: Symbolic Semantics and Applications to Security

Carlos Olarte
INRIA and LIX École Polytechnique
Pontificia Universidad Javeriana
carlos.olarte@lix.polytechnique.fr

Frank D. Valencia
CNRS and LIX École Polytechnique
frank.valencia@lix.polytechnique.fr

ABSTRACT

We introduce the *Universal Timed Concurrent Constraint Programming* (utcc) process calculus; a generalisation of Timed Concurrent Constraint Programming. The utcc calculus allows for the specification of mobile behaviours in the sense of Milner's π-calculus: Generation and communication of private channels or links. We first endow utcc with an *operational* semantics and then with a *symbolic* semantics to deal with problematic operational aspects involving infinitely many substitutions and divergent internal computations. The novelty of the symbolic semantics is to use *temporal constraints* to represent finitely infinitely-many substitutions. We also show that utcc has a strong connection with Pnueli's Temporal Logic. This connection can be used to prove reachability properties of utcc processes. As a compelling example, we use utcc to exhibit the secrecy flaw of the Needham-Schroeder security protocol.

Keywords

Concurrent Constraint Programming, Symbolic Semantics, Security, Mobility.

1. INTRODUCTION

Process calculi treat concurrent processes much like the λ-calculus treat computable functions. They provide a language in which the structure of terms represents the structure of processes together with an operational semantics to represent computational steps. Concurrent Constraint Programming (CCP) [13] is a process calculus which combines the traditional operational view of process calculi with a declarative one based upon logic. This combination allows CCP to benefit from the large body of techniques of both process calculi and logic. In fact, CCP has successfully been used in the modelling of several concurrent scenarios: E.g., Biological, Timed, Reactive and Stochastic systems [10].

Agents in CCP interact with each other by telling and asking information represented as constraints in a global store. The process $\mathbf{tell}(c)$ adds the constraint c to the store and the ask process **when** c **do** P executes P if c is entailed by the store. The process $(\mathbf{local}\, x; c)\, P$ declares a private variable x for P constrained by c and $P \parallel Q$ stands for the parallel execution of P and Q. The computations of processes are *deterministic* in that the final store is always the same regardless of the execution order of parallel components.

Generation and communication of private links or channels, i.e. *mobility*, is fundamental for the specification of systems where agents change their communication structure (*mobile systems*). This is also fundamental to specify protocols where *nonces* (i.e, randomly-generated unguessable items) are transmitted. In fact, generation and communication of private links are the central operations of one of the main representative formalisms for mobility in concurrency theory, namely the π-calculus [9].

We aim at defining a CCP-based language to specify mobility. This language ought to comply with two criteria that distinguish basic CCP from other formalisms: (1) *Logic correspondence* which provides CCP with a unique declarative view of processes and (2) *determinism* which is the source of CCP's elegant and simple characterisations (e.g., the closure operator semantics [13]). Another general criterion for our extension is (3) to be *applicable* to meaningful concurrent scenarios, in particular those not yet explored using CCP calculi.

Basic CCP is able to specify mobile behaviour using logical variables to represent channels and unification to bind messages to channels [13]. In this approach if two messages are sent through the same channel, they must be equal. In other case an inconsistency arises. In [6] this problem is solved using *Atomic CCP* where $\mathbf{tell}(c)$ adds c to the current store d if $c \wedge d$ is not inconsistent. Here a protocol is required since messages must compete for a position in a list representing the messages previously sent. Atomic tells introduce non-determinism to the calculus since the execution of $\mathbf{tell}(c)$ depends on the current store thus not adhering to Criterion (2). Furthermore, to our knowledge no correspondence between this language and logic has been given (Criterion (1)).

Mobility can be also modelled by adding *linear* parametric ask processes to CCP as done in Linear CCP [4]. A parametric ask $A(x)$ can be viewed as a process **when** c **do** P with a variable x declared as a formal parameter. Intuitively, $A(x)$ may evolve into $P[y/x]$, i.e. P with x replaced by y, if $c[y/x]$ is entailed by the store. Mobility is exhibited when y is a private variable (*link*) from some other process. This extension, however, does not adhere to Criterion (2), i.e. it is *non-deterministic:* If both $c[y/x]$ and $c[z/x]$ are entailed

by the store, $A(x)$ may evolve to either $P[y/x]$ or $P[z/x]$.

The above kind of non-determinism can be avoided by extending CCP only with *persistent* parametric asks following the semantics of the persistent π-calculus in [11]. The idea is that if both $c[y/x]$ and $c[z/x]$ are entailed by the store, a persistent $A(x)$ evolves into $A(x) \parallel P[y/x] \parallel P[z/x]$. Forcing every ask to be persistent, however, makes the extension not suitable for modelling typical scenarios where a process stops after performing its query (e.g., web-services requests).

Our strategy is therefore to extend CCP with *temporary* parametric ask operations. Intuitively, these operations behave as persistent parametric asks during a *time-interval* but may disappear afterwards. We do this by generalising the timed CCP model in [12]. We call this extension *Universal Timed CCP* (`utcc`). As explained below, we shall show that `utcc` complies with previously-mentioned criteria (1-3).

Contributions. We first give `utcc` an *operational semantics*. However, parametric ask operations pose technical problems: They may generate infinitely many substitutions thus causing divergent (i.e., infinite) internal computations. We resolve this problem by endowing `utcc` with a novel *symbolic semantics* that uses temporal constraints to represent finitely a possible infinite number substitutions. We then show that `utcc` allows for mobility and it is deterministic (Criterion 2). We shall show a strong correspondence with Linear-Time Temporal Logic (LTL) by providing a *compositional* encoding of `utcc` processes into LTL formulae (Criterion 1). This correspondence with the standard logic for the verification of concurrent systems allows for reachability analysis central to security protocols: If there is a way to reach a state where an intruder knows a secret, then there is a secrecy breach. To illustrate the applicability of `utcc` (Criterion 3), we use it to model and predict the secrecy flaw in the well-known Needham-Schroeder Protocol [7].

To our knowledge this is the first symbolic semantics for a CCP calculus as well as the first one in concurrency theory using temporal constraints as finite representations of substitutions. Furthermore, we are not aware of any other work applying a CCP calculus to model security protocols.

2. PRELIMINARIES

In this section we describe the temporal concurrent constraint (`tcc`) model [12] following the presentation in [10].

CCP calculi are parametric in a *constraint system* (cs). A cs can be represented as a pair (Σ, Δ) where Σ is a signature of function and predicate symbols, and Δ is a first-order theory over Σ. Given a cs (Σ, Δ), let $\mathcal{L}$ be its underlying first-order language with variables $x, y, \ldots$, and logic symbols $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall$, `true` and `false`. *Constraints* $c, d, \ldots$ are formulae over $\mathcal{L}$. We say that c *entails* d in Δ, written $c \vdash_\Delta d$, iff $(c \Rightarrow d) \in \Delta$ (i.e., iff $c \Rightarrow d$ is true in all models of Δ). We shall omit "Δ" in $\vdash_\Delta$ when no confusion arises. For operational reasons $\vdash$ is often required to be decidable.

We use $\vec{t}$ for a sequence of terms $t_1, \ldots, t_n$ with length $|\vec{t}| = n$. If $|\vec{t}| = 0$ then $\vec{t}$ is written as ϵ. We use $c[\vec{t}/\vec{x}]$, where $|\vec{t}| = |\vec{x}|$ and x_i's are pairwise distinct, to denote c with the free occurrences of x_i replaced with t_i. The substitution $[\vec{t}/\vec{x}]$ will be similarly applied to other syntactic entities.

The `tcc` calculus extends CCP for timed systems [12]. Time is conceptually divided into *time intervals* (or *time units*). In a particular time interval, a CCP process P gets an input c from the environment, it executes with this input as the initial *store*, and when it reaches its resting point, it *outputs* the resulting store d to the environment. The resting point determines a residual process Q which is then executed in the next time interval. The resulting store d is not automatically transferred to the next time interval.

DEFINITION 1. *Processes $P, Q, \ldots$ in `tcc` are built from constraints in the underlying cs by the following syntax:*

$$P, Q := \quad \mathbf{skip} \mid \mathbf{tell}(c) \mid \mathbf{when}\ c\ \mathbf{do}\ P \mid P \parallel Q \mid (\mathbf{local}\,\vec{x}; c)\,P \mid \mathbf{next}\,P \mid \mathbf{unless}\ c\ \mathbf{next}\,P \mid !\,P$$

with the variables in $\vec{x}$ being pairwise distinct.

The process **skip** does nothing and $\mathbf{tell}(c)$ adds c to the store in the current time interval. If in the current time interval c can eventually be derived from the store, **when** c **do** P evolves into P within the same time interval. Otherwise **when** c **do** P evolves into **skip**. $P \parallel Q$ denotes P and Q running in parallel during the current time interval and $(\mathbf{local}\,\vec{x}; c)\,P$ *binds* $\vec{x}$ in P by declaring it private to P under a constraint c. The *bound variables* $bv(Q)$ (*free variables* $fv(Q)$) are those with a bound (a not bound) occurrence in Q. We write $(\mathbf{local}\,\vec{x})\,P$ as short for $(\mathbf{local}\,\vec{x}; \mathtt{true})\,P$.

The *unit-delay* **next** P executes P in the next time interval. The *time-out* **unless** c **next** P is also an unit-delay, but P is executed in the next time unit iff c is not entailed by the final store at the current time interval. We use $\mathbf{next}^n P$ as short for $\mathbf{next} \ldots \mathbf{next}\,P$, with **next** repeated n times. Finally, the *replication* $!\,P$ means $P \parallel \mathbf{next}\,P \parallel \mathbf{next}^2 P \parallel \ldots$, i.e., unboundly many copies of P but one at a time.

3. UNIVERSAL TIMED CC

In [12] it is shown that `tcc` processes are finite-state. This means it cannot be used to describe infinite-state behaviours like those arising from mobility which is one of the main concerns in this paper. Mobility is understood in the sense of the π-calculus, i.e, communication of private variables (or links). In this section we introduce an extension to `tcc` which by following a logic approach it provides for mobile behaviour. We call this extension Universal `tcc` (`utcc`).

3.1 Abstractions

To model mobile behaviour, `utcc` replaces the `tcc` ask operation **when** c **do** P with a more general parametric ask construction, namely $(\mathbf{abs}\ \vec{x}; c)\,P$. This process can be viewed as a λ-*abstraction* of the process P on the variables $\vec{x}$ under the constraint (or with the *guard*) c. From a programming language perspective, $\vec{x}$ in $(\mathbf{local}\,\vec{x}; c)\,P$ can be viewed as the local variables of P while $\vec{x}$ in $(\mathbf{abs}\ \vec{x}; c)\,P$ can be viewed as the formal parameters of P.

From a logic perspective we shall show that the new binder has a pleasant duality with the local operator: The processes $(\mathbf{local}\,\vec{x}; c)\,P$ and $(\mathbf{abs}\ \vec{x}; c)\,P$ correspond, resp., to the *existential* and *universal* formulae $\exists\vec{x}(c \wedge F_P)$ and $\forall\vec{x}(c \Rightarrow F_P)$ where F_P corresponds to P.

DEFINITION 2 (`utcc` PROCESSES). *The `utcc` processes result from replacing in the syntax in Definition 1 the expression* **when** c **do** P *with* $(\mathbf{abs}\ \vec{x}; c)\,P$ *with the variables in $\vec{x}$ being pairwise distinct.*

Intuitively, $Q = (\mathbf{abs}\ \vec{x}; c)\,P$ performs $P[\vec{t}/\vec{x}]$ in the current time interval for *all the terms* $\vec{t}$ s.t $c[\vec{t}/\vec{x}]$ is entailed by the store. Q binds the variables $\vec{x}$ in P and c. Sets $fv(.)$ and $bv(.)$, are extended accordingly. Furthermore Q evolves into

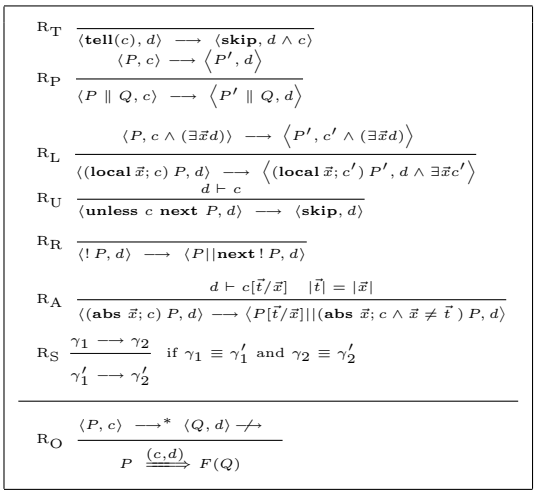

Table 1: Internal and observable reductions. $\equiv$ and F are given in Definitions 3 and 4. In $\mathrm{R_A}$, $\vec{x} \neq \vec{t}$ denotes $\bigvee_{1 \leq i \leq |\vec{x}|} x_i \neq t_i$. If $|\vec{x}| = 0$, $\vec{x} \neq \vec{t}$ is defined as `false`.

skip at the end of the time unit. We shall use **when** c **do** P for the empty abstraction (**abs** $\epsilon; c)\, P$.

Mobile links. We conclude this section with a little example illustrating how mobility is obtained from the interplay between abstractions and local processes.

EXAMPLE 1 (MOBILITY). *Let Σ be a signature with the unary predicates $\mathtt{out}_1, \mathtt{out}_2, \ldots$ and a constant 0. Let Δ be the set of axioms over Σ valid in first-order logic* [1]. *Let* $P = (\mathbf{abs}\ y; \mathtt{out}_1(y))\, \mathbf{tell}(\mathtt{out}_2(y))$ *and* $Q =$

$$(\mathbf{local}\, z)\, (\mathbf{tell}(\mathtt{out}_1(z)) \parallel \mathbf{when}\ \mathtt{out}_2(z)\ \mathbf{do\ next\ tell}(\mathtt{out}_2(0)))$$

Intuitively, if a link y is sent on channel $\mathtt{out}_1$, P forwards it on $\mathtt{out}_2$. Now, Q sends its private link z on $\mathtt{out}_1$ and if it gets it back on $\mathtt{out}_2$ it outputs 0 on $\mathtt{out}_2$. In the next section we show that $P \parallel Q$ produces $\mathtt{out}_2(0)$ in the next time unit.

3.2 Operational Semantics (SOS)

The structural operational semantics (SOS) of `utcc` considers *transitions* between process-store *configurations* $\langle P, c\rangle$ with stores represented as constraints and processes quotiented by $\equiv$. We use $\gamma, \gamma', \ldots$ to range over configurations.

DEFINITION 3. *Let $\equiv$ be the smallest congruence satisfying: (0) $P \equiv Q$ if they differ only by a renaming of bound variables (1) $P \parallel \mathbf{skip} \equiv P$, (2) $P \parallel Q \equiv Q \parallel P$, (3) $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, (4) $P \parallel (\mathbf{local}\, \vec{x}; c)\, Q \equiv (\mathbf{local}\, \vec{x}; c)\, (P \parallel Q)$ if $\vec{x} \notin fv(P)$, (5) $(\mathbf{local}\, \vec{x}; c)\, \mathbf{skip} \equiv \mathbf{skip}$. Extend $\equiv$ by decreeing that $\langle P, c\rangle \equiv \langle Q, c\rangle$ iff $P \equiv Q$.*

The SOS transitions are given by the relations $\longrightarrow$ and $\Longrightarrow$ in Table 1. The *internal* transition $\langle P, d\rangle \longrightarrow \langle P', d'\rangle$ should be read as "P with store d reduces, in one internal step, to P' with store d' ". The *observable transition* $P \xrightarrow{(c,d)} R$ should be read as "P on input c, reduces in one *time unit* to R and outputs d". The observable transitions are obtained from finite sequences of internal transitions.

We only describe some of the rules in Table 1 due to space restrictions. The other rules are standard, easily seen to realise the operational intuitions given above (see [10] for further details). As clarified below, the seemingly missing rules for "next" and "unless" processes are given by $\mathrm{R_O}$.

We dwell a little upon the description of Rule $\mathrm{R_L}$ as it may seem somewhat complex. Consider $Q = (\mathbf{local}\, x; c)\, P$ in Rule $\mathrm{R_L}$. The global store is d and the local store is c. We distinguish between the *external* (corresponding to Q) and the *internal* point of view (corresponding to P). From the internal point of view, the information about x, possibly appearing in the "global" store d, cannot be observed. Thus, before reducing P we first hide the information about x that Q may have in d by existentially quantifying x in d. Similarly, from the external point of view, the observable information about x that the reduction of internal agent P may produce (i.e., c') cannot be observed. Thus we hide it by existentially quantifying x in c' before adding it to the global store. Additionally, we make c' the new private store of the evolution of the internal process.

Rule $\mathrm{R_A}$ describes the behaviour of (**abs** $\vec{x}; c)\, P$. If the store entails $c[\vec{t}/\vec{x}]$ then $P[\vec{t}/\vec{x}]$ is executed. Additionally, the abstraction persists in the current time interval to allow other potential replacements of $\vec{x}$ in P but c is augmented with $x_i \neq t_i$ to avoid executing $P[\vec{t}/\vec{x}]$ again.

Rule $\mathrm{R_O}$ says that an observable transition from P labelled with (c, d) is obtained from a terminating sequence of internal transitions from $\langle P, c\rangle$ to a $\langle Q, d\rangle$. The process R to be executed in the next time interval is equivalent to $F(Q)$ (the "future" of Q). $F(Q)$ is obtained by removing from Q abstractions and any local information which has been stored in Q, and by "unfolding" the sub-terms within "next" and "unless" expressions.

DEFINITION 4. *Let F be a partial function defined as:*

$F(\mathbf{skip}) = \mathbf{skip}$	$F((\mathbf{abs}\ \vec{x}; c)\, Q) = \mathbf{skip}$
$F(P_1 \parallel P_2) = F(P_1) \parallel F(P_2)$	$F((\mathbf{local}\, \vec{x}; c)\, Q) = (\mathbf{local}\, \vec{x})\, F(Q)$
$F(\mathbf{next}\, Q) = Q$	$F(\mathbf{unless}\ c\ \mathbf{next}\, Q) = Q$

We can now illustrate the observable transition showing the expected behaviour in our mobility example.

EXAMPLE 2 (MOBILE OBSERVABLE TRANSITION). *Let P and Q as in Example 1. One can verify that for every c :, $P \parallel Q \xrightarrow{(c,d)} R$ with $R \equiv \mathbf{tell}(\mathtt{out}_2(0))$.*

3.2.1 Infinite behaviour within time-units

Due to abstractions the SOS may induce an infinite sequence of internal transitions within a time interval thus never producing an observable transition.

Abstraction Loops. One source of infinite internal behaviour involves looping (recursion) in abstractions. E.g. let $R = (\mathbf{abs}\ x; \mathtt{out}_1(x))\, (\mathbf{local}\, z)\, \mathbf{tell}(\mathtt{out}_1(z)))$. Each time R gets a link on $\mathtt{out}_1$, it generates a new link z on $\mathtt{out}_1$ thus causing infinite internal behaviour. (A similar problem involves several abstractions producing mutual recursive behaviour.) This kind of looping problems can be avoided by requiring for each (**abs** $x; c)\, P$ that P must be a "next" expression. This restriction, however, may also disallow behaviour which will not cause infinite internal computations.

Infinitely Many Substitutions. Another source of infinite internal behaviour involves the cs under consideration. Let $R = (\mathbf{abs}\ x; c)\, P$. If the current store d entails $c[t/x]$ for infinitely many t's, then R will have to produce $P[t/x]$ for each such t. This kind of infinite internal behaviour can be avoided by allowing only guards c so that for any

[1] Notice that the induced $\vdash$ is decidable because Δ corresponds to monadic first-order logic.

d the set $\{t \mid d \vdash c[t/x]\}$ modulo logic equivalence is finite. This seems inconvenient for the modelling of cryptographic knowledge as typically is done in process calculi: The presence of some messages entails the presence of arbitrary compositions among them. We discuss this at length in our security application in Section 6.

4. SYMBOLIC SEMANTICS

In this section we define a symbolic semantics for `utcc` whose basic idea is to use temporal constraints to represent in a finite way a possibly infinite number of substitutions as well as the information that an infinite loop may provide. It turns out that without appealing to the syntactic restrictions mentioned above, this semantics guarantees that every sequence of internal transitions is finite.

4.1 Symbolic Intuitions

Before defining the symbolic semantics let us give some intuitions of its basic principles.

A. Substitutions as Constraints. Take $R = (\mathbf{abs}\ x; c)\ P$. The operational semantics performs $P[t/x]$ for every t s.t $c[t/x]$ is entailed by the store d. Instead, the symbolic semantics dictates that R should produce $e = d \wedge \forall x(c \Rightarrow d')$ where, similarly, d' should be produced according to the symbolic semantics by P. Let t be an arbitrary term s.t $d \vdash c[t/x]$. The idea is that if e' is operationally produced by $P[t/x]$ then e' should be entailed by $d'[t/x]$. Since $d \vdash c[t/x]$ then $e \vdash d'[t/x] \vdash e'$. Therefore e entails the constraint that any arbitrary $P[t/x]$ produces.

B. Timed Dependencies in Substitutions. The symbolic semantics represents as temporal constraints dependencies between substitutions from one time interval to another. E.g., suppose that for R above, $P = \mathbf{next\ tell}(e)$. Operationally, once we move to the next time unit, the constraints produced are of the form $e[t/x]$ for those t's s.t the final store d in the *previous* time unit entails $c[t/x]$. The symbolic semantics captures this as $e' = (\ominus d) \wedge \forall x((\ominus c) \Rightarrow e)$ where $\ominus$ is the "previous" modality in temporal logic. Intuitively, $\ominus c'$ means that c' holds in the previous time interval.

4.2 Temporal Formulae and Constraints

We shall use (a fragment of) the standard Linear-Time Temporal Logic (LTL) in [8] for our symbolic semantics.

DEFINITION 5. *Given a cs with a first-order language $\mathcal{L}$, the* LTL formulae *we use are given by the syntax:*

$$F, G, \ldots := c \mid F \wedge G \mid \neg F \mid \exists x F \mid \ominus F \mid \circ F \mid \Box F.$$

where c is a constraint in $\mathcal{L}$, from now called state formula.

The modalities $\ominus F, \circ F$ and $\Box F$ state, resp., that F holds *previously, next* and *always*. We use $\forall x F$ for $\neg\exists x \neg F$, and the *eventual* modality $\Diamond F$ as an abbreviation of $\neg\Box\neg F$.

We presuppose the reader is familiar with the basic concepts of Model Theory. The non-logical symbols of $\mathcal{L}$ are given meaning in an underlying $\mathcal{L}$-structure, or $\mathcal{L}$-*model*, $\mathcal{M}(\mathcal{L}) = (\mathcal{I}, \mathcal{D})$. (They are interpreted via $\mathcal{I}$ as relations over a domain $\mathcal{D}$ of the corresponding arity.) A *state* s is a mapping assigning to each variable x in $\mathcal{L}$ a value $s[x]$ in $\mathcal{D}$. This interpretation is extended to $\mathcal{L}$-expressions in the usual way (e.g. $s[f(x)] = \mathcal{I}(f)(s[x])$). We write $s \models_{\mathcal{M}(\mathcal{L})} c$ iff c is true wrt s in $\mathcal{M}(\mathcal{L})$. The state s is an x-variant of s' iff $s'[y] = s[y]$ for each $y \neq x$. We use σ to range over infinite sequences of states. Furthermore, σ is an x-variant of σ' iff $\sigma(i)$ is an x-variant of $\sigma'(i)$ for each $i \geq 0$.

DEFINITION 6. *We say that σ* satisfies *F in an $\mathcal{L}$-structure $\mathcal{M}(\mathcal{L})$, written $\sigma \models_{\mathcal{M}(\mathcal{L})} F$, iff $\langle \sigma, 0\rangle \models_{\mathcal{M}(\mathcal{L})} F$ where:*

$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})}$ `true`		$\langle \sigma, i\rangle \not\models_{\mathcal{M}(\mathcal{L})}$ `false`
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} c$	*iff*	$\sigma(i) \models_{\mathcal{M}(\mathcal{L})} c$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \neg F$	*iff*	$\langle \sigma, i\rangle \not\models_{\mathcal{M}(\mathcal{L})} F$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} F \wedge G$	*iff*	$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} F$ *and* $\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} G$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \ominus F$	*iff*	$i > 0$ *and* $\langle \sigma, i-1\rangle \models_{\mathcal{M}(\mathcal{L})} F$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \circ F$	*iff*	$\langle \sigma, i+1\rangle \models_{\mathcal{M}(\mathcal{L})} F$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \Box F$	*iff*	*for all* $j \geq i$, $\langle \sigma, j\rangle \models_{\mathcal{M}(\mathcal{L})} F$
$\langle \sigma, i\rangle \models_{\mathcal{M}(\mathcal{L})} \exists x F$	*iff*	*for some x-variant* σ' *of* σ, $\langle \sigma', i\rangle \models_{\mathcal{M}(\mathcal{L})} F$

We say that F is LTL valid *in $\mathcal{M}(\mathcal{L})$ iff for all σ, $\sigma \models_{\mathcal{M}(\mathcal{L})} F$.*

LTL Theories. Given a cs (Δ, Σ) with first-order language $\mathcal{L}$, the LTL theory induced by Δ, $T(\Delta)$ is the set of LTL sentences that are LTL valid in all the $\mathcal{L}$-structures (or $\mathcal{L}$-models) of Δ. We write $F \vdash_{T(\Delta)} G$ iff $(F \Rightarrow G) \in T(\Delta)$. We omit "$(\Delta)$" in $\vdash_{T(\Delta)}$ when no confusion arises.

Future-free constraints. For the symbolic semantics we take the liberty of assuming that the constraints in `utcc` processes and configurations are replaced with certain LTL formulae. Namely, *future-free* formulae: Those which do not contain occurrences of $\Box$ and $\circ$. So, for example a process-store configuration of the form $\langle (\mathbf{abs}\ y; \ominus c)\ P, \ominus d\rangle$ may appear in the transitions of the symbolic semantics. We shall abuse the notation by assuming that, in the context of the symbolic semantics, $c, d, c', d', \ldots$ range over future-free formulae. In the context of the SOS, however, $c, d, c', d', \ldots$ range over state formulae only.

Future-free formulae preserve the decidability of their underlying cs.

PROPOSITION 1. *Let (Δ, Σ) be a cs with decidable entailment relation $\vdash$. Then given F, G the problem of whether $F \vdash_T G$ is decidable if F, G are future-free LTL formulae.*

4.3 Symbolic Reductions.

The internal and observable *symbolic* transitions $\longrightarrow_s, \Longrightarrow_s$ are defined as in Table 1 with $\vdash$ replaced by $\vdash_T$ and with $\mathrm{R_A}$ and $\mathrm{R_O}$ replaced, resp., by $\mathrm{R_{As}}$ and $\mathrm{R_{Os}}$ in Table 2 (with c, c' and d' representing future-free formulae rather than just state-formulae).

The rule $\mathrm{R_{As}}$ represents with the temporal constraint $\forall\vec{x}(c \Rightarrow d')$ the substitutions that its operational counterpart $\mathrm{R_A}$ would induce as intuitively explained in Section 4.1(A). Notice that in the reduction of P the variables $\vec{x}$ in d are hidden, via existential quantification, to avoid clashes with those in P.

The function F_s in $\mathrm{R_{Os}}$ is similar to its operational counterpart F. However, F_s records the final global and local stores as well as abstraction guards as past information. As explained in Section 4.1(B) this past information is needed in next time unit.

DEFINITION 7. *Let F_s a partial function from configurations to processes defined by $F_s(P,d)=\mathbf{tell}(\ominus d)\|F'(P)$ where:*

$F'(\mathbf{skip}) = \mathbf{skip}$	$F'((\mathbf{abs}\ \vec{x}; c)\ P) = (\mathbf{abs}\ \vec{x}; \ominus c)\ F'(P)$
$F'(P_1 \| P_2) = F'(P_1) \| F'(P_2)$	$F'((\mathbf{local}\ \vec{x}; c)\ Q) = (\mathbf{local}\ \vec{x}; \ominus c)\ F'(Q)$
$F'(\mathbf{next}\ Q) = Q$	$F'(\mathbf{unless}\ c\ \mathbf{next}\ Q) = Q$

where c and d represent future-free formulae.

Finally, it is easy to verify that no infinite sequence $\gamma_1 \longrightarrow_s \gamma_2 \longrightarrow_s \ldots$ can be generated by the symbolic semantics. Thus we avoid the problems discussed in Section 3.2.1.

$$\mathrm{R_{As}}\ \frac{\langle P, \exists \vec{x} d\rangle \longrightarrow_s \left\langle P', d' \wedge \exists \vec{x} d\right\rangle}{\langle (\mathbf{abs}\ \vec{x}; c)\ P, d\rangle \longrightarrow_s \left\langle (\mathbf{abs}\ \vec{x}; c)\ P', d \wedge \forall \vec{x}(c \Rightarrow d')\right\rangle}$$

$$\mathrm{R_{Os}}\ \frac{\langle P, c\rangle \longrightarrow_s^* \langle Q, d\rangle \not\longrightarrow_s}{P \stackrel{(c,d)}{\Longrightarrow}_s F_s(Q, d)}$$

Table 2: Symbolic Rules for Internal and Observable Transitions. The function F_s is given in Definition 7.

5. SEMANTIC RESULTS

We now look at the semantics results for `utcc`. First of all, as the basic CCP, `utcc` is deterministic.

THEOREM 1 (DETERMINISM). *For every P, c if $P \stackrel{(c,d)}{\Longrightarrow} Q$ and $P \stackrel{(c,d')}{\Longrightarrow} Q'$ then $Q \equiv Q'$ and $\vdash d \Leftrightarrow d'$. Similarly for the symbolic semantics, for every P, c if $P \stackrel{(c,d)}{\Longrightarrow}_s Q$ and $P \stackrel{(c,d')}{\Longrightarrow}_s Q'$ then $Q \equiv Q'$ and $\vdash_T d \Leftrightarrow d'$.*

Operational Correspondence. The correspondence between the two semantics is given below. It states that the (untimed) constraints entailed by the outputs of any arbitrary long sequence of observable reductions coincide with those of the corresponding one in the symbolic semantics.

We say that P is *abstracted-unless free* iff it has no occurrences of processes of the form **unless** c **next** Q under the scope of an abstraction.

THEOREM 2 (SEMANTIC CORRESPONDENCE). *Let P be an abstracted-unless free process. Suppose that $P \stackrel{(c_1,d_1)}{\Longrightarrow} P_1 \stackrel{(c_2,d_2)}{\Longrightarrow} P_2 \ldots$ and $P \stackrel{(c_1,d_1{}')}{\Longrightarrow}_s P_1{}' \stackrel{(c_2,d_2{}')}{\Longrightarrow}_s P_2{}' \ldots$. Then for every state formula d and $i > 0$, $d_i \vdash d$ iff $d_i' \vdash_T d$.*

We confined ourselves to abstracted-unless free processes due to the problem of representing the negation of entailment as logic formulae. Take $P = (\mathbf{abs}\ x; \mathtt{true})\, Q$ where $Q = \mathbf{unless}\ e\ \mathbf{next}\ \mathbf{tell}(e')$. Let d be the final store in the first time unit when running P. Operationally, $\mathbf{tell}(e')[t/x]$ is executed in the second time unit for those t's s.t $d \not\vdash e[t/x]$. Following Section 4.1 one may try to capture this in the symbolic semantics with $\ominus d \wedge \forall x(\ominus(\neg e) \Rightarrow e')$ as if we had $Q = \mathbf{when}\ \neg e\ \mathbf{do}\ \mathbf{next}\ \mathbf{tell}(e')$. This wrongly assumes, however, that in general $d \not\vdash e[t/x]$ is equivalent to $d \vdash \neg e[t/x]$.

Nevertheless, abstract-unless free processes represent a meaningful and practical fragment of `utcc`. They are sufficient to express all the examples and intended applications of this paper. In fact, one can show they are sufficient to model Turing Machines which bears witness to their universal expressiveness. Due to space restrictions, however, this is outside of the scope of this paper.

LTL Correspondence. The *compositional* encoding in Definition 8 gives a pleasant correspondence between `utcc` processes and LTL formulae. It shows the duality between local and abstraction operators, represented resp. as existential and universal quantified formulae.

DEFINITION 8. *Let $[\![\cdot]\!]$ a map from `utcc` processes to LTL formulae given by:*

$$\begin{array}{lcllcl}
[\![\mathbf{skip}]\!] & = & \mathtt{true} & [\![\mathbf{tell}(c)]\!] & = & c \\
[\![(\mathbf{abs}\ \vec{y}; c)\ P]\!] & = & \forall \vec{y}(c \Rightarrow [\![P]\!]) & [\![P \parallel Q]\!] & = & [\![P]\!] \wedge [\![Q]\!] \\
[\![(\mathbf{local}\ \vec{x}; c)\ P]\!] & = & \exists \vec{x}(c \wedge [\![P]\!]) & [\![\mathbf{next}\ P]\!] & = & \circ [\![P]\!] \\
[\![\mathbf{unless}\ c\ \mathbf{next}\ P]\!] & = & c \vee \circ [\![P]\!] & [\![!\, P]\!] & = & \Box [\![P]\!]
\end{array}$$

$$\begin{array}{lcl}
A \to B & : & \{m, A\}_B \\
B \to A & : & \{m, n\}_A \\
A \to B & : & \{n\}_B
\end{array}$$
(a)

$$\begin{array}{lcl}
A \to C & : & \{m, A\}_C \\
C \to B & : & \{m, A\}_B \\
B \to A & : & \{m, n\}_A \\
A \to C & : & \{n\}_C
\end{array}$$
(b)

Figure 1: Needham-Schroeder Protocol

$$\mathrm{PRJ}\ \frac{F \vdash_{\mathtt{s}} \mathtt{out}(\{m_1, m_2\})\ \ i \in \{1,2\}}{F \vdash_{\mathtt{s}} \mathtt{out}(m_i)} \qquad \mathrm{PAIR}\ \frac{F \vdash_{\mathtt{s}} \mathtt{out}(m_1) \quad F \vdash_{\mathtt{s}} \mathtt{out}(m_2)}{F \vdash_{\mathtt{s}} \mathtt{out}(\{m_1, m_2\})}$$

$$\mathrm{ENC}\ \frac{F \vdash_{\mathtt{s}} \mathtt{out}(m) \quad F \vdash_{\mathtt{s}} \mathtt{out}(k)}{F \vdash_{\mathtt{s}} \mathtt{out}(\{m\}_k)} \qquad \mathrm{DEC}\ \frac{F \vdash_{\mathtt{s}} \mathtt{out}(k^{-1}) \quad F \vdash_{\mathtt{s}} \mathtt{out}(\{m\}_k)}{F \vdash_{\mathtt{s}} \mathtt{out}(m)}$$

Table 3: Security cs entailment relation.

NOTATION 1. *Suppose that $P_1 \stackrel{(\mathtt{true},c_1)}{\Longrightarrow}_s P_2 \stackrel{(\mathtt{true},c_2)}{\Longrightarrow}_s \ldots$, and c is a state formula. Let $P = P_1$. We say that P* eventually outputs *c, written $P \Downarrow^c$, iff for some $i > 0$, $c_i \vdash_T c$.*

THEOREM 3 (LOGIC CORRESPONDENCE). *Let $[\![\cdot]\!]$ as in Definition 8. If P is abstracted-unless free and c is a state formula then $[\![P]\!] \vdash_T \Diamond c$ iff $P \Downarrow^c$.*

A key aspect of Theorem 3 is that it allows using well-established techniques from LTL for reachability analysis of `utcc` processes: E.g., to verify if a given security protocol modelled in `utcc` can reach a state where secrecy is violated.

6. APPLICATIONS

This section illustrates the use of `utcc` in the analysis of security protocols. In particular, we shall exhibit the secrecy flaw of the Needham-Schroeder protocol (NS) [7].

NS aims at distributing two *nonces* (unguessable secrets) in a secure way. Figure 1(a) shows the steps of NS where m and n represent the nonces generated, resp., by the principals A and B. The notation $\{m, A\}_B$ represents the message obtained by encrypting with B's public key the composition m with A's identifier. In [7] it is shown how an attacker (C in Figure 1(b)) can reveal n thus violating the *secrecy* property.

Modelling NS in `utcc`. As typically done we follow the Dolev and Yao thread model [5] which presupposes an attacker that can eavesdrop, disassemble, compose, encrypt and decrypt messages with available keys. It also presupposes that cryptography is unbreakable. We begin by defining a cs based on [5] which follows the Dolev and Yao model.

DEFINITION 9. *Let Σ_s be a signature with constant symbols in $\mathcal{P} \cup \mathcal{K} \cup \{\mathtt{fail}\}$, function symbols enc, pair, inv and key_p and unary predicates* `out`, `secret`, `out'`. *Let Δ_s be the closure under deduction of $\{\ F \mid \vdash_s F\ \}$ with $\vdash_s$ as in Table 3. The (secure) cs is the pair (Σ_s, Δ_s).*

Intuitively, $\mathcal{P}$ and $\mathcal{K}$ represent the principal identifiers $A, B, \ldots$ and keys $k, k', \ldots$. We use $\{m\}_k$, $\{m_1, m_2\}$, k^{-1}, resp., for $enc(m, k)$ (encryption), $pair(m_1, m_2)$(composition) and $inv(k)$ (inverse key). Furthermore, $key_p(x)$ represents the public key of x. From [5] one can show that $\vdash_{\mathtt{s}}$ is decidable. The rule ENC says that if one can infer that the message m as well as a key k are output on the global channel `out`, then one may as well infer that $\{m\}_k$ is also output on `out`. The other rules can be explained similarly.

We now specify the principals. First we need a process $W = (\mathbf{wait}\ \vec{x}; c)\ \mathbf{do}\ P$ which *waits*, possibly for several time

units, until for some t, $c[\vec{t}/\vec{x}]$ holds and then it executes $P[\vec{t}/\vec{x}]$. Given $stop, go \notin fv(P)$, W is defined as:

$$(\mathbf{local}\, stop, go)\,(\mathbf{tell}(\mathtt{out}'(go)) \parallel! \mathbf{unless}\ \mathtt{out}'(stop)\ \mathbf{next}\,\mathbf{tell}(\mathtt{out}'(go)) \parallel! (\mathbf{abs}\ \vec{x}; c \wedge \mathtt{out}'(go))\,(P \parallel! \mathbf{tell}(\mathtt{out}'(stop)))$$

The NS model uses **Init** and **Resp** describing the role of the initiator and the responder i.e. A and B in Figure 1(a). Posting messages, nonce generation, message reception are modelled, resp., using tell, local and wait processes.

$$\begin{array}{rcl}
\mathbf{Init}(i,r) & \stackrel{\text{def}}{=} & (\mathbf{local}\, m)\,\mathbf{tell}(\mathtt{out}(\{m,i\}_{key_p(r)})) \parallel \\
& & \quad (\mathbf{wait}\ x; \mathtt{out}(\{m,x\}_{key_p(i)}))\ \mathbf{do} \\
& & \qquad \mathbf{next}\,\mathbf{tell}(\mathtt{out}(\{x\}_{key_p(r)})) \\
\mathbf{Resp}(t) & \stackrel{\text{def}}{=} & (\mathbf{local}\, n)\,\mathbf{next}\,(\mathbf{wait}\ y,p; \mathtt{out}(\{y,p\}_{key_p(t)}))\ \mathbf{do} \\
& & \qquad \mathbf{next}\,\mathbf{tell}(\mathtt{out}(\{y,n\}_{key_p(p)})) \\
& & \quad \parallel! \mathbf{tell}(\mathtt{secret}(n))
\end{array}$$

Notice that since Responder's nonce cannot be known by an attacker, $\mathbf{Resp}(t)$ states that n is secret.

We also define an *observer* $\mathcal{O}$ telling `fail` whenever a secret appears unprotected on the global channel `out`. Additionally the observer remembers every message that was ever on `out` by transferring them across the time intervals.

$$\begin{array}{rcl}
\mathcal{O} & \stackrel{\text{def}}{=} & !(\mathbf{abs}\ n; \mathtt{secret}(n) \wedge \mathtt{out}(n))\,\mathbf{tell}(\mathtt{fail}) \parallel \\
& & !(\mathbf{abs}\ m; \mathtt{out}(m))\,\mathbf{next}\,\mathbf{tell}(\mathtt{out}(m))
\end{array}$$

The situation in Figure 1(b) can be modelled as:

$$\mathcal{NS} = \mathbf{Init}(A,C) \parallel \mathbf{Resp}(B) \parallel \mathcal{O} \parallel \mathbf{tell}(c_{init})$$

with $c_{init} = \mathtt{out}(key_p(C)^{-1}) \wedge \forall_{id \in \mathcal{P}}(\mathtt{out}(id) \wedge \mathtt{out}(key_p(id)))$ representing the initial knowledge available for an attacker. Namely, all the principal ids, their public keys and the compromised private key of C.

We can use the symbolic semantics to show that $\mathcal{NS}$ eventually outputs `fail` :

PROPOSITION 2. $\mathcal{NS} \Downarrow^{\mathtt{fail}}$.

Alternatively, the above proposition can be proved using Theorem 3 and LTL deduction.

Notice that we *cannot* use the SOS to exhibit an output of `fail` since the secure cs will generate infinite substitutions within a time interval (see Section 3.2.1). E.g., If $F \vdash \mathtt{out}(m)$ then $F \vdash \mathtt{out}(\{m,m\})$ and $F \vdash \mathtt{out}(\{m,m,m\})$ and so on. This illustrates the relevance of the symbolic semantics and the correspondence in Theorem 3.

7. CONCLUDING REMARKS

From a practical point of view, we have implemented in Oz (`http://www.mozart-oz.org/`) a prototype of an interpreter executing programs written in `utcc`. Following the SOS, the interpreter computes the final stores given the input sequences and the process to be executed. To avoid problems with infinitely many substitutions, we restricted m, m_1 and m_2 in Rules ENC and PAIR to have less than a fixed number of applications of *enc* and *pair*. In this way we have automatically verified Proposition 2.

Related Work In [2] and [5] a *symbolic semantics* is introduced to deal with the infinite-branching reduction relation of variants of the *spi*-calculus caused by infinitely many substitutions. Roughly, boolean constraints over names represent conditions the transition must hold to take place. These calculi, however, are conceptually different from CCP. Furthermore, here not only did the symbolic semantics deal with the infinite-branching problem but also with temporal issues and divergent internal computation in the SOS.

The language *LMNtal* [14] uses logical variables to specify mobility behaviour as basic CCP does. Since LMNtal was designed as a unifying model of concurrency, it is nondeterminisitc thus not adhering to Criterion (2). Furthermore, to our knowledge no correspondence between this languages and logic has been given (Criterion (1)).

Several process calculi have been specifically designed for the analysis of *security protocols*, e.g., [5, 2, 3]. Although `utcc` can be used to reason about certain aspects of security protocols (e.g., secrecy), it was not designed exclusively for this domain of application. Nevertheless, its LTL characterisation offers a reasoning technique for reachability, to our knowledge not provided by the above calculi.

The rather successful *logic programming* approach to security protocols in [1] is closely related to ours in `utcc`. The basic difference is that the underlying logic in [1] is Horn clauses while ours is LTL. Also our approach benefits from the operational view of process calculi.

8. REFERENCES

[1] B. Blanchet. Security Protocols: From Linear to Classical Logic by Abstract Interpretation. *Information Processing Letters*, 95(5), 2005.

[2] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. of ICALP'01*. LNCS, 2001.

[3] F. Crazzolara and G. Winskel. Events in security protocols. In *Proc. of CCS '01*. ACM Press, 2001.

[4] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Information and Computation*, 165, 2001.

[5] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proc. of the 14th IEEE Workshop on Computer Security Foundations*. IEEE Computer Society, 2001.

[6] C. Laneve and U. Montanari. Mobility in the CC-paradigm. In *Mathematical Foundations of Computer Science*, 1992.

[7] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of TACAS'96*. LNCS, 1996.

[8] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1991.

[9] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[10] M. Nielsen, C. Palamidessi, and F. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Comp.*, 9(1), 2002.

[11] C. Palamidessi, V. Saraswat, F. Valencia, and B. Victor. On the expressiveness of linearity vs persistence in the asychronous pi-calculus. In *Proc. of LICS'06*. IEEE Computer Society, 2006.

[12] V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proc. of LICS'94*. IEEE CS, 1994.

[13] V. A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.

[14] K. Ueda, N. Kato, K. Hara, and K. Mizuno. LMNtal as a unifying declarative language: Live demonstration. In *Proc. of ICLP'06*. LNCS, 2006.