Liveness and Safety in Concurrent Constraint Programs*

Andreas Podelski
Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken
podelski@mpi-sb.mpg.de

1 Temporal operators

In this section we recall the standard definitions of temporal operators in the notation of Clarke's CTL [**?**]. The operators are used to specify liveness properties ("something good will finally happen") and safety properties ("nothing bad will happen"). In non-deterministic systems one has to make precise whether the specification refers to some or to all execution sequences starting from a given state.

Let $(\mathcal{S}, \mathcal{T})$ be a transition system, *i.e.*, a set $\mathcal{S}$ of states together with a (non-deterministic) transition function $\mathcal{T} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$, *i.e.*, a mapping from $\mathcal{S}$ to the powerset of $\mathcal{S}$. Given a property of states, *i.e.*, a set $Prop \subseteq \mathcal{S}$, we can define a new set of states satisfying a CTL-style temporal property in one of the following ways.

$$\begin{aligned}
\mathsf{EF}(Prop) &= lfp(\lambda X.\ Prop \cup \mathcal{T}^{-1}(X)) \\
\mathsf{EG}(Prop) &= gfp(\lambda X.\ Prop \cap \mathcal{T}^{-1}(X)) \\
\mathsf{AF}(Prop) &= \mathcal{S} - \mathsf{EG}(\mathcal{S} - Prop) \\
\mathsf{AG}(Prop) &= \mathcal{S} - \mathsf{EF}(\mathcal{S} - Prop)
\end{aligned}$$

Here, $\mathsf{EF}(Prop)$ ("exists finally *Prop*") denotes the set of all states $s \in \mathcal{S}$ for which there exists an execution starting in s and reaching a state s' in *Prop*, *i.e.*, there exists a sequence of states $s = s_0, \ldots, s_n = s'$ such that $s_i \in \mathcal{T}(s_{i-1})$ for $i = 1, \ldots, n$.

$$\mathsf{EF}(Prop) = \bigcup_{i \geq 0} (\mathcal{T}^{-1})^i(Prop)$$

Similarly, $\mathsf{EG}(Prop)$ ("exists globally *Prop*") denotes the set of all states $s \in \mathcal{S}$ for which there exists an execution starting in s such that all reached states lie in *Prop*, *i.e.*, a (necessarily infinite) sequence of states $s = s_0, s_1, s_2, \ldots \in Prop$ such that $s_i \in \mathcal{T}(s_{i-1})$ for $i = 1, \ldots, n$.

$$\mathsf{EG}(Prop) = \bigcap_{i \geq 0} (\mathcal{T}^{-1})^i(Prop)$$

The meaning of the other two sets is obtained in the analogous way by referring to *all* executions starting in the states they contain.

2 Constraints

We assume a constraint system consisting of a class $\mathcal{L}$ of constraints φ, φ', ψ, ..., *i.e.*, a class of first-order formulas over a given signature Σ which is closed under conjunction and existential quantification. We also assume a structure $\mathcal{D}$, *i.e.*, a domain D of values together

*This is a note about preliminary results. Comments are solicited. Last change: February 25, 1997

with an interpretation of the function and relation symbols occurring in Σ. A valuation is a mapping $\alpha : \mathsf{Var} \to D$ from the given infinite set of variables Var to the domain D.

The cc programming language Oz [**?**] uses the constraint system over infinite trees. Here the constraints are equalities between variables and terms over a given signature Σ of function symbols. We may restrict ourselves to equalities of the form $x = y$ or $x = f(\bar{u})$. An infinite tree is defined as usual. One may formalize an infinite tree as a partial function from the free monoid of strings over natural numbers (a string designates a node by a path) to Σ; its domain must contain the root node (designated by the empty string ε) and, with every node p, the nodes $p.1, \ldots, p.k$ where $k \geq 0$ is the arity of the function symbol f assigned to p. The interpretation of the function symbol f is the function f that contructs a new tree. That is, if $t_1, \ldots, t_k$ are trees, then $t = f(t_1, \ldots, t_k)$ is given by $t(\varepsilon) = f$ and $t(i.p) = t_i(p)$ for all paths p. Note that the satisfiability notions of contraints over rational trees (which are employed in Prolog-II, for example [**?**]) and over infinite trees are equivalent (even their first-order theories are equivalent [**?**]). The difference lies in the satisfiability of infinite sets of constraints and, thus, appears only when infinite executions are considered.

The following property of constraint systems has already been considered by Palmgren [**?**], who proved that it is a sufficient condition for the *canonicity* of constraint logic programs. The property is relevant for negation as failure [**?**, **?**]; it says that the greatest fixpoint of the T_P-operator is obtained after omega-many iterations.

Definition 1 (Saturation) A constraint system has the saturation property if a countable set Φ of constraints is satisfiable already when every finite subset of Φ is satisfiable. That is, for all constraints φ_i, $i \geq 1$, the following holds.

$$\{\varphi_i \mid i \leq n\} \text{ satisfiable for all } n \Longrightarrow \{\varphi_i \mid i \geq 1\} \text{ satisfiable}$$

Proposition 1 ([?]) The constraint system of infinite trees has the saturation property.

Proof. Our proof is by the observation that the space of valuations over infinite trees is compact and that the solutions of equations over infinite trees are closed. □

Other examples of constraint systems with the saturation property are linear equations over the reals as in CLP(R) [**?**] and finite domain constraints. The constraint systems over finite trees and over rational trees do not have the saturation property. Take the sets $\{x = f^i(y) \mid i \leq n\}$ as a counterexample for the first case, and the sets $\{x = g(f^i(a), f^i(y)) \mid i \leq n\}$ for the second. Palmgren [**?**] has given a construction for the conservative extension of any constraint system to one with the saturation property; conservative here means that the satisfiability notion for finite sets of constraints is preserved.

3 cc* programs

We will next define the syntax of cc* programs and their interleaving semantics in terms of transition systems. Our class of programs is a subclass of concurrent constraint programs [**?**]. In Section **??**, we will see in what sense temporal properties of a cc program are approximated in terms of the corresponding cc* program. Note that our interleaving semantics coincides with the *abstract* operational semantics usually given to constraint logic programs (with a non-deterministic selection function) [**?**]; the correctness of such programs is, however, specified in terms of an input-output relation, and non-terminating executions are considered erroneous.

A cc* program P consists of the definitions of procedures $p \in \mathsf{Proc}$ (also called "processes" or "predicates") by an application of the non-deterministic choice operator "$\vee$" to $n \geq 1$ statements. Each of these statements is the parallel composition "$\wedge$" of a constraint ψ_i (standing for the corresponding "tell" operation) and the parallel composition of other procedure calls

("process invocations"). We leave the quantifiers (existential quantifiers for the statements, universal closure of the equivalences) implicit for better readability. As usual, when we apply a procedure $p(\bar{y})$, we assume consisting α-renaming of the definition of p according to the actual parameters $\bar{y}$ (and absence of variable capturing).

$$P \quad \equiv \quad \bigwedge_{p \in \mathsf{Proc}} \; p(\bar{x}) \leftrightarrow \bigvee_{i=1,\ldots,n} (\psi_i \wedge \bigwedge_{j=1,\ldots,m_i} p_{ij}(\bar{y}_{ij}))$$

We now define the transition systems that are induced by cc* programs. A state s is a pair consisting of the parallel composition of procedure calls and of the constraint store (an *always* satisfiable constraint); we also have an error state err.

$$\mathcal{S} = \{\langle \bigwedge_{i=1,\ldots,n} p_i(\bar{x}_i),\ \varphi\rangle \mid n \geq 0,\ \varphi \text{ is satisfiable}\} \cup \{\mathsf{err}\}$$

We identify states modulo the AC1-laws for parallel composition of procedure calls in the first component and modulo logical equivalence of the constraint stores in the second component. We use the same symbol $\wedge$ also for the logical 'and' composition of constraints; in both cases, *true* is the neutral element; it corresponds either to the empty parallel composition or to the empty conjunction. If the first component of the state s is *true* then we call s a *terminal state.*

The non-deterministic transition function $\mathcal{T}_P$ (we note the transition relation $\longrightarrow_{\mathcal{T}_P}$) of the system induced by the program P is defined as follows . Let $p(\bar{x})$ be defined as above by the choice of n statements. Let s be a state in which the procedure is called; *i.e.*, the first component of s can be written in the from $p(\bar{x}) \wedge E$. Let φ be the constraint store of s. Then, for any $i = 1, \ldots, n$ such that $\varphi \wedge \psi_i$ is satisfiable, there is a non-deterministic transition from s of the form:

$$\langle p(\bar{x}) \wedge E, \varphi\rangle \quad \longrightarrow_{\mathcal{T}_P} \quad \langle \bigwedge_{j=1,\ldots,m_i} p_{ij}(\bar{y}_{ij}) \wedge E, \varphi \wedge \psi_i\rangle.$$

If no such transition exists for that procedure p, then there is a transition from s into the error state err. If s is a terminal state, then there is a transition from s to itself.

$$\langle \mathit{true}, \varphi\rangle \quad \longrightarrow_{\mathcal{T}_P} \quad \langle \mathit{true}, \varphi\rangle$$

The reason for adding a loop to terminal states is purely formal. Our definition of EG does not formalize safety properties of finite execution sequences.

As is usual with concurrent systems, we need a notion of fairness. Most correctness proofs of concurrent systems rely on a fairness assumption [**?**]. The fairness notion of concurrent constraint systems (see [**?**]) is intuitive and simple. We take the identical notion for cc* systems.

Definition 2 (Fairness) An execution sequence $s_0, s_1, s_2, \ldots$ of a cc* transition system is *fair* if for every state s_i, every procedure call $p(\bar{x})$ appearing in s_i will be applied finally (*i.e.*, for some state s_{i+k} in the execution sequence).

The successor state after the application of $p(\bar{x})$ is possibly the error state err. Whereas the definition of the temporal operators is independent of err and the transitions into err, the definition of fairness does require it. Note that the error state err corresponds to the notion of *failure* in the operational semantics of constraint logic programs [**?**]; there, however, failure is part of the control strategy with backtracking. (Obviously, there is no backtracking in the execution of concurrent processes; cc programming languages such as AKL [**?**] and Oz [**?**] accomodate search through local computation spaces and situated simplification [**?**].)

4 Assertions

The following schema for specifying state properties is inspired by the abstract debugging framework for imperative programming languages by Bourdoncle [**?**], where "intermittent" or "invariant" assertions are attached to control points and sufficient conditions for their violations are derived by abstract interpretation. The difficulties in transferring it to concurrent constraint programming languages lie in the facts that (1) several (in fact, unboundedly many) control points may be reached concurrently and (2) in general, it is very hard to compute $tell_\psi^{-1}$, the inverse of the operation that adds a conjunct ψ to a constraint store. We overcome the first difficulty by adapting the approach of Lamport described in [**?**], which refers to Ashcroft's assertion framework for concurrent programs. We overcome the second difficulty by defining a setup where we need to apply $tell_\psi^{-1}$ only to sets of states of a restricted form.

The idea is to annotate procedures p with logical formulas γ_p. An *annotation* Γ is thus a function

$$\Gamma : p \in \mathsf{Proc} \to \gamma_p$$

from procedures $p \in \mathsf{Proc}$ to *assertions* γ_p. Given a procedure call $p(\bar{y})$, we obtain the appropriate assertion $\gamma_p(\bar{y})$ by corresponding renaming of the free variables in γ. We observe that an annotation $\Gamma : p(\bar{x}) \in \mathsf{Proc} \to \gamma_p(\bar{x})$ defines a set of facts $p(\bar{d})$ (where $d_1, \ldots, d_n \in D$), namely $\{p(\bar{d}) \mid \mathcal{D}, \alpha[\bar{x} \mapsto \bar{d}] \models \gamma_p(\bar{x})\}$. Starting from this observation, we make our notion of assertions and annotations formally formally precise.

Definition 3 (Annotation) An annotation Γ is a set of facts, *i.e.*, a subset of $\mathcal{B}_\mathcal{D} = \{p(\bar{d}) \mid p \in \mathsf{Proc},\ \bar{d} \in \bar{D}\}$. The annotation Γ defines the assertion $\gamma_p(\bar{x})$ for the procedure call $p(\bar{x})$ by:

$$\gamma_p(\bar{x}) \ \equiv \ p(\bar{x}) \in \Gamma.$$

A state $s = \langle p(\bar{x}), \varphi\rangle$ with a single procedure call $p(\bar{y})$ satisfies the assertion of Γ if φ is satisfiable in conjunction with $\gamma_p(\bar{y})$. Given a state $\langle \bigwedge_{i=1,\ldots,n} p_i(\bar{x}_i), \varphi\rangle$ with several procedure calls, we will require something slightly stronger than the satisfaction of the assertions of Γ by each state $\langle p_i(\bar{x}_i), \varphi\rangle$.

Definition 4 (Satisfaction of assertions) The set $\mathsf{Prop}(\Gamma)$ of states satisfying the assertions γ_p of the annotation Γ is defined by:

$$\mathsf{Prop}(\Gamma) = \{\langle \bigwedge_{i=1,\ldots,n} p_i(\bar{x}_i), \varphi\rangle \mid \varphi \wedge \bigwedge_{i=1,\ldots,n} \gamma_{p_i}(\bar{x}_i) \text{ is satisfiable}\}.$$

We say that a state satisfies the assertions given by Γ *intermittently* if it lies in the set $\mathsf{EF}(\mathsf{Prop}(\Gamma))$, and that it satisfies the assertions given by Γ *invariantly* if it lies in the set $\mathsf{EG}(\mathsf{Prop}(\Gamma))$.

If the assertions are used to specify the set of correct states in terms of the liveness property $\mathsf{EF}(\mathsf{Prop})$, we call them *intermittent assertions*. If the assertions are used to specify the set of correct states in terms of the safety property $\mathsf{EG}(\mathsf{Prop})$, we call them *invariant assertions*. The default intermittent assertion for a predicate p (without an explicit annotation) is $\gamma_p = \mathit{true}$. The default intermittent assertion is $\gamma_p = \mathit{false}$.

Two important special cases of annotations are $\Gamma_{\mathit{false}} = \emptyset$ and $\Gamma_{\mathit{true}} = \mathcal{B}_\mathcal{D}$, which correspond to the mappings $\Gamma_{\mathit{false}} : p \in \mathsf{Proc} \mapsto \mathit{false}$ and $\Gamma_{\mathit{true}} : p \in \mathsf{Proc} \mapsto \mathit{true}$ of all predicates to the Boolean constants *false* and *true*, respectively. Γ_{false} defines the *strongest* assertions. The states satisfying the strongest assertions are exactly the terminal states.

$$\mathsf{Prop}(\Gamma_{\mathit{false}}) = \{\langle \mathit{true}, \varphi\rangle \mid \varphi \text{ is satisfiable}\}$$

The strongest assertions are important for their use as intermittent assertions.

Observation 1 $\mathsf{EF}(\mathsf{Prop}(\Gamma_{\mathit{false}}))$ is the set of all states for which there exists an execution reaching a terminal state.

Γ_{true} defines the *weakest* assertions. The states satisfying the weakest assertions are exactly the non-error states.

$$\mathsf{Prop}(\Gamma_{\mathit{true}}) = \{\langle E, \varphi\rangle \mid \varphi \text{ is satisfiable}\}$$

The weakest assertions are important for their use as invariant assertions.

Observation 2 $\mathsf{EG}(\mathsf{Prop}(\Gamma_{\mathit{true}}))$ is the set of all states for which there exists an infinite execution (*i.e.*, an execution which does not go into the error state err). By duality, $\mathsf{AF}(\{\mathsf{err}\})$ is the set of all states such that all executions end with the error state.

If we transfer the above observations to the setting of constraint logic programming, we obtain the characterizations of the *success* set of a CLP program and of its *finite-failure* set in terms of temporal properties. Success is a special case of a liveness property. Finite failure is the dual of a special case of a safety property. This view seems to be new.

If we extend the syntax of programs to include assertions γ_p, we may restrict ourselves to the two special cases of the annotations Γ_{false} and Γ_{true} without loosing generality. Namely, $\mathsf{EF}(\mathsf{Prop}(\Gamma))$ with respect to the program P is equal to $\mathsf{EF}(\mathsf{Prop}(\Gamma_{\mathit{false}}))$ with respect to the program $P \vee \mathsf{Prop}$ below. We obtain this program from P by adding the assertion $\gamma_p(\bar{x})$ (which stands for: $p(\bar{x}) \in \Gamma$)) as a disjunct to the choice defining the procedure $p(\bar{x})$.

$$P \vee \mathsf{Prop}(\Gamma) \quad \equiv \quad \bigwedge_{p \in \mathsf{Proc}} p(\bar{x}_p) \leftrightarrow \bigvee_{i=1,\ldots n} (\psi_i \wedge \bigwedge_{j=1,\ldots,m_i} p_{ij}(\bar{y}_{ij})) \quad \vee \quad \gamma_p(\bar{x})$$

In the analogous way, $\mathsf{EG}(\mathsf{Prop}(\Gamma))$ with respect to the program P is equal to $\mathsf{EG}(\mathsf{Prop}(\Gamma_{\mathit{true}}))$ with respect to the program $P \wedge \mathsf{Prop}$ below. We obtain this program from P by adding by adding the assertion $\gamma_p(\bar{x})$ as a conjunct to each statement in the choice defining the procedure $p(\bar{x})$.

$$P \wedge \mathsf{Prop}(\Gamma) \quad \equiv \quad \bigwedge_{p \in \mathsf{Proc}} p(\bar{x}_p) \leftrightarrow \bigvee_{i=1,\ldots n} (\psi_i \wedge \bigwedge_{j=1,\ldots,m_i} p_{ij}(\bar{y}_{ij}) \quad \wedge \quad \gamma_p(\bar{x}))$$

Note that we did not put any restriction on the annotations Γ; *a priori*, the sets of values defined by an assertion may not even be recursive. Thus, it may be misleading to refer to $P \vee \mathsf{Prop}$ and $P \wedge \mathsf{Prop}$ as *programs*.

5 Weakest pre-conditions

The function tell_ψ applied on a state s yields the set of successor states of s (a singleton or the empty set) under the operation that adds the constraint ψ as a conjunct to the constraint store. The following observation has motivated our assertion framework.

$$\mathit{tell}_\psi^{-1}(\mathsf{Prop}(\Gamma)) = \{\langle \bigwedge_{i=1,\ldots,n} p_i(\bar{x}_i), \varphi\rangle \mid \bigwedge_{i=1,\ldots,n} \gamma_{p_i(\bar{x}_i)} \wedge \varphi \wedge \psi \text{ is satisfiable}\}$$

Given the annotation $\Gamma : p \in \mathsf{Proc} \mapsto \gamma'_p$ and the constraint ψ, we define the annotation $\Gamma \wedge \psi$ as the mapping $\Gamma \wedge \psi : p \in \mathsf{Proc} \mapsto \gamma'_p$ where $\gamma'_p = \psi \wedge \gamma_p$. Then, a more concise formulation of the observation is:

$$\mathit{tell}_\psi^{-1}(\mathsf{Prop}(\Gamma)) = \mathsf{Prop}(\Gamma \wedge \psi).$$

We will next investigate the function $\mathcal{T}_P^{-1}$ and the fixpoints of the functions $\lambda X.\ \mathsf{Prop} \cup \mathcal{T}^{-1}(X)$ and $\lambda X.\ \mathsf{Prop} \cap \mathcal{T}^{-1}(X)$.

We first define the non-deterministic transition function $\tilde{\mathcal{T}}_P$ where all procedure calls $p(\bar{x}_k)$ in the parallel composition of a state are applied simultaneously in one step. We assume that the predicates p_k are defined by:

$$p_k(\bar{x}_k) \leftrightarrow \bigvee_{i_k} (\psi_{i_k} \wedge \bigwedge_j p_{i_k j}(\bar{y}_{i_k j})).$$

Then the non-deterministic transition relation $\longrightarrow_{\tilde{\mathcal{T}}_P}$ is given by (for any choice of indices i_k):

$$\langle \bigwedge_k p_k(\bar{x}_k), \varphi \rangle \longrightarrow_{\tilde{\mathcal{T}}_P} \langle \bigwedge_k \bigwedge_j p_{i_k j}(\bar{y}_{i_k j}), \varphi \wedge \bigwedge_k \psi_{i_k} \rangle.$$

Proposition 2 Provided the execution sequences are fair, the temporal properties wrt. to the transition system $\mathcal{T}_P$ may be defined wrt. the transition function $\tilde{\mathcal{T}}_P$. That is, for all sets of states *Prop*:

$$\mathsf{EF}(Prop) = lfp(\lambda X.\ Prop \cup \tilde{\mathcal{T}}_P^{-1}(X))$$
$$\mathsf{EG}(Prop) = gfp(\lambda X.\ Prop \cap \tilde{\mathcal{T}}_P^{-1}(X))$$

Proof. The constraint store is growing monotonically during one execution sequence. Thus, the (possibly infinite) conjunctions of the constraints added to the constraint stores added during two execution sequences are logically equivalent if the two execution sequences differ only in the order of procedure calls (but not in the choices of the disjuncts composed by the choice operator). The difference between such two execution sequences stems from the non-determinism due to the interleaving semantics (as opposed to the *in*determinism, which is due to the choice operator and which generally yields non-confluent transition sequences). Eliminating the non-determinism amounts to using the *true-concurrency* semantics of Montanari and Rossi [**?**]. In the special case of $\mathsf{cc}^\star$ programs, this semantics can be described very simply, *i.e.*, by the transition function $\tilde{\mathcal{T}}_P$. □

We recall that the logical consequence operator T_P maps a set X of ground facts to the set of their immediate consequences under the programm P; *i.e.*, if $X \subseteq \mathcal{B}_{\mathcal{D}} = \{p(\bar{d}) \mid p \in \mathsf{Proc},\ \bar{d} \in \bar{\mathcal{D}}\}$, then

$$T_P(X) = \{p(\bar{d}) \mid \exists \alpha : \mathsf{Var} \to D\ \exists i \in \{1, \ldots, n\} : \begin{array}{l} \alpha(\bar{x}_p^i) = \bar{d}, \\ \mathcal{D}, \alpha \models \psi_i, \\ p_{ij}(\alpha(\bar{y}_{ij})) \in X\} \end{array}$$

where the set comprehension ranges also over all procedures $p \in \mathsf{Proc}$ defined by the program P.

Since an annotation Γ is a set of ground facts, we may apply T_P to an annotation Γ. We then obtain an annotation that characterizes the predecessor relation in the following sense.

Proposition 3 A state is a predecessor of a state satisfying the assertions of the annotation Γ if and only if it satisfies the assertions of the annotation $\mathcal{T}_P(\Gamma)$. That is:

$$\tilde{\mathcal{T}}_P^{-1}(\mathsf{Prop}(\Gamma)) = \mathsf{Prop}(T_P(\Gamma)).$$

Proof. Given the annotation $\Gamma : p \in \mathsf{Proc} \mapsto \gamma_p$, the inverse of $\tilde{\mathcal{T}}_P$ is defined as follows.

$$\tilde{\mathcal{T}}_P^{-1}(\mathsf{Prop}(\Gamma)) = \{\langle \bigwedge_k p_k(\bar{x}_k), \varphi \rangle \mid \begin{array}{l} \text{for all } k \text{ exists } i_k \text{ such that} \\ \varphi \wedge \bigwedge_k (\psi_{i_k} \wedge \bigwedge_j \gamma_{p_{i_k j}}(\bar{y}_{i_k j})) \text{ is satisfiable}\} \end{array}$$

Thus, $\tilde{\mathcal{T}}_P^{-1}(\mathsf{Prop}(\Gamma)) = \mathsf{Prop}(\Gamma' : p \in \mathsf{Proc} \mapsto \gamma'_p)$ where

$$\begin{array}{rcl} \gamma'_p(\bar{x}_p) & \equiv & \bigvee_i \psi_i \wedge \bigwedge_j \gamma_{p_{ij}}(\bar{y}_{i_k j}) \\ & \equiv & p(\bar{x}) \in T_P(\{p(\bar{d}) \mid \gamma_p(\bar{d})\}) \\ & \equiv & p(\bar{x}) \in T_P(\Gamma). \end{array}$$

□

Theorem 1 (Liveness) A state satisfies the assertions given by the annotation Γ intermittently if and only if it satisfies the assertion given by the closure of the T_P operator applied to Γ. That is:

$$\mathsf{EF}(\mathsf{Prop}(\Gamma)) = \mathsf{Prop}(\mathit{lfp}(\lambda X.(\Gamma \cup T_P(X)))).$$

Proof. In the chain of equalities below, we use Proposition 2 for (1), Proposition 3 for (2), the fact that an (infinite) disjunction of formulas is satisfiable if and only if one of these formulas is satisfiable for (3), and the fact that the T_P operator is continuous for (4).

$$\begin{aligned}
\mathsf{EF}(\mathsf{Prop}(\Gamma)) &= \bigcup_{i\geq 0} (\tilde{\mathcal{T}}_P^{-1})^i(\mathsf{Prop}(\Gamma)) && (1)\\
&= \bigcup_{i\geq 0} \mathsf{Prop}(T_P^i(\Gamma)) && (2)\\
&= \mathsf{Prop}(\bigcup_{i\geq 0} T_P^i(\Gamma)) && (3)\\
&= \mathsf{Prop}(\mathit{lfp}(\lambda X.(\Gamma \cup T_P(X)))) && (4)
\end{aligned}$$

□

We formulate the characterization above for the special case of the annotation Γ_{false}.

$$\mathsf{EF}(\mathsf{Prop}(\Gamma_{\mathit{false}}) = \mathsf{Prop}(\mathit{lfp}(T_P))$$

This yields the classical characterization of the success set of a CLP program P in terms of the least model $lm(P)$ of the logical formula corresponding to the program P ($lm(P)$ is equal to the least fixpoint of the T_P operator). In order to see this we only have to note that the success set is equal to $\mathsf{EF}(\mathsf{Prop}(\Gamma_{\mathit{false}}))$ and unfold the definition of Prop. Namely, a state $\langle p(\bar{x}), \varphi\rangle$ has a terminating execution sequence if and only if the formula $\varphi \wedge p(\bar{x}) \in \mathit{lfp}(T_P)$ has a solution.

The above theorem states the weakest pre-condition of a liveness property specified by assertions in terms of new assertions. In the special case of the "strongest assertions" *false* (which specify termination), the new assertions are defined by the least model of the program formula.

We will next study the case of safety properties. This case is not analogous to the previous one for the following reason. The satisfiability of the constraint φ with all finite conjunctions of assertions $\gamma_1 \wedge \ldots \wedge \gamma_n$ is generally not equivalent to the satisfiability of the constraint φ with the infinite conjunction of assertions $\gamma_1 \wedge \gamma_2 \wedge \ldots$. We thus need an additional requirement.

Theorem 2 (Safety) Provided the saturation property holds for the underlying constraint system, we have the following characterization with respect to fair execution sequences: A state satisfies the assertions given by the annotation Γ invariantly if and only if it satisfies the assertion given by the downward closure of the T_P operator restricted to Γ. That is:

$$\mathsf{EG}(\mathsf{Prop}(\Gamma)) = \mathsf{Prop}(\mathit{gfp}(\lambda X.(\Gamma \cap T_P(X)))).$$

Proof. In the chain of equalities below, we use Proposition 2 for (5), Proposition 3 for (6), the saturation property for (7), and the fact from [**?**] that, given the saturation property, the T_P operator reaches its greatest fixpoint after omega many iterations for (8).

$$\begin{aligned}
\mathsf{EG}(\mathsf{Prop}(\Gamma)) &= \bigcap_{i\geq 0} (\tilde{\mathcal{T}}_P^{-1})^i(\mathsf{Prop}(\Gamma)) && (5)\\
&= \bigcap_{i\geq 0} \mathsf{Prop}(T_P^i(\Gamma)) && (6)\\
&= \mathsf{Prop}(\bigcap_{i\geq 0} T_P^i(\Gamma)) && (7)\\
&= \mathsf{Prop}(\mathit{gfp}(\lambda X.(\Gamma \cap T_P(X)))) && (8)
\end{aligned}$$

□

We formulate the characterization above for the special case of the annotation Γ_{true}.

$$\mathsf{EF}(\mathsf{Prop}(\Gamma_{true}) = \mathsf{Prop}(gfp(T_P))$$

This characterization yields a result which seems to be new in the theory of constraint logic programming. We obtain the characterization of the set of all states for which infinite executions exist (and, by the dual, the characterization of the "finite-failure" set of a CLP program P) in terms of the greatest model of the program formula P (which is equal to the greatest fixpoint of the T_P operator). Note that infinite executions subsume those reaching a terminal state in our formalization of transition systems and of temporal properties (which adds a loop to each terminal states).

Corollary 1 Provided the saturation property holds for the underlying constraint system, a state $\langle p(\bar{x}), \varphi \rangle$ has an infinite execution sequence ("does not finitely fail") if and only if the formula $\varphi \wedge p(\bar{x}) \in gfp(T_P)$ has a solution. □

Theorem 2 states the weakest pre-condition of a safety property specified by assertions in terms of new assertions. In the special case of the "weakest assertions" *true* (which specify the absence of a systems failure), the new assertions are defined by the greatest model of the program formula.