Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Uncomputability of Supremal Local Supports in Distributed Diagnosis

J.G. Thistle and R. Su

Abstract— **In distributed diagnosis it may be useful to achieve *local consistency* among local estimates. For that purpose the *Computational Procedure for Local Consistency* (CPLC) was proposed to achieve the *supremal local support*, which represents one type of local consistency. It has been shown that if CPLC terminates then the result is in fact the supremal local support. However, in this paper it is shown that, even if all initial estimates are regular languages, the termination of CPLC is undecidable. Moreover, these difficulties are not confined to this specific procedure: it is undecidable whether the supremal local support corresponding to an arbitrary collection of regular initial languages is componentwise empty; consequently, the supremal local support is effectively uncomputable.**

I. Introduction

In many distributed estimation problems, achieving consistency is crucial. For instance, belief propagation problems [7], [1] require consistency among local posterior probabilities based on newly acquired information; constraint satisfaction problems [5], [14], [12] require consistency among assignments to local variables, which can be categorized as either global consistency or local consistency (i.e. arc consistency [5]); in distributed fault diagnosis problems there is also a global consistency issue and a local consistency issue [8] depending on the priority between quality of diagnosis and scalability of the algorithm. It is well known that, in some cases – such as constraint satisfaction problems featuring a finite number of variables over finite domains – consistency can always be achieved, although the computational complexity is NP-complete [4]; but in others – such as belief propagation [7], [6] – computational procedures for consistency may not terminate.

In this paper we address the local consistency issue in the framework of distributed diagnosis of discrete-event systems. In [8] the authors defined a concept called the *supremal local support*, which was used to capture the interaction between pairs of local components modelled by formal languages during fault diagnosis. A *computational procedure for local consistency* (CPLC) – was proposed to compute the supremal local support – similar to those of [10], [2]. But CPLC may not always terminate: counterexamples were given in [11]. This raises the question of decidability of termination. If it were possible to determine whether CPLC would terminate, we could choose approximate solutions in cases of nontermination, as may be done in loopy belief propagation problems [6] when an exact solution is unattainable. In [9] it was proved that the termination of CPLC is undecidable if initial languages are Turing recognizable; in this paper we show that the termination of CPLC is still undecidable even if all initial languages are restricted to regular ones.

The paper is organized as follows. In Section II we introduce CPLC, and then provide the undecidability result in Section III. In section IV the results are summarized and future work is discussed.

II. Brief Review of CPLC

We assume that readers are familiar with languages and operations such as natural projection and synchronous product; if not, an introduction can be found in [13]. The motivation for CPLC is described in [11] and [8]; here we simply review the algorithm. Let I be a finite index set. Suppose we have a collection of event sets $\{\Sigma_i | i \in I\}$. For each $i, j \in I$ let $\mathrm{P}_{i,j} : \Sigma_i^* \to (\Sigma_i \cap \Sigma_j)^*$ be the natural projection. Let $\mathcal{L} := \{\mathrm{L}_i \subseteq \Sigma_i^* | i \in I\}$ be a collection of languages. Let $\mathrm{Gr} =< V, E >$ be a graph, where $V := I$ and $E \subseteq V \times V$ is such that

$$(\forall i, j \in V)\,(i,j) \in E \iff i \neq j \,\&\, \Sigma_i \cap \Sigma_j \neq \varnothing$$

Clearly, if $(i,j) \in E$ then so is (j,i). Thus Gr is an undirected graph. A *simple path* between two different nodes $i, j \in V$ is a sequence of edges $(v_0, v_1), \cdots, (v_{k-1}, v_k) \in E$ such that

$$v_0 = i \,\&\, v_k = j \,\&\, (\forall l, r : 0 \leq l, r \leq k)\, l \neq r \Rightarrow v_l \neq v_r$$

For each $i, j \in V$ let $\mathrm{Path}(i,j)$ be the set of all simple paths between i and j. For each $w \in \mathrm{Path}(i,j)$ we use $|w|$ to represent the length of the simple path w. For simplicity, we assume that the graph is *connected*, meaning that for any two different nodes there is a simple path connecting them. If Gr is not connected then, since V is finite, the graph consists of a finite number of disjoint connected subgraphs, which in our interpretation represent a finite number of disjoint subsystems. Thus the following theory will apply to each connected subgraph.

Let $d : \mathrm{Ver} \times \mathrm{Ver} \to \mathbb{N}$ with

$$(i,j) \mapsto d(i,j) := \begin{cases} \min_{w \in \mathrm{Path}(i,j)} |w| & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

Research partially supported by grant number RGPIN-155594-01 from the Natural Sciences and Engineering Research Council of Canada.

J.G. Thistle is affiliated with Department of Electrical and Computer Engineering, University of Waterloo, 200 University Avenue West, Waterloo, Ontario N2L 3G1, Canada. jthistle@kingcong.uwaterloo.ca

R. Su is affiliated with System Architecture and Networking Group (SAN) in Department of Mathematics and Computer Science, Eindhoven University of Technology, PO Box 513, 5600 MB Eidnhoven, The Netherlands. R.Su@tue.nl

0-7803-9568-9/05/$20.00 ©2005 IEEE

be a distance map. We (arbitrarily) pick one node as the root node of Gr, say node 1. Let $\phi : \text{Ver} \to \{1, \cdots, |V|\}$ be a one-to-one map such that

$$(\forall i, j \in V)\,(i,j) \in E \Rightarrow [d(1,i) < d(1,j) \Rightarrow \phi(i) < \phi(j)]$$

In general there may exist more than one choice for ϕ. Fix one choice of ϕ and define the following binary relation among vertices,

$$(\forall i, j \in V)\, i \downarrow j \iff (i,j) \in E \ \&\ \phi(i) < \phi(j)$$

We call i a *father node* of j and the binary relation $\downarrow$ the *father-son* relation. It is easy to see that $(\text{Gr}, \downarrow)$ is acyclic. Therefore all nodes can be partitioned into several special pairwise disjoint sets such that a recursive computational procedure can be designed. The following proposition describes these special sets.

Proposition 2.1: Let $\text{Gr} = (V, E)$ be a graph as described above, where $V = I$. Suppose a father-son relation $\downarrow$ among nodes in V is given. Then there exists a partition $\{V_0, V_1, \cdots, V_k\}$ on V such that for the sets V_i $(0 \le i \le k)$ the following two conditions hold,

1) $(\forall j \in V_i)\{r \in I | r \downarrow j\} \subseteq \cup_{m=0}^{i-1} V_m$ where $\cup_{m=0}^{-1} V_m := \varnothing$
2) $(\forall j \in V_i)\{r \in I | j \downarrow r\} \subseteq \cup_{m=i+1}^{k} V_m$ where $\cup_{m=k+1}^{k} V_m := \varnothing$

Proof: The proof is given in Proposition 2.9 of [11]. ■

The first condition in Proposition 2.1 says that no node in V_0 has a father node, and for any other set V_i $(1 \le i \le k)$, the father nodes of any node $j \in V_i$ are contained in $\cup_{m=0}^{i-1} V_m$. The second condition says that no node in V_k has a son node, and for any other set V_i $(0 \le i \le k-1)$, the son nodes of any node $j \in V_i$ are contained in $\cup_{m=i+1}^{k} V_m$.

We now present the algorithm CPLC.

Computational Procedure for Local Consistency:

1) Initialization: $(\forall i \in I)\, \mathrm{M}_i^0 := \mathrm{L}_i$
2) At odd rounds $n \in 2\mathbb{N}+1$, starting from V_k and ending at V_1, for each $i \in V_j$ $(k \ge j \ge 1)$ compute

$$\mathrm{M}_i^n := \begin{cases} \mathrm{M}_i^{n-1} & \text{if } (\nexists r \in I)\, i \downarrow r \\ \mathrm{M}_i^{n-1} \Big|\Big| (||_{r:i\downarrow r} \mathrm{P}_{r,i}(\mathrm{M}_r^n)) & \text{otherwise} \end{cases}$$

3) At even rounds $n \in 2\mathbb{N}^+$, starting from V_1 and ending at V_k, for each $i \in V_j$ $(1 \le j \le k)$ compute

$$\mathrm{M}_i^n := \begin{cases} \mathrm{M}_i^{n-1} & \text{if } (\nexists r \in I)\, r \downarrow i \\ \mathrm{M}_i^{n-1} \Big|\Big| (||_{r:r\downarrow i} \mathrm{P}_{r,i}(\mathrm{M}_r^n)) & \text{otherwise} \end{cases}$$

4) Termination: $(\exists n \in \mathbb{N}^+)(\forall i \in I)\, \mathrm{M}_i^n = \mathrm{M}_i^{n+1}$ □

Proposition 2.1 guarantees that in step 2 before we compute M_i^n all those M_r^n with $i \downarrow r$ have been computed. Similarly in step 3 before we compute M_i^n all those M_r^n with $r \downarrow i$ have been computed. Therefore CPLC is well defined. The main feature of CPLC is its *scalability*, in the sense that when we add a new node to the graph or remove one from the graph, only a few adjacent nodes need to update their local communication protocol, which is usually done by simply changing their lists of father and son nodes.

Figure 1 depicts how CPLC procedure works in a simple example, where the father-son relation is illustrated by the

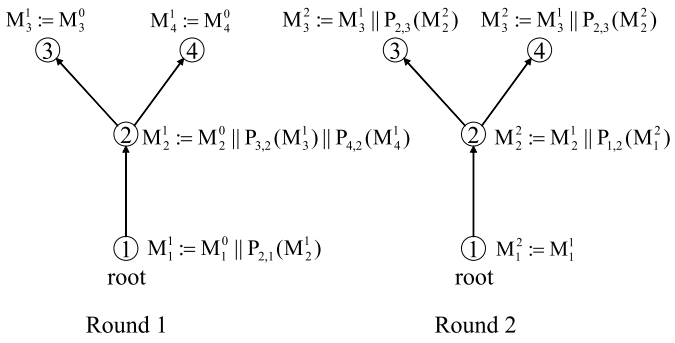

Fig. 1. 2-Round CPLC

arrow-headed edges:

$$1 \downarrow 2 \text{ and } 2 \downarrow 3 \text{ and } 2 \downarrow 4$$

So the partition is $\{V_0 = \{1\}, V_1 = \{2\}, V_2 = \{3,4\}\}$. We can see that during the computational process, in odd-numbered rounds computation starts from nodes in V_2, then passes to the node in V_1 and finally to the node in V_0. In even-numbered rounds computation goes in the opposite direction, namely starting from the node in V_0, and then passing to the node in V_1 and finally to the nodes in V_2. So the computational process is well organized. ♢

In the next section we will show that the termination of CPLC is undecidable, even when all initial languages $\{\mathrm{M}_i^0 | i \in I\}$ are *regular*.

III. Undecidability of termination of CPLC

To prove the undecidability of termination of CPLC, we first define a *Chaotic Distributed Communication Procedure* (CDCP), which is similar to the one used in [2]. We will first show that CDCP has the same termination properties as CPLC. Then we show that the termination of CDCP is undecidable, which establishes the undecidability result for termination of CPLC. In practical applications, CPLC usually converges faster than CDCP, if termination is achievable. There is another major difference between CPLC and CDCP when the network Gr is a tree: if the initial languages in $\mathcal{L}$ are not regular, e.g. context-free, then checking the termination condition for CDCP may not be feasible because language identity may be undecidable [3]; but CPLC does not need to check the termination condition at all because in [11] it has been shown that the termination condition of CPLC is guaranteed to hold no later than round 2. The following proof serves to establish undecidability results for both CPLC and CDCP.

A. Chaotic Distributed Communication Procedure (CDCP)

As before, let I be a finite index set and $\{\Sigma_i | i \in I\}$ a collection of alphabets not necessarily pairwise disjoint. For each $i, j \in I$ let $P_{i,j} : \Sigma_i^* \to (\Sigma_i \cap \Sigma_j)^*$ be the natural projection. Let Gr $= (V, E)$ be the graph induced from $\{\Sigma_i | i \in I\}$ as described in Section II. For each $v \in V$ we define $NB(v) := \{v' \in V | (v, v') \in E\}$ as the *neighbor set* of v. Let $\mathcal{L} := \{L_i \subseteq \Sigma_i^* | i \in I\}$ be the initial set.

Chaotic Distributed Communication Procedure (CDCP):

1. Initialization: $(\forall i \in I)\, L_i^0 := L_i$
2. Iteration:

$$(\forall k \in \mathbb{N}^+)(\forall i \in I)\, L_i^k := L_i^{k-1} \Big\| [||_{j \in NB(i)} \mathrm{P}_{j,i}(L_j^{k-1})]$$

3. Termination: $(\exists n \in \mathbb{N})(\forall i \in I)\, L_i^n = L_i^{n+1}$

Proposition 3.1: Given $\mathcal{L} := \{L_i \subseteq \Sigma^* | i \in I\}$, CDCP terminates if and only if CPLC terminates.

Proof: By the description of CDCP and CPLC, both have the same undirected graph Gr. In CPLC Gr is equipped with the binary relation $\downarrow$ so that $(\mathrm{Gr}, \downarrow)$ is equivalent to an acyclic directed graph. By Proposition 2.1 the set V can be partitioned into $\{V_1, \cdots, V_m\}$ such that the two conditions of Proposition 1 hold. The number $m \in \mathbb{N}^+$ can be regarded as the height of the graph from the leaf nodes (each of which has no son nodes) to the root node. We use L_i^k to denote the language associated with node $i \in I$ at round k in CDCP, and M_i^r for the language related to node $i \in I$ at round r in CPLC. Then we have the following claim:

(a) $(\forall k \in \mathbb{N})(\forall i \in I)\, \mathrm{M}_i^{2k} \subseteq L_i^k$
(b) $(\forall r \in \mathbb{N})(\forall i \in I)\, L_i^{mr} \subseteq \mathrm{M}_i^r$

The proof of the claim is given in a full version of this paper. We now continue the proof of Proposition 3.1.

(1) (IF): Suppose CDCP terminates. Then

$$(\exists n \in \mathbb{N}^+)(\forall i \in I)\, L_i^n = L_i^{n+1}$$

Clearly, for each $i \in I$ we have $L_i^n = L_i^{m(2n+1)}$. Since

$$(\forall i \in I)\, L_i^{m(2n+1)} \overset{(b)}{\subseteq} \mathrm{M}_i^{2n+1} \subseteq \mathrm{M}_i^{2n} \overset{(a)}{\subseteq} L_i^n$$

we have $\mathrm{M}_i^{2n} = \mathrm{M}_i^{2n+1}$ for each $i \in I$, which means the termination condition of CPLC holds.

(2) (ONLY IF): Suppose CPLC terminates. Then

$$(\exists n \in \mathbb{N}^+)(\forall i \in I)\, \mathrm{M}_i^n = \mathrm{M}_i^{n+1}$$

Clearly, for each $i \in I$ we have $\mathrm{M}_i^n = \mathrm{M}_i^{2(mn+1)}$. Since

$$(\forall i \in I)\, \mathrm{M}_i^{2(mn+1)} \overset{(a)}{\subseteq} L_i^{mn+1} \subseteq L_i^{mn} \overset{(b)}{\subseteq} \mathrm{M}_i^n$$

we have $L_i^{mn} = L_i^{mn+1}$ for each $i \in I$, which means the termination condition of CDCP also holds. By (1) and (2), the proposition is true. ■

If we can show that the termination of CDCP is undecidable when all initial languages in $\mathcal{L}$ are regular, then by Proposition 3.1, so is the termination of CPLC. We shall now establish the undecidability of the termination of CDCP..

B. Undecidability of Termination of CDCP and CPLC

We shall reduce the halting problem for an arbitrary deterministic Turing Machine (TM) **M** to that of checking termination of a three-node, regular-language instance of CDCP. The idea of the proof is that one of the three regular initial languages will essentially encode the structure of the transition relation on Turing machine quintuples. This language will be based on two disjoint alphabets – one for the encoding of predecessor quintuples, and the other for the encoding of their successors. In fact, the language can be thought of as specifying the input-output relation of a finite state machine that, when presented at its input with the successive symbols of an encoding of a predecessor TM configuration over the first alphabet, successively outputs the symbols of the corresponding successor configuration, encoded over the second alphabet. The other two regular languages are chosen so that, under CDCP, they implement a 'feedback loop' that returns the successor configuration from the output of the finite-state machine to the input, where it is treated as a new predecessor configuration: to do so, one node transliterates, symbol by symbol, successor configurations into a third, intermediate alphabet, disjoint from the other two; the remaining node transliterates from this intermediate alphabet into the input alphabet of the finite-state machine. The result is that as the execution of CDCP proceeds, the network simulates the action of the Turing machine on the empty input, and the computation terminates if and only if the Turing machine halts.

The key to making the construction with regular languages is that pairs of encodings of configurations, whether of a predecessor and a successor configuration, or of the same configuration over two disjoint alphabets, are interleaved. In machine-theoretic terms, this means that strings can be processed a bounded number of symbols at a time, without the need to store arbitrarily large encodings of configurations.

Given a deterministic Turing machine **M** let Σ be the *input alphabet*, $\Sigma \cup \{\sqcup\}$ with $\sqcup \notin \Sigma$ the *tape alphabet*, where $\sqcup$ denotes the blank symbol, and Q the *state set*. To represent the transitions of **M** we have a finite set of Turing *quintuples* $\mathcal{T} \subseteq Q \times (\Sigma \cup \{\sqcup\}) \times (\Sigma \cup \{\sqcup\}) \times \{R, L, *, \} \times Q$. Each element (q_i, a, b, m, q_j) means that the current state is q_i, the tape head is on a cell containing the symbol a and the next state is q_j; as part of the transition, the cell contents a are replaced by b; if $m = R$ then the tape head moves to the adjacent cell to the right after the transition; if $m = L$ then the tape head moves towards the left; and if $m = *$ then the tape head doesn't move. No two different quintuples share the same first two elements, which means that the next state q_j is uniquely determined by the current

state q_i and the current tape cell content a.

Each input of $\mathbf{M}$ is a finite string $w \in \Sigma^*$ not containing the blank symbol. A *configuration* of $\mathbf{M}$ is an three-tuple (u, q, av), meaning that the current tape contents are uav, the tape head is pointing to a and the current state is q. We only consider a left-bounded tape. So $u \in (\Sigma \cup \{\sqcup\})^*$ left of the tape head is finite, but $av \in (\Sigma \cup \{\sqcup\})^\omega$ is infinite because the part of the tape to the right of the tape head is infinite. It is convenient to arrange for the blank symbol to act as a delimiter of the tape contents, in the sense that any symbol to the right of a blank symbol is itself the blank symbol. To this end, we modify $\mathbf{M}$ as follows: (1) Add a new event μ to Σ to get a new alphabet $\Sigma' := \Sigma \cup \{\mu\}$; (2) Replace each quintuple $(q, a, \sqcup, m, q')$ with (q, a, μ, m, q'); (3) For each quintuple $(q, \sqcup, b, m, q')$ add a new one, (q, μ, b, m, q'), without replacing $(q, \sqcup, b, m, q')$.

Suppose the new Turing machine is $\mathbf{M}'$. Then $\mathbf{M}$ halts on $w \in \Sigma^*$ if and only if $\mathbf{M}'$ halts on w. Since halting of $\mathbf{M}$ on w is undecidable, halting of $\mathbf{M}'$ on w is also undecidable. The set of configurations of $\mathbf{M}'$ can be encoded as a regular language $\Sigma'^* Q \Sigma'^* \sqcup$, where the encoding of each configuration ends with $\sqcup$. A string $uqav\sqcup$ is interpreted as meaning that the current state is q, the tape head is pointed to a and the tape contents are $uav\sqcup$; if av is the empty string, $uq\sqcup$ means the tape head sits on the first cell containing $\sqcup$. Henceforth, we use $\mathbf{M}$ to mean the corresponding $\mathbf{M}'$ (therefore Σ means Σ' which contains the special event μ). The new set of configurations is encoded as $C := \Sigma^* Q \Sigma^* \sqcup$. Two configurations $u_1 a_1 q_1 b_1 v_1$, $u_2 a_2 q_2 b_2 v_2$ are *consecutive* if $u_2 a_2 q_2 b_2 v_2$ is any one of the following strings:

1) $u_2 a_2 q_2 b_2 v_2 = u_1 q_2 a_1 b' v_1$ if $(q_1, b_1, b', L, q_2) \in \mathcal{T}$
2) $u_2 a_2 q_2 b_2 v_2 = u_1 a_1 b' q_2 v_1$ if $(q_1, b_1, b', R, q_2) \in \mathcal{T}$
3) $u_2 a_2 q_2 b_2 v_2 = u_1 a_1 q_2 b' v_1$ if $(q_1, b_1, b', *, q_2) \in \mathcal{T}$

$uaqbcv \in C$ is called a *halting configuration* if there is no quintuple $(q_i, x, y, m, q_j) \in \mathcal{T}$ such that $q_i = q$ and $x = b$.

Proposition 3.2: Termination of CPLC is undecidable.
Proof: Consider an arbitrary deterministic Turing machine $\mathbf{M}$ of the form described above. We first construct a language encoding all pairs of consecutive configurations. Let $\hat{\Sigma}$ be a new alphabet such that there is a bijection

$$f : \Sigma \cup \{\sqcup\} \to \hat{\Sigma} \cup \{\hat{\sqcup}\} : \sigma \mapsto f(\sigma) := \hat{\sigma}$$

Let $\hat{Q}$ be a new state set such that there is a bijection

$$g : Q \to \hat{Q} : q \mapsto g(q) := \hat{q}$$

In a configuration $uaqbv$ the *critical information* is the state q, the contents of the cell pointed to by the tape head and the position of the tape head. Two consecutive configurations are different from each other only on the critical information. Let

$$\begin{aligned} T_s &:= \{qb\hat{q'}\hat{b'} | (q, b, f^{-1}(\hat{b'}), *, g^{-1}(\hat{q'})) \in \mathcal{T}\} \\ T_s^{\sqcup} &:= \{q \sqcup \hat{q'}\hat{b'}\hat{\sqcup} | (q, \sqcup, f^{-1}(\hat{b'}), *, g^{-1}(\hat{q'})) \in \mathcal{T}\} \end{aligned}$$

T_s encodes all possible changes of critical information after a transition when the tape head does not sit on the leftmost blank cell and remains stationary. For example, $qb\hat{q'}\hat{b'}$ denotes that the Turing machine makes a transition from state q to state q', meanwhile replacing b with b' without moving the tape head. We use $\hat{q'}$ and $\hat{b'}$ instead of q' and b' only for the transliteration purpose. $T_s^{\sqcup}$ encodes changes of critical information when the tape head sits on the leftmost blank cell and remains stationary after a transition. Similarly, we can encode changes of critical information when the tape head moves left or right; or the Turing machine halts:

$$\begin{aligned} T_l &:= \{aqb\hat{q'}\hat{a}\hat{b'} | (q, b, f^{-1}(\hat{b'}), L, g^{-1}(\hat{q'})) \in \mathcal{T}\} \\ T_l^{\sqcup} &:= \{aq \sqcup \hat{q'}\hat{a}\hat{b'} | (q, \sqcup, f^{-1}(\hat{b'}), L, g^{-1}(\hat{q'})) \in \mathcal{T}\} \\ T_r &:= \{qb\hat{b'}\hat{q'} | (q, b, f^{-1}(\hat{b'}), R, g^{-1}(\hat{q'})) \in \mathcal{T}\} \\ T_r^{\sqcup} &:= \{q \sqcup \hat{b'}\hat{q'}\hat{\sqcup} | (q, \sqcup, f^{-1}(\hat{b'}), R, g^{-1}(\hat{q'})) \in \mathcal{T}\} \\ T_h &:= \{qb | (\nexists (q_i, x, y, m, q_j) \in \mathcal{T})\, q_i = q \,\&\, x = b\} \\ T_h^{\sqcup} &:= \{q \sqcup | (\nexists (q_i, x, y, m, q_j) \in \mathcal{T})\, q_i = q \,\&\, x = \sqcup\} \end{aligned}$$

Let

$$A := \{\sigma\hat{\sigma} \in \Sigma\hat{\Sigma} | \sigma \in \Sigma \,\&\, \hat{\sigma} = f(\sigma)\}$$

Define the regular template language

$$\begin{aligned} LT_{\mathbf{M}} := \; & A^*(T_s + T_l + T_r)A^* \sqcup \hat{\sqcup} + A^*(T_s^{\sqcup} + T_l^{\sqcup} + T_r^{\sqcup}) \\ & + \Sigma^* T_h \Sigma^* \sqcup + \Sigma^* T_h^{\sqcup} \end{aligned}$$

where LT stands for the *Language of Transitions*.

Notation: Given two sets X and Y with a bijection $\phi : X \mapsto Y$, we use $X \to_\phi Y$ to mean that each element $x \in X$ is replaced by $\phi(x) \in Y$. Therefore, given a string s we use $s(X \to_\phi Y)$ to mean a new string s' created by replacing each symbol x in s with $\phi(x)$. For a set of strings W we use $W(X \to_\phi Y)$ to mean a new set of strings created by performing the replacement operation on each string $s \in W$. We use $\hat{c}$ to mean $c(\Sigma \cup \{\sqcup\} \to_f \hat{\Sigma} \cup \{\hat{\sqcup}\}; Q \to_g \hat{Q})$. Let $\psi : C \cup \{\epsilon\} \to C \cup \{\epsilon\}$ with

$$x \mapsto \psi(x) := \begin{cases} \text{next configuration} & \text{if } x \text{ not halts} \\ \epsilon & \text{if } x = \epsilon \text{ or } x \text{ halts} \end{cases}$$

Then for each $c \in C$ and $k \in \mathbb{N}^+$, $\psi^k(c)$ is interpreted as the kth consecutive configuration of c if no halting configuration appears. Given two alphabets Σ, Σ' let $P_{\Sigma,\Sigma'} : \Sigma \to (\Sigma \cap \Sigma')^*$ be the natural projection.

Let $\Gamma := \Sigma \cup \{\sqcup\} \cup Q$ and $\hat{\Gamma} := \hat{\Sigma} \cup \{\hat{\sqcup}\} \cup \hat{Q}$. We first make some claims about $LT_{\mathbf{M}}$:

Claim 1: "$LT_{\mathbf{M}}$ encodes all configurations of C", i.e.

$$\mathrm{P}_{\Gamma \cup \hat{\Gamma}, \Gamma}(LT_{\mathbf{M}}) = \Sigma^* Q \Sigma^* \sqcup$$

Claim 2: "$LT_{\mathbf{M}}$ encodes all pairs of consecutive configurations", i.e.

$$(\forall c \in C)(\exists s \in LT_{\mathbf{M}})\, \mathrm{P}_{\Gamma \cup \hat{\Gamma}, \Gamma}(s) = c \,\&\, \mathrm{P}_{\Gamma \cup \hat{\Gamma}, \hat{\Gamma}}(s) = \widehat{\psi(c)}$$

Claim 3: "the next configuration is uniquely determined by the current configuration", i.e.

$$(\forall s, s' \in LT_{\mathbf{M}})\, s = s' \iff \mathrm{P}_{\Gamma\cup\hat{\Gamma},\Gamma}(s) = \mathrm{P}_{\Gamma\cup\hat{\Gamma},\Gamma}(s')$$

Let $\tilde{\Sigma}$ be a third alphabet such that there is a bijection

$$h : \Sigma \cup \{\sqcup\} \to \tilde{\Sigma} \cup \{\tilde{\sqcup}\} : \sigma \mapsto h(\sigma) := \tilde{\sigma}$$

and $\tilde{Q}$ a third state set such that there is a bijection

$$p : Q \to \tilde{Q} : q \mapsto p(q) := \tilde{q}$$

Let

$$\begin{aligned} B &:= \{\sigma\tilde{\sigma} | \sigma \in \Sigma \,\&\, \tilde{\sigma} = h(\sigma)\} \\ D &:= \{qb\tilde{q}\tilde{b} | b \in \Sigma \,\&\, q \in Q \,\&\, \tilde{b} = h(b) \,\&\, \tilde{q} = p(q)\} \\ F &:= \{q \sqcup \tilde{q}\tilde{\sqcup} | q \in Q \,\&\, \tilde{q} = p(q)\} \end{aligned}$$

We construct another regular template language as follows:

$$LL_{\mathbf{M}} := B^* D B^* \sqcup \tilde{\sqcup} + B^* F$$

where LL stands for the *Language of transLiteration*. Let $\tilde{\Gamma} := \tilde{\Sigma} \cup \{\tilde{\sqcup}\} \cup \tilde{Q}$. Then it is not difficult to show that

$$\mathrm{P}_{\Gamma\cup\tilde{\Gamma},\Gamma}(LL_{\mathbf{M}}) = C$$

In other words, $LL_{\mathbf{M}}$ simply attaches to each encoding $c \in C$ of a configuration of $\mathbf{M}$ a copy $\tilde{c}$ which is obtained by replacement operation $c(\Sigma \cup \{\sqcup\} \to_h \tilde{\Sigma} \cup \{\tilde{\sqcup}\}; Q \to_p \tilde{Q})$. Now we are ready to build a finite network of regular languages to simulate the run of $\mathbf{M}$ on a given input $w \in \Sigma^*$.

We construct three nodes L_1, L_2 and L_3 based on the given $(\mathbf{M}, \Sigma\cup\{\sqcup\}, w \in \Sigma^*)$. Let $\Sigma_1 := \Gamma\cup\hat{\Gamma}$ be the alphabet for L_1, $\Sigma_2 := \Gamma\cup\tilde{\Gamma}$ for L_2 and $\Sigma_3 := \hat{\Gamma}\cup\tilde{\Gamma}$ for L_3. The initial configuration is $q_0 w\sqcup$, which means that the initial state is q_0, the tape content is $w\sqcup$ and the tape head is located at the left-hand cell (i.e. the first left symbol of w). For the given initial configuration $q_0 w\sqcup \in C$, by Claims 1,3 there exists a unique string $s_0 \in LT_{\mathbf{M}} \subseteq \Sigma_1^*$ such that $\mathrm{P}_{\Sigma_1,\Gamma}(s_0) = q_0 w\sqcup$. Let

$$\begin{aligned} N_1 &:= LT_{\mathbf{M}} \subseteq \Sigma_1^* \\ N_2 &:= LL_{\mathbf{M}} \subseteq \Sigma_2^* \\ W_3 &:= LL_{\mathbf{M}}(\Sigma \cup \{\sqcup\} \to_f \hat{\Sigma} \cup \{\hat{\sqcup}\}; Q \to_g \hat{Q}) \\ &\quad \cap \mathrm{P}^{-1}_{\Sigma_3,\Sigma_1}(\mathrm{P}_{\Sigma_1,\Sigma_3}(N_1)) \subseteq \Sigma_3^* \\ N_3 &:= W_3 \cup (\mathrm{P}_{\Sigma_2,\Sigma_3}(N_2) - \mathrm{P}_{\Sigma_3,\Sigma_2}(W_3)) \subseteq \Sigma_3^* \end{aligned}$$

For each configuration $c \in C$ we use $\tilde{c}$ to denote $c(\Sigma \cup \{\sqcup\} \to_h \tilde{\Sigma} \cup \{\tilde{\sqcup}\}; Q \to_p \tilde{Q})$; By Claim 2 we have that, for each string $s \in N_1$, if we let $c := P_{\Sigma_1,\Gamma}(s)$ then $P_{\Sigma_1,\hat{\Gamma}}(s) = \widehat{\psi(c)}$. Furthermore, by Claim 3, s is uniquely determined by c.

Now we define $L_2 := N_2^*$, $L_3 := N_3^*$, and let

$$L_1 := \mathrm{P}_{\Sigma_1,\hat{\Gamma}}(s_0) N_1^*$$

Figure 2 depicts the network structure of $\{L_1, L_2, L_3\}$. After applying CDCP on $\{L_1, L_2, L_3\}$, we can show that

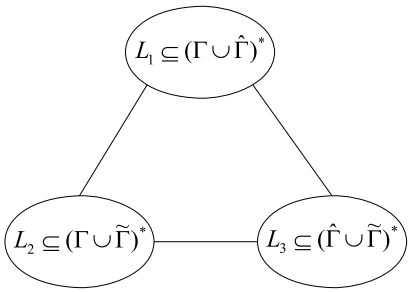

Fig. 2. Graph Gr of $\{L_1, L_2, L_3\}$

the Turing machine $\mathbf{M}$ halts on the initial input w if and only if CDCP terminates on the initial set $\{L_1, L_2, L_3\}$. Since the (revised) Turing machine halting problem is undecidable, the termination of CDCP is also undecidable, as required. ■

Corollary 3.1: The termination of CPLC is undecidable.
Proof: Follows directly from Propositions 3.1 and 3.2. ■

Corollary 3.2: It is undecidable whether the supremal local support corresponding to an arbitrary collection of regular initial languages is componentwise empty.
Proof: Let L_1, L_2 and L_3 be constructed as in Proposition 3.2. If Turing machine $\mathbf{M}$ halts on the given input w, then CDCP terminates at a finite round k such that (by Proposition 3.2) $L_1^k \neq \varnothing, L_2^k \neq \varnothing, L_3^k \neq \varnothing$. By Proposition 2.10 in [11], $\{L_1^k, L_2^k, L_3^k\}$ is the supremal local support of $\{L_1, L_2, L_3\}$. If $\mathbf{M}$ does not halt then there is a monotonically decreasing sequence of regular languages for every local component. Therefore, the set-theoretic limit for each sequence exists. Fix $i \in I$. If the Turing machine does not halt, then by the proof of Proposition 3.2, the minimum length of strings in L_i^k is a strictly increasing function of k. But the languages L_i^k contain only finite-length strings, so the intersection of the infinite sequence of L_i^k must be empty. Thus the limit language is empty. Let $\{E_i \subseteq L_i | i = 1, 2, 3\}$ be the supremal local support. Then by the proof of Lemma 2.3 in [11] we have

$$(\forall k \in \mathbb{N})(\forall i = 1, 2, 3)\, E_i \subseteq L_i^k$$

Therefore, when $\mathbf{M}$ does not terminate, the collection of empty limit sets is the supremal local support. Thus, $\mathbf{M}$ terminates if and only if the supremal local support is not componentwise empty. Since the halting of $\mathbf{M}$ is undecidable, the corollary is true. ■

Any computational procedure must employ some form of finite representation of the supremal local support; for example, each element in the local support may be represented by a formal grammar, a Petri net, or a finite-state automaton. Not every representation allows effective testing of language emptiness in every instance; for example the Turing machine emptiness problem is undecidable. A representation of the supremal local support is *effective*

if it admits a procedure for checking the emptiness of component languages that terminates for any component languages that are regular.

Corollary 3.3: There exists no algorithm that computes, for every finite collection of regular initial languages, an effective representation of the corresponding supremal local support.

Proof: Notice that in the proof of Proposition 3.2, no matter whether **M** halts or not, the supremal local support of $\mathcal{L} = \{L_1, L_2, L_3\}$ is componentwise regular. If the corollary does not hold, then there exists an algorithm that computes an effective representation of $\mathcal{L}$ which permits the testing of its componentwise emptiness. By the proof of Corollary 3.2, we can then decide whether **M** halts, contradicting the fact that the halting of **M** is undecidable. Therefore the corollary must be true. ■

IV. Conclusion

In this paper we showed that the termination of CPLC is undecidable even if initial languages $\mathcal{L}$ are required to be regular (and, by extension, if they are merely required to belong to some larger language class). Consequently, in practical applications we must either focus on a restricted class of systems, whose structure guarantees the termination of CPLC [11], or achieve an approximate result by stopping CPLC without convergence. For the latter case, the quantification of the tradeoff between computational effort and accuracy is an interesting and challenging problem. Future reports will address the former problem by providing new sufficient conditions for termination. Finally, we also showed that there is no algorithm that computes, for every finite collection of regular initial languages, a representation of the supremal local support that allows effective emptiness-checking of any regular component languages.

References

[1] A.P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[2] E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Dynamic Systems*, 15(1):33–84, March 2005.

[3] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall Inc., Englewood, New Jersey, 1981.

[4] A.K. Mackworth. Constraint satisfaction. *In S. C. Shapiro, editor, Encyclopedia of Artificial Intelligence, 2nd Edition, John Wiley & Sons*, pages 285–293, 1992.

[5] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[6] K.P. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: an empirical study. In *Proc. 15th Conference on Uncertainty in Artificial Intelligence*, pages 467–475, Stockholm, Sweden, 1999.

[7] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[8] R. Su and W.M. Wonham. A model of component consistency in distributed diagnosis. In *Proc. 2004 IFAC Workshop on Discrete Event Systems (WODES'04)*, pages 427–432, Reims, France, September 2004.

[9] R. Su and W.M. Wonham. Undecidability of termination of cplc for computing supremal local support. In *Proc. 42nd Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 2004.

[10] R. Su, W.M. Wonham, J. Kurien, and X. Koutsoukos. Distributed diagnosis for qualitative systems. In *Proc. 6th International Workshop on Discrete Event Systems (WODES'02)*, pages 169–174, Zaragoza, Spain, October 2002.

[11] Rong Su. *Distributed Diagnosis for Discrete-Event Systems*. PhD Thesis, ECE Dept., Univ. of Toronto, URL:www.control.utoronto.ca/∼surong/SR.zip, 2004.

[12] Rainer Weigel and Boi Faltings. Compiling constraint satisfaction problems. *Artificial Intelligence*, 115(2):257–287, 1999.

[13] W. M. Wonham. *Supervisory Control of Discrete-Event Systems*. Systems Control Group, Dept. of ECE, University of Toronto. URL: www.control.utoronto.ca/DES, 2004.

[14] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.