Goal-Directed Context Validation for Adaptive Ubiquitous Systems[1]

Chang Xu* Shing-Chi Cheung* Wing-Kwong Chan#
* Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China
# Department of Computer Science, City University of Hong Kong, Kowloon Tong, Kowloon, Hong Kong, China
{changxu, scc}@cse.ust.hk, wkchan@cs.cityu.edu.hk

ABSTRACT
Ubiquitous systems adaptive to their dynamic environments find their roles useful in many modern applications. Their adaptability, however, can be unexpectedly impaired if the environments are incorrectly perceived. Our earlier work has studied evaluating the constraints that govern the consistency of perceived environmental information (or *contexts*). The work generates links as the evaluation result for explaining how the constraints are satisfied or violated. However, the link generation technique may generate many redundant links, which would consume large unnecessary time and space. To address the problem, we in this paper present a goal-directed technique to improve our earlier link generation technique. We evaluate the technique analytically, and show how the improved technique helps reduce the number of generated redundant links in context validation.

1. INTRODUCTION
Adaptive ubiquitous systems are receiving increasing research and industrial attention with the advancement of radio frequency, sensor and wireless communication technologies. These systems typically perceive their computing environments continually and adapt their behavior accordingly. A system's *contexts* refer to the pieces of information that characterize its running environment. Examples of context include network condition, geographical location, temperature and user identity. A system is *context-aware* if its behavior is adaptive to its context changes. For example, a smart phone controlled by context-aware software would only vibrate and keep silent in a concert hall, but beep loudly during a football match to alert its user.

Our previous work [20] has shown the need for context validation for adaptive ubiquitous systems. Otherwise, the use of inconsistent contexts (e.g., obsolete, corrupted or inaccurate contexts) can easily lead to anomalous behavior of adaptive ubiquitous systems. For example, the aforementioned context-aware smart phone may roar during the most important moment of a wedding ceremony if it mistakes the current situation as a football match. In those situations where a precise oracle for judging the validity of each context is generally unavailable or too costly to apply, specifying consistency constraints on contexts and checking them for possible inconsistency are a feasible alternative. Let us explain this approach using an example.

To detect inconsistent contexts for a location-aware application Call Forwarding [17], one may specify a constraint such as "*nobody can be in two different places at the same time*". Such a constraint can be expressed in a First Order Logic (FOL) based language like: $\forall \gamma_1 \in \mathsf{LOC}$ (not ($\exists \gamma_2 \in \mathsf{LOC}$ (difPlc(γ_1, γ_2)))), where LOC refers to a set of contexts, each of which represents an object's location collected within a certain period of time. Let difPlc be the function that accepts two location contexts as input and returns true if the two contexts refer to the same object (e.g., the same person) appearing at different locations (e.g., two different rooms) simultaneously. Suppose that l_1 and l_2 are two location contexts in the LOC set that satisfy the difPlc function (condition). The above constraint would be violated when $\gamma_1 = l_1$ and $\gamma_2 = l_2$. As such, a link associated with this constraint would be generated to indicate how this violation has occurred: (violated, {(γ_1, l_1), (γ_2, l_2)}). This information is named a *link*, because it indicates whether a specific variable assignment that binds concerned variables to their corresponding context values would cause satisfaction or violation to this constraint. In this example, the variable assignment in the link exactly causes a violation to the constraint, and thus the two location contexts are *inconsistent* with respect to the constraint.

The original definition of *link* comes from the xlinkit work [12][13], which describes how combinations of elements (e.g., contexts in pervasive computing) fulfill a constraint or breach it. Our previous work [20] proposed to construct such links incrementally and efficiently. However, either approach can generate *redundant* links in evaluating a consistency constraint's sub-formulae. Such links do not contribute to the calculation of the entire constraint's satisfaction or violation. In Section 2, we will illustrate the generation of redundant links using two examples. In the two examples, up to 80% generated links could be redundant. This percentage value is significant. The constraints studied in the two examples were derived from a constraint survey we conducted. In the survey, 30 participants were asked to specify consistency constraints for three ubiquitous applications published in the literature, which included the Call Forwarding application discussed earlier. Over 42% of the specified consistency constraints (or *constraints* for short) are either identical or close to the constraints we studied in the two examples. They are all subject to the generation of redundant links.

Therefore, we present in this paper a goal-directed context validation technique to address the problem. The technique is evaluated in terms of *used link rate* and *effective link rate*. Both rates are improved from 70.8% to 91.7% and from 33.3% to 100%, respectively, as compared to the original context validation technique without our goal direction. The results suggest that our proposed new goal-directed technique can effectively reduce generating redundant links in the context validation for ubiquitous systems.

The rest of this paper is organized as follows. Section 2 introduces background knowledge about link generation for context validation and presents two motivating examples. Section 3 proposes our goal-directed context validation technique for reducing the generation of redundant links. Section 4 analyzes the properties of our

[1] Crafted version: Made in Feb 2017 (originally published at SEAMS in May 2007). Contact: Chang Xu (changxu@nju.edu.cn).

$\mathcal{L}[\forall \gamma \in s\ (f)]_\alpha =$

$\{\ l \mid l \in (\mathsf{violated}, \{(\gamma, x)\}) \otimes \mathcal{L}[f]_{\mathsf{bind}((\gamma, x), \alpha)} \wedge x \in s$

$\wedge\ \mathcal{T}[f]_{\mathsf{bind}((\gamma, x), \alpha)} = \bot\}.$

$\mathcal{L}[(f_1)\ \mathsf{and}\ (f_2)]_\alpha =$

(1) $\mathcal{L}[f_1]_\alpha \underline{\otimes}\ \mathcal{L}[f_2]_\alpha$,
if $\mathcal{T}[f_1]_\alpha = \mathcal{T}[f_2]_\alpha = \top$;

(2) $\mathcal{L}[f_1]_\alpha \cup \mathcal{L}[f_2]_\alpha$,
if $\mathcal{T}[f_1]_\alpha = \mathcal{T}[f_2]_\alpha = \bot$;

(3) $\mathcal{L}[f_2]_\alpha$,
if $\mathcal{T}[f_1]_\alpha = \top$ and $\mathcal{T}[f_2]_\alpha = \bot$;

(4) $\mathcal{L}[f_1]_\alpha$,
if $\mathcal{T}[f_1]_\alpha = \bot$ and $\mathcal{T}[f_2]_\alpha = \top$.

Figure 1. Link generation semantics for universal and and formulae.

technique and evaluates its effectiveness. Finally, Section 5 discusses related work in recent years and Section 6 concludes this paper.

2. BACKGROUND

In this section, we introduce link generation for context validation and discuss two examples showing the generation of redundant links.

2.1 Link Generation for Context Validation

We present an overview of our previous work [20] on context validation for ubiquitous systems.

Links connect multiple contexts (e.g., location contexts in the Call Forwarding application) that satisfy or violate specified consistency constraints. There are two types of link: *satisfaction link* and *violation link*. They are represented as (satisfied, *assignment*) and (violated, *assignment*), respectively. Satisfied and violated are two keywords indicating whether a concerned constraint is satisfied or violated. *Assignment* is a variable assignment, which contains mappings between variables defined in the constraint and contexts bound to these variables. Under this particular variable assignment, the concerned constraint is satisfied or violated according to the type of this link. One example is the violation link (violated, $\{(\gamma_1, l_1), (\gamma_2, l_2)\}$) we discussed earlier in Section 1. This violation link indicates that its associated constraint has been violated due to the context pair l_1 and l_2, when they are assigned to variables γ_1 and γ_2, respectively.

We then introduce how to generate links for a universal and an and formulae as example. Other formula types (e.g., existential, or, implies and not formulae) are discussed in our technical report [19]. We omit them here for simplicity.

Figure 1 gives our link generation semantics for a universal and an and formulae. Given a formula, function $\mathcal{L}$ generates links for this formula under the given variable assignment α. In the semantics, $\top$ means true and $\bot$ means false. Function bind adds a new mapping of variable and context into an existing variable assignment, and returns the updated variable assignment as output. This function is used when one evaluates a universal formula's sub-formula. The interpretation for operators such as $\otimes$ and $\underline{\otimes}$ is not essential to understanding our later explanation about redundant link generation. One only needs to know that they are algebraic operators on links. Informally, the $\otimes$ operator is used to construct links for a universal formula with the current variable assignment for this formula and the links returned from its sub-formulae (multiple sub-formula instances with different variable assignments). The $\underline{\otimes}$ operator is used to construct links for an and formula with links returned from its two sub-formulae (first and second sub-formulae).

From the semantics in Figure 1, one can observe that the links generated for a universal formula relate only to the links returned from those of its sub-formulae when such sub-formulae are evaluated to false. Here, $\mathcal{T}$ is a truth value evaluation function, which works similarly as the $\mathcal{L}$ function. For an and formula, it is a little bit more complicated. One can observe that the links generated for an and formula relate to the links returned from both of its sub-formulae only when the two sub-formulae have the same truth value, either true or false (see Case (1) and Case (2)). Otherwise, only the links from one sub-formula are selected and thus relate to the links generated for the and formula (see Case (3) and Case (4)). This observation is useful for our later proposed goal-directed context validation technique for reducing redundant link generation, as we observe that some links would never be used after their generation.

For better understanding, we explain more about the link generation semantics for the two formulae types, i.e., universal and and formulae.

Given a universal formula, the link generation semantics examines all possible variable assignments for its sub-formula to see whether any one of them can cause violation to the sub-formula, and records such variable assignments in terms of links. This is because a variable assignment that makes a universal formula's sub-formula violated also explains why this universal formula itself is violated. Given an and formula, the link generation semantics partitions all possibilities into four cases and constructs links accordingly. For example, in Case (3) the and formula's first sub-formula f_1 is evaluated to true and its second sub-formula f_2 to false, then the and formula is evaluated to false due to and only due to f_2 (violated). Therefore, the links generated for, and thus returned from, f_2, which explain why f_2 is evaluated to false, also explain why the and formula itself is evaluated to false. We omit the discussion of the other three cases for simplicity.

A full treatment of all formula types and their examples can be found in our technical report [19].

2.2 Revisiting Link Generation Semantics

The link generation semantics in Figure 1 are not optimal in the sense that they may generate redundant links, which are never used for explaining why constraints are satisfied or violated. Let us illustrate redundant link generation using two examples.

From the point of view of users, there are two ways of specifying constraints. One is the *necessary condition representation*, which is from the validation perspective and most common form [16]. One example is the consistency constraint we discussed earlier in Section 1: $\forall \gamma_1 \in \mathsf{LOC}$ (not ($\exists \gamma_2 \in \mathsf{LOC}$ (difPlc(γ_1, γ_2)))). For this constraint, we are interested in its violation, because any violation indicates that the necessary condition specified by this constraint does not hold, and that thus the contexts of concerned adaptive systems are no longer valid. On the other hand, one can also use the *sufficient condition representation* to specify this constraint in another way: $\exists \gamma_1 \in \mathsf{LOC}$ ($\exists \gamma_2 \in \mathsf{LOC}$ (difPlc(γ_1, γ_2))). For this representation, one would be interested in its satisfaction, because this representation exactly specifies a scenario where contexts are consid-

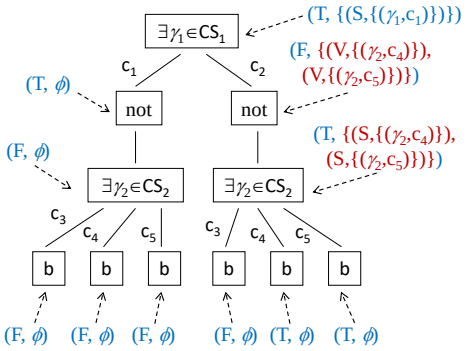

Figure 2. Example 1: satisfaction links are expected.

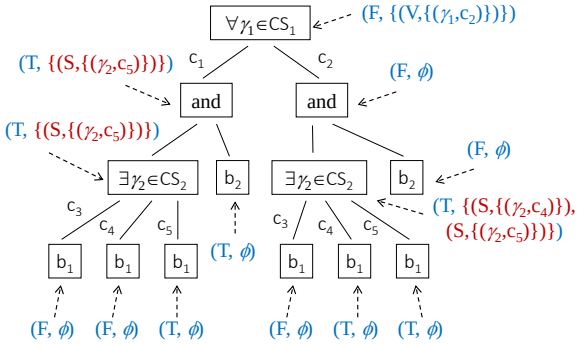

Figure 3. Example 2: violation links are expected.

ered as invalid. We note that both representations are useful because users may have different preferences in specifying consistency constraints. Since different representations imply different interests to users (e.g., expecting violation or satisfaction of a consistency constraint), some links generated by the link generation semantics in Figure 1 may be redundant.

Figure 2 illustrates how links are generated for an example consistency constraint: $\exists\gamma_1\in CS_1$ (not ($\exists\gamma_2\in CS_2$ (b(γ_1, γ_2)))), in which CS_1 and CS_2 are two context sets and b is a Boolean function. Their semantics do not essentially relate to our discussed problem and therefore are omitted. Suppose that $CS_1 = \{c_1, c_2\}$ and $CS_2 = \{c_3, c_4, c_5\}$, and all c_1, c_2, c_3, c_4 and c_5 are contexts. According to the semantics of this constraint, we enumerate all possible variable assignments for function b. Suppose that the function returns true for b(c_2, c_4) and b(c_2, c_5), and false for the other four variable assignments. Figure 2 illustrates this value information, where T represents true and F represents false. Besides, the information on generated links is also annotated besides the T/F information in Figure 2: S represents satisfied, V represents violated, and ϕ represents that no link is generated for a particular sub-formula node. For example, the information shown besides the rightmost b node at the bottom level is "(T, ϕ)", which means that this b node (with variable assignment {(γ_1, c_2), (γ_2, c_5)}) is evaluated to true and no link is generated for this node. The information shown besides the root node "$\exists\gamma_1\in CS_1$" is "(T, {(S, {(γ_1, c_1)})})", which means that the root node is evaluated to true and a satisfaction link is generated for this node: (satisfied, {(γ_1, c_1)}). This link indicates that the consistency constraint is satisfied due to the particular variable assignment of $\gamma_1 = c_1$. In other words, other variable assignments (e.g., $\gamma_1 = c_2$, $\gamma_2 = c_3$, $\gamma_2 = c_4$, …) are immaterial to the satisfaction of this constraint.

Suppose that our goal is to find satisfaction links for a given existential constraint, i.e., we use a sufficient condition representation to specify the constraint. While we can obtain satisfaction links at the root node like "$\exists\gamma_1\in CS_1$" as illustrated in Figure 2, some links generated during the tree traversal in context validation, e.g., {(violated, {(γ_2, c_4)}), (violated, {(γ_2, c_5)})} from the right not node and {(satisfied, {(γ_2, c_4)}), (satisfied, {(γ_2, c_5)})} from the right "$\exists\gamma_2\in CS_2$" node, are indeed redundant. This is because the final satisfaction link set {(satisfied, {(γ_1, c_1)})} generated for the root node essentially does not use any information from these two link sets. A closer study shows that link generation for formula "$\exists\gamma_1\in CS_1$ (…)" cares only about satisfaction links from its sub-formula, but the sub-formula not node on the right branch in Figure 2 happens to generate violation links only, which are not expected by the "$\exists\gamma_1\in CS_1$ (…)" root node. Therefore, such generated violation links become redundant.

f ::= $\forall\gamma\in s$ (*f*) | // Universal formula
$\exists\gamma\in s$ (*f*) | // Existential formula
(*f*) and (*f*) | // and formula
(*f*) or (*f*) | // or formula
(*f*) implies (*f*) | // implies formula
not (*f*) | // not formula
b(γ_1, …, γ_n) // Terminal

Figure 4. Constraint language syntax.

Figure 3 illustrates another constraint checking example, whose goal is to find violation links for a given universal constraint, i.e., one uses the necessary condition representation to specify the consistency constraint. While we can obtain violation links at the root node, some links generated during the tree traversal in context validation, e.g., {(satisfied, {(γ_2, c_5)})} from the left and node and the left "$\exists\gamma_2\in CS_2$" node, {(satisfied, {(γ_2, c_4)}), (satisfied, {(γ_2, c_5)})} from the right "$\exists\gamma_2\in CS_2$" node, are redundant. This is because the final violation link set {(violated, {(γ_1, c_2)})} generated for the root node does not use any information from them at all. We do not elaborate on details due to similar explanations.

The preceding two examples suggest the inadequacy of existing link generation semantics, which essentially adopt the strategy of "generate and then select". For example, in Case (3) of the link generation semantics for and formula in Figure 1, the links generated for the first sub-formula f_1 are actually abandoned and thus wasted.

3. GOAL-DIRECTED CONTEXT VALIDATION

Our new approach is to replace the original link generation strategy of "generate and select" with that of "generate as required". We model the requirement of specific link generation as a *goal*, which specifies what type of link is expected for a given formula. Due to the fact that each constraint is recursively constructed using FOL formulae, the goal on a given constraint can be further divided into sub-goals on sub-formulae that construct this constraint.

Figure 4 gives our constraint language syntax, with which one specifies consistency constraints. The language syntax follows traditional FOL interpretations, where *f* represents a formula, *s* is a set that contains finite contexts, and *b* is a function that takes contexts as input and returns a truth value as output. Note that we are only interested in well-formed formulae, which contain no free variable. *Free variable* refers to a variable that is used without definition.

$\mathcal{G}[\forall \gamma \in s\ (f)]_\alpha$:

(1) $\mathcal{G}[f]_{\text{bind}((\gamma, x), \alpha)} = \bot \mid x \in s$, if $\mathcal{G}[\forall \gamma \in s\ (f)]_\alpha = \bot$;
(2) $\mathcal{G}[f]_{\text{bind}((\gamma, x), \alpha)} = \text{null} \mid x \in s$, otherwise.

$\mathcal{G}[\exists \gamma \in s\ (f)]_\alpha$:

(1) $\mathcal{G}[f]_{\text{bind}((\gamma, x), \alpha)} = \top \mid x \in s$, if $\mathcal{G}[\exists \gamma \in s\ (f)]_\alpha = \top$;
(2) $\mathcal{G}[f]_{\text{bind}((\gamma, x), \alpha)} = \text{null} \mid x \in s$, otherwise.

$\mathcal{G}[(f_1)\ \text{and}\ (f_2)]_\alpha$:

$\mathcal{G}[f_1]_\alpha = \mathcal{G}[f_2]_\alpha = \mathcal{G}[(f_1)\ \text{and}\ (f_2)]_\alpha$.

$\mathcal{G}[(f_1)\ \text{or}\ (f_2)]_\alpha$:

$\mathcal{G}[f_1]_\alpha = \mathcal{G}[f_2]_\alpha = \mathcal{G}[(f_1)\ \text{or}\ (f_2)]_\alpha$.

$\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$:

(1) $\mathcal{G}[f_1]_\alpha = \neg\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$,
$\mathcal{G}[f_2]_\alpha = \mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$,
if $\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha \neq \text{null}$;
(2) $\mathcal{G}[f_1]_\alpha = \mathcal{G}[f_2]_\alpha = \text{null}$, otherwise.

$\mathcal{G}[\text{not}\ (f)]_\alpha$:

(1) $\mathcal{G}[f]_\alpha = \neg\mathcal{G}[\text{not}\ (f)]_\alpha$, if $\mathcal{G}[\text{not}\ (f)]_\alpha \neq \text{null}$;
(2) $\mathcal{G}[f]_\alpha = \text{null}$, otherwise.

Figure 5. Goal semantics.

For example, variable γ_2 in formula "$\exists \gamma_1$ in s (f(γ_2))" is a free one, and therefore this formula is not a well-formed formula.

The recursive nature of the constraint language allows consistency constraints to be hierarchically structured as shown in Figure 2 and Figure 3. Based on such hierarchy, our new link generation would work as follows:

- Given a consistency constraint, the answer to question "what type of link is expected" decides the goal on this constraint itself.
- A top-down analysis, i.e., from each formula node to its sub-formula nodes, derives sub-goals on sub-formula nodes according to the goal on the constraint.
- A bottom-up process generates links according to sub-goals on sub-formula nodes in a post-order traversal until the root node, which represents the whole constraint.

The first step is straightforward because one can easily tell what type of link is expected from a consistency constraint's representation. We elaborate on the second and third steps in Sections 3.1 and 3.2, respectively.

3.1 Goal Semantics

To reduce the generation of redundant links, we need to know whether evaluating a particular sub-formula has to generate links and what type of link is actually expected. As such, we define the value set for a goal to be {$\top$, $\bot$, null}, where $\top$ means that a consistency constraint should generate satisfaction links, $\bot$ means the opposite (i.e., violation links), and null means that no link is expected (i.e., no need for generating any type of link).

Our goal semantics is based on the observation that some formulae can generate only one particular type of link, and that if that type of link is not expected, one can choose not to generate such links at all. For example, a universal formula generates only violation links, which explain why the formula has been violated (see the link generation semantics in Figure 1). Therefore, a goal of generating satisfaction links for a universal formula can be ignored for avoiding generating redundant (violation) links. We model this idea using the following goal semantics.

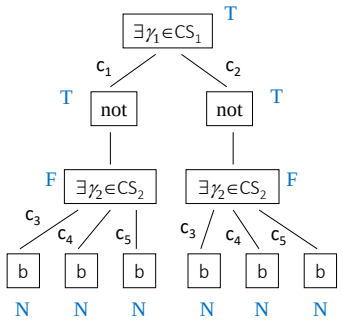

Figure 6. Derived sub-goals.

Let $\mathcal{G}$ be a goal function that accepts a formula and a variable assignment as input, and returns a goal for this formula. We consider six formula types (not all seven as in Figure 4), because they have at least one sub-formula for sub-goal derivation. We give our goal semantics in Figure 5.

Our goal semantics are straightforward. For example, for a universal formula, each of its sub-formula instances (a sub-formula node with a certain variable assignment as in Figure 2 and Figure 3) has a sub-goal of $\bot$, only if this formula's goal is $\bot$. Otherwise, the sub-goal of each sub-formula instance is set to null. This is because a universal formula can generate only violation links, as we discussed earlier, and a generated violation link explaining why a sub-formula instance is violated also explains why the universal formula itself is violated. For an implies formula "(f_1) implies (f_2)", if its goal is $\top$, then its first sub-formula f_1 has a goal of $\bot$ and the second sub-formula f_2 has a goal of $\top$. This is because a generated violation link explaining why f_1 is violated or a satisfaction link explaining why f_2 is satisfied can also explain why this implies formula is satisfied according to its built-in semantics. In other words, a goal of $\top$ for an implies formula should be divided into a sub-goal of $\bot$ for its first sub-formula f_1 and a sub-goal of $\top$ for its second sub-formula f_2. Other cases are similar and thus omitted due to page limit.

Let us apply the goal semantics to our first example shown in Figure 2, where satisfaction links are expected and thus the constraint's goal is $\top$. All sub-goals on its sub-formula nodes are then derived and shown in Figure 6, where T represents a goal of $\top$, F represents $\bot$, and N represents null. We observe that in Figure 6, the sub-goals on the right not and "$\exists \gamma_2 \in CS_2$" nodes are $\top$ and $\bot$, respectively. They would help reduce generating redundant links as in Figure 2. We explain our new link generation guided by our goal semantics in the following.

3.2 Link Generation

The third step reuses the original link generation semantics in Figure 1, except that we make some small modifications in order to exploit our derived goals in the second step.

Let $\mathcal{L}$ be the original link generation function we discussed earlier in Section 2.1, and $\mathcal{L}'$ be our new link generation function. $\mathcal{L}'$ differs

$\mathcal{L}'[\forall\gamma{\in}s\ (f)]_\alpha$ =

(1) $\mathcal{L}[\forall\gamma{\in}s\ (f)]_\alpha$, if $\mathcal{G}[\forall\gamma{\in}s\ (f)]_\alpha = \bot$;

(2) ϕ, otherwise.

$\mathcal{L}'[\exists\gamma{\in}s\ (f)]_\alpha$ =

(1) $\mathcal{L}[\exists\gamma{\in}s\ (f)]_\alpha$, if $\mathcal{G}[\exists\gamma{\in}s\ (f)]_\alpha = \top$;

(2) ϕ, otherwise.

Figure 7. New link generation semantics for universal and existential formulae.

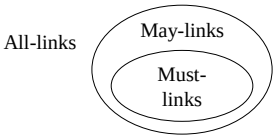

Figure 9. Relationships among all-links, must-links and may-links.

from $\mathcal{L}$ only for universal and existential formulae (i.e., for any other formula *f*, we have $\mathcal{L}'[f]_\alpha = \mathcal{L}[f]_\alpha$). We give the new link generation semantics for universal and existential formulae in Figure 7.

We revisit the example in Figure 6. Using the new link generation function $\mathcal{L}'$, checking the right "$\exists\gamma_2{\in}\mathsf{CS}_2$" node would generate no (redundant) link at all, since this situation falls into Case (2) of the $\mathcal{L}'$ function for existential formula. For its upper not node, its checking also does not generate any (redundant) link, since its child node "$\exists\gamma_2{\in}\mathsf{CS}_2$" returns no link. Thus, the preceding four redundant links (Figure 2) generated for the two sub-formula nodes are completely avoided by not generating them at all (Figure 8). This is the exact meaning of our "generate as required" strategy. The other example in Figure 3 can also be handled in the same way, and we can observe that the preceding four redundant links are prevented from generation, too. We omit their details due to page limit.

4. DISCUSSION OF PROPERTIES

In this section, we study properties of our goal-directed context validation technique. The technique includes both goal semantics and revised link generation semantics. We refer to them as "new semantics" for short. We first give a theorem below:

Theorem 1. *Given a formula, if its goal is* $\top$*, then the new semantics generate only satisfaction links (if any); if its goal is* $\bot$*, the new semantics generate only violation links (if any); otherwise, its goal is* null *and the new semantics would generate no link.* □

Sketch of proof. We prove the theorem by induction. We do not give a complete proof here, since a complete treatment of all formula types is long and tedious. Suppose that we have proved the basic step and proceed with the inductive step. We take and formula for example. Suppose that an and formula has a goal of $\top$, which means that satisfaction links are expected for this formula. According to the new semantics, both its sub-formulae have a goal of $\top$, i.e., both sub-formulae are expected to generate satisfaction links. For and formula, its new link generation semantics is the same as the original one, and thus we revisit the semantics in Figure 1. For Case (1), both sub-formulae are evaluated to true and generate satisfaction links according to the basic step. Then the and formula is also evaluated to true and generates satisfaction links. For the other three cases, at least one sub-formula is evaluated to false and can generate only violation links according to the basic step.

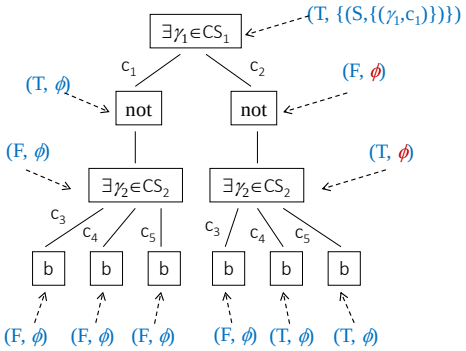

Figure 8. Improved link generation (redundant links prevented from generation).

However, since each sub-formula's goal is $\top$, it would generate no (violation) link. Then the and formula also generates no link, since for these three cases, its links are generated based on violation links from its sub-formulae, but in the new semantics no violation link can be generated from its sub-formulae. In summary, the and formula generates only satisfaction links or no link at all, if its goal is $\top$. The goal of $\bot$ can also be proved similarly. This completes the proof (sketch). □

With Theorem 1, we discuss three properties of our new semantics: *soundness*, *completeness* and *conciseness*.

4.1 Soundness and Completeness

Given a consistency constraint, we use *all-links* to refer to the links generated by the original link generation semantics (referred to as "original semantics" for short). All-links can be divided into two parts: *must-links* and *may-links*. *Must-links* refer to those links that have to be generated in order to generate final links for the root note associated with the constraint, while *may-links* refer to those links, whose generation can be avoided without affecting the generation of final links. For example, in Figure 2 the link set {(satisfied, {(γ_1, c_1)})} generated for the root node "$\exists\gamma_1{\in}\mathsf{CS}_1$" contains must-links since they are expected. However, the other two link sets, {(violated, {(γ_2, c_4)}), (violated, {(γ_2, c_5)})} and {(satisfied, {(γ_2, c_4)}), (satisfied, {(γ_2, c_5)})}, generated for the right not and "$\exists\gamma_2{\in}\mathsf{CS}_2$" nodes, respectively, contain may-links, since they are redundant links, as we discussed earlier.

We use Figure 9 to illustrate relationships among all-links, must-links and may-links. We discuss soundness and completeness next. *Soundness* means that no link other than all-links has ever been generated in the new semantics. It is important for the correctness of our link generation. *Completeness* means that all must-links have been generated by the new semantics. It is also important because any must-link should not be missed. Otherwise, when a consistency constraint is violated, one may not be able to fully explain how it has occurred. This would impair the usefulness of our goal-directed context validation.

We have the following theorem about the soundness and completeness of our new semantics:

Theorem 2. *The new semantics are sound in that no link other than all-links would be generated, and complete in that all must-links would be generated.* □

Sketch of proof. We prove the theorem by induction, too. Soundness is easy to prove since the new semantics do not generate any

Table 1. Conciseness analysis.

Formula	U-LR (%)		E-LR (%)	
	New	Original	New	Original
$\forall \gamma \in s$ (*f*)	100	50	100	33.3
$\exists \gamma \in s$ (*f*)	100	50	100	33.3
(*f*) and (*f*)	83.3	75	100	33.3
(*f*) or (*f*)	83.3	75	100	33.3
(*f*) implies (*f*)	91.7	75	100	33.3
not (f)	100	100	100	33.3
Averaged	**93.1**	**70.8**	**100**	**33.3**

new link other than those by the original semantics (see Section 3.2). To prove completeness, we take and formulae for example. Suppose that an and formula has a goal of $\top$. Then this formula generates only satisfaction links according to Theorem 1. Let us examine whether all "must" satisfaction links can be generated. According to the original semantics in Section 2.1, only Case (1) can generate satisfaction links, and these links are constructed from satisfaction links from the and formula's two sub-formulae. According to the new semantics and Theorem 1, both sub-formulae have a goal of $\top$ and thus they generate satisfaction links only. From the induction, these generated satisfaction links are complete for sub-formulae, and thus the constructed satisfaction links are also complete for the and formula. The proof for other goals and formula types is omitted due to similarity and page limit. This completes the proof (sketch). □

4.2 Conciseness Analysis

Conciseness is another important property about our new semantics. *Conciseness* means that no may-link would be generated by the new semantics, and this is an ideal goal. Since our new semantics aim to reduce the generation of redundant links (or may-links) as many as possible, we want to know the extent our new semantics can suppress the generation of such links.

We in the following analyze the conciseness of our new semantics. For any formula *f* that contains sub-formulae (i.e., universal, existential, and, or, implies, and not formulae), we measure its *used link rate* and *effective link rate*. Both metrics relate to the problem of how many redundant links are generated, and are thus concerned with the conciseness analysis.

The *used link rate* is defined as $B / A * 100\%$, where A is the number of formula *f*'s sub-formulae for which links have been generated, and *B* is the number of *f*'s sub-formulae for which links are generated and also *used* for generating *f*'s links. Since the links generated for a specific sub-formula are not necessarily used later, *B* is typically less than *A*. The used link rate measures the extent generated links are useful for later link generation.

The *effective link rate* is defined as $D / C * 100\%$, where *C* is the number of links that have been generated for formula *f*, and *D* is the number of links that are generated for *f* and also required. We note that even if a link is used to construct new links in the middle of context validation, the constructed links may not be actually required for generating final links for the concerned constraint (i.e., they are eventually abandoned). Therefore, *D* is typically less than *C*. The effective link rate measures the extent generated links are necessary for final link generation.

To calculate analytically the used link rate and effective link rate, we assume that each sub-formula has the same probability of being evaluated to true or false, and that all sub-formulae are independent of each other. For example, for an and formula "$f = (f_1)$ and (f_2)", f_1 and f_2 are two sub-formulae. We would assume that both of them have a 50 % probability of being evaluated to true or false, and that f_1's truth value is independent of that of f_2. Besides, we also assume that each sub-formula has the same probability of having a goal of $\top$, $\bot$ or null, i.e., three cases are averaged for statistical purposes.

With the above assumptions, we derive Table 1, which compares our new semantics and the original semantics in terms of used link rate (U-LR) and effective link rate (E-LR).

From the comparison, we observe that the new semantics exceed the original semantics in both used link rate and effective link rate. The averaged improvement (from 70.8% to 93.1%) in the used link rate indicates that 22.3% more links generated for sub-formulae are useful for later link generation in checking a consistency constraint. Besides, the averaged improvement (from 33.3% to 100%) in the effective link rate indicates that 66.7% more links generated for sub-formulae are actually required for generating final links for the concerned constraint. Therefore, both results suggest that the generation of redundant links has been greatly suppressed by our new semantics.

What is worth noticing is that even our new semantics do not realize a perfect used link rate of 100%. This is because there are still some, although much fewer, redundant links that could be generated in checking and, or and implies formulae. For example, when the goal of an and formula is $\top$, the goals of its two sub-formulae are also $\top$. If the first sub-formula is evaluated to true and the second one to false, then the first sub-formula should generate satisfaction links and the second one violation links. This is Case (3) in the semantics. However, only links generated for the second sub-formula are used, but those for the first one are abandoned, since the and formula itself needs to generate violation links. These abandoned links are redundant.

In summary, we have realized partial conciseness by the new semantics, but the comparison results are still promising, considering that we have incorporated only small modifications into the link generation semantics.

5. RELATED WORK

Ubiquitous computing is receiving increasing attention from researchers and developers for its flexible and adaptive computing capability. The capability is built on application contexts, which are collected from environments of ubiquitous computing. From an early piece of representative work, Context Toolkit [4], to nowadays sophisticated middleware infrastructures or programming frameworks [6][8][11][14][15], various programming support and context-aware services have been offered, and practical applications have been developed and deployed. With such success, context validation, however, has not been adequately studied in the existing literature. A few studies on context-awareness [4][7] are concerned with either frameworks that support context abstraction or data structures that support context queries and topic subscriptions. Some research projects, e.g., Gaia [14][15], have been proposed to provide middleware support for ubiquitous computing, but they focus mainly on the organization of, and cooperation among, computing devices and services. Other infrastructures such as hybrid observer model [6] and mobility coordination model in [8][11] are

mostly concerned with support of context processing, reasoning and programming. Little attention has been paid to context validation for adaptive ubiquitous systems.

Validating contexts for adaptive ubiquitous systems follows a constraint perspective, which concerns to some extent the automated theorem proving issue. Related work [10] considered how to realize automated theorem proving by mapping logics to an artificial intelligence problem. Theorem-proving techniques can also be used for automated synthesis of combinational logic [9]. Nevertheless, we in this paper do not study how to realize automated theorem proving or apply theorem-proving techniques to practical problems, but focus on a related problem, i.e., how to recognize and reduce unnecessary effort in logic evaluation, in particular, during context validation for adaptive ubiquitous systems.

Context validation also shares some observations with consistency management for software artifacts, which include application profile [1], triggered action [2], data structure [3] and UML model [5][13]. Our previous work [18] proposed to use context-related Event-Condition-Action rules and semantic reasoning to recognize inconsistent contexts with ontology support. The work has limitations in expressive power and detection performance. Then our follow-up work [20] studied how to detect inconsistent contexts based on an incremental constraint checking technique. The work presented in this paper further improves the technique by guiding its link generation with specific goals. This can substantially suppress the generation of redundant links, as we analyzed in Section 4.2.

6. CONCLUSION

Having a vast amount of redundant links is a severe obstacle to applying context validation techniques to realistic adaptive systems. In this paper, we have proposed a goal-directed technique to effectively reduce the generation of redundant links in context validation for adaptive ubiquitous systems.

We have evaluated our technique through proofs and qualitative analysis. The analysis shows that our technique may potentially generate much fewer redundant links than the original link generation. When our results can be further confirmed, it would greatly improve the applicability of context validation to adaptive ubiquitous systems. In future, we plan to consider ways of fully eliminating the generation of redundant links in context validation, and conduct realistic experiments with practical case studies to further validate our findings.

ACKNOWLEDGMENTS

The research is partially supported by a grant from the Research Grants Council of Hong Kong (Project No. 612306).

REFERENCES

[1] L. Capra, W. Emmerich and C. Mascolo, "CARISMA: Context-Aware Reflective Middleware System for Mobile Applications", *IEEE Transactions on Software Engineering 29(10)*, pp. 929-945, Oct 2003.

[2] J. Chomicki, J. Lobo and S. Naqvi, "Conflict Resolution Using Logic Programming", *IEEE Transactions on Knowledge and Data Engineering 15(1)*, pp. 244-249, Jan/Feb 2003.

[3] B. Demsky and M. Rinard, "Data Structure Repair Using Goal-Directed Reasoning", In *Proceedings of the 27th International Conference on Software Engineering*, pp. 176-185, St. Louis, USA, May 2005.

[4] A.K. Dey, G.D. Abowd and D. Salber, "A Context-Based Infrastructure for Smart Environments", In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments*, pp. 114-128, Dublin, Ireland, Dec 1999.

[5] A. Egyed, "Instant Consistency Checking for the UML", In *Proceedings of the 28th International Conference on Software Engineering*, pp. 381-390, Shanghai, China, May 2006.

[6] W.G. Griswold, R. Boyer, S.W. Brown and M.T. Tan, "A Component Architecture for an Extensible, Highly Integrated Context-Aware Computing Infrastructure", In *Proceedings of the 25th International Conference on Software Engineering*, pp. 363-372, Portland, USA, May 2003.

[7] K. Henricksen and J. Indulska, "A Software Engineering Framework for Context-Aware Pervasive Computing", In *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications*, pp. 77-86, Orlando, USA, Mar 2004.

[8] C. Julien and G.C. Roman, "EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications", *IEEE Transactions on Software Engineering 32(5)*, pp. 281-298, May 2006.

[9] W.C. Kabat, "Automated Synthesis of Combinational Logic Using Theorem-Proving Techniques", *IEEE Transactions on Computers 34 (7)*, pp. 610-632, Jul 1985.

[10] D.W. Loveland, "Automated Theorem Proving: Mapping Logic into AI", In *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems*, pp. 214-229, Knoxville, Tennessee, USA, Oct 1986.

[11] A.L. Murphy, G.P. Picco and G.C. Roman, "LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents", *ACM Transactions of Software Engineering and Methodology 15(3)*, pp. 279-328, Jul 2006.

[12] C. Nentwich, L. Capra, W. Emmerich and A. Finkelstein, "xlinkit: A Consistency Checking and Smart Link Generation Service", *ACM Transactions on Internet Technology 2(2)*, pp. 151-185, May 2002.

[13] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer, "Flexible Consistency Checking", *ACM Transactions on Software Engineering and Methodology 12(1)*, pp. 28-63, Jan 2003.

[14] A. Ranganathan, J. Al-Muhtadi and R.H. Campbell, "Reasoning about Uncertain Contexts in Pervasive Computing Environments", *IEEE Pervasive Computing 3(2)*, pp. 62-70, Apr/Jun 2004.

[15] A. Ranganathan and R.H. Campbell, "An Infrastructure for Context-Awareness Based on First Order Logic", *Personal and Ubiquitous Computing 7*, pp. 353-364, 2003.

[16] P. Tarr and L.A. Clarke, "Consistency Management for Complex Applications", In *Proceedings of the 20th International Conference on Software Engineering*, pp. 230-239, Kyoto, Japan, Apr 1998.

[17] R. Want, A. Hopper, V. Falcao and J. Gibbons, "The Active Badge Location System", *ACM Transactions on Information Systems 10(1)*, pp. 91-102, Jan 1992.

[18] C. Xu and S.C. Cheung, "Inconsistency Detection and Resolution for Context-Aware Middleware Support", In *Proceed-*

ings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 336-345, Lisbon, Portugal, Sep 2005.

[19] C. Xu and S.C. Cheung, "Incremental Context Consistency Checking", *Technical Report HKUST-CS05-15*, Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China, Oct 2005.

[20] C. Xu, S.C. Cheung and W.K. Chan, "Incremental Consistency Checking for Pervasive Context", In *Proceedings of the 28th International Conference on Software Engineering*, pp. 292-301, Shanghai, China, May 2006.