To appear in *SEAMS 2007*.

Goal-Directed Context Validation for Adaptive Ubiquitous Systems

Chang Xu and S.C. Cheung
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Kowloon, Hong Kong, China
{*changxu, scc*}*@cse.ust.hk*

W.K. Chan
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong, China
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 badly impaired if the environments are incorrectly perceived. Our earlier work has proposed a technique to evaluate the constraints that govern the consistency of perceived environmental information. This technique generates links as the evaluation result to explain how the constraints are satisfied or violated. However, the technique may generate redundant links and may not effectively utilize resources in both time and space. To address the problem, this paper presents a goal-directed technique to enhance our earlier link generation semantics. We evaluate the technique analytically, and show how the enhanced semantics helps reduce the number of generated redundant links for context validation.

1. Introduction

Adaptive ubiquitous systems are receiving increasing research and industrial attention with the advancement of radio frequency, sensors, and wireless technologies. These systems typically perceive their computing environments and adapt their behaviors accordingly. A system's *contexts* refer to the pieces of information that characterize its environment. Examples of contexts include network conditions, geographical locations, temperatures, and user identities. 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 vibrate and keep silent in a concert hall, but beep loudly during a football match to alert its user.

Our earlier work [20] has shown the need for adaptive ubiquitous systems for context validation; otherwise the use of inconsistent contexts (e.g., obsolete, corrupted, or inaccurate contexts) may lead to anomalous behaviors of adaptive ubiquitous systems. For example, the context-aware smart phone mentioned previously may roar during the most important moment of a wedding ceremony if it mistakes the situation for a football match. In those situations where a precise oracle to judge the validity of a context is unavailable or too costly to be applied, specifying consistency constraints on contexts is a feasible alternative. Let us explain this approach using an example.

To detect inconsistent contexts for the location-aware application Call Forwarding [17], we specify the constraint that "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 as follows: $\forall \gamma_1 \in \mathsf{LOC}$ (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 within a particular time interval. Let difPlc be a function that accepts two location contexts and returns true if the two contexts refer to the same object (e.g., a 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. The above constraint is violated when $\gamma_1 = l_1$ and $\gamma_2 = l_2$. As such, a link associated with the constraint is generated to indicate how this violation occurs: (violated, $\{(\gamma_1, l_1), (\gamma_2, l_2)\}$). The above information is called a *link* because it indicates whether a specific binding of variables to the two location contexts would cause the satisfaction or violation of the above constraint. In this example, the binding causes a violation to the constraint, and hence the two location contexts are *inconsistent* with respect to the above constraint.

A link in xlinkit [12] [13] describes how combinations of elements (e.g., contexts in pervasive computing) fulfill a constraint or beach it. Our previous work [20] constructs links incrementally. However, either approach may generate *redundant* links during the evaluation of sub-formulae in a constraint. These links do not contribute to the computation of the entire constraint's satisfaction or violation. In Section 2, we illustrate redundant links and their generation using two examples. In these two examples, 80% of the generated links are redundant. The proportion is significant. The constraints studied in these two examples are derived from a survey we conducted. In this survey, 30 participants were asked to specify consistency constraints for three published ubiquitous applications, which include the Call Forwarding application we 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.

We present in the paper a goal-directed technique to address this problem. The technique is evaluated in terms of *used link ratio* and *effective link ratio*. Two ratios are improved from 70.8% to 91.7%, and from 33.3% to 100%, respectively, compared to the original approach. The results suggest that the proposed goal-directed technique can effectively reduce redundant link generation during the context validation for ubiquitous systems.

In this paper, Section 2 introduces background knowledge about link generation and two motivating examples; Section 3 proposes a goal-directed technique for reducing the generation of redundant links; Section 4 discusses the properties of this technique. Finally, Section 5 reviews the related work in recent years and Section 6 concludes this paper.

2. Background

2.1. Link generation for context validation

In the following we present an overview of our earlier work [19] on context validation for ubiquitous systems.

Links connect multiple contexts (e.g., location contexts in the Call Forwarding application we discussed in Section 1) that satisfy or violate specified constraints. There are two types of links: *satisfaction link* and *violation link*. They are represented, respectively, as (satisfied, *bindings*) and (violated, *bindings*). Satisfied and violated are two keywords that show whether the constraint concerned is satisfied or violated, respectively. *Bindings* is a variable assignment that contains mappings between variables in the constraint and contexts bound to these variables, and under this variable assignment, this constraint is satisfied or violated according to the type of this link. An example is the violation link (violated, $\{(\gamma_1, l_1), (\gamma_2, l_2)\}$) we discussed in Section 1. This violation link shows that its associated constraint has been violated due to the context pair l_1 and l_2.

We then introduce how to generate links for a universal and an "and" formulae for 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.

The following gives the link generation semantics for a universal and an "and" formulae:

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

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

(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 $\mathcal{T}[f_2]_\alpha = \top$

Given a formula (e.g., a universal or an "and" formula), function $\mathcal{L}$ generates links for this formula under the given variable assignment α. In the above semantics, $\top$ means true and $\bot$ means false. The function bind adds a new mapping into a variable assignment. This function is used when we evaluate the sub-formulae of the given universal or "and" formula. The interpretation of operators such as $\otimes$ and $\underline{\otimes}$ is not essential to the understanding of our following explanation about redundant links and their generation. We only need 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 assignments related to this formula and the links returned from its sub-formulae, and the $\underline{\otimes}$ operator is used to construct links for an "and" formula with links returned from its two sub-formulae.

From the above semantics, we observe that the links for a universal formula are only related to the links returned from its sub-formulae if these sub-formulae are evaluated to be false. For an "and" formula, it is a little bit more complicated. We observe that the links for an "and" formula are related to the links returned from both of its sub-formulae only when these two sub-formulae have the same truth value, true or false (see Case (1) and Case (2)); otherwise, only the links of one sub-formula are selected and thus related to the links generated by the "and" formula (see Case (3) and Case (4)). This observation is useful for our later proposed goal-directed technique for reducing the number of generated redundant links as we observe from the above link generation semantics that some generated links are no longer used later.

We explain more about the link generation semantics for these two formulae types, that is, a universal and an "and" formulae, in the following.

Given a universal formula, the link generation semantics examine all possible variable assignments in this formula to check whether they cause any violation in this formula's sub-formula, and record such variable assignments in terms of links. This is because a variable assignment that makes a sub-formula of this universal formula violated can also explain how this universal formula is violated. Given an "and" formula, the link generation semantics partition all situations into four cases and construct links accordingly. For example, in Case (3) the "and" formula's first sub-formula f_1 is evaluated to be true and its second sub-formula f_2 to be false, then this "and" formula is evaluated to be false due to and only due to f_2. Hence, the links generated (and thus returned) by f_2, which explain how f_2 is evaluated to be false, also explain how the "and" formula is evaluated to be false (and how the constraint represented by the "and" formula is violated). We omit discussion of the other three cases for simplicity.

More detailed explanations and illustrative examples of link generation can be found in our technical report [19].

2.2. Revisiting the link generation semantics

The above link generation semantics are not optimized in the sense that they may generate redundant links that are never used for explaining how constraints are satisfied or violated. Let us illustrate the generation of these redundant links in the following two examples.

From the point of view of users, there are two representations for specifying constraints. One is the *necessary condition representation*, which is from the validation perspective and the most common form [16]. An example is the constraint we discussed in Section 1: $\forall\gamma_1\in \text{LOC}$ (not ($\exists\gamma_2\in \text{LOC}$ (difPlc(γ_1, γ_2)))), where we are interested in its violation because any violation indicates that a necessary condition does not hold and hence the contexts of adaptive systems are no longer valid. On the other hand, we can also use the *sufficient condition representation* to express the same constraint in the following: $\exists\gamma_1\in \text{LOC}$ ($\exists\gamma_2\in \text{LOC}$ (difPlc(γ_1, γ_2))), where we are interested in its satisfaction because this representation directly describes a scenario where contexts are judged as invalid. Both representations are useful because users may have different preferences. Since different representations imply different interests to users (i.e., expecting violation or satisfaction of constraints), some generated links by the original link generation semantics may be redundant because the latter essentially does not take users' interests into account.

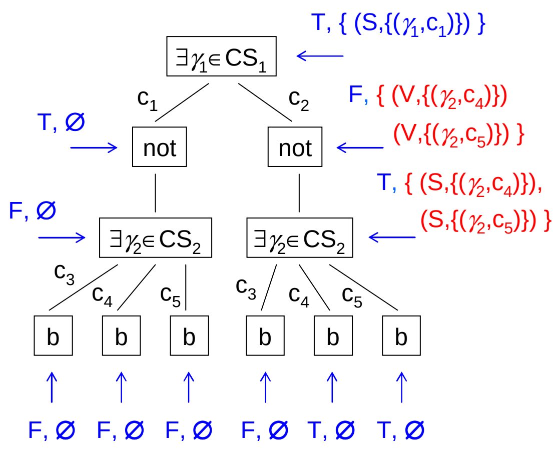


Figure 1. Satisfaction links are expected

Figure 1 illustrates the link generation process for an example 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 function. We do not elaborate on their semantics here, because they are not essentially related to the problem of generating redundant links. We suppose that $CS_1 = \{c_1, c_2\}$ and $CS_2 = \{c_3, c_4, c_5\}$, and c_1, c_2, c_3, c_4, c_5 are five contexts we collected at some time. We enumerate all combinations for function b and get true for b(c_2, c_4) and b(c_2, c_5) and false for other combinations. Figure 1 illustrates the above information, where T means true and F means false. Besides, link information is annotated with T/F information in the figure: S means satisfied, V means violated, and ∅ means no links generated at this node. For example, the information displayed besides the rightmost b node at the bottom level of the figure is "T, ∅", which means that this b node (with the variable assignment of $\gamma_1 = c_2$ and $\gamma_2 = c_5$) is evaluated to be true and there is no link generated for this node. The information displayed 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 be true and a satisfaction link is generated: (satisfied, {(γ_1, c_1)}). This satisfaction link shows that when $\gamma_1 = c_1$, the constraint is satisfied, no matter which context the variable γ_2 is bound to.

Suppose that our goal is to find satisfaction links of the given existential constraint, that is, we use a sufficient condition representation to specify this constraint. While we get satisfaction links at the root node "$\exists\gamma_1\in CS_1$" as illustrated in Figure 1, some links generated during 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 redundant. This is because the final satisfaction link set {(satisfied, {(γ_1, c_1)})} returned by the root node does not use any information from these two link sets at all. A closer study shows that the link generation for "$\exists\gamma_1\in CS_1$ (…)" only cares about satisfaction links from its sub-formula, but the sub-formula "not (…)" on the right branch happens to generate violation links that are not expected by "$\exists\gamma_1\in CS_1$ (…)". Therefore, these links become redundant.

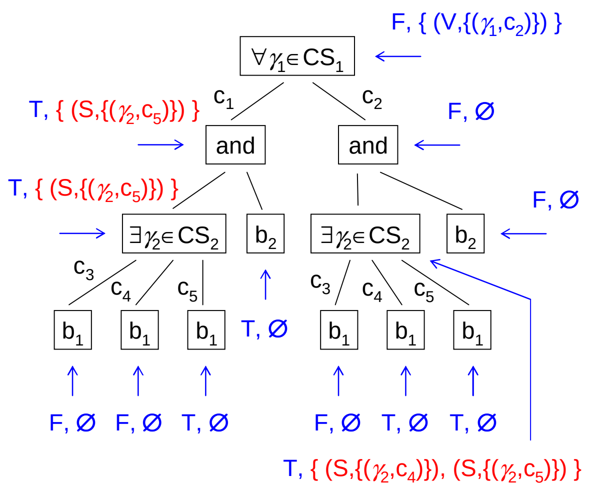


Figure 2. Violation links are expected

Figure 2 illustrates another example whose goal is to find violation links from the given universal constraint, that is, we use the necessary condition representation to specify this constraint. While we get violation links at the root node, some links generated in the 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_4)})} from the right "$\exists\gamma_2\in CS_2$" node, are redundant because the final violation link set {(violated, {(γ_1, c_2)})} does not use any information from them at all. We do not elaborate on the details because of the similarity to the first example.

The above two examples illustrate the inadequacy of existing link generation semantics, which actually adopts the strategy of "generate and select". For example, in Case (3) of the link generation semantics for "and" formulae in Section 2.1, the links generated by the first sub-formula f_1 are abandoned.

3. Goal-directed context validation

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

The following gives our constraint language syntax:

$f ::= \forall\gamma\in s\ (f)\ \mid$	/* universal formula */
$\exists\gamma\in s\ (f)\ \mid$	/* existential formula */
(f) and (f) |	/* "and" formula */
(f) or (f) |	/* "or" formula */
(f) implies (f) |	/* "implies" formula */
not (f) |	/* "not" formula */
$b(\gamma_1, \ldots, \gamma_n)$	/* terminal */

The above language syntax follows traditional FOL interpretations. f represents a formula, s is a set that contains finite contexts, and b is a function that takes contexts as inputs and returns a truth value as the output. Note that we are only interested in well-formed formulae that contain no free variables. *Free variable* refers to a variable that is used without definition. For example, the variable γ_2 in formula "$\exists\gamma_1$ in s (f(γ_2))" is a free variable. Therefore, it is not a well-formed formula.

The recursive nature of the constraint language allows constraints to be hierarchically structured as shown in Figure 1 and Figure 2. Based on this hierarchy, our improved link generation works as follows:

(1) Given a constraint, the answer to the question "what type of links is expected?" decides the ultimate goal on this constraint;

(2) A top-down analysis, i.e., from every formula to its sub-formulae, derives the sub-goal on every sub-formula according to the ultimate goal;

(3) A bottom-up process generates links according to the sub-goal on every sub-formula in a post-order traversal until the root node.

The first step is straightforward because we can easily tell what type of links is expected from a constraint's representation, which constitutes the goal of context validation using this constraint. We elaborate on the second and third steps in Sections 3.1 and 3.2, respectively.

3.1. Goal semantics

In order to reduce the number of generated redundant links, we want to know whether a particular evaluation of sub-formula may require generating a link. As such, we extend the value set for a goal from $\{\top, \bot\}$ to include a null value, where logical operators are applicable to only the values $\top$ and $\bot$. In other words, $\top$ means that the given constraint should provide satisfaction links; $\bot$ means the opposite; whereas null means that no link is needed (or expected).

The goal semantics is based on the observation that some formulae can only generate a particular type of links. For example, a universal formula only generates violation links, which indicate how they violate the formula (see the link generation semantics in Section 2.1). Therefore, the goal of generating satisfaction links for a universal formula can be ignored. This idea is formally modeled by the following goal semantics.

Let $\mathcal{G}$ be a goal function that accepts a formula and a variable assignment, and returns a goal for this formula. We only consider the following six formula types because all of them have at least one sub-formula. The goal semantics is as follows:

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

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

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

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

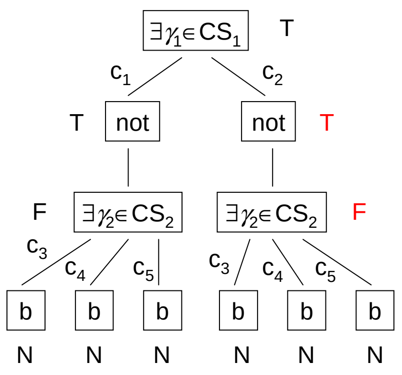
5. $\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$:
   - (1) $\mathcal{G}[f_1]_\alpha = \neg\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$,
     $\mathcal{G}[f_2]_\alpha = \mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha$,
     if $\mathcal{G}[(f_1)\ \text{implies}\ (f_2)]_\alpha \neq \text{null}$;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

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

The interpretation of the above goal semantics is straightforward. For example, for a universal formula, each of its sub-formulae has a goal of $\bot$ only if this formula has a goal of

$\perp$. Otherwise, the goal of each sub-formula is set to null. This is because universal formulae can only generate violation links according to our earlier discussion, and the violation links explaining how a sub-formula is violated also explain how a universal formula 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 $\perp$ and the second sub-formula f_2 has a goal of $\top$. This is because the links explaining how f_1 is violated (that is, violation links) or the links explaining how f_2 is satisfied (that is, satisfaction links) can explain how this "implies" formula is satisfied. In other words, a goal of $\top$ for this "implies" formula should be divided into a sub-goal of $\perp$ for f_1 and a sub-goal of $\top$ for f_2. Other cases are similar and thus explanation is unnecessary.

Let us apply the above semantics to the early example illustrated in Figure 1, where satisfaction links are expected and therefore the ultimate goal is $\top$. All sub-goals on sub-formulae are derived and illustrated in Figure 3, where T means a goal of $\top$, F means a goal of $\perp$, and N means a goal of null. From Figure 3, the goals on the right "not" and "$\exists\gamma_2\in CS_2$" nodes are $\top$ and $\perp$, respectively. These would help reduce the generation of redundant links showed in Figure 1. We explain our new link generation semantics in the next section.



$\exists\gamma_1\in CS_1$
T
c_1
c_2
T
not
not
T
F
$\exists\gamma_2\in CS_2$
$\exists\gamma_2\in CS_2$
F
c_3
c_4
c_5
c_3
c_4
c_5
b
b
b
b
b
b
N
N
N
N
N
N

Figure 3. Derived sub-goals on sub-formulae

3.2. Generating links

The third step reuses the original link generation semantics except that a small modification is made to utilize our goals derived in the second step.

Let $\mathcal{L}$ be the original link generation function in Section 2.1, and $\mathcal{L}$' be the new link generation function. $\mathcal{L}$' differs from $\mathcal{L}$ only for universal and existential formulae (that is, for other formula f, $\mathcal{L}'[f]_\alpha = \mathcal{L}[f]_\alpha$):

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

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

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

(2) $\varnothing$,

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) $\varnothing$,

otherwise

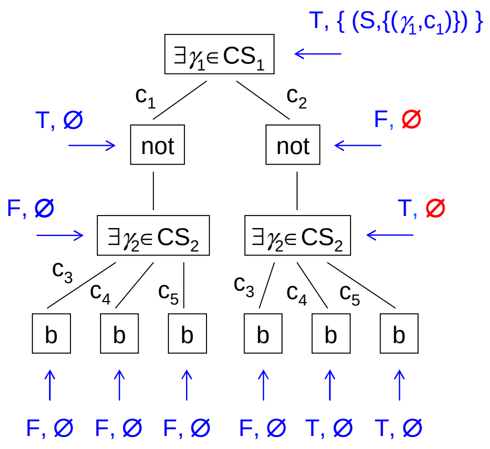

T, { (S,{(γ_1,c_1)}) }
$\exists\gamma_1\in CS_1$
c_1
c_2
T, ∅
F, ∅
not
not
F, ∅
T, ∅
$\exists\gamma_2\in CS_2$
$\exists\gamma_2\in CS_2$
c_3
c_4
c_5
c_3
c_4
c_5
b
b
b
b
b
b
F, ∅
F, ∅
F, ∅
F, ∅
T, ∅
T, ∅

Figure 4. Improved link generation

We revisit the example in Figure 3. Using the new function $\mathcal{L}$', checking the right "$\exists\gamma_2\in CS_2$" node would generate no redundant links because this situation falls into Case (2) of $\mathcal{L}$' function for existential formulae. For the upper "not" node, its validation also does not generate any redundant links since its child node "$\exists\gamma_2\in CS_2$" returns no links. Thus, the previous redundant links from these two sub-formulae are avoided by not generating them at all (see Figure 4). This is the exact meaning of our "generate as required" strategy. The other example in Figure 2 can also be handled in the same way and we see that the previous redundant links are avoided, too. We omit the details here.

4. Discussion of properties

In this section we study some properties of our proposed goal-directed technique. From the previous, the technique includes the goal semantics and modified 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 $\top$, the new semantics only generate satisfaction links (if any); if the goal is $\perp$, the new semantics only generate violation links (if any); if the goal is null, the new semantics generate no links. □

We use induction to prove the above theorem. We do not give a complete proof here, because a complete treatment of all formula types is very long. Suppose that the basic step has been proved and we continue with the inductive step. We take an "and" formula for example. Suppose that an "and" formula has a goal of $\top$, which means that satisfaction links are expected from this formula. According to our goal semantics, both its sub-formulae have a goal of $\top$, i.e., both expect satisfaction links to be returned from them. We revisit the original link generation semantics for an "and" formula in Section 2.1. In Case (1), both sub-formulae are evaluated to be true and return satisfaction links according to the induction, and then the "and" formula is also evaluated to be true and returns satisfaction links according to the link generation semantics. In the other three cases, at least one sub-formula is evaluated to be false and should return violation links. However, since its goal is $\top$, it would return no links according to the induction. Then the "and" formula also returns no links since in these three cases, its returned links are generated based on the violation links returned from its sub-formulae but in the new semantics no violation links are returned from its sub-formulae. In summary, the "and" formula only returns satisfaction links or no links if its goal is $\top$. The goal of $\perp$ can also be proved in the same way.

With Theorem 1, we discuss three properties of the new semantics. They are soundness, completeness, and conciseness.

4.1. Soundness and completeness

Given a constraint, we use *all-links* to refer to the links generated by the original link generation semantics. 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 the final links to be returned, whereas *may-links* refer to those links, the generation of which can be avoided without affecting the generation of the final links. For example, in Figure 1 the link set {(satisfied, {(γ_1, c_1)})} returned by the root node "$\exists \gamma_1 \in CS_1$" contains must-links because they are expected, whereas the two link sets {(violated, {(γ_2, c_4)}), (violated, {(γ_2, c_5)})} and {(satisfied, {(γ_2, c_4)}), (satisfied, {(γ_2, c_5)})} returned by the right "not" and "$\exists \gamma_2 \in CS_2$" nodes, respectively, contain may-links, because they are redundant links as discussed earlier in Section 2.2.

Figure 5 illustrates the relationships among all-links, must-links, and may-links.

We discuss soundness and completeness in the following. *Soundness* means that no links other than all-links are generated in the new semantics. It is important for the correctness of generated links. *Completeness* means that all must-links are generated by the new semantics. It is also important because any must-links should not be missed. Otherwise, when a constraint is violated, we may not be able to explain how it occurs. This would impair the significance of our context validation.

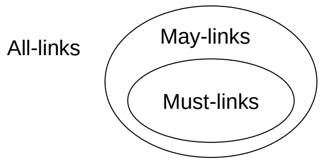


Figure 5. All-links, must-links, and may-links

We have the following theorem about the soundness and completeness:

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

Theorem 2 can be proved using induction, too. We do not show the proof details for simplicity. Soundness is relatively easy to prove because we do not generate any new links (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 only generates satisfaction links according to Theorem 1. Let us examine whether all satisfaction links are generated. According to the original link generation semantics in Section 2.1, only Case (1) would generate satisfaction links, and these satisfaction links are constructed from the satisfaction links returned from the "and" formula's two sub-formulae. According to our goal semantics and Theorem 1, both sub-formulae have a goal of $\top$ and hence they would return satisfaction links only (if any). From the induction, these returned satisfaction links are complete, and therefore the constructed satisfaction links for the "and" formula are also complete. The proofs for other goals and formula types are omitted because they are similar.

4.2. Conciseness analysis

Conciseness is another important property. *Conciseness* means that no may-links are generated by the new semantics. Since our objective is to reduce the generation of redundant links (or may-links), we want to know to what degree our proposal suppress these links.

Let us here 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 evaluate its *used link ratio* and *effective link ratio*. These two measurements relate to the problem of how many redundant links are generated, and are thus concerned with the conciseness issue.

Used link ratio is defined as B / A * 100%, where A is the number of *f*’s sub-formulae for which links are generated, and B is the number of *f*’s sub-formulae for which links are generated and actually used for generating *f*’s links. Since the links generated by a sub-formula are not necessarily used later, B is less than A. Used link ratio evaluates the extent that generated links are used by the later link generation.

Effective link ratio is defined as D / C * 100%, where C is the number of links that are generated for *f*, and D is the number of links that are generated for *f* and actually required. Even if a link is used to construct new links, the constructed links may not be required for generating the final links to be returned. Therefore, D is less than C. Effective link ratio evaluates the extent that generated links are required for generating final links that are expected.

To calculate used link ratio and effective link ratio, we assume that every sub-formula has the same probability to be evaluated to be true or false, and that they 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. Both have a 50 % probability of being evaluated as true or false, and the truth value of f_1 is independent of that of f_2. Another assumption is about the goal on every sub-formula. We assume that every 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 the difference between the new semantics and the original link generation semantics in terms of used link ratio (or U-LR for short) and effective link ratio (or E-LR for short).

From the comparison in Table 1, we see that the new semantics exceed the original link generation semantics in both used link ratio and effective link ratio. The improvement (from 70.8% to 91.7%) in the used link ratio shows that more links returned from sub-formulae are used in the later link generation for the whole constraint, whereas the improvement (from 33.3% to 100%) in the effective link ratio shows that more links generated by a formula are actually required for generating the final links for the whole constraint. Therefore, both results imply that the generation of redundant links is greatly reduced by our new semantics.

Table 1. Conciseness analysis

	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*)	83.3	75	100	33.3
not (*f*)	100	100	100	33.3
Average	**91.7**	**70.8**	**100**	**33.3**

What is worth noticing is that we do not realize a used link ratio of 100%. This is because there are still some, although much fewer, redundant links generated during the context validation for "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 $\top$ according to our goal semantics in Section 3.1. If the first sub-formula is evaluated to be true and the second one to be false, then the first sub-formula would return satisfaction links (if any) and the second one would return violation links (if any) according to Theorem 1. This is Case (3) in the original link generation semantics in Section 2.1. However, from the link generation semantics of Case (3), we observe that only the links returned from the second sub-formula are used but those returned from the first one are abandoned. These abandoned links are redundant.

In summary, we only realize partial conciseness, but the comparison results from Table 1 are still promising, considering that we have only incorporated a small modification into the original link generation semantics, and that the modification only affects universal and existential formulae (see Section 3.2).

5. Related work

Ubiquitous computing is receiving increasing attention from academic researchers and software developers for its adaptive computing capability. This capability is built on the contexts acquired from the environments of ubiquitous computing. From an early 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 provided 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 projects, for example, Gaia [14] [15], have been proposed to provide middleware support for ubiquitous computing, but they mainly focus on the organization of, and cooperation among, computing devices and services. Other infrastructures such as the hybrid observer model in [6] and the mobility coordination model in [8] [11] are mostly concerned with the support of context processing, reasoning and programming. Little attention has been paid to the context validation for adaptive ubiquitous systems.

Validating the contexts of adaptive ubiquitous systems follows a constraint perspective that concerns to some extent the automated theorem proving issue. Related work [10] considers how to realize automated theorem proving by mapping logics to an AI (that is, artificial intelligence) problem. Theorem-proving techniques can be also used for automated synthesis of combinational logic [9]. 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, that is, how to identify and reduce unnecessary effort in the logic evaluation, in particular, during the context validation in ubiquitous computing.

Context validation also shares observations with consistency management for many software artifacts, which include application profiles [1], triggered actions [2], data structures [3], and UML models [5] [13]. Our previous work [18] proposed to use context-related ECA (that is, Event-Condition-Action) rules and semantics reasoning to capture inconsistent contexts with the support of an ontology database. The work has limitations in expressive power and detection performance. Our follow-up work [20] studied how to detect inconsistent contexts based on an incremental constraint checking technique. This paper further enhances this technique by augmenting the link generation with goals. This can substantially reduce the generation of redundant links.

6. Conclusion

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

We evaluate our approach through proofs and qualitative analysis. The evaluation shows that our technique may potentially generate much fewer redundant links than the original approach. When our results can be further confirmed, it greatly improves the applicability of context validation to adaptive ubiquitous systems. In the future, we will consider ways to eliminate generating redundant links during formulae evaluation, and conduct more experiments with practical case studies to confirm our preliminary findings.

Acknowledgment

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, October 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, January-February 2003.

[3] B. Demsky and M. Rinard, “Data Structure Repair Using Goal-Directed Reasoning”, *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”, *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments*, pp. 114-128, Dublin, Ireland, December 1999.

[5] A. Egyed, “Instant Consistency Checking for the UML”, *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”, *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”, *Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications*, pp. 77-86, Orlando, USA, March 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, July 1985.

[10] D.W. Loveland, “Automated Theorem Proving: Mapping Logic into AI”, *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems*, pp. 214-229, Knoxville, Tennessee, USA, October 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, July 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, January 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, April-June 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”, *Proceedings of the 20th International Conference on Software Engineering*, pp. 230-239, Kyoto, Japan, April 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, January 1992.

[18] C. Xu and S.C. Cheung, “Inconsistency Detection and Resolution for Context-Aware Middleware Support”, *Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pp. 336-345, Lisbon, Portugal, September 2005.

[19] Chang 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, October 2005.

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