ASERE: Assuring the Satisfiability of Sequential Extended Regular Expressions*

Naiyong Jin and Huibiao Zhu

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
{nyjin,hbzhu}@sei.ecnu.edu.cn

Abstract. One purpose of Property Assurance is to check the satisfiability of properties. The Sequential Extended Regular Expressions (SEREs) play important roles in composing PSL properties. The SEREs are regular expressions with repetition and conjunction. Current assurance method for LTL formulas are not applicable to SEREs.

In this paper, we present a method for checking the satisfiability of SEREs. We propose an extension of Alternating Finite Automata with internal transitions and logs of universal branches (IAFA). The new representation enables *memoryful* synchronization of parallel words. The compilation from SEREs to IAFAs is in linear space. An algorithm, and two optimizations are proposed for searching satisfying words of SEREs. They reduce the stepwise search space to the product of universal branches' guard sets. Experiments confirm their effectiveness.

Keywords: Alternating Automata, Satisfiability, Memoryful Synchronization.

1 Introduction

The correctness of functional specifications is important. Conflicting properties will put design and verification effort into vain. Property Assurance [4] [20] aims at a methodology for checking the existence of behaviors which satisfy a set of properties, and the satisfaction of given properties for all possible behaviors of a system. With property assurance, designers can develop a better understanding and have stronger confidence in their specifications.

PSL [11] is an industry standard specification language (IEEE-1850) for circuit and embedded system design. The core logic of PSL is an extension of LTL with the Sequential Extended Regular Expressions (SEREs). SEREs are operands of PSL's LTL operators. Therefore, the satisfiability of SEREs is critical to the correctness of PSL specifications. Bloem *et al.* have addressed the satisfiability problem of LTL in [20]. In this paper, we present a method for Assuring the satisfiability of clocked SEREs [9] (ASERE).

* This paper is supported by the "863" project (2007AA01302) of Ministry of Science and Technology of China, and the "Dengshan Project"(067062017) of the Science and Technology Commission of Shanghai Municipality.

T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS 17, pp. 237–251, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Related Work

The automata-theoretic approach for proving the satisfiability (SAT) of regular expressions starts with converting an expression r into an automaton A_r, then checks the non-emptiness of the language L accepted by A_r, that is $L(A_r) \neq \phi$. The formula holds if we can find a word w such that $w \in L(A_r)$.

The topic of automata construction for intersection-extended regular expressions (IERE) was carefully studied by Ben-David *et al.* in [1]. Firstly, they transform an IERE r into an *Alternating Finite Automaton* (AFA), then to a *Non-deterministic Finite Automaton* (NFA). The state complexity of such a transformation is $2^{O(|r|)}$, where $| r |$ refers to the size of r. The construction from SEREs to AFA is linear. The exponential increase takes place in transforming the AFA of $r_1 \&\& r_2$ to NFA. $r_1 \&\& r_2$ requires that words of r_1 and r_2 should both start and terminate simultaneously. One important reason that forces them to further transforming AFA to NFA is due to the fact that the acceptance of $r_1 \&\& r_2$ depends on the universal activeness of the accepting states of r_1 and r_2. However, a traditional AFA does not have the accepting states for universal branches.

Vardi *et al.* gave a systematic analysis on the time complexity of the language emptiness problem by automata-theoretic approach in [22] [15] [13]. For traditioanl alternating automata, their time complexity is in exponential time. In the last ten years, the progress was driven by the LTL model checking [2] [17] [10]. The expressiveness of LTL is equivalent to that of the star-free ω-regular expressions. Runs of the alternating automata for LTL are *memoryless* [13]. But SEREs are not *memoryless*. They can repeat infinitively, and conjunct with other SEREs to form new ones, like $r_1 \&\& r_2$. Therefore, we must explore new methods for the SAT problem of SEREs.

Contribution

In summary, the contribution of this paper includes:

1. An extesion of Alternating Finite Automata with internal transitions and universal branching logs. The new representation enables *memoryful* synchronization of parallel words. The construction from SEREs to IAFA is in linear space.
2. An algorithm for proving the satisfiability of SEREs by IAFA. Two optimization methods are proposed. They reduce the stepwise search space to the product of universal branches' guard sets.

The rest of this paper is organized as follows. Section 2 presents the syntax and the semantics of SERE. Section 3 reviews the evolution of Alternating Automata, and justifies their use in representing SEREs. The new features to be enhanced for the satisfiability problem of SERE are explored. Section 4 introduces the Internal-transition-enhanced AFA (IAFA) as the operating representation of SEREs. In Section 5, we develop a DPLL-like stepwise search algorithm and use zchaff [18] as the building block in each step. After that, we propose two optimizations which are able to reduce the search space of each step from exponential to linear. Experiments and analysis are presented in Section 6. And Section 7 discusses our future works.

We leave the detailed proofs, the automaton constructions and the diagrams in a technical report which, in along with the source code of `ASERE`, is available at [23].

2 SERE: Syntax and Semantics

The syntax of SEREs supported by ASERE is defined recursively as follows:

Definition 2.1. *(SEREs)*

$r ::= \epsilon$ *empty expression*
$\mid b$ *Boolean expression*
$\mid \{r\}$ *bracketed SERE*
$\mid r;\ r$ *sequential concatenation*
$\mid r\&\&r$ *length − matching conjunction*
$\mid r\&r$ *non − length − matching conjunction*
$\mid r \mid r$ *disjunction*
$\mid r[*]$ *repeating r for zero or more times*
$\mid r[*k]$ *repeating r for k times*
$\mid r[*n : m]$ *repeating r for n to m times, n can be* 0, *and m can be infinitive*

In accordance with [7], we define the semantics of *SERE* with finite and infinite words over $\Sigma = 2^V \cup \{\epsilon\}$, where V refers to the predicate variable set, and ϵ refers to an empty word. We denote by $Bool_V$ the set of Boolean expressions over V, $Bool_V = 2^{2^V}$.

We denote a letter from Σ by l, and a word by w. $\sharp w$ denotes the length of w. The empty word ϵ has length 0, a finite non-empty word $w = l_0 l_1 \ldots l_n$ has length $n + 1$, and an infinite word has length ∞. We denote the $(i + 1)$th letter of w by w^i, the suffix of w starting at w^i by $w^{i\cdots}$, and the finite sequence of letters starting from w^i and ending in w^j by $w^{i\cdots j}$. If $i > \sharp w$, then $w^i = \epsilon$. $w_1 w_2$ denotes the sequential concatenation of w_1 and w_2. If w_1 is infinite, then $w_1 v = w_1$. W^* denotes finite words whose letters are from W. For the empty word ϵ, $w\epsilon^* = \epsilon^* w = w$.

$l \Vdash b$ denotes that the letter l satisfies the Boolean expression b. The Boolean satisfaction relation $\Vdash \subseteq \Sigma \times Bool_V$ behaves in the usual manner.

Definition 2.2. *(Boolean Satisfaction) For letter $l \in \Sigma$, atomic proposition $p \in V$, and Boolean expressions $b, b_1, b_2 \in Bool_V$, then*

1. $l \Vdash p$ **iff** $p \in l$
2. $l \Vdash \neg b$ **iff** $l \nVdash b$
3. $l \Vdash$ **true** $\wedge\ l \nVdash$ **false**, *4)*
4. $l \Vdash b_1 \wedge b_2$ **iff** $f \Vdash b_1 \wedge f \Vdash b_2$

$w \models r$ denotes that the word w satisfies the SERE r tightly.

Definition 2.3. *(SERE Tight Satisfaction)*

$w \models \epsilon$ **iff** $\sharp w = 0$
$w \models b$ **iff** $\sharp w = 1 \wedge w^0 \Vdash b$
$w \models \{r\}$ **iff** $w \models r$
$w \models r_1;\ r_2$ **iff** $\exists w_1, w_2 \bullet w = w_1 w_2 \wedge w_1 \models r_1 \wedge w_2 \models r_2$
$w \models r[*]$ **iff** $w \models \epsilon \vee \exists w_1 \neq \epsilon, w_2 \bullet w = w_1 w_2 \wedge w_1 \models r \wedge w_2 \models r[*]$
$w \models r[*k]$ **iff** $w \models \overbrace{r;\ \ldots;\ r}^{k \text{ times}}$
$w \models r[*n : m]$ **iff** $w \models \overbrace{r;\ \ldots;\ r}^{n \text{ to } m \text{ times}}$
$w \models r_1\ \&\&\ r_2$ **iff** $w \models r_1 \wedge w \models r_2$

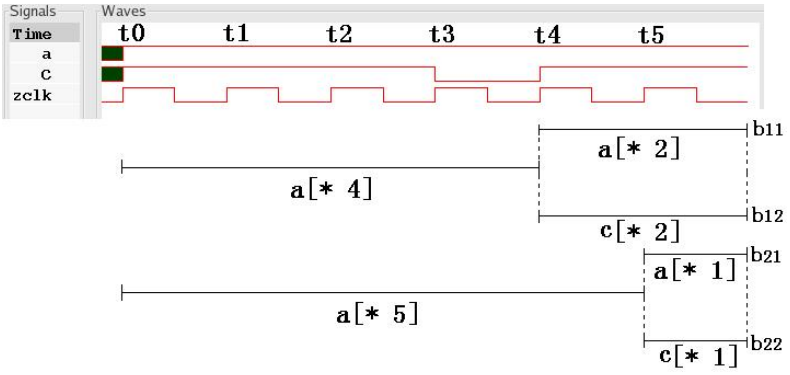

Fig. 1. Runs of $a[*4:5];\ \{a[*1:2]\&\&c[*1:2]\}$

$w \models r_1 \mid r_2$ **iff** $w \models r_1 \vee w \models r_2$

$w \models r_1 \,\&r_2$ **iff** $w \models \{r_1 \,\&\&\, \{r_2;\ \ \textbf{true}[*]\}\} \mid \{\{r_1;\ \ \textbf{true}[*]\} \,\&\&\, r_2\}$

SERE can describe non-deterministic behaviors succinctly [16]. One may have different interpretations of a SERE over a finite word. For instance, **Fig.1** illustrates two interpretations of

$$a[*4:5];\ \{a[*1:2]\&\&c[*1:2]\}$$

over the word given in the wave form. For the first interpretation, after 4 clocks of a, it branches at t_4, and its branches, $a[*2]$ (b_{11}) and $c[*2]$ (b_{12}), last two clocks. For the second interpretation, after 5 clocks of a, it branches at t_5, and its branches, $a[*1]$ (b_{21}) and $c[*1]$ (b_{22}), last only one clock cycle. Though b_{12} and b_{22} all reach their accepting states, b_{11} should length-match with b_{12}, not b_{22}. This example tells us that word length is not sufficient enough for us to synchronize words of parallel SEREs.

3 A Review of Alternating Automata

By **Definition 2.3**, we know that SERE supports two types of branches. They are the existential branch, $r_1 \mid r_2$, and the universal branch, $r_1 \&\& r_2$. No cooperation takes place between spawned processes in both types of branches, until the time comes to decide the acceptance of the input. That kind of concurrency is termed as *weak concurrency* [12].

NFA is the counterpart of *existential* branching in the automaton world. AFA further enrich NFA with universal branches. An *AFA on finite words* is a tuple of $A =< \Sigma, S, s_0, \rho, F >$, where Σ is the input letter set, S is a finite set of states, s_0 is the initial state, and F is a finite set of accepting states. $\rho : S \times \Sigma \rightarrow 2^{2^S}$ is a transition function. The target of a transition is not a state of S, but a subset of S. A state may transit to multiple target sets to express non-deterministic behavior. For instance, a transition $\rho(s, l) = \{\{s_1, s_2\}, \{s_3, s_4\}\}$ states that A accepts a letter l from state s, and it activates both s_1 and s_2, or both s_3 and s_4. Chandra *et al.* [5] have proved that AFA is doubly exponentially more succinct than Deterministic Finite Automata (DFA). Thus, we have the following observation.

Observation 3.1. *AFA is a promising representation of SERE.*

Traditionally, runs of AFAs are expressed in terms of trees [22] [13]. A finite tree is a finite non-empty set $T \subseteq \mathbb{N}^*$ such that forall $x \cdot c \in T$, with $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, we have $x \in T$. The elements of T are called nodes, and the empty word ϵ is the root of T. The level of a node x, denoted $\mid x \mid$, is x's distance from the root ϵ. Particularly, $\mid \epsilon \mid = 0$. A run of A on a finite word $w = l_0 \cdot l_1 . \, . \, l_{n-1}$ is a S-labelled tree $< T_R, R >$, where T_R is a tree and $R : T_R \rightarrow S$ maps each node of T_R to a state in S. For $< T_R, R >$, the followings hold:

- $R(\epsilon) = s_0$
- Let $x \in T_R$ with $R(x) = s$ and $\rho(s, l_{|x|}) = \mathbb{S}'$. There is a (possible empty) set $S_K = \{s_1, \ldots, s_k\}$ such that there exists a $S_y \subseteq S_K$ with $S_y \in \mathbb{S}'$, and for all $1 \leq c \leq k$, we have $x \cdot c \in T_R$ and $R(x \cdot c) = s_c$

A word W is accepted iff there is an *accepting* run on it. A run is *accepting* if all nodes at depth n are labelled by states in F.

By the above two statements, we can imply that in AFA, there is no accepting state for universal branches. Usually, there should be certain extra-mechanism which monitors whether the acceptance condition holds or not. This method is not elegant in addressing expressions like $\{\{r_1 \&\& r_2\}; \ r_3\}$, where r_3 starts after an accepting run of $\{r_1 \&\& r_2\}$. A better ways seems to be setting a special state whose activeness indicates a successful synchronization of r_1 and r_2. With the state, we can concatenate the automata of $\{r_1 \&\& r_2\}$ and r_3 by usual approaches. Those acceptance states are internal. They are not activated by input letters.

Observation 3.2. *The synchronization states for universal branches are necessary for keeping the elegance of automata-theoretic approach.*

Another weakness of tree-represented AFA is that AFA do not constrain the breadth of a level. An active state will move to sets of target states whenever an input letters satisfy some corresponding guards. So with the verification process continuing, the memory grows without restrictions.

Kupferman and Vardi [14] [13] proposed to merge *similar* target states of transitions into a single one. That results in representing runs of AFAs by Directed Acyclic Graphs (DAG). For two nodes x_1 and x_2, they are *similar* iff $\mid x_1 \mid = \mid x_2 \mid$ and $R(x_1) = R(x_2)$. Recently, the DAG approach [8] [3] is accepted in static verification (model checking) for LTL properties. The intuition is that the LTL formulas are equivalent to star-free words. For AFAs converted from LTL formulas, they do not have loops other than self loops. Hence, runs of traditional LTL-AFAs are *memoryless* [13]. During verification, one only needs to look in the future, but never the past. In other words, *similar* states correspond to the same future mission: to accept the suffixes which satisfy a common property.

Kupferman *et al.* represent a *memoryless* run $< T_R, R >$ by a DAG $G_R =< V, E >$, where

1. $V \subseteq S \times \mathbb{N}$ is such that $< s, l > \in V$ iff there exists $x \in T_R$ with $\mid x \mid = l$ and $R(x) = s$. For example, $< s_0, 0 >$ is the only vertex of G_R in $S \times \{0\}$.

2. $E \subseteq \bigcup_{l\geq 0}(S \times \{l\} \times (S \times \{l+1\}))$ is such that $E(< s, l >, < s', l+1 >)$ iff there exists $x \in T_R$ with $\mid x \mid = l$, $R(x) = s$ and $R(x.c) = s'$ for some $c \in \mathbb{N}$.

Configurations $C_i \subseteq S$ are sets of active states, where i refers to the level of a DAG. It is evident that, by DAG, every configuration contains at most $\mid S \mid$ states which are roots of different subtrees. A DAG is acceptable if $C_i \subseteq F$ holds.

The branches of AFA's DAGs resemble the requirements of universal choices [8]. A DAG is just a single path through the existential choices of an AFA. One may have to try breadth-first search, depth-first search or backward search [6] in looking for an accepting run. That is why the *EXPTIME* complexity of AFA is unavoidable [21].

We find that if a memory can distinguish different universal branching histories, then the runs with different branching histories will not be able to activate the synchronization states. Thus a *memoryful* DAG with synchronization states can enable concurrent runs of both *existential* and *universal* branches.

Observation 3.3. *A* memoryful *DAG can accommodate all possible runs of an AFA.*

4 Representing SEREs by IAFA

We formalize the Internal-transition-enhanced AFA (IAFA) as follows

Definition 4.1. *An IAFA is a tuple of* $A = (V, \Sigma_A, S, \mathbf{H}, s_0, \rho, F)$*, where*

- V is a set of variables of the SERE under assuring.
- S is a set of states.
- $\mathbf{H} = 2^{\mathbb{N}^*}$ is a set of historical log sets. A historical log $h \in \mathbb{N}^*$, namely $h = t_0 t_1 \ldots t_{n-1} t_n$, is a finite sequence of time-stamps. It records the important timing information of runs which make a state active. We have H range over the sets of historical logs.
- $\Sigma_A = Bool_V \cup FOP_{SH}$ is the letter set of A. FOP_{SH} refers to the first order predicates over $S \cup S \times \mathbf{H}$. We distinguish $\mathbf{true}_V$ and $\mathbf{true}_{SH}$. $\mathbf{true}_V$ stands for logic **true** over V and $\mathbf{true}_{SH}$ stands for logic **true** over $S \cup S \times \mathbf{H}$.
- s_0 is the initial state.
- F is the set of states for tight acceptance.
- In IAFA, a transition ρ is in type of $S \times \Sigma_A \times U \times 2^S$, where
  - Σ_A specifies the guarding conditions.
  - U is a set of assignments which update historical logs whenever a transition takes place.
  - The target of a transition is a subset of S. All elements of the subset should be active after the transition.

We classify IAFA transitions into *external* and *internal* ones. Our SEREs are synchronous. The external transition can be triggered only on clock events. Therefore the guarding conditions of external transitions are in form of $Bool_V \wedge clock\_event$. For conciseness, we do not attach the *clock_event* in reasoning external transitions. The guarding conditions of *internal* transitions are in type of FOP_{SH}. We call a special type of internal self-loop transitions as τ transitions. τ-typed transitions are in form like

$(s, g_s, u_s, \{s\})$. A τ transition keeps a state active between two external transitions. We have ρ_e, ρ_i and ρ_τ to represent the set of *external*, *internal* and τ transitions. Notation $s.\tau$ denotes s's τ transition.

A configuration C gives out the status and historical logs of active states. Configurations are in type of $2^{S\times\mathbf{H}}$. Notation $C.ST$ denotes the set of active states in C, that is $C.ST = \{s \mid (s, H) \in C\}$. In a configuration, a state can have only one set of historical logs. That is if $(s, H_1) \in C$ and $(s, H_2) \in C$, then $H_1 = H_2$. Therefore, we represent the historical log set of s by $s.H$. For two active states s_1 and s_2, they are synchronizable if and only if they share common historical logs. That is $s_1.H \cap s_2.H \neq \phi$

A predicate $g \in FOP_{SH}$ holds under the configuration C, if there exists some pairs $(s, s.H) \in C$ which make g **true**. We represent the case by $C \models g$.

Definition 4.2. *Let $A = (V, \Sigma_A, S, H, s_0, \rho_A, F)$ be a IAFA, runs of A over a word $w = w_0 w_1 w_2 \ldots w_k$ is a sequence of configurations $\Delta = C_0 C_1 \ldots C_n$, where*

1. $C_0 = \{(s_0, \phi)\}$
2. *Given a letter w_i, if $\exists\, s \in C_i.ST, (s, g, u, S') \in \rho_e$, and $w_i \Vdash g$, then $S' \subseteq C_{i+1}.ST$*
3. *Given a state $s \in C_i.ST$, if $(s, g, u, S') \in \rho_i$, and $C_i \models g$, then $S' \subseteq C_{i+1}.ST$.*
4. *Given a state $s \in C_i.ST$, if $s.\tau \in \rho_\tau$, and $s \in C_{i+1}.ST$, then for all $(s, g, u, S') \in \rho_i$, $C_i \models \neg g$ holds.*

The second and the third clause of **Definition 4.2** state that for external and internal transitions, whenever their guarding conditions hold, they should take place immediately. That amounts to an identical treatment towards both universal and existential branches. However, such a treatment will not impact the correctness of assuring SEREs. Because in SEREs, the acceptance of universal branches asks for synchronization of peers. For $r_1 \&\& r_2$, if the branches of r_1 and r_2 are not able to synchronize on their termination, then the condition for the tight satisfaction of $r_1 \&\& r_2$ will fail.

Due to the existence of clock events, the external and internal transitions take place interleavingly. An external transition increases the word length. However, an internal

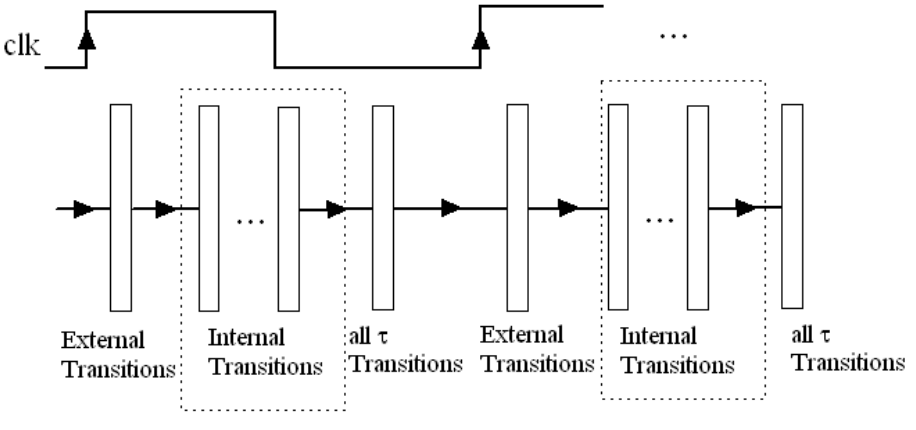

Fig. 2. Transitions of IAFAs

transition resembles the empty letter ϵ. The last clause of **Definition 4.2** says that a τ transitions are triggered only when the corresponding states do not have other enabled internal transitions. **Fig.2** illustrates the running patterns of IAFAs. Supposing the clock events of external transitions are the posedges of clk, then on each *posedge*(*clk*), an IAFA will sample the values of V and trigger enabled external transitions. After that, there comes a sequence of internal transitions until all states have only τ transitions enabled.

Proposition 4.3. *For each SERE r, there is an IAFA A_r, such that $w \models r$* **iff** *$w \in L(A_r)$ and A_r has $O(|\ r\ |)$ states.*

Proof: In [23].

Here, we give out the IAFA construction for r_1 && r_2, as illustrated in **Fig. 3**. Given $A_i = (V, \Sigma, S_i, \mathbf{H_i}, s0(r_i), \rho_i, \{ss(r_i)\})$ are IAFAs of r_i, then

$S(r_1 \&\& r_2) = S(r_1) \cup S(r_2) \cup \{s_0, s_f\}$

$s_0(r_1 \&\& r_2) = s_0$

$\rho_A(r_1 \&\& r_2)$

$= \rho_A(r_1) \cup \rho_A(r_2)$

$\quad \cup\{(s_0,\ \mathbf{true}_{LV}, u_i, \{s_0(r_i)\}) \mid$

$\qquad u_i = \{l\_s\_0(r_i).H.include(push(l\_s\_0.H, t))\}\}$

$\quad \cup\{(ss(r_1), g, u, \{s_f\}), (ss(r_2), f\_ss\_1, \phi, \phi)\}$

$F(r_1 \&\& r_2) = \{s_f\}$

$\quad$ where, $g = f\_ss\_r\_2 \wedge (f\_ss\_r\_1.H \cap f\_ss\_r\_2.H \neq \phi)$

$\qquad u = \{T := f\_ss\_r\_1.H \cap f\_ss\_r\_2.H;$

$\qquad\quad f'\_s\_f := 1;\ f'\_s\_f.H.include(T.pop);\ \}$

In the above clause, all target states of s_0 inherit s_0's history logs, and have a new universal branching time as the last time stamp of their history logs. s_f is the tight accepting state of $r_1 \&\& r_2$. Before reaching s_f, we shall synchronize on the tight accep-

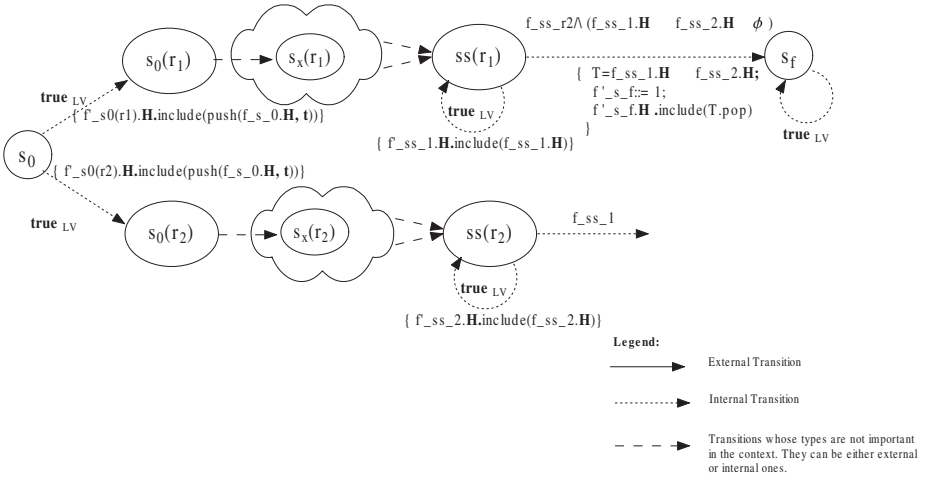

Fig. 3. The IAFA of $r_1 \&\& r_2$

tance of both r_1 and r_2. For the transition from $ss(r_1)$, the guarding condition
$f\_ss\_r\_2 \wedge (f\_ss\_r\_1.H \cap f\_ss\_r\_2.H \neq \phi)$
conveys the idea that for a successful synchronization, both $ss(r_1)$ and $ss(r_2)$ shall be active and both of them have common histories of universal choices. The update part activates s_f and assigns the $T.pop$ as histories to s_f. T is a temporary variable. It is just the common histories of $ss(r_1)$ and $ss(r_2)$. $T.pop$ removes the branching time which is pushed into logs on leaving s_0. Once the automata reaches s_f, both $ss(r_1)$ and $ss(r_2)$ are deactivated.

By this example, we can see the effect of the τ transitions. According to the semantics of PSL [7], The length of a clocked SERE is counted on clock events. Internal transitions within two external transitions do not take time. It may takes different numbers of internal transitions to reach $ss(r_1)$ and $ss(r_2)$. With the τ transition, $ss(r_1)$ will not miss the synchronization with $ss(r_2)$ only if $ss(r_2)$ could be active in the current clock cycle.

Proposition 4.4. *The time complexity of the satisfiability problem of SERE is* $O(2^{d \cdot |V|})$*, where* $|V|$ *is the number of variables in V and d is the search depth.*

Proof: In [23].

5 The Implementation and Optimization

The search process of ASERE follows the classic DPLL algorithm [19], which is the base of most Boolean SAT solvers. For SAT solvers, a *conflict* is an implied assignment in which some variables are assigned both **true** and **false** . If no conflict is detected in the preprocessing, a DPLL SAT solver starts by assigning a value to an unassigned variable. If all variables are assigned, a solution is found. Otherwise, the solver will *deduce* values of other variables through a process called *Boolean Constraint Propagation* (BCP). If a conflict is detected, it will perform *backtrack* to undo some decisions. If all decisions have to be undone, then one can conclude the unsatisfiability of a boolean expression. The *deduce* and *backtrack* processes form the inner loop. It stops if no more deduction is possible and the deductions do not imply conflicts. After that, the solver will decide the next branch provided that there are still unassigned variables.

Our ASERE algorithm is similar. The target of each decision is to find a letter which can trigger the *external* transitions such that the IAFA can move forward. The *internal* transitions take the role of BCP. The search process of ASERE is illustrated in **Fig.4**.

1. Line 1 says that ASERE will initially try for internal transitions.
2. If an internal trial returns TRIAL_PASS, ASERE will commit the trial by *configuration_update()*. If in sat_check, no states of F becomes active, ASERE will continue for more internal transitions.
3. The codes from line 13 to line 20 state that if ASERE finds that all internal transitions are τs, it shifts to external trials. Before the next external trial, ASERE will check whether it reaches the maximal search depth. If so, ASERE *backtrack*s to the previous configuration for the last external trial by *configuration_retreat()*.

```
trial_type = INTERNAL_TRIAL;
while( !terminate ){
    cont = false;
    trial_ret = TRY_STEP_FORMARD(trial_type);
    switch ( trial_ret ){
      case EXTERNAL_FAIL:
             if (current_search_depth == 0)
               terminate = true;
             else configuration_retreat();
             cont = true;
             break;

      case INTERNAL_TA0:
             configuartion_update(trial_type);
             if ( reach_max_search_depth )
               configuration_retreat();

             trial_type = EXTERNAL_TRIAL;
             cont = true;
             break;

      case INTERNAL_FAIL:
             if (current_search_depth == 0)
               terminate = true;
             else configuration_retreat();

             cont = true;
             break;

      case TRIAL_PASS:
      default        :;
    }

    if ( cont )
      continue;

    configuration_update(trial_type);
    satisfied = sat_check();
    if ( !satisfied ){
      if ( reach_max_search_depth )
        configuration_retreat();

      trial_type = INTERNAL_TRIAL;
    } else return SAT;
}
return UNSAT;
```

Fig. 4. The Algorithm of ASERE

4. For the external trials, ASERE calls a SAT solver to find a letter l which should satisfy the disjunction of the guarding conditions of the active states. That is

$$l \models \bigvee_k g_k \ \ with \ \ (s, g_k, u, S') \in \rho_e \wedge s \in C_i.ST \tag{1}$$

   If such a l does exist, the external trial will return TRIAL_PASS and some external transitions of active states take place. Accordingly, ASERE updates the configuration as specified in line 37.
5. However, if the external trial returns EXTERNAL_FAIL and the search depth is greater than 0, ASERE will *backtrack* as well. That amounts to the deduction that the prefix word after the last external trial can not lead to a satisfying word. Then, ASERE continues with external trials for other words.
6. The codes from lines 38 to line 44 state that after updating a configuration, if some states of F become active, then a tight satisfying word is found. Otherwise, ASERE will try all possible internal trials to propagate the influence of the last letter. As in 3, *backtrack*ing is necessary if ASERE reaches the maximal search depth.

Proposition 4.4 tells us that the time complexity of SERE's satisfiability problem is $(2^{|V|})^d$. The base $2^{|V|}$ gives the search space of each *external trial*. The complexity increases exponentially along with the search depth. The exponent d is unavoidable. Our optimization effort focuses on reducing the step-wise search space by extra constraints. We propose 2 optimization methods, they are the the *Post-Trial Check* and the *No-Repeated-Transition Check*.

Opt.1(*Post-Trial Check*)

The motivation is to utilize the structure information of SEREs. For instance, to the *length-matching conjunction* $r_1 \&\& r_2$, whenever an external transition of r_1 (r_2) takes place, then at least one external transition of r_2 (r_1) has to take place as well. If a branch has no more active states after a trial, it is impossible for further runs to synchronize. And we can halt the search process earlier. Therefore, we must ensure the simultaneous activeness of universal branches' states. That criteria applies to *internal trials* too.

Given a temporary configuration C and an IAFA A_r generated from r, predicate $PTC(C, A_r)$ holds if C passes the *Post-Trial Check*. We setup a temporary configuration C'_i for the result of C_i after the current trial. Only when C'_i passes PTC, can *configuration_update()* commits the trial by assigning C'_i to C_{i+1}. We define $PTC(C, A_r)$ as follows.

Definition 5.1.

- For $r = \epsilon$, $r = b$, $r = r_1[*n]$, $r = r_1[*n : m]$ and $r = r[*]$
  $PTC(C, A_r) =_{df} \bigvee_{s \in S(r)} s \in C.ST$
- For $r = r_1;\ r_2$ and $r = r_1 \mid r_2$
  $PTC(C, A_r) =_{df} PTC(C, A_{r_1}) \vee PTC(C, A_{r_2}) \bigvee_{s \in S(r) - S(r_1) - S(r_2)} s \in C.ST$
- For $r = r_1 \&\& r_2$ and $r = r_1 \& r_2$[1]

[1] Please refer the automata construction of $A_{r_1\ \&\&\ r_2}$ and $A_{r_1\ \&\ r_2}$ for s_0, s_f, s_{f1} and s_{f2}.

$$PTC(C, A_r) =_{df} s_0 \in C.ST \vee s_f \in C.ST \vee ((PTC(C, A_{r_1}) \vee s_{f1} \in C.ST) \wedge (PTC(C, A_{r_2}) \vee s_{f2} \in C.ST))$$

Now, let us have a look at the time complexity of Opt.1. Suppose r_1 and r_2 are two concurrent SEREs, C is a configuration, and $CG_1(C)$ and $CG_2(C)$ are the *candidate guard* sets of A_{r1} and A_{r2}.

$$CG_i(C) = \{g \mid \exists\, u, S, s \in C.ST \cap S(r_1) \bullet (s, g, u, S) \in \rho(r_i)\}$$

Let $LET(C)$ be the set of letters which can trigger external transitions of C's active states and the resulting C' can pass the *Post-Trial Check*. Then,

$$LET(C) = \{l \mid \exists\, g_{1i} \in CG_1(C), g_{2j} \in CG_2(C) \bullet l \Vdash g_{1i}g_{2j}\} = \bigcup_{g_{1i}, g_{2j}} \{l \mid l \Vdash g_{1i}\} \cap \{l \mid l \Vdash g_{2j}\}$$

Let $size(g)$ be the size of the letter set whose elements satisfy g. Then,

$$\mid LET(C) \mid \leq \sum_{g_{1i}, g_{2j}} min(size(g_{1i}), size(g_{2j})) \quad (2)$$

Formula (2) gives the upper bound of the letter space restricted by Opt.1 in each external trial. The letter set characterized by g is a subset of 2^V. Though the exponent in $size(g)$ is not removed, if the biggest $size(g)$ are small, then $\mid LET(C) \mid$ will be small.

Opt.2(*No-Repeated-Transition Check*)

Opt.1 reduces the search space from 2^V to $LET(C)$. It is observed that there is still redundancy in $LET(C)$. It is possible for different letters to trigger an identical bunch of external transitions which lead to identical suffix words. It is reasonable to cancel a trial if it does not contribute new transitions. That is the motivation of our second optimization (Opt.2), the *No-Repeated-Transition Check*. Opt.2 reduces the search space to the product of the guard sets of universal branches. In terms of Opt.1, the search space of Opt.2 is $CG_1(C) \times CG_2(C)$. Consequently, its size is $\mid CG_1(C) \mid \times \mid CG_2(C) \mid$. Now, we get an algorithm whose time complexity is linear in each external trial. However, Opt.2 may not always reduce the number of external trials. Because, the check is applied after external trials.

6 Experiments and Analysis

By **Proposition 4.3**, the size of A_r is linear to $\mid r \mid$, our experiment focuses on the time aspect in concluding an unsatisfiable SERE and searching finite words of a satisfiable one. We carry out the algorithm comparison on a ThinkPad with dural 1.83GHz CPUs, and 2GB RAM. The parallel feature of the machine is not exploited. We adopt *zchaff* [18] (version 2007.3.12) as the engine for external trials. Currently, there is no standard benchmark for the satisfiability of SEREs. We forge the test cases with the object as covering as many SERE constructs as possible.

Table.1.2.3.4 demonstrate some test results. In those tests, the minimal search depth is 1, the maximal search depth is 22, and the upper bound for external trials is 100,000. If the value is reached before finding any satisfying word, we can not conclude on the satisfiability of the SEREs Under Assuring (SUA).

The performance of our optimizations is encouraging. If a SUA has abundant features of concurrency, as in experiment 1 and 2, the performance promotion is significant. It confirms our prediction on the algorithm complexity. The effect of *Opt.2* is dominant in experiment 3 and 4. It is rather quick in concluding the unsatisfiability of the SUAs. And the joint use of *Opt*.1 and *Opt*.2 is preferable if satisfiability is the only pursuit.

It is interesting to investigate **Table.2**, which aims at the first 30 satisfying words. We find that if we turn off *Opt*.2, the ratio of *external trial* against *internal trial* is less than 1. But it is greater than 1 when *Opt*.2 = 0. That means quite a number of *external trials* are not committed if they do not bring new transitions. Consequently, turning on *Opt*.2 can work out more diversified words.

Table 1. Experiment 1

SERE	$\{\{\{a\ xor\ b\}[*1:4]\}\&\&\{\{b\ xor\ c\}[*2:5]\};\ \{\{[*2:4];\ !a[*2:5]\}[*2]\}\&\{[*5];\ a[*4]\}\}$				
Target	To find the first satisfying word				
Opt. 2	Opt. 1	NO. External-Trial	NO. Internal-Trial	Time(s)	NO. Found
0	0	100000	11145	64.79	0
0	1	59	97	0.09	1
1	0	1357	759	0.78	1
1	1	32	82	0.07	1

Table 2. Experiment 2

SERE	$\{\{\{a\ xor\ b\}[*1:4]\}\&\&\{\{b\ xor\ c\}[*2:5]\};\ \{\{[*2:4];\ !a[*2:5]\}[*2]\}\&\{[*5];\ a[*4]\}\}$				
Target	To find the first 30 satisfying words				
Opt. 2	Opt. 1	NO. External-Trial	NO. Internal-Trial	Time(s)	NO. Found
0	0	100000	11145	65.15	0
0	1	106	420	0.36	30
1	0	49632	29066	29.19	30
1	1	3588	2743	2.78	30

Table 3. Experiment 3

SERE	$\{[*10];\ \{a\}\&\&\{!a\}\}$				
Target	To find the first satisfying word				
Opt. 2	Opt. 1	NO. External-Trial	NO. Internal-Trial	Time(s)	NO. Found
0	0	100000	33431	16.94	0
0	1	6141	9210	2.45	0
1	0	99	65	0.03	0
1	1	33	43	0.01	0

Table 4. Experiment 4

SERE	$[*10];\ \{a[*3]\}\&\&\{b;\ TRUE;\ !a\}$				
Target	To find the first satisfying word				
Opt. 2	Opt. 1	NO. External-Trial	NO. Internal-Trial	Time(s)	NO. Found
0	0	100000	20057	22.96	0
0	1	100000	103598	22.08	0
1	0	256	104	0.09	0
1	1	62	52	0.03	0

7 Future Works

In this paper, we have presented the algorithms behind the tool kit ASERE for assuring the satisfiability of SEREs. We have formalized the IAFA as the representation of SERE. Essentially, the IAFA conception is a *memoryful*, *synchronization-enabled* and *multi-tape* computing model.

We have proposed a DPLL-like search process and discussed two optimizations aiming at reducing the number of external trials. Experiments have confirmed our prediction on their performance.

In the future, we will carry out our research in the following directions.

- Enabling the fusion operator : (overlapping concatenation) in ASERE.
- Extending ASERE to LTL so that we can solve the assurance problem for formulas in PSL's simple subset. A possible solution is to combine the alternating automaton approach with the SNF approach [4] [20]. The SNF addresses the LTL constructs and the alternating automaton addresses the embedded SEREs. The challenging work lies in searching the satisfying words of r **until** b, which requires a new search for r on each clock event until the assertion of b. However, the length of the satisfying words of r is so non-deterministic that it is rather hard to decide the depth at which we can assert b .

References

1. Ben-David, S., Bloem, R., Fisman, D., Griesmayer, A., Pill, I., Ruah, S.: Automata construction algorithm optimized for PSL. Technical Report Delivery 3.2/4, PROSYD (July 2005)
2. Benedetti, M., Cimatti, A.: Bounded model checking for past ltl. In: Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 18–33 (2003)
3. Bloem, R., Cimatti, A., Pill, I., Roveri, M., Semprini, S.: Symbolic implementation of alternating automata. In: H. Ibarra, O., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 208–218. Springer, Heidelberg (2006)
4. Bloem, R., Cavada, R., Esiner, C., Pill, I., Roveri, M., Semprini, S.: Manual for property simulation and assurance tool. Technical Report Deliverable D1.2/4-5, PROSYD (2005)
5. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. Journal of ACM 28(1), 113–114 (1981)
6. Feikbeiner, B., Sipma, H.: Checking finite traces using alternating automata. Formal Methods in System Design 24(2), 101–127 (2004)
7. Fisman, D., Eisner, C., Havlicek, J.: Formal syntax and Semantics of PSL: Appendix B of Accellera's Property Specification Language Reference Manual, 1.1 edn. Accellera (March 2004)
8. Hammer, M.: Linear Weak Alternating Automata and The Model Checking. PhD thesis (2005)
9. Havlicek, J., Fisman, D., Eisner, C.: Basic results on the semantics of accellera PSL 1.1 foundation language (2004)
10. Heljanko, K., Junttila, T.A., Keinänen, M., Lange, M., Latvala, T.: Bounded model checking for weak alternating büchi automata. In: Proceedings of the 18th International Conference on Computer Aided Verification, pp. 95–108 (2006)
11. IEEE. IEEE 1850-2005 Standard for Property Specification Language (PSL) (2005)
12. Kupferman, O., Ta-Shma, A., Vardi, M.Y.: Concurrency counts (2001)

13. Kupferman, O., Vardi, M.Y.: Weak alternating automata and tree automata emptiness. In: Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, pp. 224–233 (1998)
14. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Transactions on Computational Logic (TOCL) 2(3), 408–429 (2001)
15. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM 47(2), 312–360 (2000)
16. Lange, M.: Linear time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 90–104. Springer, Heidelberg (2007)
17. Latvala, T., Biere, A., Heljanko, K., Junttila, T.: Simple is better: Efficient bounded model checking for past LTL. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 380–395. Springer, Heidelberg (2005)
18. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th Design Automation Conference (DAC 2001), pp. 530–535 (2001)
19. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in sat-based formal verification. International Journal on Software Tools for Technology Transfer 7, 156–173 (2005)
20. Roveri, M.: Novel techniques for property assurance. Technical Report Deliverable D1.2/2, PROSYD (2004)
21. Vardi, M.Y.: Alternating automata and program verification. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 471–485. Springer, Heidelberg (1995)
22. Vardi, M.Y.: An automata-theoretic approach to linaer temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
23. http://sites.sei.ecnu.edu.cn/Teachers/nyjin/e/asere_e.html