Well-structured Model Checking of Multiagent Systems*

N.V. Shilov, N.O. Garanina

Institute of Informatics Systems, Russian Academy of Science
6, Lavrentiev ave., 630090, Novosibirsk, Russia
shilov, garanina@iis.nsk.su

Abstract. We address model checking problem for combination of Computation Tree Logic (CTL) and Propositional Logic of Knowledge (PLK) in finite systems with the perfect recall synchronous semantics. We have published already an (update+abstraction)-algorithm for model checking with detailed time upper bound. This algorithm reduces model checking of combined logic to model checking of CTL in a finite abstract space (that consists of some finite trees). Unfortunately, the known upper bound for size of the abstract space (i.e. number of trees) is a non-elementary function of the size of the background system. Thus a straightforward use of a model checker for CTL for model checking the combined logic seems to be infeasible. Hence it makes sense to try to apply techniques, which have been developed for infinite-state model checking. In the present paper we demonstrate that the abstract space provided with some partial order on trees is a well-structured labeled transition system where every property expressible in the propositional μ-Calculus, can be characterized by a finite computable set of maximal elements. We tried feasibility of this approach to model checking of the combined logic in perfect recall synchronous environment by automatic model checking a parameterized example.

1 Introduction

Combinations of traditional program logics [20, 8, 25] with logics of knowledge [9, 24] become a current research topic due to the importance of study of interactions between knowledge and actions for reasoning about multiagent systems. A number of techniques for (semi)automatic processing of a number of combined logics have been under study [15, 6, 7, 23, 14, 16, 17, 13].

Paper [13] has studied the model checking problem in trace-based synchronous perfect recall systems for pairwise fusion of the program logics Computation Tree Logic extended by actions (*Act*-CTL) and the propositional μ-Calculus (μC) with the epistemic logics Propositional Logic of Knowledge for n agents (PLK_n) and Propositional Logic of Common Knowledge for n agents (PLC_n).

* Supported by joint grant RFBR 05-01-04003-a (DFG project COMO, GZ: 436 RUS 113/829/0-1) and by grant RFBR 06-01-00464-a.

'Trace-based' means that semantics of formulas is defined on traces, i.e. finite sequences of states and actions[1]. Each element of a trace represents a state of the system at some moment of time. So, 'synchronous' means that agents distinguish traces of different lengths. 'Perfect recall' means that every agent can distinguish traces with different sequences of information available for him/her. If $\mathcal{L}$ stands for any of acronym of program logics *Act*-CTL or μC, and PL$\mathcal{X}$ stands for any of acronym of epistemic logics PLK$_n$ or PLC$_n$, then let acronym $\mathcal{L}$-$\mathcal{X}$ stand for fusion of logics $\mathcal{L}$ and PL$\mathcal{X}_n$. For example, *Act*-CTL-K$_n$ denotes fusion of *Act*-CTL and PLK$_n$.

It has been demonstrated in [13] that the model checking problem in the class of finitely-generated trace-based synchronous systems with perfect recall is undecidable for *Act*-CTL-C$_n$, μPLK$_n$, and μPLC$_n$ (where $n > 1$), but is decidable for *Act*-CTL-K$_n$ (with a non-elementary lower bound). It was a 'decidability in principle', and is not oriented towards any implementation.

Paper [26] presents a 'direct' (update+abstraction)-algorithm for model checking *Act*-CTL-K$_n$ in perfect recall synchronous environments. This (update+abstraction)-algorithm has been inspirited by [21]. It is based on a simple transformation of *Act*-CTL-K$_n$ formulas into formulas of Act^{+n}-CTL (i.e. *Act*-CTL with n fresh action symbols) and on a reduction of infinite synchronous perfect recall system to finite model $TR_k(E)$ which consists of k-trees (special finite trees of height k). Thus the resulting model checking algorithm simply solves formulas of Act^{+n}-CTL on k-trees.

Unfortunately, the upper bound for size of this finite model is a non-elementary function of the size of the background finite system [26]. Hence a straightforward use of a model checker for CTL for model checking Act^{+n}-CTL on k-trees is likely to be a non-feasible task. Roughly speaking, this space is too big to be treated as a finite. It implies that for model checking Act^{+n}-CTL on k-trees it makes sense to try techniques which have been developed for infinite-state model checking.

A very popular approach to infinite-state model checking is formalism of well-structured labeled transition systems. Fundamental papers [1, 10] have proved the decidability of liveness (reachability) and progress (eventuality) properties in well-structured single action labeled transition systems. Roughly speaking, a well-structured single action labeled transition system is provided with (pre-)order where transition 'preserves' this (pre-)order, and its labeling forms cones with respect to this (pre-)order. Paper [19] has generalized cited decidability results for disjunctive formulas of the propositional μ-Calculus [18] in well-structured labeled multi-action transition systems.

In the present paper we demonstrate that model $TR_k(E)$ provided with a special sub-tree partial order forms a well-structured labeled transition system where every property expressible in the μ-Calculus can be characterized by a finite computable set of maximal trees that enjoy the property. We tried feasibility

[1] Let us remark that we work with traces that are finite sequences. Every finite sequence represents current state of the system (that is the last element of the sequence) and system's past (that is maximal prefix of the sequence).

of this approach to model checking of *Act*-CTL-K$_n$ in trace-based synchronous perfect recall synchronous environment by automatic model checking simple, but big example[2].

2 Background Logics and their Fusion

Logics we are going to discuss are propositional polymodal logics. Semantics of these logics is defined in models which are called Kripke structures or labeled transition systems (LTS); it is defined in terms of satisfiability relation $\models$.

Definition 1. Let $\{true, false\}$ be Boolean constants, *Prp* and *Rlt* be disjoint finite alphabets of propositional variables and relational symbols. *Syntax* of our logics consists of formulas which are constructed from Boolean constants, propositional variables, and connectives[3] $\neg$, $\wedge$, $\vee$ and some modalities.

Definition 2. A *transition system* (synonym: Kripke frame) is a tuple (D, I), where the domain D is a non-empty set of elements that are called states (or worlds), and the interpretation I is a total mapping $I : Rlt \to 2^{D \times D}$. For every $r \in Rlt$ an *r-run* is a maximal sequence of states $ws = s_1 \dots s_i s_{i+1} \dots$ such that for all adjacent states within the sequence $(s_i, s_{i+1}) \in I(r)$. For every finite i within ws let ws_i stands for the element s_i. *Kripke model* or *labeled transition system (LTS)* M is a triple (D, I, V), where (D, I) is a Kripke frame, and the valuation V_M maps propositional variables into subsets of D.

Definition 3. A *satisfiability relation* $\models$ between models, worlds, and formulas can be defined inductively with respect to a structure of formulas as follows[4] . For Boolean constants $w \models_M true$ and $w \not\models_M false$ for any world w and model $M = (D, I, V)$. For propositional variables we have: $w \models_M p$ iff $w \in V(p)$. For connectives $\models$ is defined in the standard manner: $w \models_M \neg\phi$ iff $w \not\models_M \phi$, $w \models_M \phi \wedge \psi$ iff $w \models_M \phi$ and $w \models_M \psi$, $w \models_M \phi \vee \psi$ iff $w \models_M \phi$ or $w \models_M \psi$. Definition of $\models$ for modalities is specific for every particular propositional polymodal logic.

A particular example of propositional polymodal logics is Propositional Logic of Knowledge (PLK) [9]. It is the simplest epistemic logic. Informally speaking, PLK is a polymodal variant of the basic propositional modal logic **S5** [3]. A special terminology, notation and models are used in this framework.

Definition 4. *(of Propositional Logic of Knowledge for n agents PLK$_n$)*
Let $n > 0$ be an integer. The alphabet of relational symbols consists of a set of natural numbers $[1..n]$ representing names of agents. Notation for modalities is: if $i \in [1..n]$ and ϕ is a formula, then $(K_i\phi)$ and $(S_i\phi)$ are formulas[5]. For

[2] The size of the initial background environment E is 120000 and the size of the corresponding generated finite model $TR_k(E)$ is about 10^{36000}.
[3] Standard abbreviations $\to$ and $\leftrightarrow$ are admissible too.
[4] Throughout the paper $\not\models$ stays for negation of $\models$.
[5] They are read as '(an agent) i knows ϕ' and '(an agent) i supposes ϕ'.

every agent $i \in [1..n]$ in every model $M = (D, I, V)$ interpretation $I(i)$ is an equivalence, i.e. a symmetric, reflexive, and transitive binary relation on D. Every model M, where all agents in $[1..n]$ are interpreted in this way, is denoted as $(D, \overset{1}{\sim}, \ldots, \overset{n}{\sim}, V)$ instead of (D, I, V) with $I(i) = \overset{i}{\sim}$ for every $i \in [1..n]$. In particular, for every $i \in [1..n]$ and every ϕ,

- $w \models_M (S_i\phi)$ iff for some w': $w \overset{i}{\sim} w'$ and $w' \models_M \phi$,
- $w \models_M (K_i\phi)$ iff for every w': $w \overset{i}{\sim} w'$ implies $w' \models_M \phi$.

Another propositional polymodal logic *Act*-CTL is a variant of the basic propositional branching time temporal logic Computational Tree Logic (CTL) [8, 4, 5] extended by action symbols.

Definition 5. *(of Act-CTL)*
In the case of *Act*-CTL the alphabet of relational symbols consists of action symbols *Act*. Notation for basic modalities is: if $a \in Act$ and ϕ is a formula, then $(\mathbf{AX}^a\phi)$ and $(\mathbf{EX}^a\phi)$ are formulas. Syntax of *Act*-CTL has also some other special constructs associated with action symbols: if $a \in Act$, ϕ and ψ are formulas, then $(\mathbf{AG}^a\phi)$, $(\mathbf{AF}^a\phi)$, $(\mathbf{EG}^a\phi)$, $(\mathbf{EF}^a\phi)$, $(\mathbf{A}\phi\mathbf{U}^a\psi)$, and $(\mathbf{E}\phi\mathbf{U}^a\psi)$[6] are formulas too. For every model $M = (D, I, V)$ semantics of 'universal' special constructors follows:

- $w \models_M \mathbf{AX}^a\phi$ iff $ws_2 \models_M \phi$ for every a-run ws with $ws_1 = w$,
- $w \models_M \mathbf{AG}^a\phi$ iff $ws_j \models_M \phi$ for every a-run ws with $ws_1 = w$ and every $1 \leq j \leq |ws|$,
- $w \models_M \mathbf{AF}^a\phi$ iff $ws_j \models_M \phi$ for every a-run ws with $ws_1 = w$ and some $1 \leq j \leq |ws|$,
- $w \models_M \mathbf{A}(\phi\mathbf{U}^a\psi)$ iff $ws_j \models_M \phi$ and $ws_k \models_M \psi$ for every a-run ws with $ws_1 = w$, for some $1 \leq k \leq |ws|$ and every $1 \leq j < k$,

Semantics of 'existential' constructors $\mathbf{EX}^a$, $\mathbf{EG}^a$, $\mathbf{EF}^a$, $\mathbf{EU}^a$ is similar but refers to some a-run.

The standard CTL is *Act*-CTL with a singleton alphabet *Act*.

We are going to define a combined Propositional Logic of Knowledge and Branching Time *Act*-CTL-K$_n$.

Definition 6. *(of Act-CTL-K$_n$)*
Let $[1..n]$ be a set of agents ($n > 0$), and *Act* be a finite alphabet of action symbols. Syntax of *Act*-CTL-K$_n$ admits all knowledge modalities K_i, and S_i for $i \in [1..n]$, and all branching-time constructs $\mathbf{AX}^a$, $\mathbf{AG}^a$, $\mathbf{AF}^a$, $\mathbf{AU}^a$, $\mathbf{EX}^a$, $\mathbf{EG}^a$, $\mathbf{EF}^a$, $\mathbf{EU}^a$. Semantics is defined in terms of satisfiability $\models$. An environment is a tuple $E = (D, \overset{1}{\sim}, \ldots, \overset{n}{\sim}, I, V)$ such that $(D, \overset{1}{\sim}, \ldots, \overset{n}{\sim}, V)$ is a model for PLK$_n$ and (D, I, V) is a model for *Act*-CTL. Satisfiability is defined by induction

[6] **A** is read as 'for all futures', **E** – 'for some futures', **X** – 'next time', **G** – 'always', **F** – 'sometime', **U** – 'until', and a sup-index ^{a} is read as 'in a-run(s)'.

according to semantics of propositional (def. 3), knowledge (def. 4), and branching time constructs (def. 5). For every environment E and every formula ϕ let $E(\phi)$ be the set $\{w \mid w \models_E \phi\}$ of all worlds that satisfies formula ϕ in E.

We are mostly interested in trace-based perfect recall synchronous environments generated from background finite environments. In these environments states are sequences of worlds of initial environments with history of actions that generate them. Agent does not distinguish these sequences if the background system performs the same sequence of actions, and if these sequences have the same number of worlds, and agent can not distinguish these sequences world by world (in the background environment). We can transit from a sequence to another one with respect to an action a by extending the sequence by a state that can be reached by a from the last state of the sequence. Propositionals are evaluated at the last state of sequences with respect to their evaluations in the background environment.

Definition 7. *(of Perfect Recall Synchronous environment)*
Let E be an environment $(D, \overset{1}{\sim}, \ldots, \overset{n}{\sim}, I, V)$. A trace-based Perfect Recall Synchronous environment generated by E is another environment $(D_{PRS(E)}, \overset{1}{\underset{\text{prs}}{\sim}}, \ldots, \overset{n}{\underset{\text{prs}}{\sim}}, I_{PRS(E)}, V_{PRS(E)})$, where

- $D_{PRS(E)}$ is the set of all pairs (ws, as), where[7]
  $ws \in D^+$, $as \in Act^*$, $|ws| = |as| + 1$, and
  $(ws_j, ws_{j+1}) \in I(as_j)$ for every $j \in [1..|as|]$;
- for every $i \in [1..n]$ and for all (ws', as'), $(ws'', as'') \in D_{PRS(E)}$,
  $(ws', as') \overset{i}{\underset{\text{prs}}{\sim}} (ws'', as'')$ iff
  $as' = as''$ and $ws'_j \overset{i}{\sim} ws''_j$ for every $j \in [1..|ws|]$;
- for every $a \in Act$ and for all (ws', as'), $(ws'', as'') \in D_{PRS(E)}$,
  $((ws', as'), (ws'', as'')) \in I_{PRS(E)}(a)$ iff[8]
  $as'' = as'^{\wedge}a$, and $ws'' = ws'^{\wedge}w''$, $(w', w'') \in I(a)$, where w' is the last element in ws';
- for every $p \in Prp$ and for every $(ws, as) \in D_{PRS(E)}$,
  $(ws, as) \in V_{PRS(E)}(p)$ iff $ws_{|ws|} \in V(p)$.

3 Bounded Knowledge Update and Abstraction

Below in this section we recall definitions and results from [26] (slightly reformulated for the lack of space) that lead to (update+abstraction)-algorithm and evaluation of its non-elementary complexity. We examine the model checking problem for Act-CTL-K$_n$ in perfect recall synchronous environments generated from finite environments. The following formalization of the problem has been introduced in [26].

[7] For every set S let S^+ be the set of all non-empty finite sequences over S and S^* be the set of all finite sequences over S.

[8] Operation $^{\wedge}$ stands for the concatenation of finite words.

Definition 8. *(of the model checking problem)*
The model checking problem for *Act*-CTL-K$_n$ in perfect recall synchronous environments is to validate or refute $(ws, as) \models_{PRS(E)} \phi$, i.e. whether ϕ is satisfiable on (ws, as) in $PRS(E)$, where E is a finite environment, $(ws, as) \in D_{PRS(E)}$, ϕ is a formula of *Act*-CTL-K$_n$.

Definition 9. The *knowledge depth* of a formula is the maximal nesting of knowledge operators in that formula. For every $k \geq 0$ let *Act*-CTL-K$_n^k$ be sublogic of *Act*-CTL-K$_n$ with knowledge depth bounded by k.

It is obvious that *Act*-CTL-K$_n = \bigcup_{k\geq 0}$ *Act*-CTL-K$_n^k$.

For every integer $k \geq 0$ we define by mutual recursion a set $\mathcal{T}_k$ of *k-trees over E*, and a set $\mathcal{F}_k$ of *forests of k-trees over E*.

Definition 10. Let $\mathcal{T}_0$ be a set of all tuples of the form $(w, \emptyset, \ldots, \emptyset)$, where w is a world and the number of copies of the empty set $\emptyset$ is equal to the number of agents n. Once $\mathcal{T}_k$ has been defined, let $\mathcal{F}_k$ be the set of all subsets of $\mathcal{T}_k$. Now, define $\mathcal{T}_{k+1}$ as the set of all tuples of the form $(w, U_1, \ldots, U_n)$, where w is a world and $U_i \neq \emptyset$ is in $\mathcal{F}_k$ for each $i \in [1..n]$. Let us denote $\bigcup_{k\geq 0} \mathcal{T}_k$ by $\mathcal{T}$.

Intuitively, a k-tree is a finite tree of height k whose vertices are labeled by worlds of the environment E and edges are labeled by agents. In a tuple $(w, U_1, \ldots, U_n)$, the world w represents the actual state of the universe, and for each $i \in [1..n]$ the set U_i represents knowledge of the agent i.

The following *update functions* G_k^a generate k-trees obtained from some k-tree after action a taking into account knowledge of every agent.

Definition 11. For every number $k \geq 0$, $a \in Act$ and $i \in [1..n]$, functions $G_k^a : \mathcal{T}_k \times D \to \mathcal{T}_k$ and $H_{k,i}^a : \mathcal{F}_k \times D \to \mathcal{F}_k$, are defined by induction on k and mutual recursion. Let $G_0^a(tr, w) = (w, \emptyset, \ldots, \emptyset)$ iff[9] $(root(tr), w) \in I(a)$. Once G_k^a has been defined, we can define for each $i \in [1..n]$ the function $H_{k,i}^a(U, w) = \{G_k^a(tr, w') \mid tr \in U \text{ and } w' \overset{i}{\sim} w\}$. Now let $G_{k+1}^a((w, U_1, \ldots, U_n),\ w')$ be

$$(\ w'\ , H_{k,1}^a(U_1, w'),\ \ldots\ , H_{k,n}^a(U_n, w')\ ) \text{ iff } (w, w') \in I(a).$$

The following model can be associated with the synchronous environment with perfect recall $PRS(E)$.

Definition 12. *(of model $TR_k(E)$)*
For every $k \geq 0$ let $TR_k(E)$ be the following model $(D_{TR_k(E)}, I_{TR_k(E)}, V_{TR_k(E)})$:

- $D_{TR_k(E)}$ is the set of all 0-, …, k-trees over E for n agents;
- for $a \in Act$: $I_{TR_k(E)}(a) = \{(tr', tr'') \in D_{TR_k(E)} \times D_{TR_k(E)} \mid$
  $tr'' = G_j^a(tr', w)$ for some $j \in [0..k]$ and some $w \in D_E\ \}$;
  for $i \in [1..n]$: $I_{TR_k(E)}(i) = \{(tr', tr'') \in D_{TR_k(E)} \times D_{TR_k(E)} \mid$
  $tr'' \in U_i$ and $tr' = (w, U_1, \ldots, U_n)$ for some $w \in D_E\ \}$;

[9] Operation *root* on trees returns the root of the argument. In particular, for a k-tree $root(w, U_1, \ldots, U_n)$ returns w.

– $V_{TR_k(E)}(p) = \{tr \mid root(tr) \in V_E(p)\}$ for $p \in Prp$.

Definition 13. *(of Act^{+n}-CTL)*
Let Act^{+n} be $Act \cup [1..n]$. A natural translation of formulas of *Act*-CTL-K$_n$ to formulas of Act^{+n}-CTL is simple: just replace every instance of K_i and S_i by corresponding $\mathbf{AX}^i$ and $\mathbf{EX}^i$, respectively ($i \in [1..n]$). For every formula ϕ of *Act*-CTL-K$_n$, let us denote by ϕ^{+n} the resulting formula of Act^{+n}-CTL.

Definition 14. Complete tree is a k-tree $(w, U_1, ..., U_n)$ such that $\{root(tr) | tr \in U_i\} = \{w' \in D | w \overset{i}{\sim} w'\}$ and all trees in U_i are complete trees for every $i \in [1..n]$. Since a state in the root of a complete tree defines the tree uniquely, let us denote the complete tree with root w by $tr(w)$.

Definition 15. Let E be an environment, and $k \geq 0$. A correspondence $tree_k$ between $D_{PRS(E)}$ and k-trees $tree_k : (ws, as) \mapsto tree_k(ws, as)$ is defined by the following.

1. Let tr_1 be complete k-tree $tr(ws_1)$;
2. for every $l \in [2..|ws|]$ let tr_l be $G_k^{as_{l-1}}(tr_{l-1}, ws_l)$;
3. let $tree_k(ws, as)$ be $tr_{|ws|}$.

The following proposition summarizes propositions 4, 5, and 6 from [26].

Proposition 1.
For every integer $k \geq 0$ and $n \geq 1$ and every environment E, for every formula ϕ of Act-CTL-K$_n$ with the knowledge depth k at most there exists bijective correspondence $tree_k : D_{PRS(E)} \to D_{TR_k(E)}$ that

$$(ws, as) \models_{PRS(E)} \phi \text{ iff } tree_k(ws, as) \models_{TR_k(E)} \phi^{+n}.$$

The following (update+abstraction) model checking algorithm is based on the above proposition 1:

1. Input a formula ϕ of *Act*-CTL-K$_n$ and count its knowledge depth k;
2. convert ϕ into the corresponding formula $\psi \equiv \phi^{+n}$ of Act^{+n}-CTL;
3. input a finite environment E and construct the finite model $TR_k(E)$;
4. input a trace (ws, as) and construct the corresponding k-tree tr;
5. model check ψ on tr in $TR_k(E)$.

Its correctness immediately follows from the proposition. In contrasts, complexity of the algorithm is not so straightforward. Paper [26] has proved that its upper bound nonelementary depends on the size of the formula, the number of states, the knowledge depth k and the number of agents n.

4 TR_k as Ideal-based Model

In principle size of $TR_k(E)$ is finite, but it is simply too big to be treated as finite. Due to this reason for model checking $TR_k(E)$ we would like to try

techniques that are in use for model checking infinite systems. In particular, we try a formalism of well-structured labeled transition systems [1, 10] that is a very popular approach to infinite-state model checking.

In well-structured labeled transition systems we can represent a set of states (that is semantics of some formula) by some subset that is usually much smaller than the set itself. Usually this representative subset is a collection of minimal or maximal elements of the set of interest. In this case model checking can compute representative subsets and then restore complete semantics of formulas.

Definition 16. Let D be a set. A *partial order* is a reflexive, transitive, and antisymmetric binary relation R on D. A *preorder* is a reflexive and transitive binary relation R on D. We use prefix, infix and postfix notation for preorders: $R(d', d'')$, $d'(R)d''$ (or $d'Rd''$) , and $(d', d'') \in R$. A *well-preorder* is a preorder R where every infinite sequence $d_1, \ldots d_i, \ldots$ of elements of D contains a pair of elements d_m and d_n so that $m < n$ and $d_m(R)d_n$.

Definition 17. Let (D, R) be a *well-preordered set* (i.e. a set D provided with a well-preorder R). An *ideal (synonym: cone)* is an upward closed subset of D, i.e. a set $C \subseteq D$ such that for all $d', d'' \in D$, if $d'(R)d''$ and $d' \in C$ then $d'' \in C$. Every $d \in D$ generates a cone $(\uparrow d) \equiv \{e \in D \mid d(R)e\}$. For every subset $S \subseteq D$, a *basis* of S is a subset $B \subseteq S$ such that for every $s \in S$ there exists $b \in B$ that $b(R)s$.

Definition 18. A *well-preordered transition system (WPTS)* is a triple (D, R, I) such that (D, R) is a well-preordered set and (D, I) is a Kripke frame.

We are mostly interested in well-preordered transition systems with decidable and compatible well-preorder and interpretation. The standard decidability condition for the well-preorder is straightforward: $R \subseteq D \times D$ is decidable.

Definition 19. Let (D, R, I) be a WPTS.

- *Decidability (tractable past) condition*: there exists a computable total function $BasPre : D \times Act \to 2^D$ such that for every $w \in D$, for every $a \in Act$, $BasPre(w, a)$ is a finite basis of $\{u \in D \mid (u, v) \in I(a)$ and $w(R)v\}$.
- *Compatibility condition*: preorder R is compatible with interpretation $I(a)$ of every action symbol $a \in Act$. (Three equivalent definitions for compatibility of R and $I(a)$ are presented in Tab. 1.)

Definition 20. *(of ideal-based model)*
A well preordered transition system is said to be well-structured transition system (WSTS) iff its preorder is decidable, it meets tractable past and compatibility conditions. A well-structured labeled transition system (WSLTS) is a quadruple (D, R, I, V), where (D, I, V) is a labeled transition system, and (D, R, I) is a well-structured transition system. An ideal-based model is a well-structured labeled transition system (D, R, I, V), where V interprets every propositional variable $p \in Prp$ by a cone.

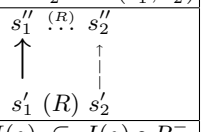
notation	
logic	$\forall s'_1, s''_1, s'_2\ \exists s''_2\ :$ $s'_1 \xrightarrow{I(a)} s''_1\ \&\ R(s'_1, s'_2)\ \Rightarrow$ $\Rightarrow s'_2 \xrightarrow{I(a)} s''_2\ \&\ R(s''_1, s''_2)$
diagram	
algebraic	$R^- \circ I(a) \subseteq I(a) \circ R^-$

Table 1. Equivalent compatibility conditions

The μ-Calculus of D.Kozen (μC) [18] is a very powerful propositional program logic with fixpoints. It is widely used for specification and verification of properties of finite state systems. We would like to skip formal definition of μC due to space limitations. Please, refer to [25] for the elementary introduction to μC. A comprehensive definition of μC can be found in a monograph [2]. Paper [19] has demonstrated that model checking problem in ideal-based models is decidable for μC formulas without negation, conjunction, boxes, and greatest fixpoints.

In the following we assume we are given an environment E and $k, n \geq 0$.

Definition 21. Let us define *binary relation* $\succ$ *on* $D_{TR_k(E)}$. For all trees of equal height $tr' = (w', U'_1, \ldots, U'_n)$ and $tr'' = (w'', U''_1, \ldots, U''_n)$ in $D_{TR_k(E)}$, let us write $tr' \succ tr''$ (and say that tr' has a subtree tr'') iff $w' = w''$ and for every $i \in [1..n]$, for every $st'' \in U''_i$ there exists $st' \in U'_i$ that $st' \succ st''$.

Theorem 1. *Binary relation* $\succ$ *is a partial order on* k*-trees such that model* $TR_k(E)$ *provided with this partial order becomes an ideal-based model, where semantics of every formula of* μC *is a cone with computable finite basis.*

Proof.

First, $\succ$ is a partial order since $tr' \succ tr''$ iff all branches in both trees have equal length and the set of vertexes and edges of tr'' is a subset of the set of vertexes and edges of tr' (i.e. just some branches are skipped). It is decidable relation due to the same argument. It is also a well-preorder since $D_{TR_k(E)}$ is finite.

Second, $\succ$ enjoy tractable past since, in principle, we can find preimage of every tree for every 'action' transition and for every 'knowledge' transition (defined by $I_{TR_k(E)}(a)$ and $I_{TR_k(E)}(i)$, respectively for $a \in Act$ and $i \in [1..n]$, in def. 12) by scanning finite space $TR_k(E)$. But there is more effective technic to find preimages based on the notion of complete trees. More efficient algorithm follows. Let $(w, U_1, ...U_n)$ be a k-tree. If $a \in Act$ then for every state u such that $(u, w) \in I(a)$ construct complete tree $tr(u)$; collect all subtrees tr of any of these complete $tr(u)$ (i.e. $tr(u) \succeq tr$) such that $(w, U_1, ...U_n) \succeq G^a_k(tr, w)$. If $i \in [1..n]$ is an agent then just collect all $tr \in U_i$.

But the most efficient[10] algorithm can be described as follows. We consider the case of one agent only, which is easily generalized to the case of n agents. Let us find preimage by induction on a height of a tree. Let $tr = (w, \emptyset)$ be a tree of height 0. Its preimage with respect to an action a is the set of trees of height 0 with roots which are in the preimage for w: $Pre^a(tr) = \{(w', \emptyset) | (w', w) \in I(a)\}$. Let $tr = (w, U)$ be a tree of height k, where root w is a world, and U is a set of trees, which roots are in the set $roots(U) = \{s | s = root(t), t \in U\}$. Its preimage with respect to action a $Pre^a(tr) = Tr'_1 \cup Tr'_2 \cup Tr'$ includes all trees of form $tr'_1 = (w', U'_1) \in Tr'_1$ and $tr'_2 = (w', U'_2) \in Tr'_2$, such that $(w', w) \in I(a)$ and $U'_1 = \{(s', V') | s' \sim w'$, and $\exists s \in root(U) : G^a_{k-1}((s', V'), s) \in U\}$, and $U'_2 = U'_1 \cup \{(s', tr(s') | s' \sim w'$, and $\forall s'' \in a(s') : s'' \nsim w\}$ (Note that each tree of Tr'_1 is k-subtree of some tree in Tr'_2). The set Tr' is defined as follows: $Tr' = \{tr' | tr'_1 \prec tr' \prec tr'_2$ for some $tr'_1 \in Tr'_1$ and $tr'_2 \in Tr'_2\}$. So, roots of trees in U'_1 are all worlds s', which the agent can not distinguish with w', and there exists a world $s \in roots(U)$, such that $(s', s) \in I(a)$, and subtrees of these trees are computed in according to induction assumption. Roots of trees in U'_2 are $roots(U_1)$ supplemented by worlds s'' whose images $a(s'')$ are distinguished with w by the agent, and subtrees corresponding to these roots s'' are complete trees. In addition, preimage includes all "intermediate" trees. We can compute these trees easily due to finiteness of D. Due to definition of update function, it is obvious that $tr = G^a_1(tr', w)$ for every $tr' \in Pre^a(tr)$. There are no other trees in preimage since other roots are impossible, and U'_2 can be extended only by trees with roots which are transformed by action a to worlds indistinguishable with w, but the images of such trees include tr as a subtree.

Third, $\succ$ is compatible with all 'action' transitions and all 'knowledge' transitions. For every action $a \in Act$, and for every pair of trees $tr' \succ tr''$ there exists some sup-tree tr which is a-image of the greater tree tr' and this sup-tree includes a-image of the smaller tree tr'' because a-images are computed recursively by processing each vertex of trees (def. 11). Again we consider the case of one agent only, which is easily generalized to the case of n agents. Let $tr_1 = (w, U_1)$ and $tr_2 = (w, U_2)$ be trees of height k and $tr_1 \prec tr_2$. Note that for every $t_1 \in U_1$ there exists $t_2 \in U_2$ such that $t_1 \prec t_2$ by definition of $\prec$. Let $tr'_1 = (w', U'_1) \in G^a_k(tr_1, w')$. Let us find $tr'_2 = (w', U'_2) \in G^a_k(tr_2, w')$, such that $tr'_1 \prec tr'_2$. Due to def. 11, $(w, w') \in I(a)$ and $U'_1 = \{G^a_{k-1}(tr, w'') | tr \in U_1$ and $w'' \sim w'\}$. By the same definition $U'_2 = \{G^a_{k-1}(tr, w'') | tr \in U_2$ and $w'' \sim w'\}$. Then it is obvious that for every $t'_1 \in U'_1$ there exists $t'_2 \in U'_2$ such that $t'_1 \prec t'_2$, hence $tr'_1 \prec tr'_2$.

For every 'knowledge' action $i \in [1..n]$, and for every pair of trees $tr' \succ tr''$ there exists some sup-tree tr which is i-image of the greater tree tr' and this sup-tree includes i-image of the smaller tree tr'' because computing of i-images of some tree is based on transition to subtrees of this tree (def. 12). Once again we consider the case of one agent only. Let $tr_1 = (w, U_1)$ and $tr_2 = (w, U_2)$ be trees of height k and $tr_1 \prec tr_2$. Note that for every $t_1 \in U_1$ there exists $t_2 \in U_2$ such that $t_1 \prec t_2$ by definition of $\prec$. $(tr_1, t_1) \in I_{TR_k(E)}(1)$ holds for every $t_1 \in U_1$

[10] to the best of our knowledge

and $(tr_2, t_2) \in I_{TR_k(E)}(1)$ holds for every $t_2 \in U_2$ due to def. 12. it is obviously implies that $\prec$ is compatible with 'knowledge' transitions.

Fourth, $TR_k(E)$ is an ideal-based model. It is obvious that valuation of every propositional variable forms a cone with basis consisting of complete trees with roots which are states where this propositional variable holds, due to def. 12: $V_{TR_k(E)}(p) = \{tr | root(tr) \in V_E(p)\} = \uparrow \{tr(w) | w \in V_E(p)\}$ for $p \in Prp$ since p is satisfiable in every k-subtree of every $tr \in V_{TR_k(E)}(p)$. There are no other trees with this property since the set includes all complete trees with these roots. Note, that negation of propositional variable is a cone also: $V_{TR_k(E)}(\neg p) = \uparrow \{tr(w) | w \notin V_E(p)\}$ for $p \in Prp$.

Finally we prove that semantics of every formula of μC is a cone with computable finite basis by induction on structure of normal formulas in which negation is used in literals[11] only. (Every μC formula is equivalent to some normal formula [25].) Induction basis deals with literals; for propositional variables it is proved already, for their negations proof is similar. Induction step consists of a number of cases: for disjunction $\vee$, conjunction $\wedge$, box $[\,]$ and diamond $\langle\,\rangle$.

Basis of disjunction of formulas is union of bases of these formulas.

Basis of conjunction of formulas consists of maximal trees which are subtrees of trees from bases of these formulas simultaneously. Let basis of formula ϕ be B_ϕ and basis of formula ψ – B_ψ. Hence, the set $B_{\phi\wedge\psi} = \{tr | tr \prec tr_\phi \in B_\phi$ and $tr \prec tr_\psi \in B_\psi\}$ is computable and finite due to finiteness of sets B_ϕ and B_ψ. This set is a basis for semantics of $\phi \wedge \psi$.

Bases of a box- or diamond-formula is a computable cone due to properties of tractable past and compatibility. Let us find basis of semantics of formula $\phi = \langle a \rangle \psi$[12]. For simplicity let basis of ψ consists of single tree: $B_\psi = \{tr\}$. Denote preimage of tr with respect to action a as $Pre^a(tr)$, defined above. Hence, preimage of all trees in semantics of ψ with respect to action a is the set of all k-subtrees from $Pre^a(tr) = Tr'_1 \cup Tr'_2 \cup Tr'$. Every tree in $Tr'_1 \cup Tr'$ is k-subtree of some tree in Tr'_2. Hence, basis of preimage of semantics of ψ with respect to action a is the finite set $Pre^a(\psi) = Tr'_2$, due to definition of Tr'_2 and knowledge update function a-image of every k-subtree of every tree in Tr'_2 is some tree in the cone generated by tree tr. By definition of diamond modality this set is a basis of semantics of formula $\phi = \langle a \rangle \psi$ also.

The least $\mu x.\phi$ and the greatest fixpoints $\nu x.\phi$ in finite models[13] are equivalent to 'infinite disjunctions' and 'infinite conjunctions':

- $false \vee \phi_x(false) \vee \phi_x(\phi_x(false)) \vee \ldots = \bigvee_{j \geq 0} \phi_x^j(false)$,
- $true \wedge \phi_x(true) \wedge \phi_x(\phi_x(true)) \wedge \ldots = \bigwedge_{j \geq 0} \phi^j(true)$,

where $\phi_x(\psi)$ is a result of substitution of a formula ψ instead of x, $\phi_x^0(\psi)$ is ψ, and $\phi_x^{j+1}(\psi)$ is $\phi_x(\phi_x^j(\psi))$ for $j \geq 0$. In $TR_k(E)$ one can assume these infinite disjunctions and conjunctions to be finite and bounded by the number of k-trees

[11] A literal is a propositional variable or its negation.
[12] Basis of semantics of $[a]\psi$ is treated analogously.
[13] by the finite-case Tarski-Knaster fixpoint theorem

in $TR_k(E)$. This observation reduces the case of fixpoints to combination of cases for disjunction, conjunction, box and diamond that are proved already. ■

Note that semantics of *every* formula of μC in model $TR_k(E)$ is a computable cone in contrast to [19] where arbitrary ideal-based models have been studied.

It is well-known that standard CTL is expressible in μC (see for example [25]). This translation of CTL to μC can be generalized easily to *Act*-CTL.

$$\begin{array}{ll}
\mathbf{AX}^a\varphi \leftrightarrow [a]\varphi & \mathbf{EX}^a\varphi \leftrightarrow \langle a\rangle\varphi \\
\mathbf{AG}^a\varphi \leftrightarrow \nu x.\ (\varphi \wedge [a]x) & \mathbf{AF}^a\varphi \leftrightarrow \mu x.\ (\varphi \vee [a]x) \\
\mathbf{EG}^a\varphi \leftrightarrow \nu x.\ (\varphi \wedge \langle a\rangle x) & \mathbf{EF}^a\varphi \leftrightarrow \mu x.\ (\varphi \vee \langle a\rangle x) \\
\mathbf{A}(\varphi\mathbf{U}^a\psi) \leftrightarrow \mu x.\ (\psi \vee (\varphi \wedge [a]x)) & \mathbf{E}(\varphi\mathbf{U}^a\psi) \leftrightarrow \mu x.\ (\psi \vee (\varphi \wedge \langle a\rangle x))
\end{array}$$

It implies the following corollary.

Corollary 1. *Semantics of every formula of Act^{+n}-CTL in $TR_k(E)$ is a cone with respect to $\succ$ with computable finite bases.*

5 Conclusion

In this paper we have shown that space $TR_k(E)$ provided with sub-tree partial order forms a well-structured labeled transition system where every property expressible in the propositional μ-Calculus, can be characterized by a finite computable set of maximal trees that enjoy the property. We tried feasibility of this approach to model checking of *Act*-CTL-K$_n$ in trace-based synchronous perfect recall synchronous environment by automatic model checking simple, but big example (the size of model TR_k is about 10^{36000}). Data structures that are used in the experiment are so-called vector-affine trees [12]. A presentation of a background theory and of our experimental model checker is a topic for a future publication.

To the best of our knowledge, the only reported (experimental) model checker for perfect recall synchronous systems is MCK [11]. It works in a linear as well as branching time settings. For perfect recall synchronous systems in temporal dimension MCK supports 'next' operator only, but neither 'always', 'sometimes', nor 'until' (although the model checking theory for the full combination of knowledge with Propositional Logic of Linear Time has been already developed [22]). The present paper has developed a technique that may lead to practical model checking the full combination of knowledge with branching time logics (a là Computation Tree Logic) with 'next' operator as well as with 'always', 'sometimes', and 'until'.

References

1. Abdulla P.A., Ĉerâns K., Jonsson B., and Tsay Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. Information and Computation, v.160(1-2), 2000, p.109-127.
2. Arnold A. and Niwinski D. *Rudiments of μ-calculus.* North Holland, 2001.

3. Bull R.A., Segerberg K. Basic Modal Logic. In: Handbook of Philosophical Logic. Vol.II. Reidel Publishing Company, 1984 (1-st ed.), Kluwer Academic Publishers, 1994 (2-nd ed.). p. 1–88.
4. Burch J.R., Clarke E.M., McMillan K.L., Dill D.L., Hwang L.J. Symbolic Model Checking: 10^{20} states and beyond. Information and Computation, 1992. v.98(2), p. 142–170.
5. Clarke E.M., Grumberg O., Peled D. Model Checking. MIT Press, 1999.
6. Dixon C., Fernandez Gago M-C., Fisher M., and van der Hoek W. Using Temporal Logics of Knowledge in the Formal Verification of Security Protocols. In: Proceedings of TIME 2004, 1st-3rd July 2004, Tatihou, Normandie, France. IEEE.
7. Dixon C., Nalon C. and Fisher M. Tableau for Logics of Time and Knowledge with Interactions Relating to Synchrony, Journal of Applied Non-Classical Logics, v.14, n.4, p.397-445, 2004.
8. Emerson E.A. Temporal and Modal Logic. In: Handbook of Theoretical Computer Science. v.B, Elsevier and MIT Press, 1990, p. 995–1072.
9. Fagin R., Halpern J.Y., Moses Y., Vardi M.Y. Reasoning about Knowledge. MIT Press, 1995.
10. Finkel A., Schnoebelen Ph. *Well-structured transition systems everywhere!* Theor. Comp. Sci., v.256(1-2), 2001, p.63-92.
11. Gammie P. and van der Meyden R. MCK: Model Checking the Logic of Knowledge. Springer-Verlag Lect. Notes Comp. Sci., v.3114, 2004, p.479-483.
12. Garanina N.O. Verification of Distributed Systems on base of Affine Data representation and Logics of Knowledge and Actions. Ph.D Thesis, A.P. Ershov Institute of Informatics Systems, 2004 (in Russian).
13. Garanina N.O., Kalinina N.A. and Shilov N.V. Model checking knowledge, actions and fixpoints. Proc. of Concurrency, Specification and Programming Workshop CS&P'2004, Germany, 2004, Humboldt Universitat, Berlin, Informatik-Bericht Nr.170, v.2, p.351-357.
14. Halpern J. Y., van der Meyden R., and Vardi M. Y. Complete Axiomatizations for Reasoning about Knowledge and Time. SIAM Journal on Computing, v.33(3), 2004, p. 674-703.
15. van der Hoek W. and Wooldridge M.J. Model Checking Knowledge and Time. Lecture Notes in Computer Science, v.2318, p.95-111, 2002.
16. Kacprzak M., Lomuscio A., Penczek W. Unbounded Model Checking for Knowledge and Time. Proceedings of the CS&P'2003 Workshop, Warsaw University, v.1, p.251-264.
17. Kacprzak M., Penczek W. Model Checking for Alternating-Time mu-Calculus via Translation to SAT. Proc. of Concurrency, Specification and Programming Workshop CS&P'2004, Germany, 2004, Humboldt Universitat, Berlin, Informatik-Bericht Nr.170, v.2.
18. Kozen D. *Results on the Propositional Mu-Calculus.* Theoretical Computer Science, v.27, n.3, 1983, p.333-354.
19. Kouzmin E.V., Shilov N.V., Sokolov V.A. Model Checking μ-Calculau in Well-Structured Transition Systems. Proceedings of 11th International Symposium on Temporal Representation and Reasoning (TIME 2004), France 2004, IEEE Press, p. 152-155.
20. Kozen D., Tiuryn J. Logics of Programs. In: Handbook of Theoretical Computer Science, v.B., Elsevier and MIT Press, 1990, p. 789–840.
21. van der Meyden R. Common Knowledge and Update in Finite Environments. Information and Computation, 1998, v.140(2), p. 115–157.

22. van der Meyden R., Shilov N.V. Model Checking Knowledge and Time in Systems with Perfect Recall. Springer-Verlag Lect. Notes Comput. Sci., 1999, v.1738, p. 432–445.
23. van der Meyden R. and Wong K. Complete Axiomatizations for Reasoning about Knowledge and Branching Time. Studia Logica, v.75(1), 2003, p. 93-123.
24. Rescher N. Epistemic Logic. A Survey of the Logic of Knowledge. University of Pitsburgh Press, 2005.
25. Shilov N.V. and Yi K. How to find a coin: propositional program logics made easy. In: Current Trends in Theoretical Computer Science, World Scientific, v. 2, 2004, p.181-213.
26. Shilov N.V., Garanina N.O., and Choe K.-M. 2.7. Update and Abstraction in Model Checking of Knowledge and Branching Time. Fundameta Informaticae, v.72, n.1-3, 2006, p.347-361.