Development of global specification for dynamically adaptive software

Yongwang Zhao
School of Computer Science & Engineering
Beihang University
zhaoyw@act.buaa.edu.cn

22/02/2013

About me

- Assistant prof in National Key Lab of Software Development Environment, Beihang University
- Research interests: formal verification of service-based and real-time systems
- Teaching: formal languages and automata theory, foundation of services computing, middleware technology – web services
- Manager of SCENE: a web services middleware of China
- Convener of ISO/IEC JTC1 SC7 WG7/SG 12207-SOA

Growing complexity of software system

- Very large scales
  - million of entities
- Ad hoc (amorphous) structures/behaviors
  - p2p/hierarchical architecture
- Dynamic
  - entities join, leave, move, change behavior
- Heterogeneous
  - capability, connectivity, reliability, guarantees, QoS
- Unreliable
  - components, communication
- Lack of common/complete knowledge
  - number, type, location, availability, connectivity, protocols, semantics, etc.

Autonomic Computing Attributes

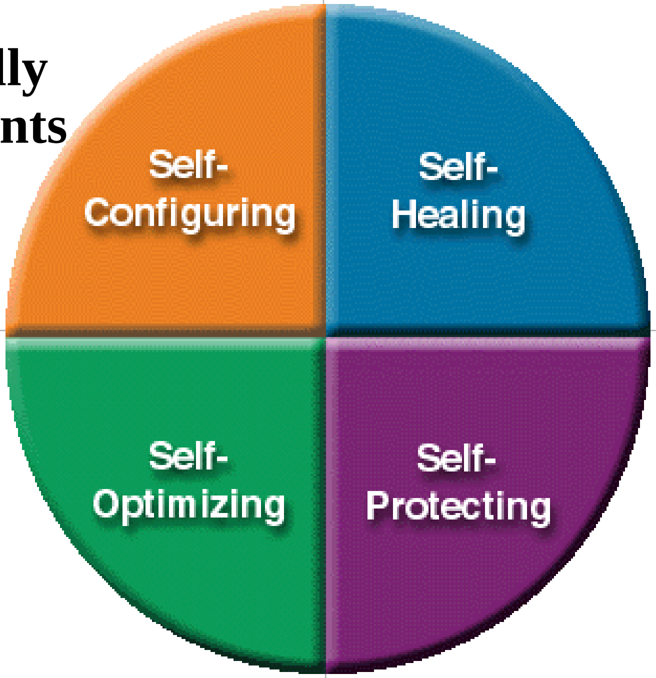
- Self-managing systems that deliver

Increased Responsiveness

Adapt to dynamically changing environments

Business Resiliency

Discover, diagnose, and act to prevent disruptions



Operational Efficiency

Tune resources and balance workloads to maximize use of IT resources

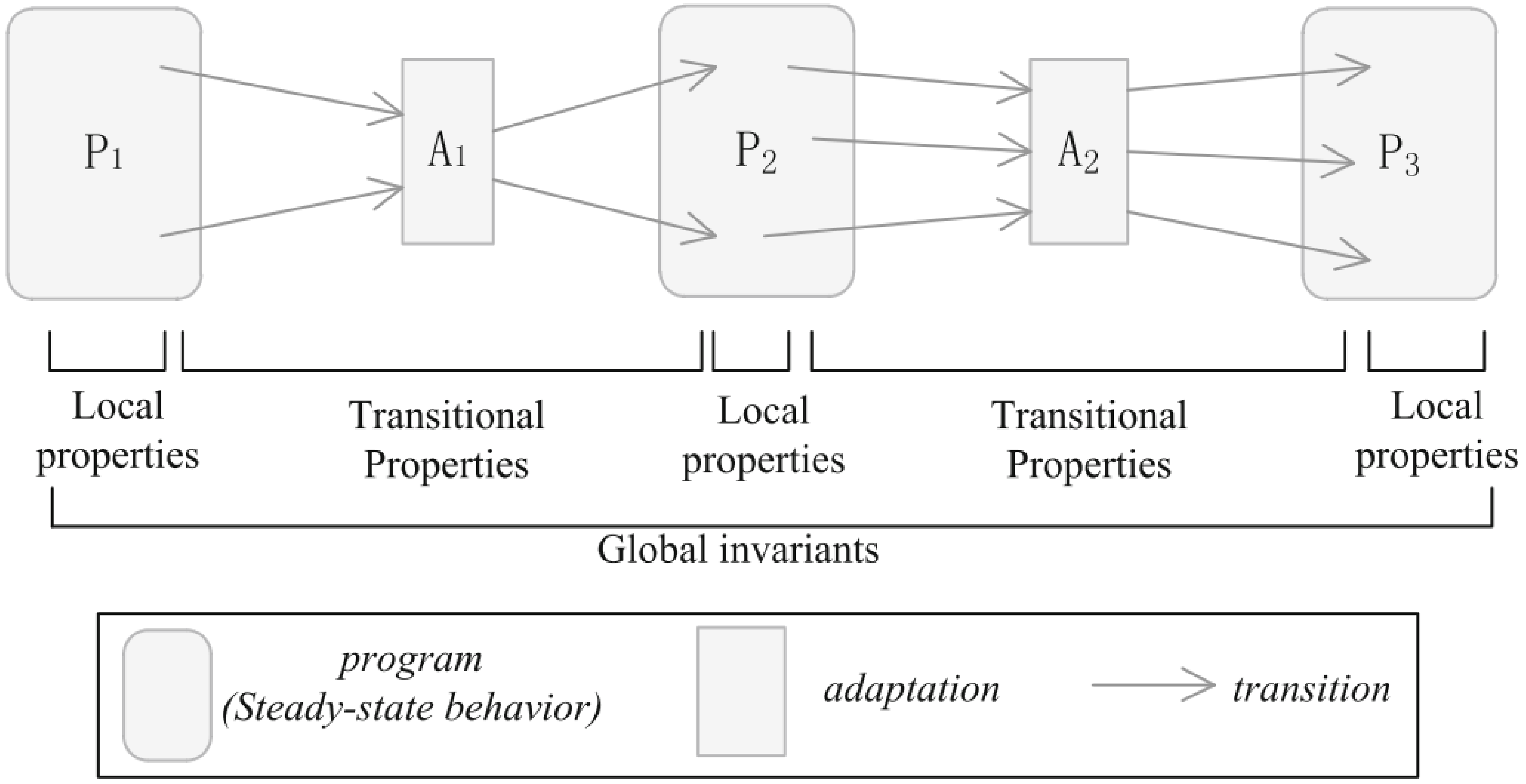
Secure Information and Resources

Anticipate, detect, identify, and protect against attacks

Dynamic Adaptation

- To decrease cost and time of human supervision for continue operation, computer software must modify its structure and behavior dynamically in response to the changes in its execution environment
- dynamic adaptation
  - modify its structure and behavior dynamically at runtime
  - in response to changes in its operating environment – end user input, external hardware device and sensors
- Verification
  - Correctness of adaptive programs becomes most crucial, especially when they are applied in safety-critical domains.
  - Correctness means that adaptive programs running correctly before, during and after adaptation
  - By high complexity, adaptive programs are generally more difficult to specify, verify, and validate. Assurance of high dependability of these programs is a great challenge

Adaptive program and specification



The properties

- Local properties: ----- LTL
  - properties define the specification of a specific non-adaptive program. They specify the steady-state behavior of an adaptive program operating in a behavioral mode. They may be such properties as safety, liveness, and invariants, etc.
- Transitional properties: ----- A-LTL
  - properties that hold during the adaptation process. They specify the dynamic adaptation between two non-adaptive programs, and should be satisfied during interval state when the adaptive program is being adapted from one behavioral mode to another.
- Global invariants: ----- LTL
  - properties to be satisfied by the adaptive program throughout its execution. They consider steady-state behaviors and adaptations as a whole regardless of the adaptations.

A-LTL

- **Adapt operator-extended LTL**
- **the adapt operator:** $\phi \overset{\Omega}{\rightarrow} \varphi$
- **Semantics**
  - If σ is an infinite state sequence and ϕ is an LTL formula, then $\sigma \vDash \phi$ in A-LTL if and only if $\sigma \vDash \phi$ in LTL.
  - If σ is a finite state sequence and ϕ is an A-LTL formula, then $\sigma \vDash_f \phi$ iff $\sigma' \vDash \phi$, where σ' is an infinite state sequence constructed by repeating the last state of σ.
  - $\sigma \vDash \phi \overset{\Omega}{\rightarrow} \varphi$ iff there exists a finite state sequence $\sigma' = (s_0, s_1, \dots, s_k)$ and an infinite state sequence $\sigma'' = (s_{k+1}, s_{k+2}, \dots)$, such that $\sigma = \sigma' \frown \sigma''$, $\sigma' \vDash_f \phi$, $\sigma'' \vDash \varphi$, and $(s_k, s_{k+1}) \vDash_f \Omega$, where ϕ,φ and Ω are A-LTL formulae.

A-LTL

- an adaptive program satisfies $\phi \xrightarrow{\Omega} \varphi$, if this program initially satisfies ϕ, and in a certain state *A*, it stops being constrained by ϕ, then in the next state *B*, it starts to satisfy φ. And the two-state sequence (*A*,*B*) satisfies Ω.
- adaptive TCP routing protocol
  - only trusted nodes are selected for packet delivery in “safe” configuration and any packet must be encrypted before being transferred to an untrusted node in “normal” configuration. The transitional property that must be satisfied by executions adapting from “safe” configuration to “normal” configuration could be expressed by the A-LTL formula

$$\Box(!unsafe) \overset{true}{\rightharpoonup} \Box(unsafe \rightarrow (!sent\ \mathcal{U}\ encrypted))$$

The problems

- Software systems are becoming increasingly complex systems
  - adaptation and its target behavior of the adaptive program may be unknowable until system operation
  - there may be large amount of (unknowable) adaptations in these systems when evolving
- Transitional specification is between two explicit non-adaptive programs. it assumes that all adaptation choices are known in advance
- To a certain extent, all possible alternative adaptations to be designed are unknowable during requirements engineering
- global specification
  - the requirement for adaptive programs from the perspective of adapting process at the macro level.
  - For instance, the degraded system functions dealing with encountering errors will eventually be upgraded after some adaptations.

A global property

- the reachability of the behavioral mode satisfied by *SPEC* in an adaptive program that has three non-adaptive programs and two adaptations
- mLTL formula

$$\Diamond_m[SPEC]$$

- A-LTL formula

$$(SPEC \overset{true}{\rightharpoonup} true \overset{true}{\rightharpoonup} true) \vee (true \overset{true}{\rightharpoonup} SPEC \overset{true}{\rightharpoonup} true) \vee (true \overset{true}{\rightharpoonup} true \overset{true}{\rightharpoonup} SPEC)$$

  - A-LTL is explicit to the adapting process of the adaptive program, the temporal relationship of multiple adaptations considering the global process of all adaptations is difficult to be specified.
  - A-LTL is not effective for large amount of adaptations
- LTL is more cumbersome than A-LTL on this problem, because A-LTL is at least exponentially more succinct than LTL in specifying transitional properties

Formal model of adaptive programs

Definition 1 (*Non-adaptive program*) A non-adaptive program $\mathcal{P}$ is a finite state machine. $\mathcal{P}$ is a tuple $\langle v, S, Act, \sigma, s_0, AP, L\rangle$, where

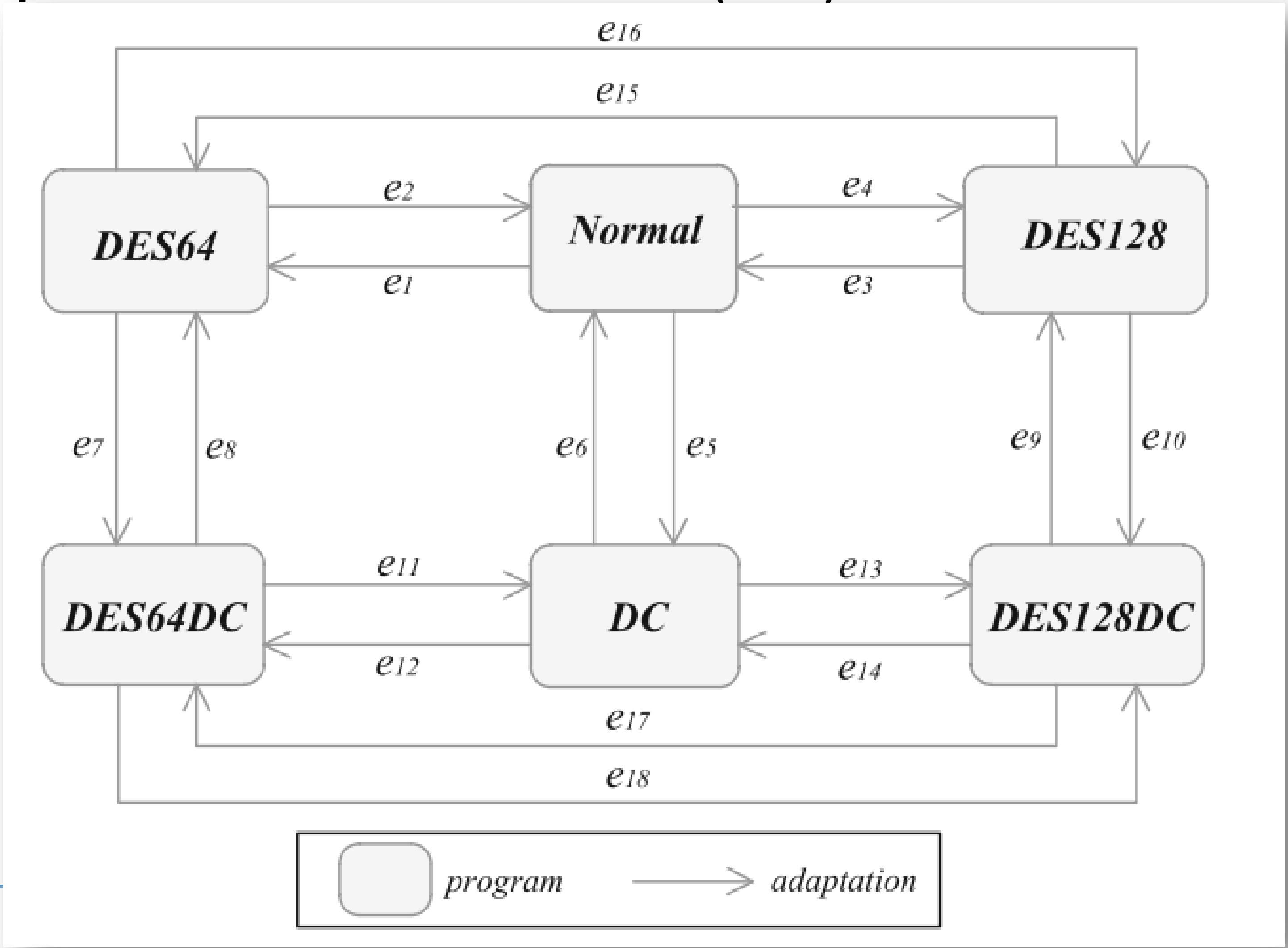
- v: name of this program
- S: a set of states
- Act: a set of actions
- $\sigma : S \times Act \to S$ is a state transition relation
- $s_0 \in S$ is the initial state
- $L : S \to 2^{AP}$ is a labelling function.

Definition 2 (*Adaptive program*) An adaptive program $\mathcal{AP}$ is a finite state machine. $\mathcal{AP}$ is a tuple $\langle A, E, \varphi, \mathcal{P}_0\rangle$, where

- A: a set of non-adaptive programs, with each of them being a mode, and for each program $\mathcal{P}_i \in A$, $\mathcal{P}_i = \langle v_i, S_i, Act_i, \sigma_i, s_{0_i}, AP_i, L_i\rangle$
- E: a set of adaptations
- $\mathcal{P}_0 \in A$ is the initial program of $\mathcal{AP}$
- $\varphi : A \times \bigcup_{\mathcal{P}\in A} S(\mathcal{P}) \times E \to A \times \bigcup_{\mathcal{P}\in A} S(\mathcal{P})$ is the adaptation relation, where $\bigcup_{\mathcal{P}\in A} S(\mathcal{P})$ is the set of all states of programs in A. For each $(\mathcal{P}, s, e, \mathcal{Q}, t) \in \varphi$, $s \in S(\mathcal{P})$ and $t \in S(\mathcal{Q})$.

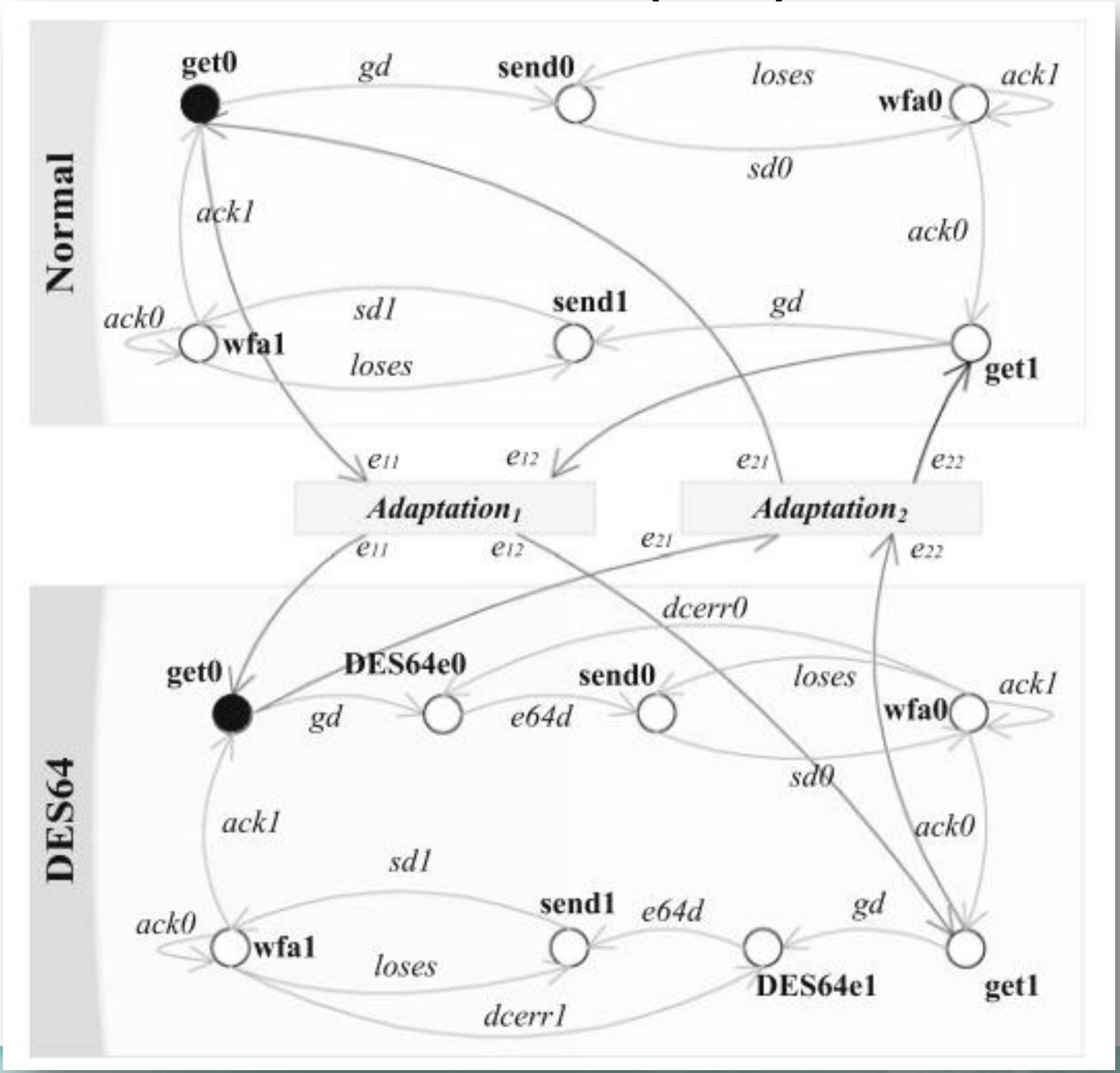
A study case

- **Adaptive Communication Protocol (ACP)**



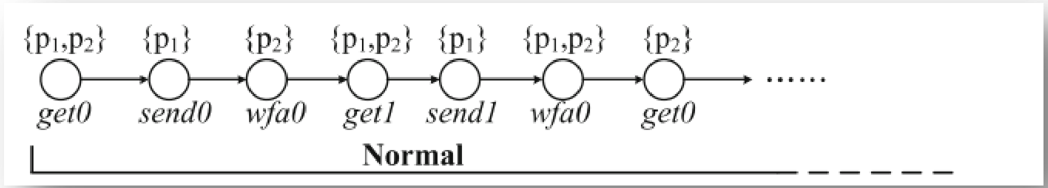
A study case

- **Adaptive Communication Protocol (ACP)**



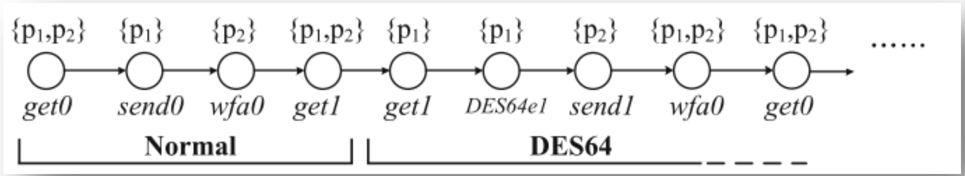
The state traces

- **Non-adaptive state trace**



{p1,p2} {p1} {p2} {p1,p2} {p1} {p1,p2} {p2}
get0 send0 wfa0 get1 send1 wfa0 get0
......
Normal

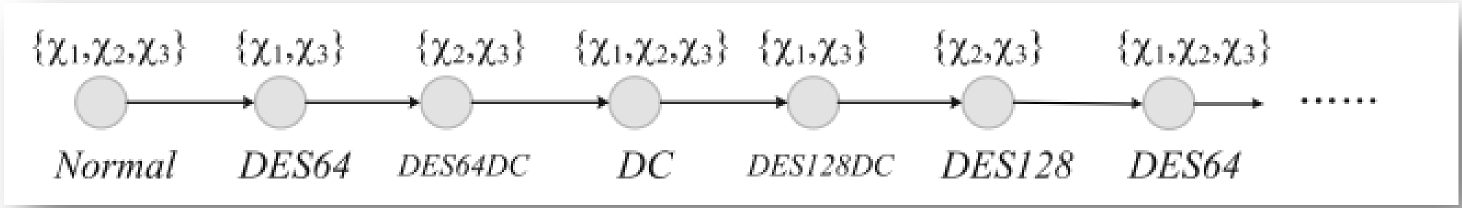
- **Adaptive state trace**



{p1,p2} {p1} {p2} {p1,p2} {p1} {p1} {p2} {p1,p2} {p1,p2}
get0 send0 wfa0 get1 get1 DES64e1 send1 wfa0 get0
......
Normal
DES64

- **Intervals & Mode trace**
  - Adaptive state trace $t = s_0 s_1 s_2 s_3 \ldots \ldots$
  - Mode trace $mt = i_1 i_2 i_3 \ldots.$

$$t = \widetilde{i_1} \frown \widetilde{i_2} \frown \ldots \frown \widetilde{i_n}$$



{χ1,χ2,χ3} {χ1,χ3} {χ2,χ3} {χ1,χ2,χ3} {χ1,χ3} {χ2,χ3} {χ1,χ2,χ3}
Normal DES64 DES64DC DC DES128DC DES128 DES64
......

Specifying adaptive programs- mLTL

- **Syntax**

  $\kappa ::= \mathbf{true} \mid [\chi] \mid @v \mid \kappa \wedge \kappa \mid \neg\kappa \mid \bigcirc_m \kappa \mid \kappa \mathcal{U}_m \kappa$

- **Semantics**
  - $t = s_0 s_1 s_2 \ldots$ be an adaptive state trace with *n* intervals $i_1, \; i_2, i_3, \ldots, i_n$.
  - The mode trace *mt* of *t* is $i_1 i_2 i_3 \ldots$
  - $t \vDash \mathbf{true}$.
  - $t \vDash [\chi]$ if and only if $mt \vDash_m [\chi]$.
  - $t \vDash @v$ if and only if $mt \vDash_m @v$.
  - $t \vDash \bigcirc_m \kappa$ if and only if $mt \vDash_m \bigcirc_m \kappa$.
  - $t \vDash \kappa_1 \mathcal{U}_m \kappa_2$ if and only if $mt \vDash_m \kappa_1 \mathcal{U}_m \kappa_2$.
  - $t \vDash \kappa_1 \wedge \kappa_2$ if and only if $t \vDash \kappa_1$ and $t \vDash \kappa_2$.
  - $t \vDash \neg\kappa$ if and only if $\neg t \vDash \kappa$.

Specifying adaptive programs- mLTL

- $mt \models_m$ **true**.
- $mt \models_m [\chi]$ if and only if $i_1 \models_m [\chi]$, which means $\widetilde{i_1} \models_{fin} \chi$.
- $mt \models_m @v$ if and only if $i_1 \models_m @v$, which means $\widetilde{i_1} \models_{fin} \Box v$.
- $mt \models_m \kappa_1 \wedge \kappa_2$ if and only if $mt \models_m \kappa_1$ and $mt \models_m \kappa_2$.
- $mt \models_m \neg\kappa$ if and only if $\neg mt \models_m \kappa$.
- $mt \models_m \bigcirc_m \lambda$ if and only if $mt' \models_m \lambda$, where $mt' = i_2 i_3 \ldots$.
- $mt \models_m \kappa_1 \mathcal{U}_m \kappa_2$ if and only if for some $j = 1, 2, \ldots, i_j \models_m \kappa_2$ and $i_1 \models_m \kappa_1, \ldots, i_{j-1} \models_m \kappa_1$.

- **Always, eventually**

$$\Diamond_m \kappa \overset{def}{=} [\textbf{true}]\mathcal{U}_m \kappa \ \ (\textit{eventually})$$

$$\Box_m \kappa \overset{def}{=} \neg\Diamond_m \neg\kappa \ \ (\textit{always})$$

Global specification of ACP in mLTL

$$\kappa = \Box_m(@Normal \rightarrow [\Box(INPUT \rightarrow \Diamond SENT)])$$
$$\kappa = \Box_m(@Normal \rightarrow [\Box\Diamond(INPUT)])$$
$$\kappa = \Box_m[\Box(INPUT \rightarrow \Diamond SENT)]$$

- if the program could not infinitely often input data before any other adaptation occurs, its next program could do that.

$$\kappa = \Box_m[[!\Box\Diamond INPUT] \rightarrow \bigcirc_m[\Box\Diamond INPUT]]$$

- eventually reach a program in which data are compressed

$$\kappa = \Diamond_m[\Box(INPUT \rightarrow \Diamond DATACOMPRS)]$$

- never reach a program in which data are encrypted by DES64e and DES128e filters simultaneouslv

$$\kappa = \Box_m([\Box(INPUT \rightarrow \Diamond ENCRYPERR)] \rightarrow \Diamond_m[\Box(INPUT \rightarrow \Diamond SENT)])$$

- a program with data encryption error will be eventually repaired.

$$\kappa = \Box_m[\Box((DES64ENCD \rightarrow \neg\Diamond DES128ENCD) \wedge (DES128ENCD \rightarrow \neg\Diamond DES64ENCD))]$$

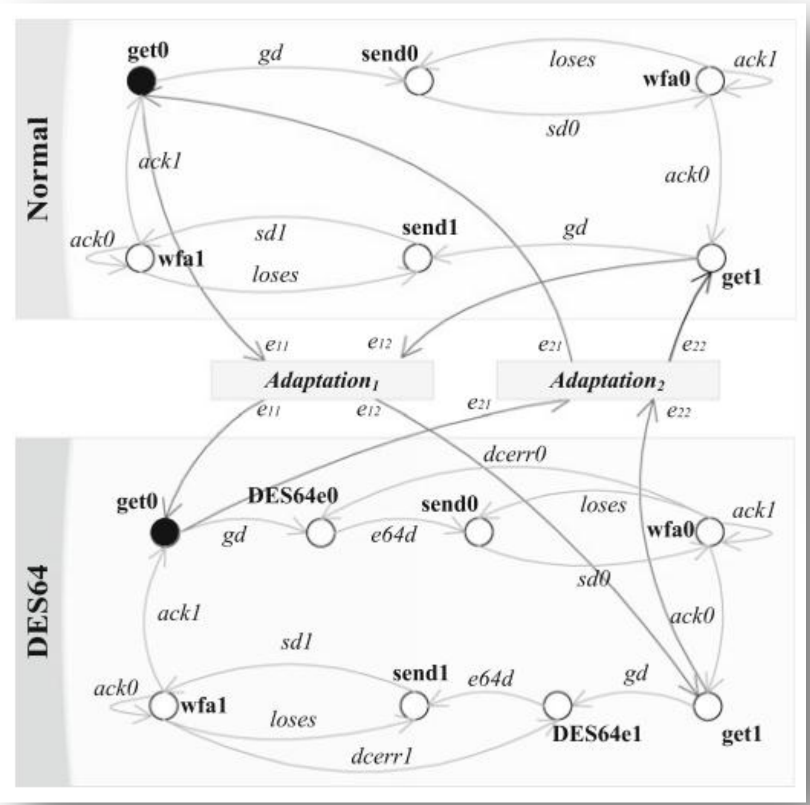
mLTL model checking

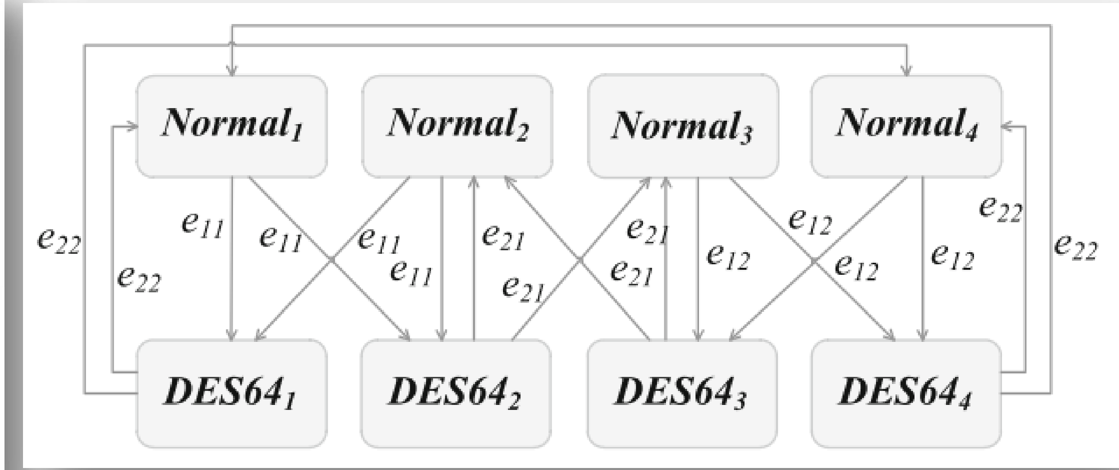
- The mLTL model checking could be decomposed into LTL checking on the state sequence of intervals and mLTL checking on the mode traces of the adaptive program.

- Four steps:
  - (1) extract the LTL formula set from the mLTL formula and translate the mLTL formula to another LTL formula,
  - (2) checking LTL formula set from (1) on each non-adaptive program
  - (3) construct a global transition system
    - a non-adaptive program is a state, the adaptations are transitions, label function indicates the LTL formulae satisfied on each state
  - (4) checking the LTL formula from (1) on the global transition system.

mLTL model checking

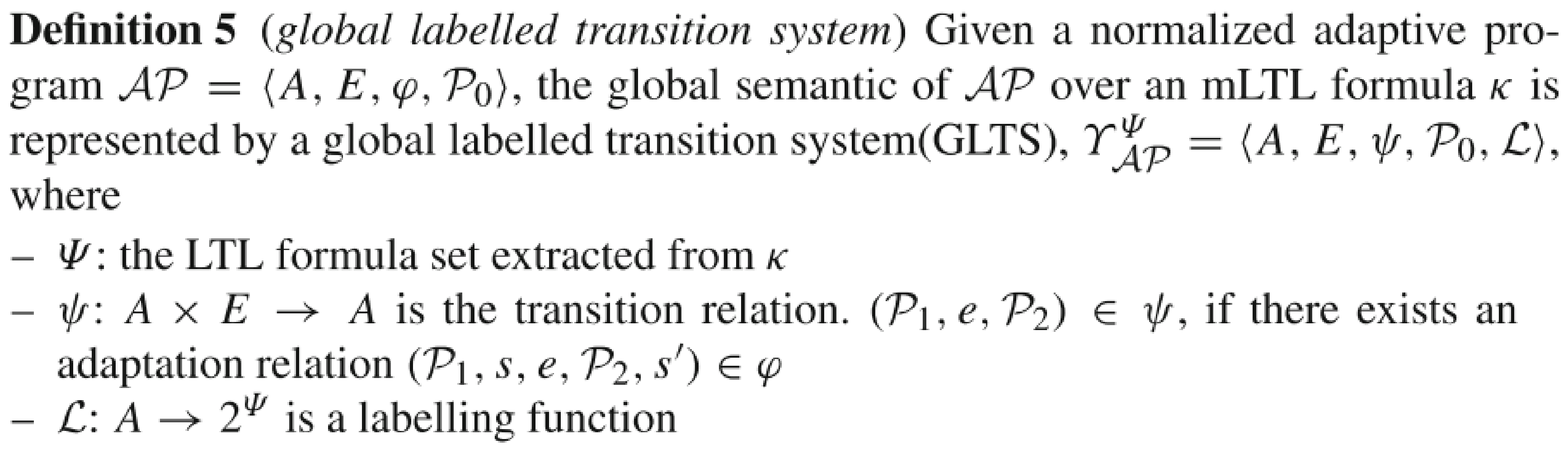
- But the non-adaptive program may exhibit different behaviors





Normalization:
An adaptive program *AP* is trace equivalent to its normalization

Global semantic of adaptive programs



Definition 5 (*global labelled transition system*) Given a normalized adaptive program $\mathcal{AP} = \langle A, E, \varphi, \mathcal{P}_0 \rangle$, the global semantic of $\mathcal{AP}$ over an mLTL formula κ is represented by a global labelled transition system(GLTS), $\Upsilon^{\Psi}_{\mathcal{AP}} = \langle A, E, \psi, \mathcal{P}_0, \mathcal{L} \rangle$, where

- Ψ: the LTL formula set extracted from κ
- ψ: $A \times E \rightarrow A$ is the transition relation. $(\mathcal{P}_1, e, \mathcal{P}_2) \in \psi$, if there exists an adaptation relation $(\mathcal{P}_1, s, e, \mathcal{P}_2, s') \in \varphi$
- $\mathcal{L}$: $A \rightarrow 2^{\Psi}$ is a labelling function

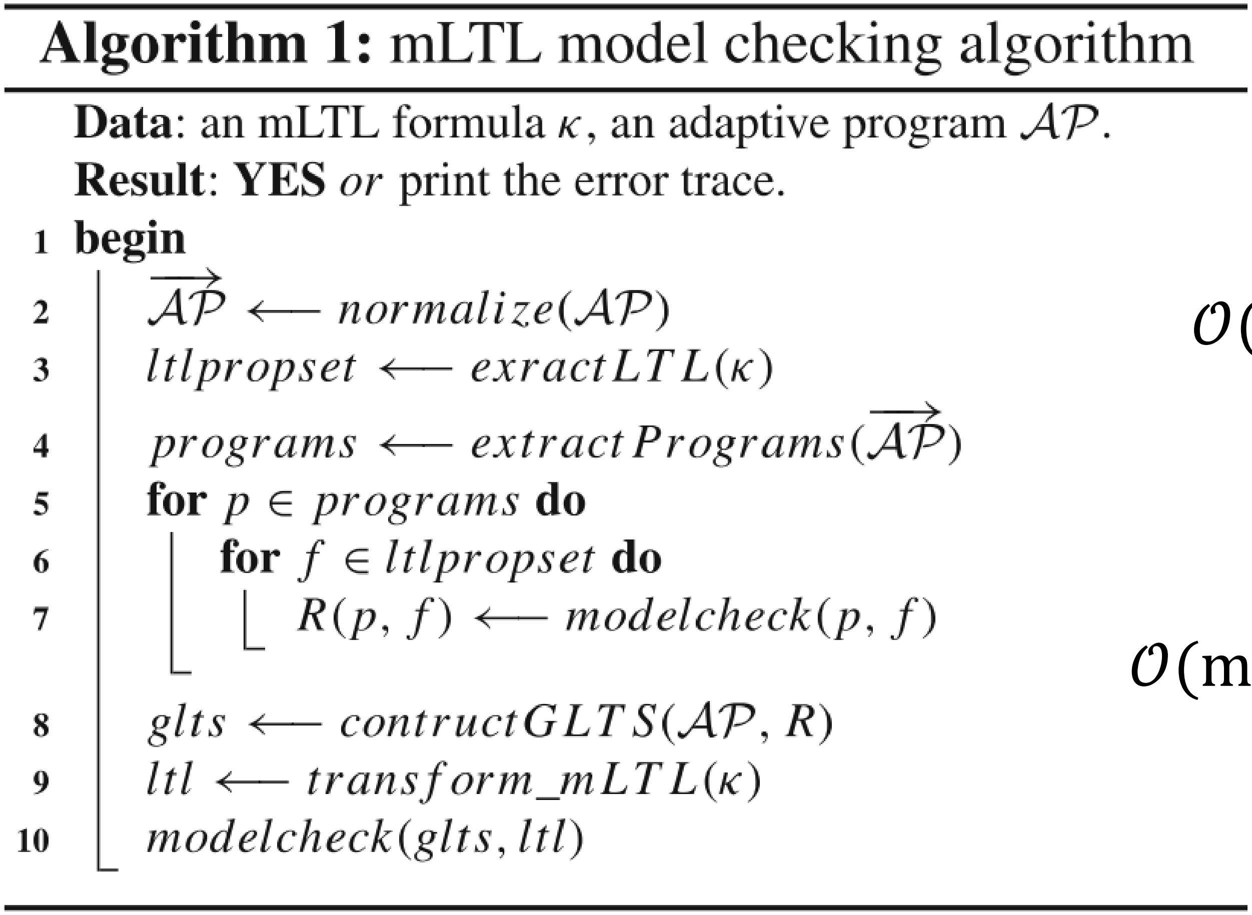
- **a GLTS is constructed for each mLTL formula on *AP***
- **we extract an LTL formula χ from the $[\chi]$part, and $\Box v$ from the $@v$ part of the mLTL formula**
- **Labelling function $\mathcal{L}$: indicates that which LTL formula is satisfied on the non-adaptive program.**
  - $<\mathcal{P}, \{\chi_1, \chi_2, \chi_3\}> \in \mathcal{L}$ **means that LTL formulae χ_1, χ_2, χ_3 hold on the non-adaptive program $\mathcal{P}$**

Translate the mLTL formula to LTL

$$\begin{aligned}
\mathcal{T}(\mathbf{true}) &= \mathbf{true} \\
\mathcal{T}([\chi]) &= p(\chi) \\
\mathcal{T}(@v) &= p(\Box v) \\
\mathcal{T}(\kappa_1 \wedge \kappa_2) &= \mathcal{T}(\kappa_1) \wedge \mathcal{T}(\kappa_2) \\
\mathcal{T}(\neg\kappa_1) &= \neg\mathcal{T}(\kappa_1) \\
\mathcal{T}(\bigcirc_m \kappa_1) &= \bigcirc\mathcal{T}(\kappa_1) \\
\mathcal{T}(\kappa_1 \mathcal{U}_m \kappa_2) &= \mathcal{T}(\kappa_1)\mathcal{U}\mathcal{T}(\kappa_2) \\
\mathcal{T}(\Diamond_m \kappa_1) &= \Diamond\mathcal{T}(\kappa_1) \\
\mathcal{T}(\Box_m \kappa_1) &= \Box\mathcal{T}(\kappa_1) \\
\mathcal{T}(\kappa_1 \vee \kappa_2) &= \mathcal{T}(\kappa_1) \vee \mathcal{T}(\kappa_2) \\
\mathcal{T}(\kappa_1 \rightarrow \kappa_2) &= \mathcal{T}(\kappa_1) \rightarrow \mathcal{T}(\kappa_2) \\
\mathcal{T}(\kappa_1 \leftrightarrow \kappa_2) &= \mathcal{T}(\kappa_1) \leftrightarrow \mathcal{T}(\kappa_2)
\end{aligned}$$

Theorem 1 *An mLTL formula κ holds on an adaptive program $\mathcal{AP}$, if and only if $\mathcal{T}(\kappa)$ holds on the GLTS of $\overrightarrow{\mathcal{AP}}$.*

Algorithm of mLTL model checking



Algorithm 1: mLTL model checking algorithm

```
Data: an mLTL formula $\kappa$, an adaptive program $\mathcal{AP}$.
Result: YES or print the error trace.
begin
    $\overrightarrow{\mathcal{AP}} \longleftarrow normalize(\mathcal{AP})$
    $ltlpropset \longleftarrow exractLTL(\kappa)$
    $programs \longleftarrow extractPrograms(\overrightarrow{\mathcal{AP}})$
    for $p \in programs$ do
        for $f \in ltlpropset$ do
            $R(p, f) \longleftarrow modelcheck(p, f)$
    $glts \longleftarrow contructGLTS(\mathcal{AP}, R)$
    $ltl \longleftarrow transform\_mLTL(\kappa)$
    $modelcheck(glts, ltl)$
```

time complexity:

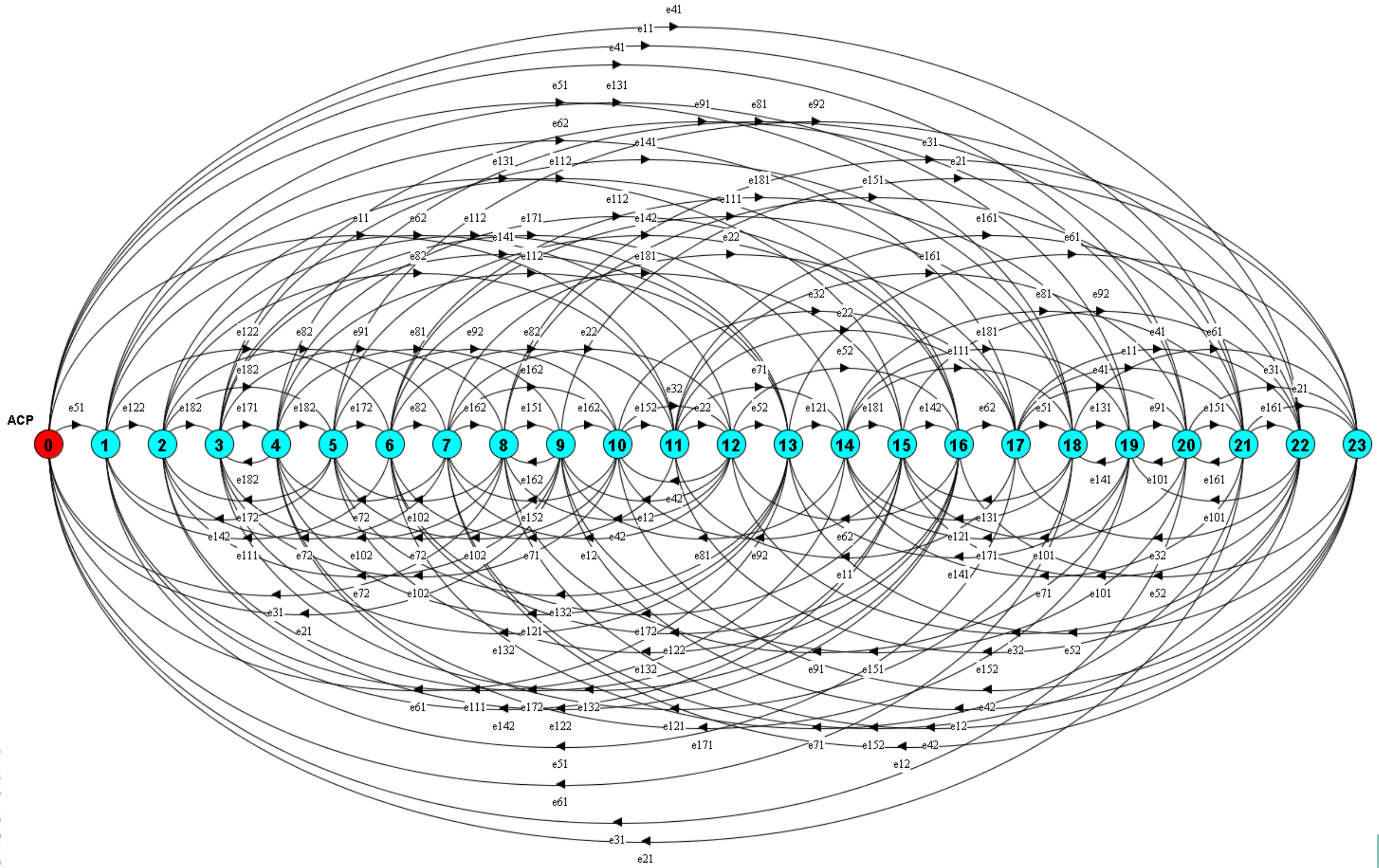
$$\mathcal{O}(n * k * 2^{|\chi|} * |\mathcal{P}| + 2^{|\mathcal{T}(k)|} * |\overrightarrow{AP}|)$$

space complexity:

$$\mathcal{O}(\max(2^{|\chi|} * |\mathcal{P}|, 2^{|\mathcal{T}(k)|} * |\overrightarrow{AP}|))$$

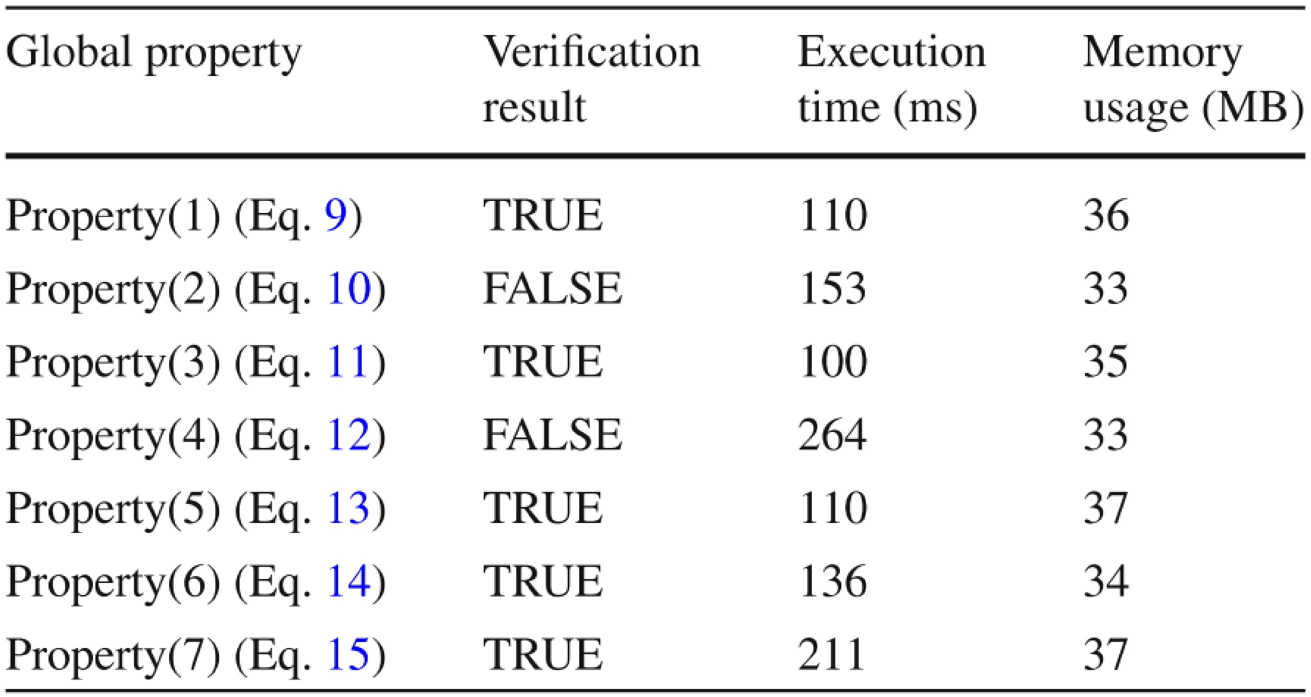
Experiments in LTSA

- normalization of ACP has 24 non-adaptive programs and 144 adaptations



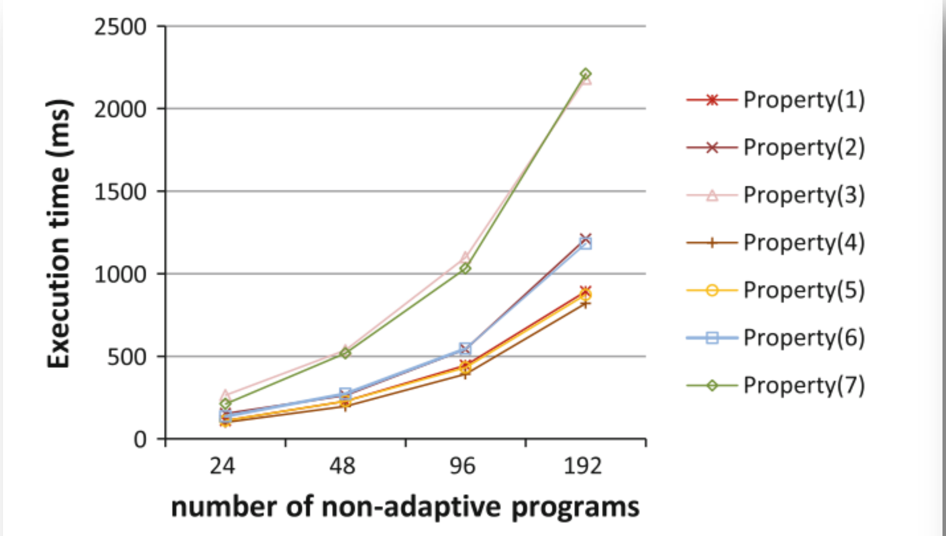
Experiments in LTSA

(1) Gm (@ n o r m a l − > [[ ] (INPUT − > <> SENT)])
(2) Gm (@ n o r m a l − > [[ ] <> INPUT])
(3) Gm ([[ ] (INPUT − > <> SENT)])
(4) Gm ([! [ ] <> INPUT] − > Xm [[ ] <> INPUT])
(5) Fm ([[ ] (INPUT − > <>DATACOMPRS)])
(6) Gm ([[ ] ((DES64ENCD − > ! < > DES128ENCD) && (DES128ENCD − > ! < > DES64ENCD))])
(7) Gm ([[ ] (INPUT − > <>ENCRYPERR)] − > Fm ([[ ] (INPUT − > <>SENT)]))

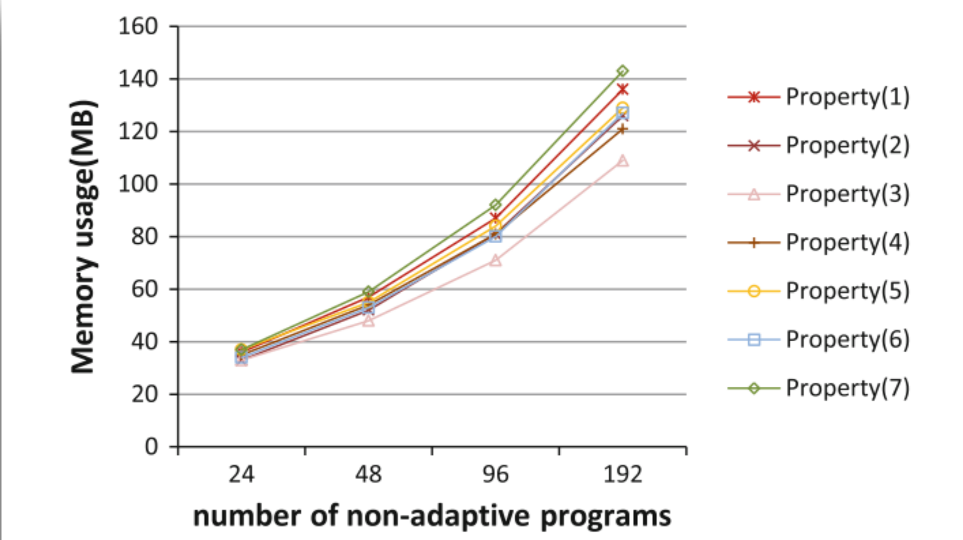


Global property	Verification result	Execution time (ms)	Memory usage (MB)
Property(1) (Eq. 9)	TRUE	110	36
Property(2) (Eq. 10)	FALSE	153	33
Property(3) (Eq. 11)	TRUE	100	35
Property(4) (Eq. 12)	FALSE	264	33
Property(5) (Eq. 13)	TRUE	110	37
Property(6) (Eq. 14)	TRUE	136	34
Property(7) (Eq. 15)	TRUE	211	37

Experiments in LTSA



(a) Execution time changes with number of non-adaptive programs



(b) Memory usage changes with number of non-adaptive programs

- **the time consumed by our approach is linear to the number of non-adaptive programs in a normalized adaptive program.**
- **the memory usage of our approach is approximately linear to the number of non-adaptive programs in a normalized adaptive program.**

Conclusion

- we propose an approach to formal development of global specifications for dynamically adaptive programs.
- mLTL does not require all possible alternative adaptations to be specified during designing systems and is effective for large scale of adaptations
- mLTL integrated with LTL and A-LTL could specify the adaptive software effectively.

Thanks

Q&A