Test generation using verification

Thierry Jéron
Irisa/Inria Rennes, France
http://www.irisa.fr/vertecs/

Adapted from

B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva,
Symbolic Test Selection based on Approximate Analysis,
in TACAS'05, LNCS 3440, Edinburgh (Scottland), April 2005.

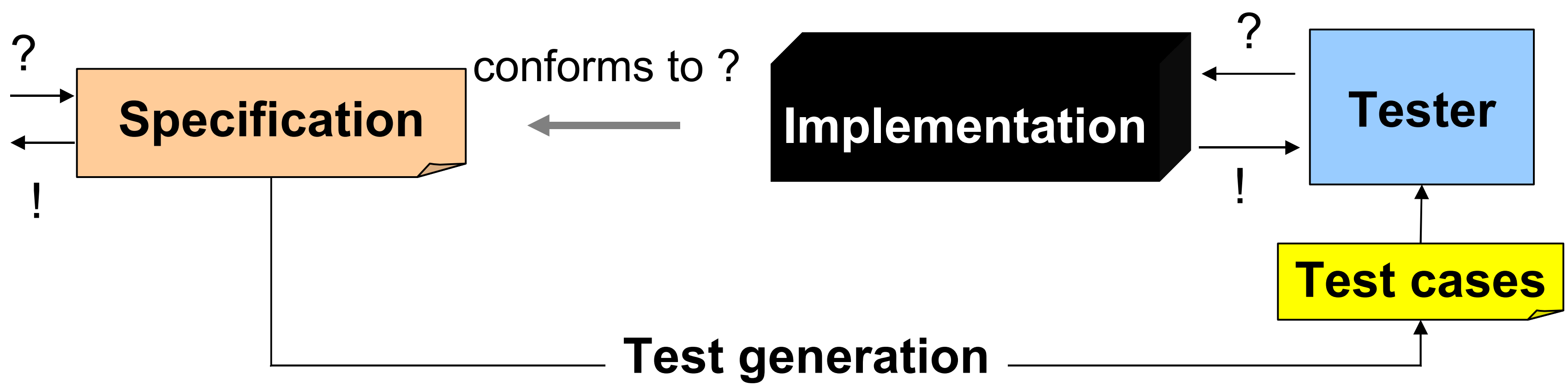
Vlad Rusu, Hervé Marchand, Thierry Jéron,
Automatic Verification and Conformance Testing for Validating Safety Properties of Reactive Systems,
in Formal Methods 2005 (FM05), July 2005.

Conformance Testing of reactive systems

Problem:

check if a black-box implementation of a reactive system conforms to a given specification by experimenting with test cases

- the source code of the implementation is unknown (or not used)
- only the interface is known
- and can be controlled/observed by the environment



Outline

- **The *ioSTS* model**
- **Conformance Testing Theory with ioco**
- **Test selection using approximate analysis**
- **Conclusion**

1. The *ioSTS* model

$\mathcal{S} = (V_S, \Theta_S, \Sigma, T_S)$

with

- V_S: variables $\ni$ loc
- Θ_S: initial condition

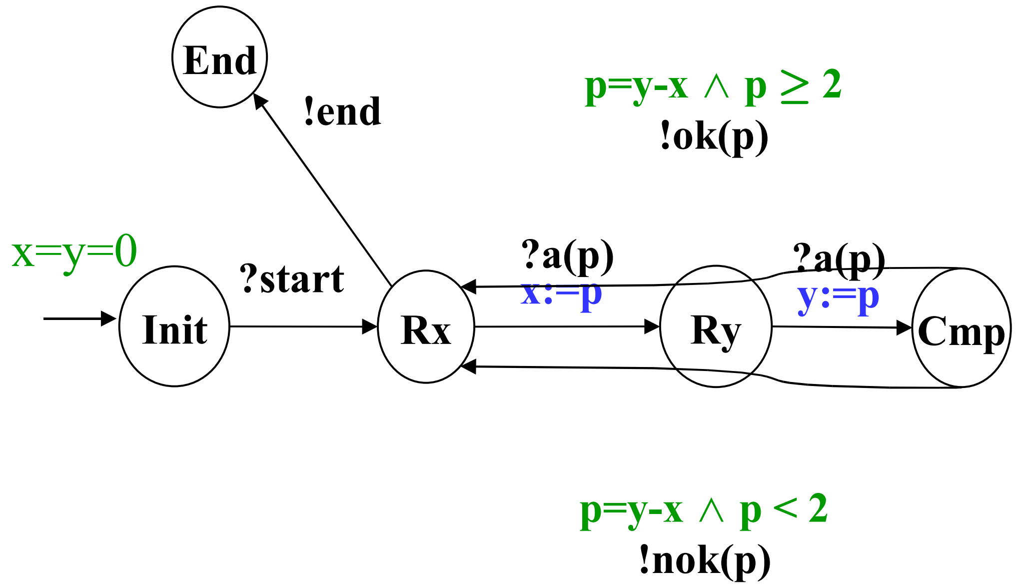
  with unique solution
- $\Sigma = \Sigma_! \cup \Sigma_? \cup \Sigma_\tau$:

  alphabet of actions

  with comm. parameters P
- T_S: transition relation

  $[a(p) : G(v_S,p);\ v_S := A(v_S,p)]$

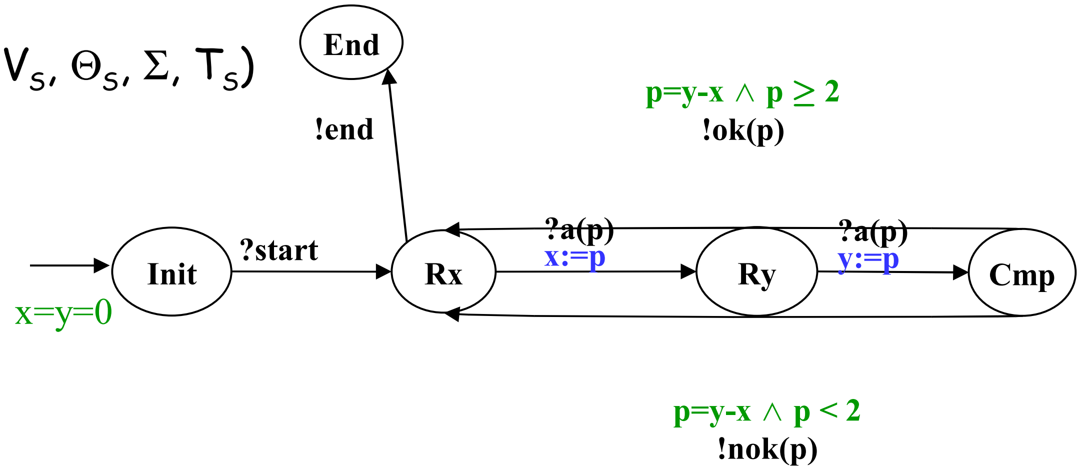
  action guard assignment

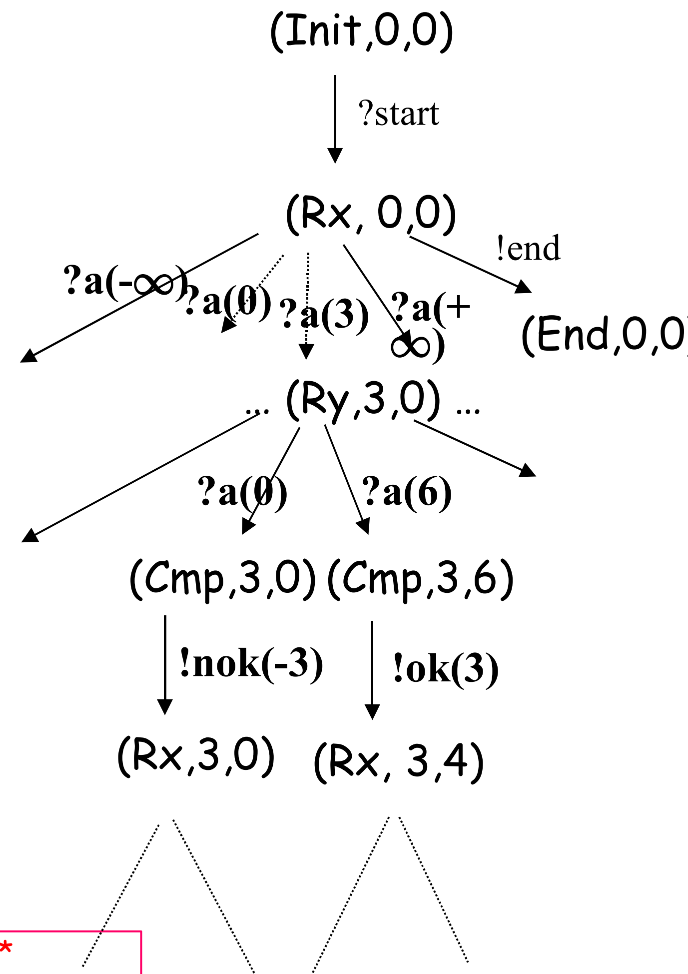


Hyp: satisfiability of guards is decidable.

ioLTS semantics of *ioSTS*

$\mathcal{S} = (V_S, \Theta_S, \Sigma, T_S)$





$[[\mathcal{S}]] =$ $S = (Q, Q_0, \Lambda, \rightarrow)$ with

- $Q = \times_{v \in V_S} Dom(v)$ i.e. $q = \langle v_0, \ldots v_n \rangle$
- $Q_0 = \Theta_S = \{\langle v_0, \ldots v_n \rangle\}$
- $\Lambda = \{\langle a,\pi \rangle \mid a \in \Sigma$ Æ $\pi \in Dom(sig(a))\}$
- $[a,p,G,A] \in T,\ q \in Q, \exists\ \pi \in Dom(sig(a)),\ G(q,\pi)$

  $q - a(\pi) \rightarrow q' = A(q,\pi)$

Runs(S): $q_0 \rightarrow^{a_1(\pi_1)} q_1 \rightarrow^{a_2(\pi_2)} q_2 \ldots \in Q_0.(\Lambda.Q)^*$

Tr (S) : $proj_{\Lambda_{vis}}(runs(S))$

Simplifying asumptions on *ioSTS* (for this talk)

1. *ioSTS* are supposed to be deterministic

Determinisation of *ioSTS* into *ioSTS* is not always possible

$\Rightarrow$ restrictions needed e.g.

- No internal loop
- Finite lookahead

2. Quiescence is not treated here

$Æ_{a \in \Sigma_! \cup \Sigma_\tau} Æ_{t=[a,p,G,A]} \forall p, \neg G(v,p)$

$!\delta$

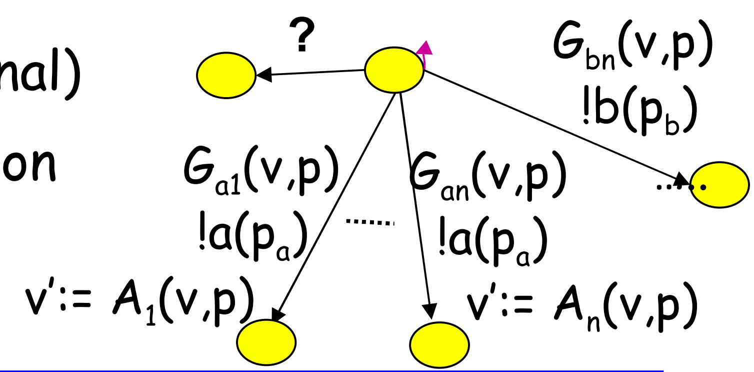
$\Delta(\mathcal{S})$: explicit quiescence by adding loops with $!\delta$

in all quiescent states (no fireable output/internal)

augment guard model with universal quantification

not a real problem

STr(S)= Tr(Δ(S))



2. Conformance Testing Theory with ioco [Tretmans 96]

- Specification: known ioLTS S (semantics of an ioSTS)
- Implementation: unknown ioLTS I

- Conformance: I ioco S : partial inclusion of STr(I) in STr(S)

- Test cases : ioSTS TC + Verdicts
  - Execution: parallel composition $\Delta(I) \,||\, TC$
  - Verdicts: TC may fail I

- Test generation: gen_test: $S \rightarrow TS = \{TC_1\, TC2, \ldots\}$

  Requested Properties of TS: may fail $\leftrightarrow \neg$ I ioco S

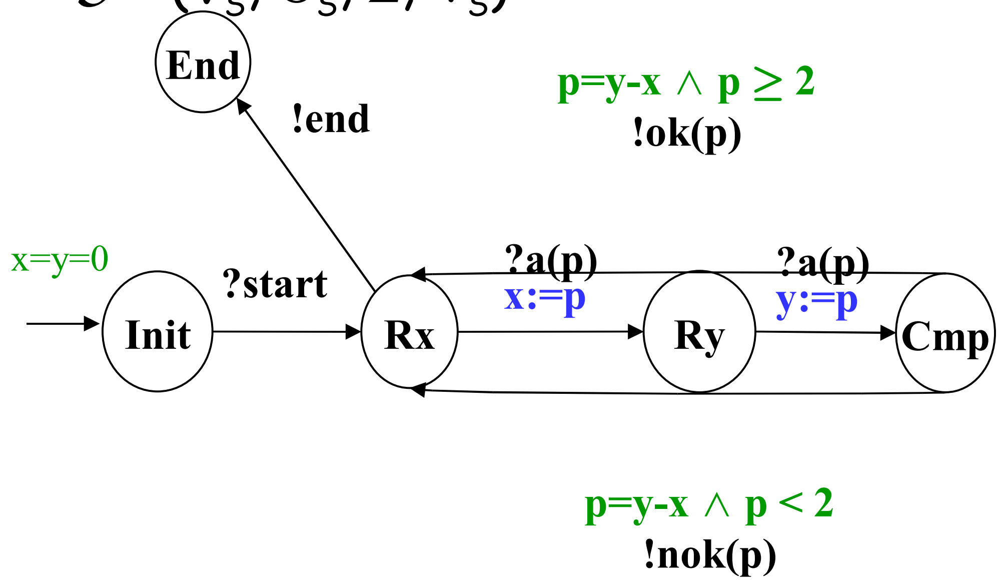
  (soundness, limit exhaustiveness)

Simplification: an automata/language point of view

Conformance relation

I ioco S ≜ ∀ σ ∈ STr(S),
Out(Δ(I) after σ) ⊆ Out(Δ(S) after σ)

$\mathcal{S} = (V_S, \Theta_S, \Sigma, T_S)$



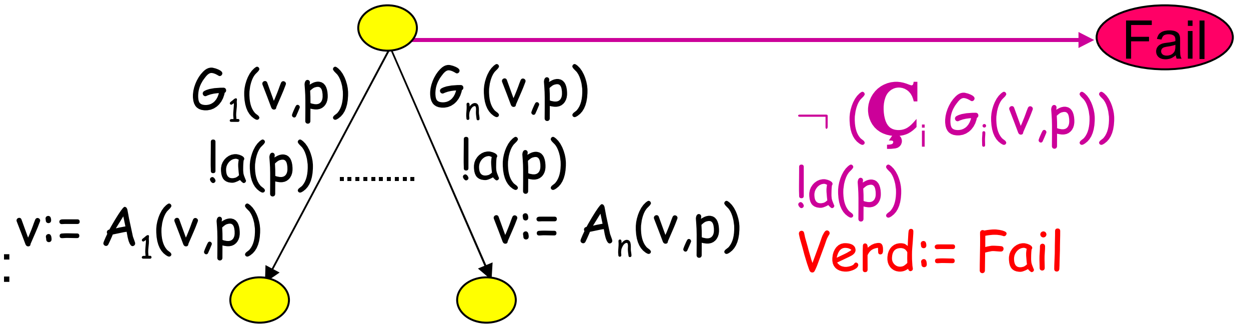
Conformant traces, e.g.
?start . ?a(4) . ?a(6) . !ok(2)
?start . !end
?start. ?start
(unspecified input allowed)

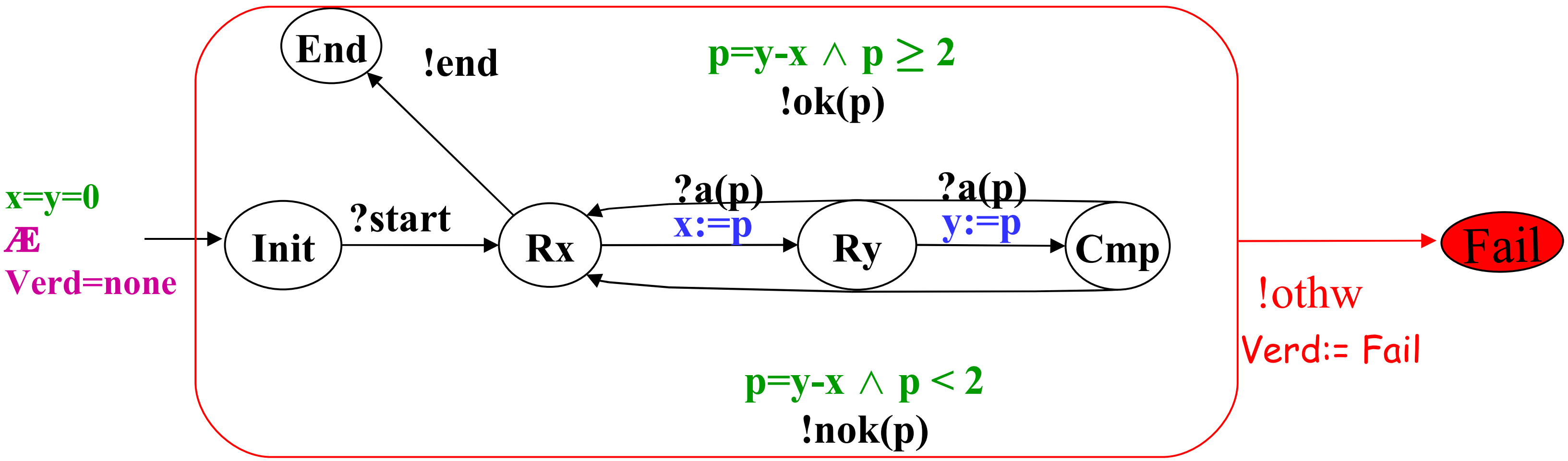
Non-conformant traces, e.g.
?start . ?a(5) . ?a(7) . !ok(3)
?start . ?a(6) . ?a(8) . !nok(2)

Prop: I ioco S ⇔ STr(I) Å [STr(S) . $\Lambda_!^\delta$ \ STr(S)] = ∅
[non-conformant behaviours]

Canonical Tester Can($\mathcal{S}$): observer of non-conformant behaviours

1. Add new variable Verd with initial value none
   i.e. Θ Æ Verd = none
2. ∀ ouput !a, ∀ t carrying !a :





$$STr_{Fail}(Can(\mathcal{S})) = [STr(\mathcal{S}).\ \Lambda_!^{\delta} \setminus STr(\mathcal{S})]$$

$$\Rightarrow I \text{ ioco } S \Leftrightarrow STr(I) \text{ Å } STr_{Fail}(Can(\mathcal{S})) = \emptyset$$

Test cases, test execution, verdicts and properties

Test Case: deterministic ioSTS TC= $(V_{TC}, \Theta_{TC}, \Sigma, T_{TC})$

+ verdict boolean variables: Fail, Pass, Inconc, …

plays the role of an observer delivering verdicts

$Tr_{Fail}(TC)$

Test suite: (infinite) set of test cases TS= $\{TC_1, TC_2, \ldots\}$

Test execution: TC || Δ(I) synchronization on common actions

Possible rejection of I by TC:

reachability of Fail in TC || Δ(I) i.e.

TC may fail I $\triangleq$ STr(I) Å $Tr_{Fail}(TC) \neq \emptyset$

TS may fail I $\triangleq$ STr(I) Å $_{TC \in TS} Tr_{Fail}(TC) \neq \emptyset$

Test suite properties

Possible rejection by a TC should correspond to non-conformance and vice-versa

$\neg$ TS may fail I $\Leftrightarrow$ STr(I) Å $_{TC \in TS} Tr_{Fail}(TC) = \emptyset$

I ioco S $\Leftrightarrow$ STr(I) Å $Tr_{Fail}(Can(S)) = \emptyset$

TS is sound $\triangleq \forall$ I, (I ioco S $\Rightarrow \neg$ TS may fail I)

$\Leftrightarrow \quad _{TC \in TS} Tr_{Fail}(TC) \subseteq Tr_{Fail}(Can(S))$

TS is exhaustive $\triangleq \forall$ I, ($\neg$ TS may fail I $\Rightarrow$ I ioco S)

$\Leftrightarrow \quad _{TC \in TS} Tr_{Fail}(TC) \supseteq Tr_{Fail}(Can(S))$

3. Test selection for *ioSTS*

TS = {Can(S)} is a sound and exhaustive test suite but

- has too many (infinite) behaviours
- does not allow to control the implementation during testing

$\Rightarrow$ Test selection

- renounce to exhaustiveness in practice,
- focus on some targetted behaviours of Can(S)
  select a finite TS likely to discover non-conformances
- use test purposes

Test purposes

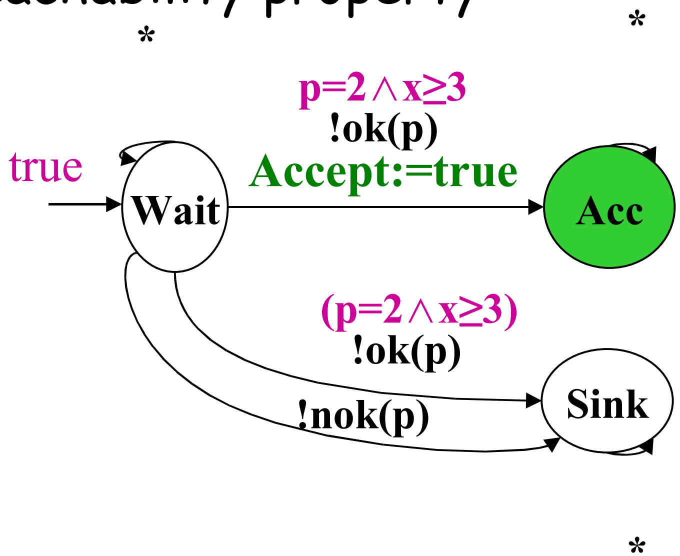
$\mathcal{TP} = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})$

Observer of actions and variables of S

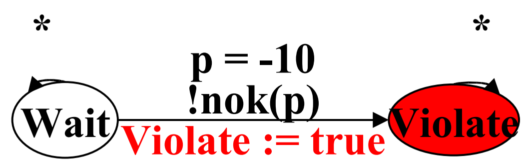
$[a(p) : G(v_S, v_{TP}, p);\ v_{TP} := A(v_S, v_{TP}, p)] \in T_{TP}$

Hyp : complete and deterministic

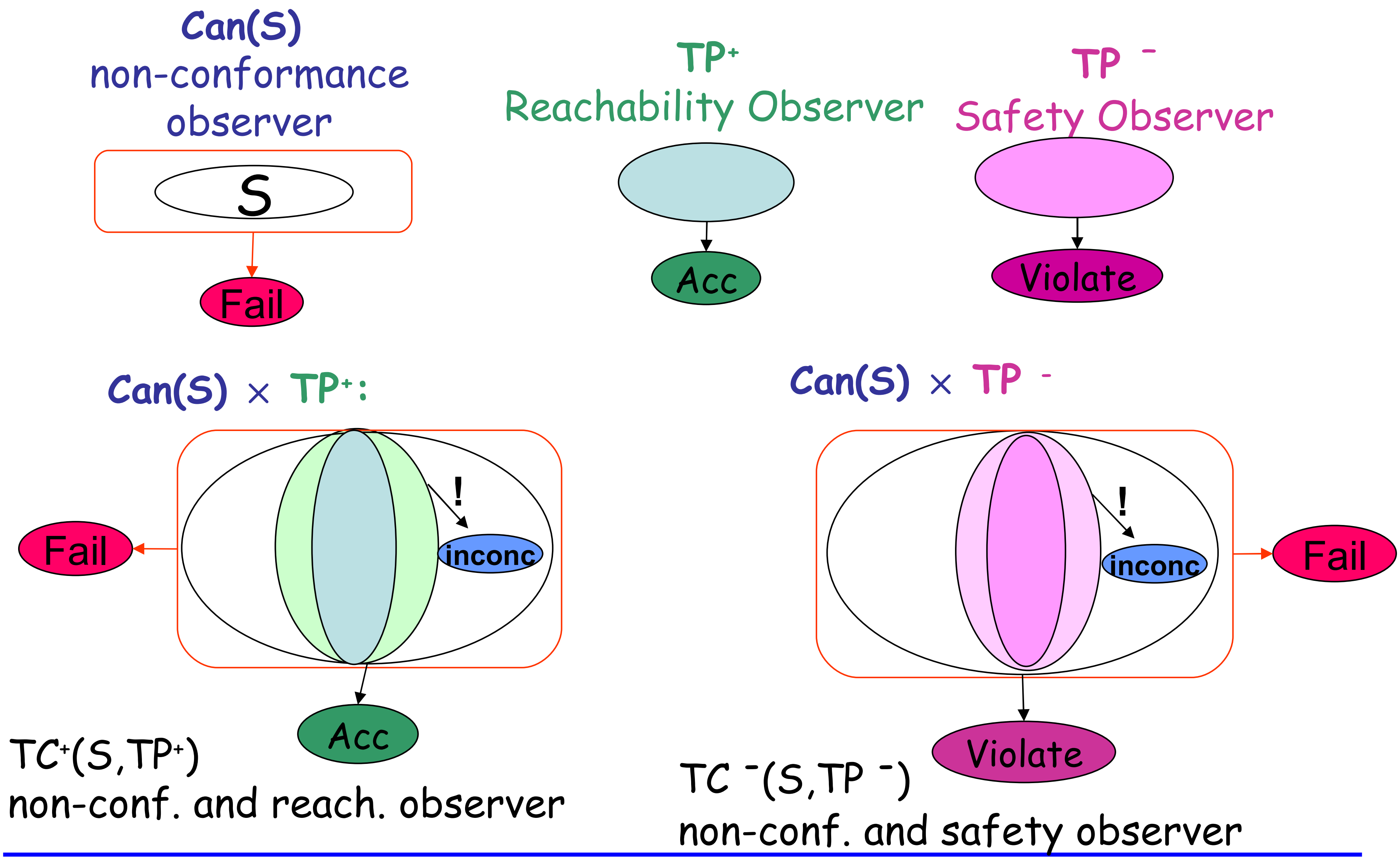
$\mathcal{TP}$+: reachability property



$\mathcal{TP}$ -: negation of safety property



General principle



Syntactical product $\mathcal{Can}(\mathcal{S}) \times \mathcal{TP}$

$\mathcal{Can}(\mathcal{S}) = (V_A, \Theta_A, \Sigma, T_A)$

$G_1(v_S,p)$
$a(p)$
$v'_S := A_1(v_S,p)$

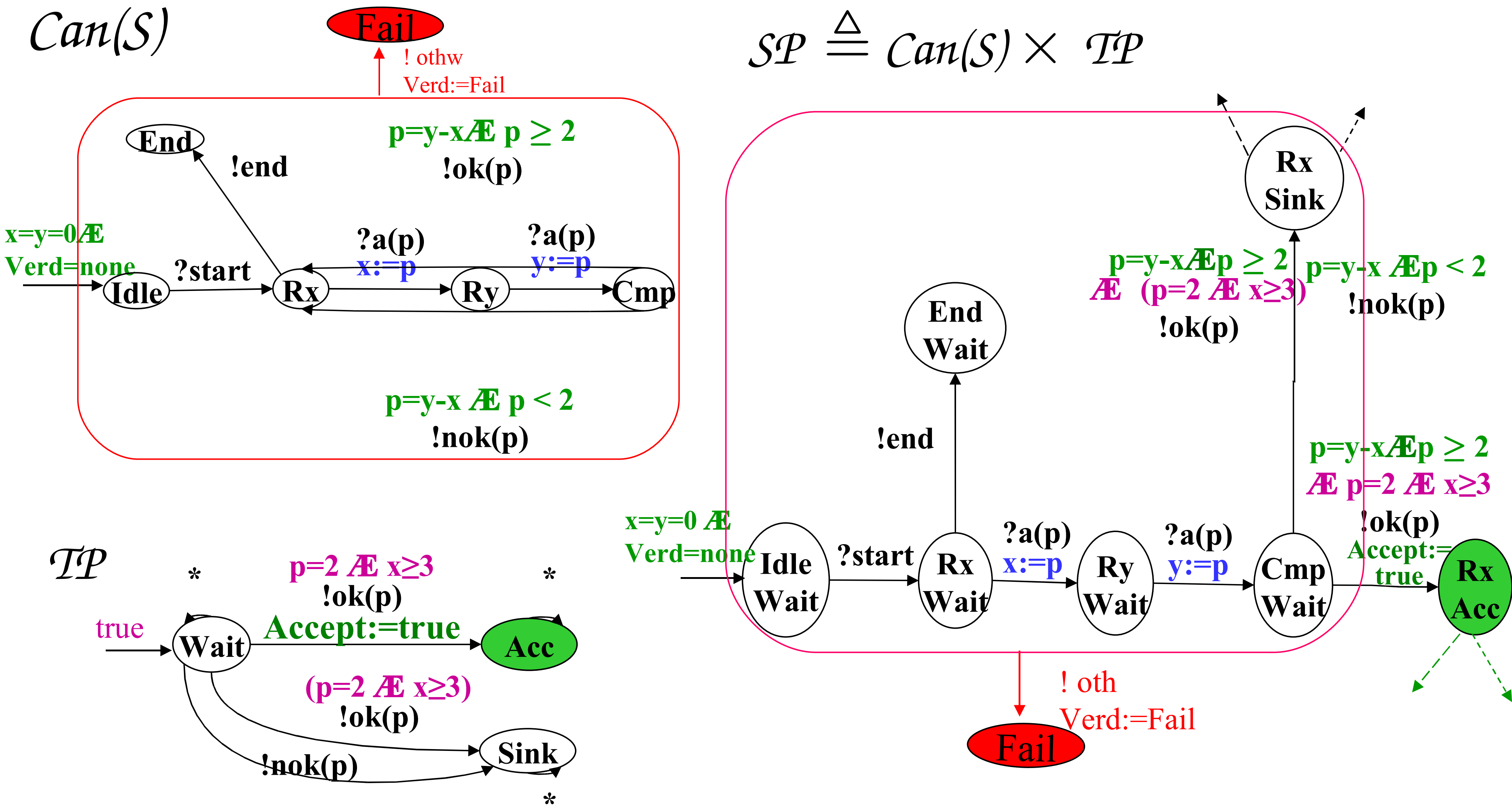
$\mathcal{TP} = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})$

$G_2(v_S,v_{TP},p)$
$a(p)$
$v'_{TP} := A_2(v_S, v_{TP},p)$

$\mathcal{SP} = \mathcal{Can}(\mathcal{S}) \times \mathcal{TP} =$
$(V_S \cup V_{TP}, \Theta_S$ Å $\Theta_{TP}, \Sigma, T_{S\times TP})$

$G_1(v_S,p)$ Æ $G_2(v_S, v_{TP}, p)$
$a(p)$
$\langle v'_S; v'_{TP}\rangle :=$
$\langle A_1(v_S,p);A_2(v_S, v_{TP},p)\rangle$

Syntactical product



1. Assignment of Pass verdicts

Pass: $Tr_{Accept}(\mathcal{SP})$ $\rightarrow$ Observer of $Tr_{Accept}(SP)$

$\mathcal{C}an(\mathcal{S}) \times \mathcal{TP}$

G(v,p)
a(p)
v:= A(v,p)

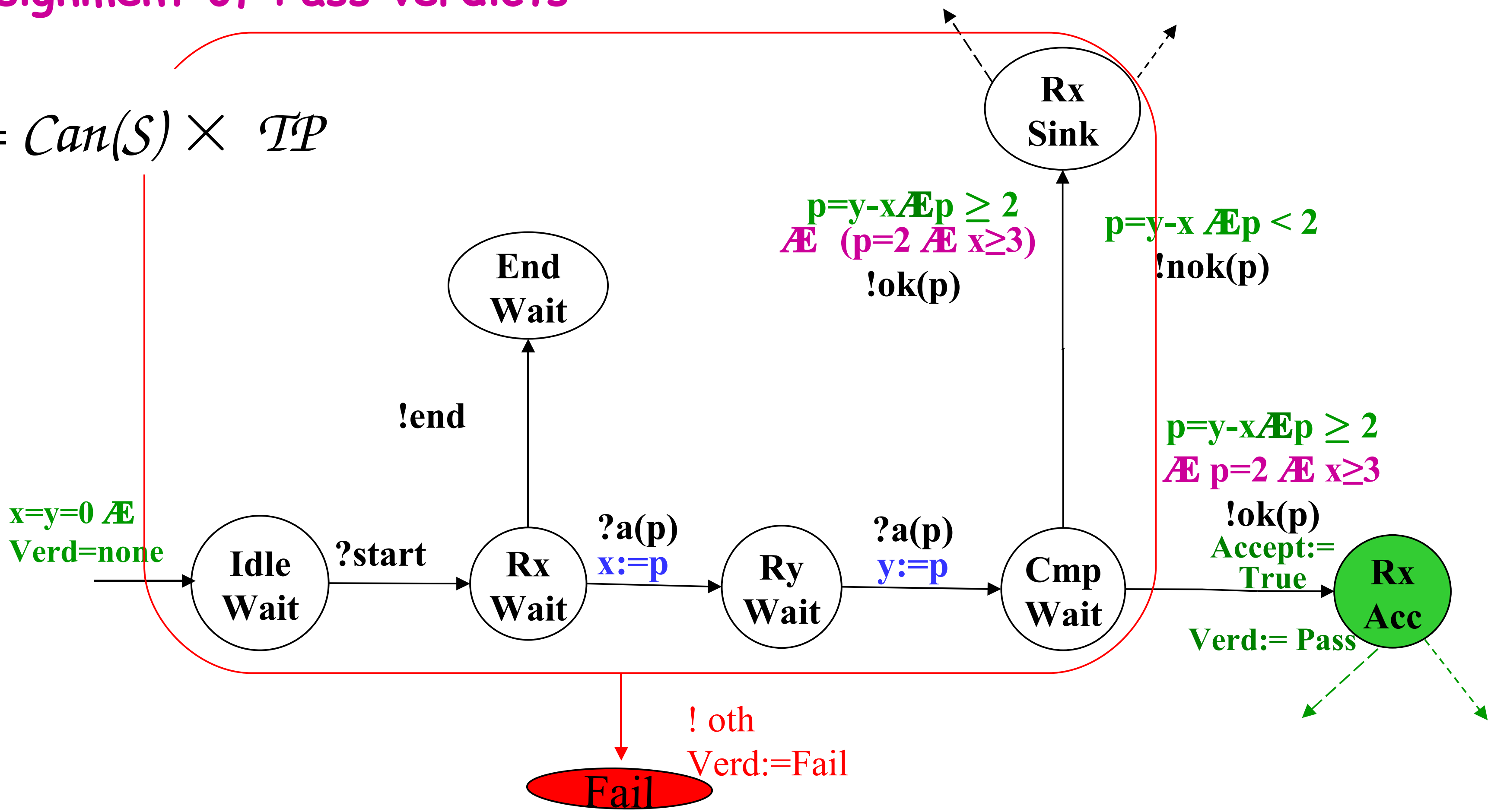
G(v,p) Æ Verd = none
a(p)
v:= A(v,p)
Verd :=if A_{Accept} then Pass
else Verd

$\mathcal{CTG}$

Syntactical Test Selection (1)

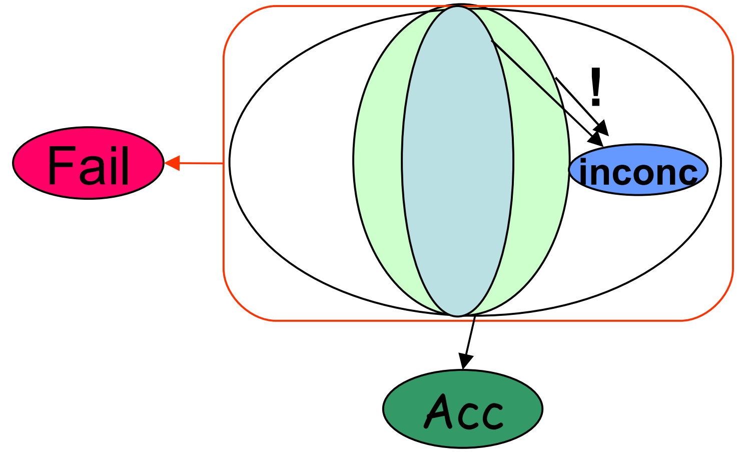
1. Assignment of Pass verdicts

$$\mathcal{SP} = Can(\mathcal{S}) \times \mathcal{TP}$$



After product we get

- $Tr(SP) = Tr(Can(S))$
- $Tr_{Fail}(SP) = Tr_{Fail}\ (Can(S))$
- $Tr_{Pass}\ (SP) = Tr_{Accept}\ (TP)$ ÅSTr(S)



SP is both a non-conformance and reachability observer but has too much behaviours: (Tr(Can(S))

Goal of selection:

focus on Tr_{Accept} (TP) ÅSTr(S),

detect unfeasible traces to Accept

Amounts to compute co-reach(Accept)

Undecidable $\Rightarrow$ over-approximate

Syntactical Test Selection (2)

2. Selection and assignment of Inconc verdicts

coreach(Accept) not computable $\Rightarrow$ compute over-approximation:

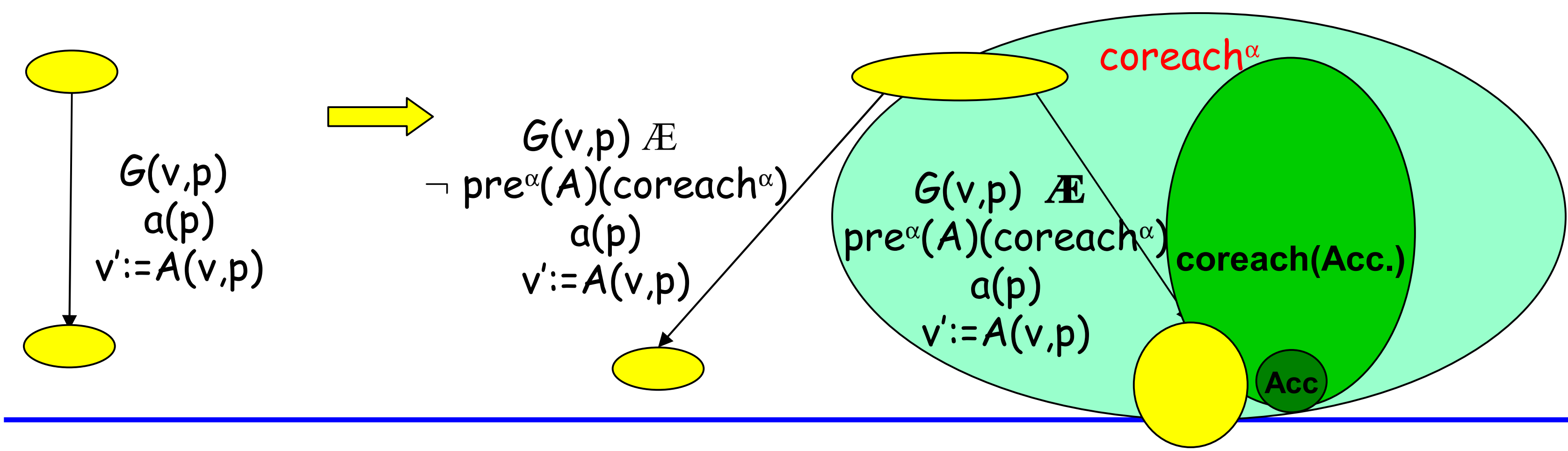
$$coreach^{\alpha} \supseteq coreach(Accept)$$

$$\forall \text{ assignment } A,\ pre^{\alpha}(A)(coreach^{\alpha}) \supseteq pre(A)(coreach^{\alpha})$$

Idea: $pre^{\alpha}(A)(coreach^{\alpha})$ = Nec. Cond. to go into $coreach^{\alpha}$

$\neg pre^{\alpha}(A)(coreach^{\alpha})$ = Suf. Cond. to go outside $coreach^{\alpha}$

$\subseteq$ outisde coreach(Accept)

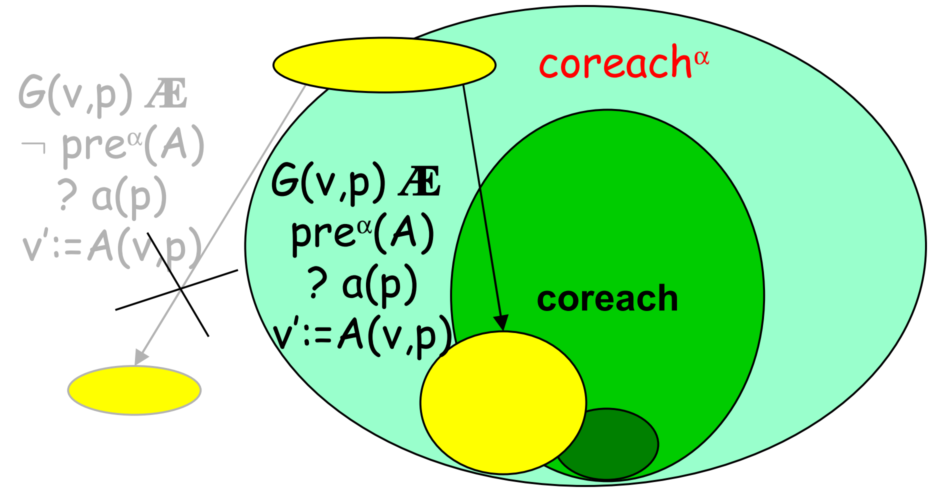


Syntactical test selection (3): guard strengthening

Rule for inputs of *s*:

keep conditions leading to coreach$^{\alpha}$,

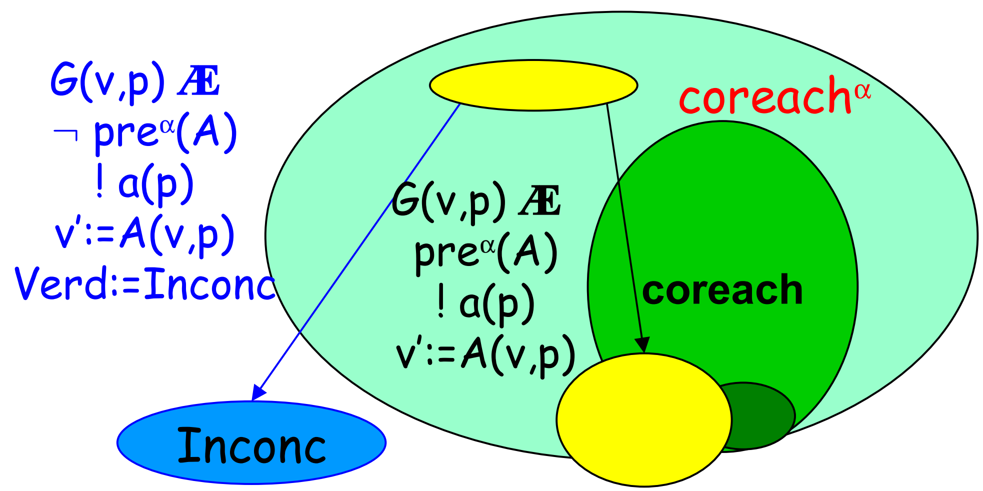
cut other ones (controllable):



Rule for outputs of *s*

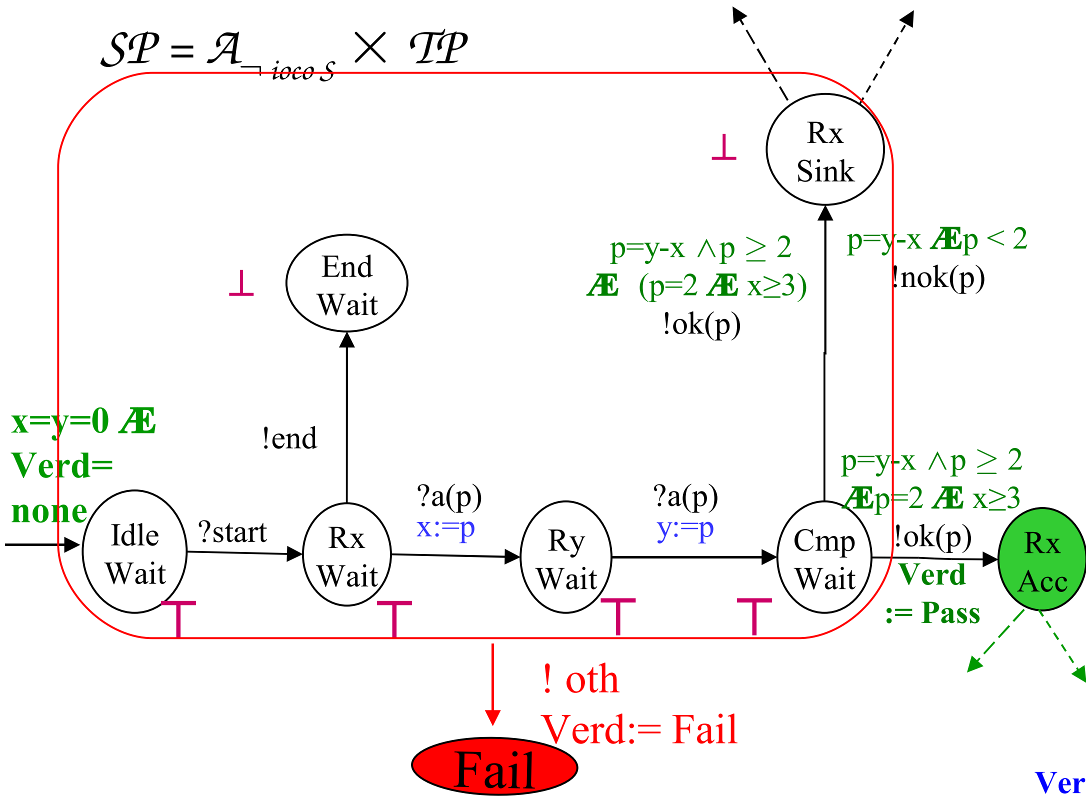
keep all conditions (uncontrollable),

those leading outside coreach$^{\alpha}$ produce Inconc:

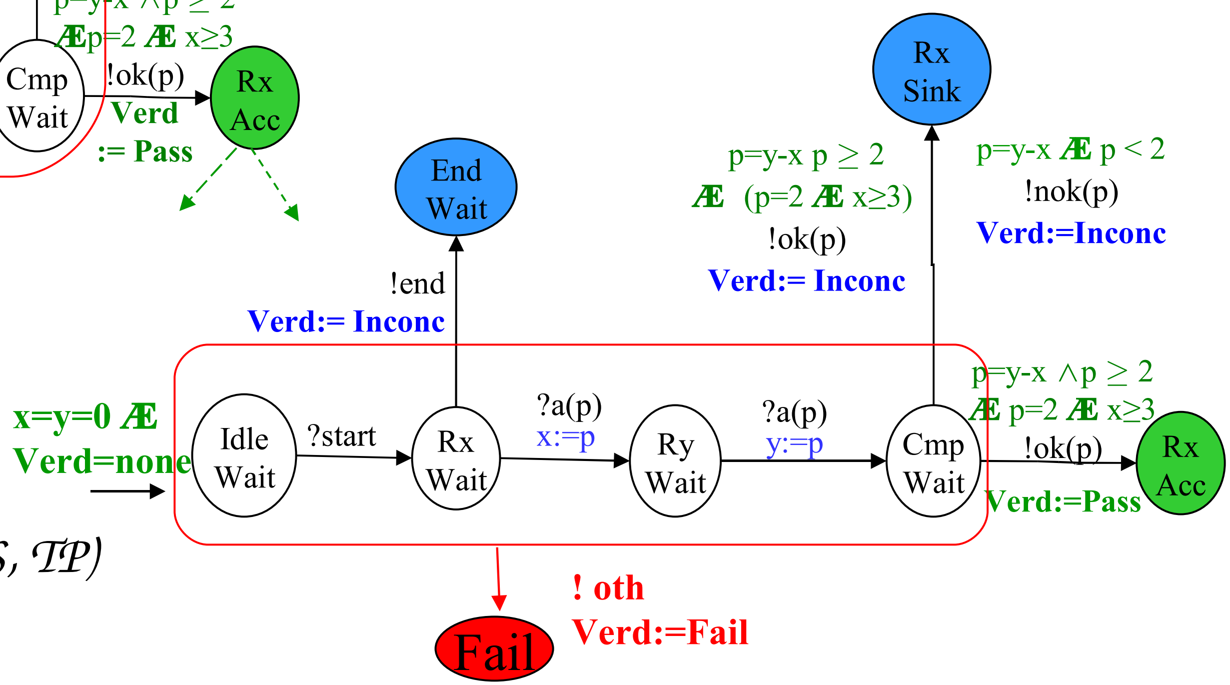


Test selection: example
1st over-approximation : control

$\mathcal{SP} = \mathcal{A}_{\neg\, ioco\ \mathcal{S}} \times \mathcal{TP}$

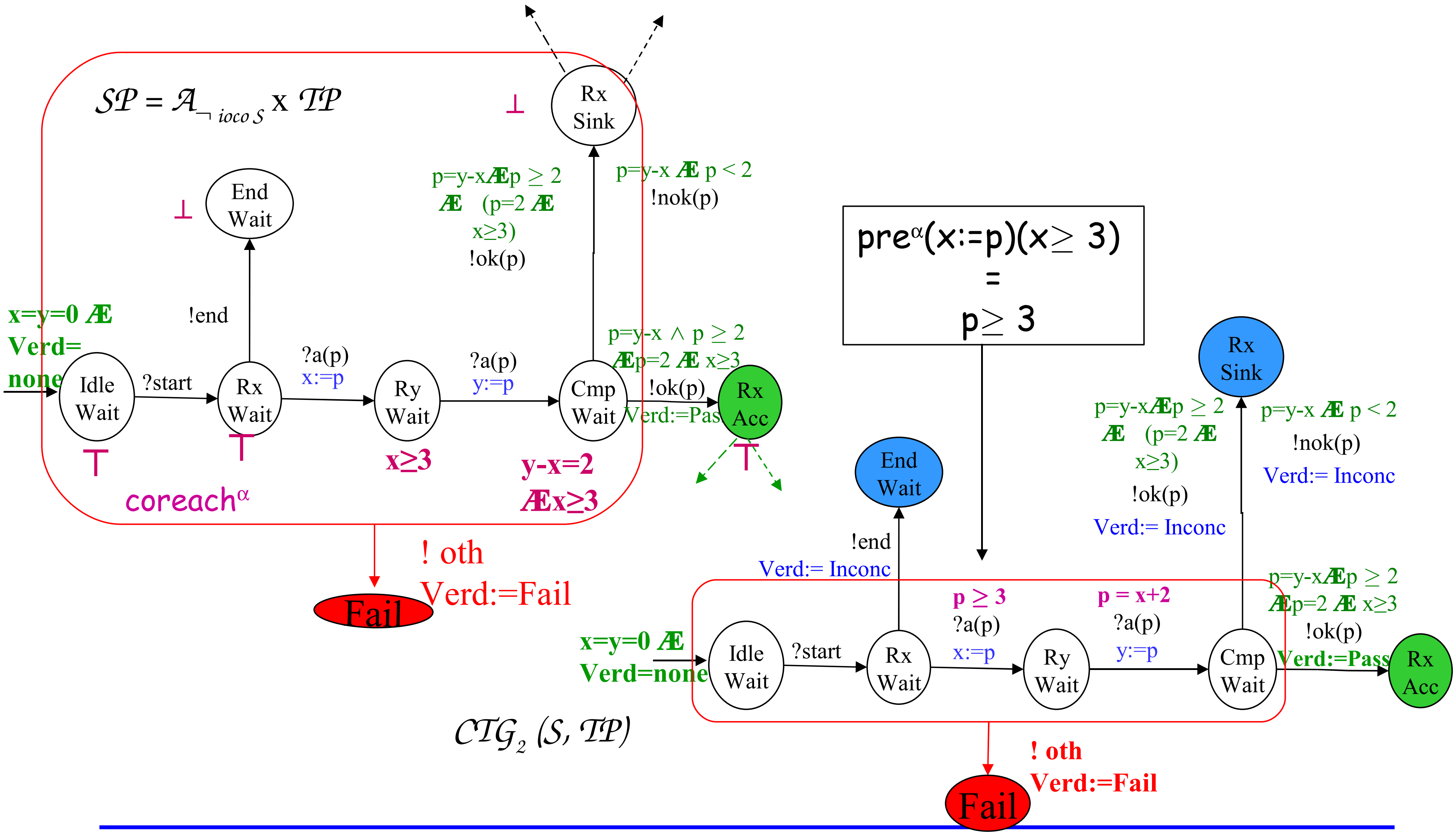


Abstraction on control:
only the location is taken into account in coreach$^{\alpha}$

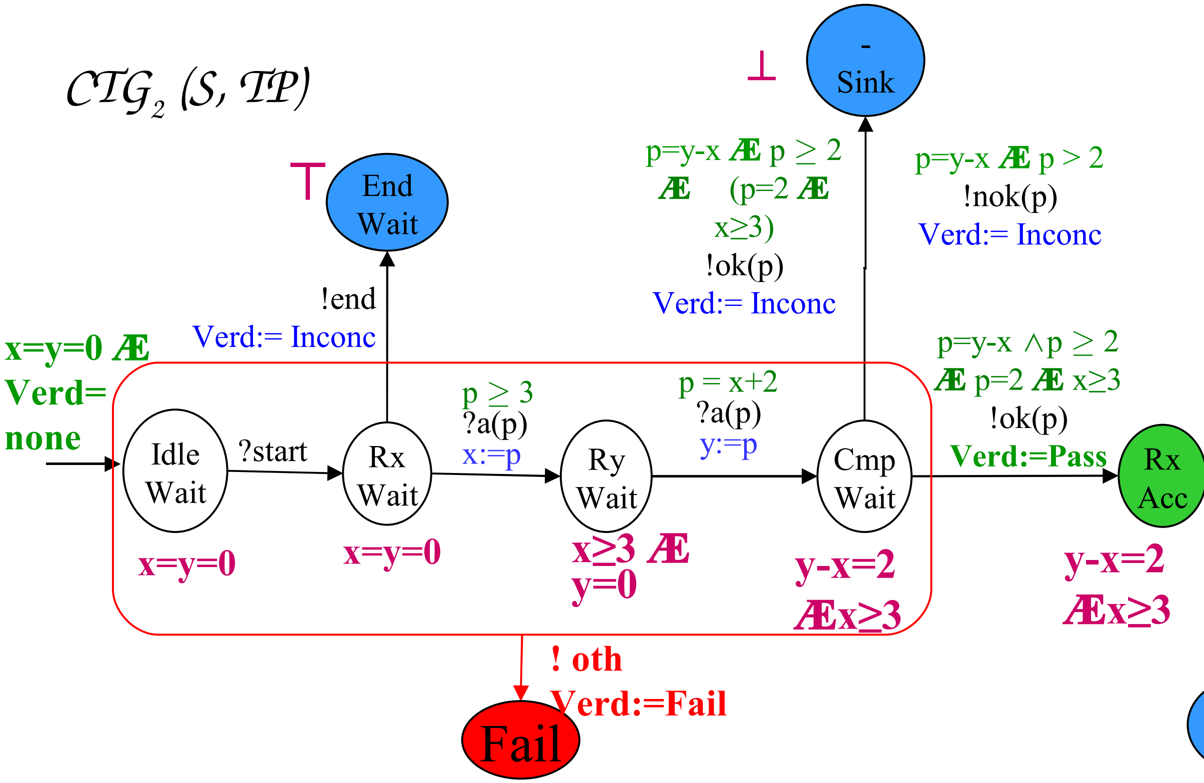


$\mathcal{CTG}_1\ (\mathcal{S}, \mathcal{TP})$

Test selection: example
2nd approximation computed by NBAC (convex polyhedra)

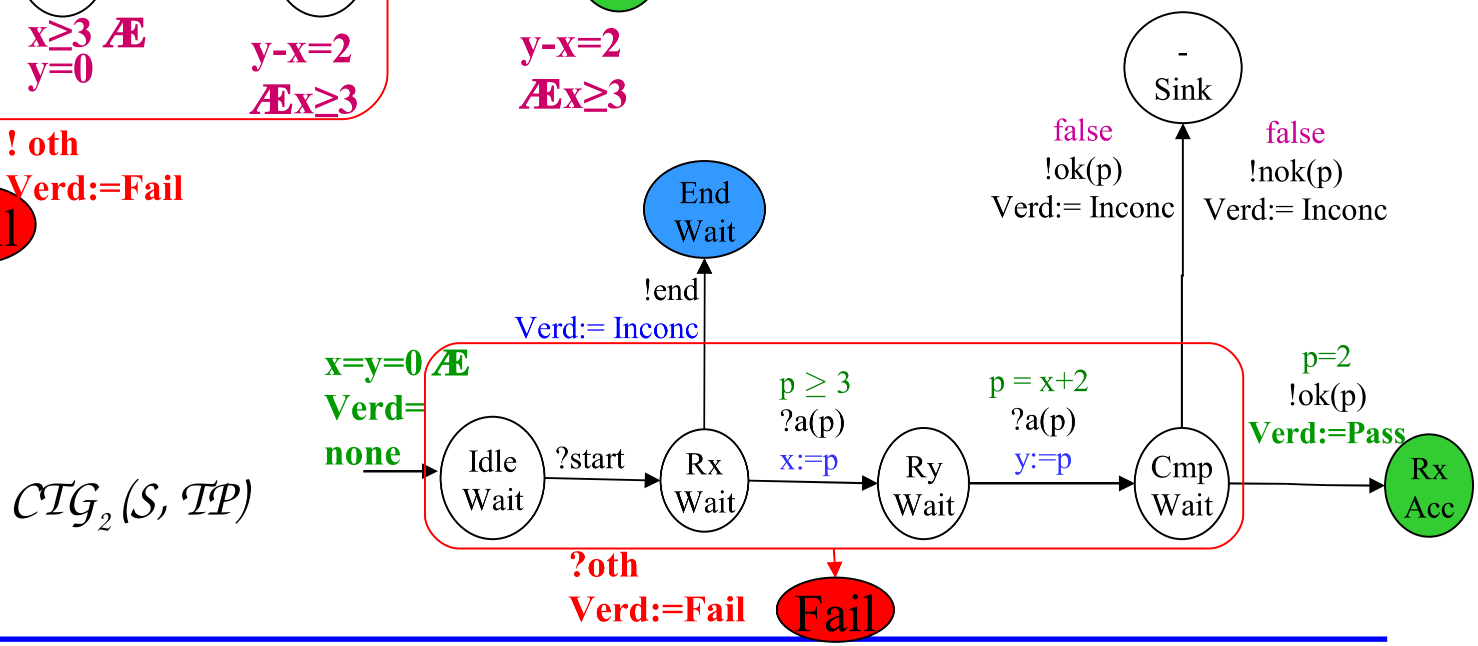


Simplification: over-approximation $reach^{\alpha}$ of $reach(\Theta)$



Simplify guards according to $reach^{\alpha}$ (false $\Rightarrow$ cut)

NB: semantics is unchanged



Consequences of over-approximation on test cases

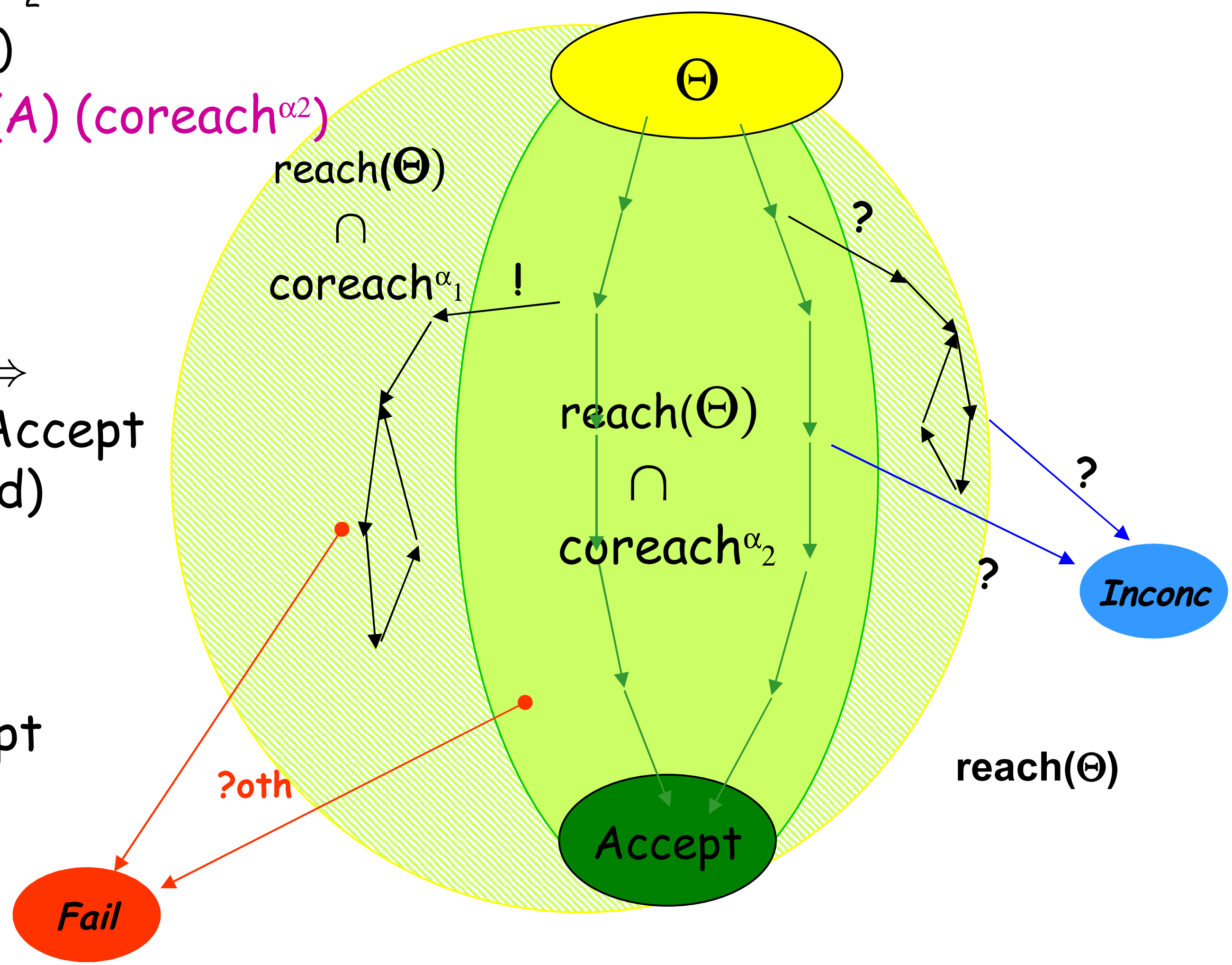
For two abstractions α_1 and α_2

(e.g. α_1: control vs α_2: polyhedra)

$pre^{\alpha 1}(A)\ (coreach^{\alpha 1}) \supseteq pre^{\alpha 2}(A)\ (coreach^{\alpha 2})$

$\Rightarrow$

$Tr(\mathcal{CTG}_1) \supseteq Tr(\mathcal{CTG}_2)$



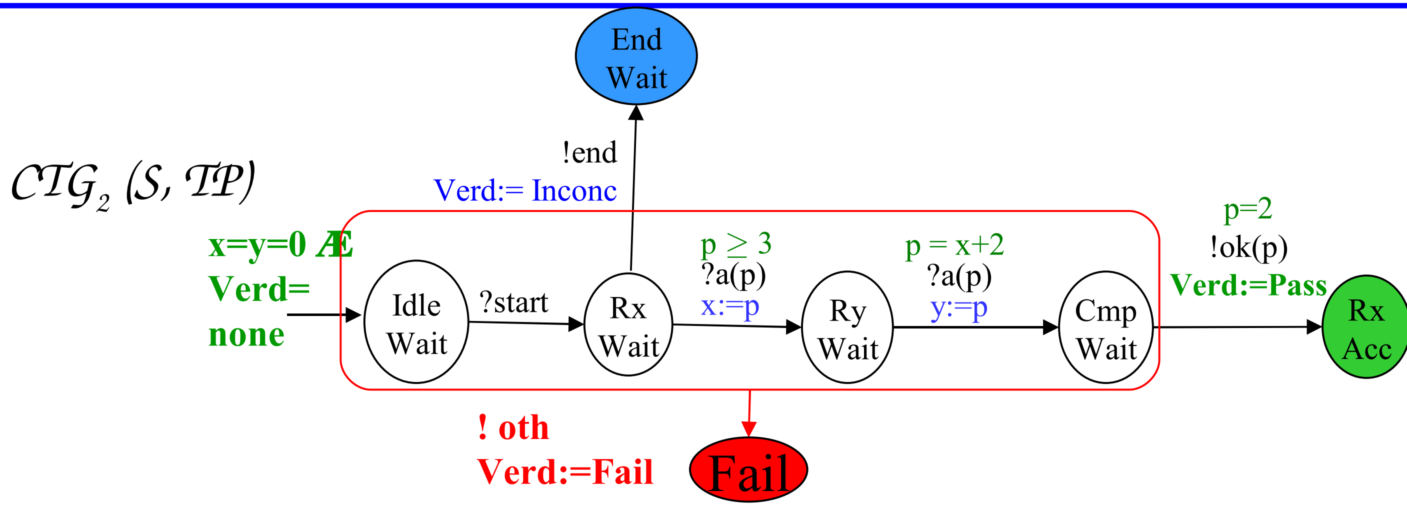
Less precise approximation $\Rightarrow$

- More infeasible traces to Accept
- More fail verdicts (all sound)

Limit cases:

- exact analysis:
  best guiding to Accept
- no analysis:
  no guiding to Accept

Test execution



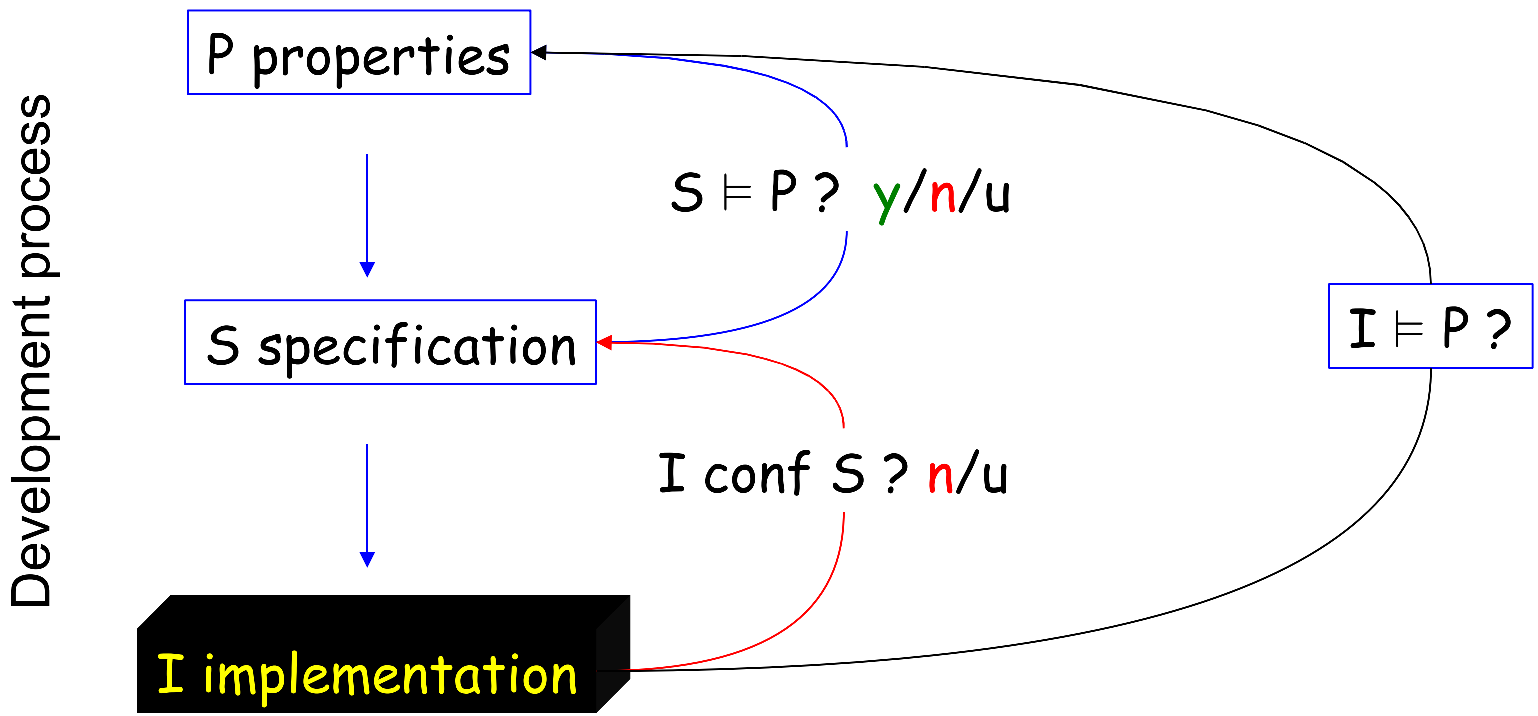
Inputs: [? a(p) : G(v,p); v:=A(v,p)] : v is known,
choose π s.t. G(v,π),by constraint solving
send a(π), assign v:=A(v, π)

Outputs: [! a(p) : G(v,p); v:=A(v,p)]: v is known,
receive a(π)
evaluate G(v,π)
if true, assign v:= A(v,π) (input complete)

?start . ?a(4) . ?a(6) . !ok(2) : Pass
?start . !end : Inconc

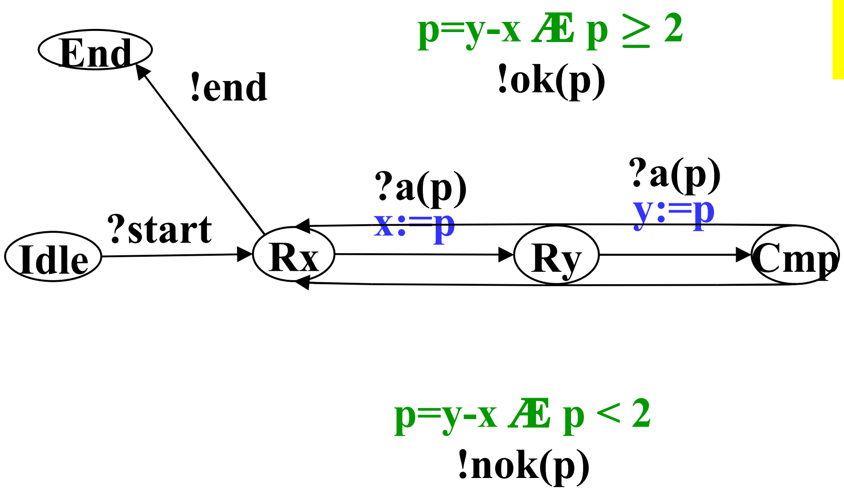
?start . ?a(5) . ?a(7) . !ok(3) : Fail
?start . ?a(6) . ?a(8) . !nok(2) : Fail

Verification and Testing



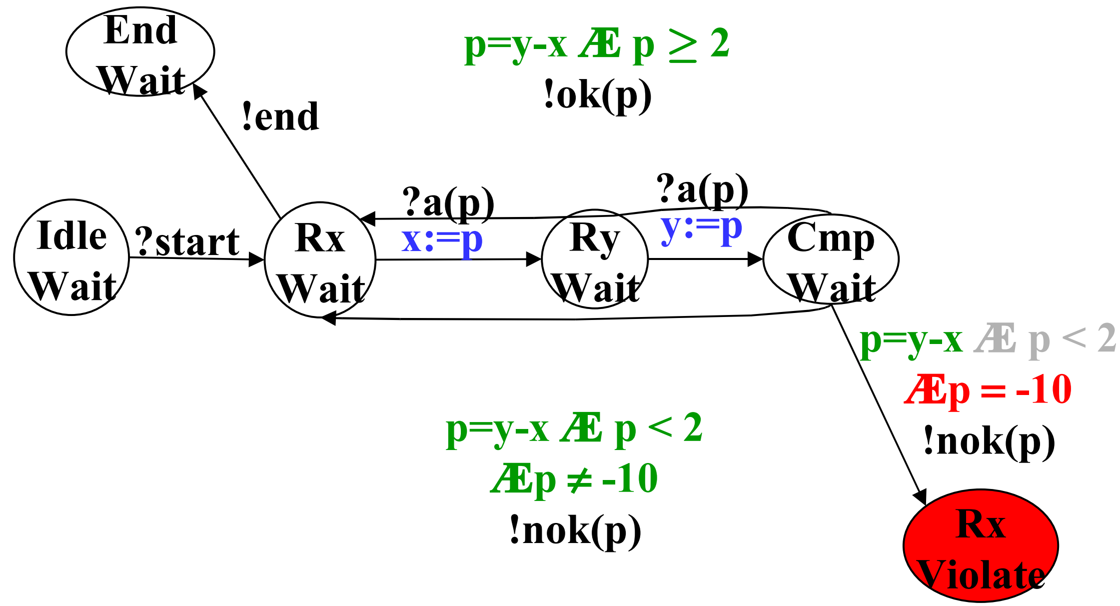
Model-checking a safety property

$\mathcal{S}$

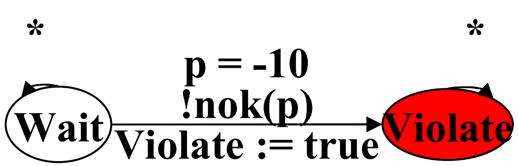


Model-checking $\mathcal{S} \models P$? reduces to

reachability in $\mathcal{S} \times \mathcal{A}_{\neg P}$ (undecidable)

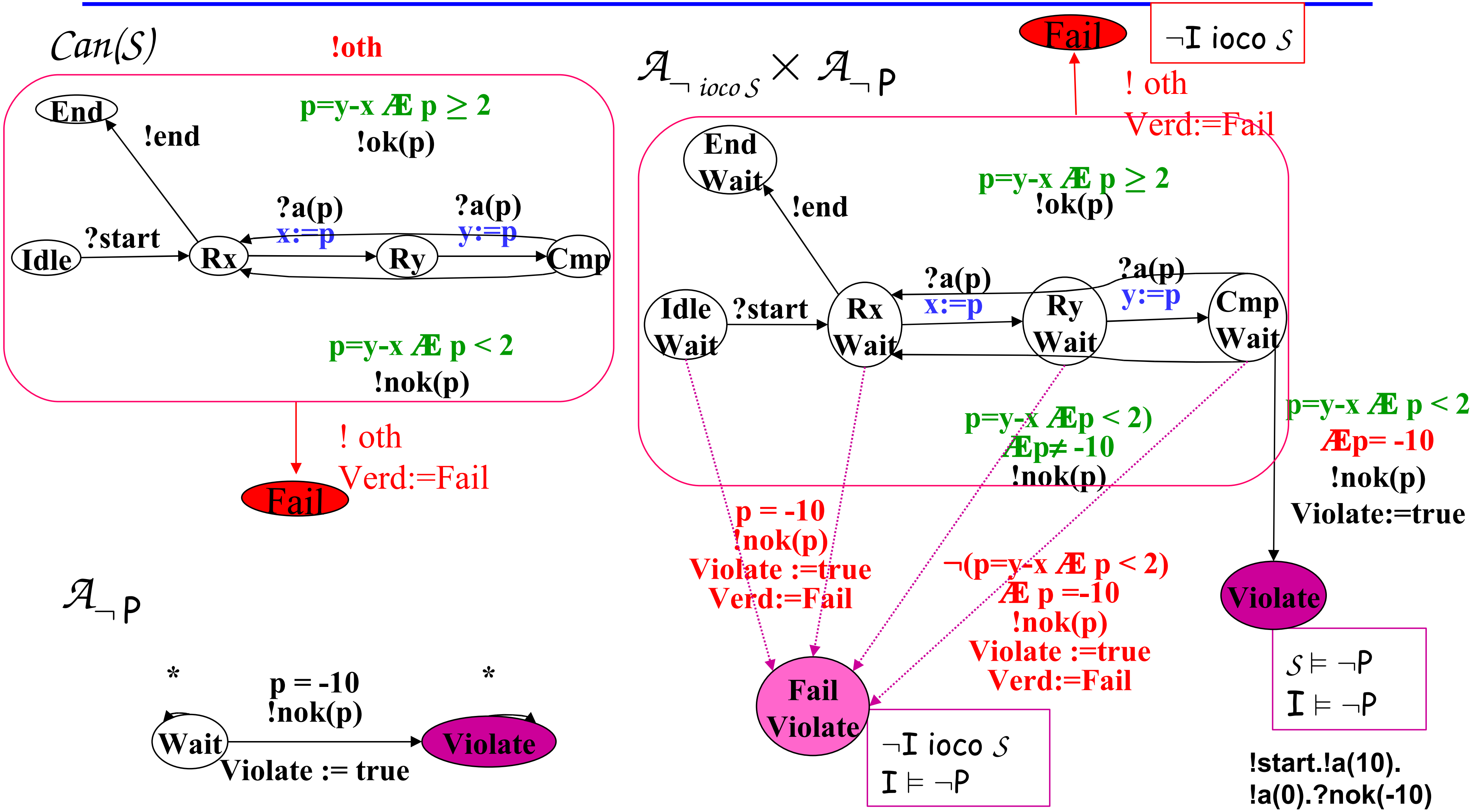


$\mathcal{TP}^{-} = \mathcal{A}_{\neg P}$



$S \models \neg P$:?start.?a(10).?a(0).!nok(-10) $\rightarrow$ (Rx, Violate)
With any abstraction $S^{\alpha} \models \neg P$? is Yes
but $S^{\alpha} \models \neg P \not\Rightarrow S \models \neg P$
Result of $S \models P$? could be Unknown

Test selection from a safety observer



Some links between Model-checking and Conformance Testing

- Test selection using model-checking :
  - S deter., controllable, P reachability: TC $\simeq$ counter-exple of $S \models \neg P$

    [Engels et al. 97, Gargantini et al.99]
  - Extension to coverage using CTL [Hong et al.02] or observers [Blom et al.04]
  - Non-controllable case is more complex (this talk)

- Checking properties on the implementation
  - Black-box checking [Peled et al.] : learn I by experiment, model-check $I \models P$

Conclusion

Simplified and general framework for Ioco-based Test selection

- For finite ioLTS and infinite *ioSTS*
- Unified For Reachability and Safety Observers
- Using verification: coreachability analysis, over-approximations
- Completing verification (case of safety)

More research work needed for, e.g.

- Theories and algorithms for other models of reactive systems e.g. with time, data, stack, probabilities …and combinations
- Coverage : measures, selection
- Links with structural testing techniques
- ….