Modeling and Verification: Runtime Monitoring and Recovery of Web Service Conversations

Marsha Chechik
Department of Computer Science
University of Toronto

CMU – April 2010

Quality software

- Dependable software:
  - that can justifiably be depended upon, in safety- and mission-critical settings
  - main concern: prevent catastrophes

BUT… I will not write software for trains and nuclear power plants!
What is in it for me? “

What Software Engineers Need Are …

- Tools that support effective analysis while remaining easy to use

- And at the same time, are
  - … fully automatic
  - … (reasonably) easy to use
  - … provide (measurable) guarantees
  - … come with guidelines and methodologies to apply effectively
  - … apply to real software systems

A simple research map

Multi-Valued logics + Model Checking

Reasoning with partial and inconsistent information

Abstraction

General study of models for representing abstractions

Software Model Checking

Checking behavioral properties of programs

Vacuity Detection

How to trust automated analysis

Temporal Logic Query Checking

Computer-aided model exploration

Understanding Counterexamples

Understanding and exploring results of automated analysis

Model Management

Synthesis, merge, analysis of structural and behavioral models

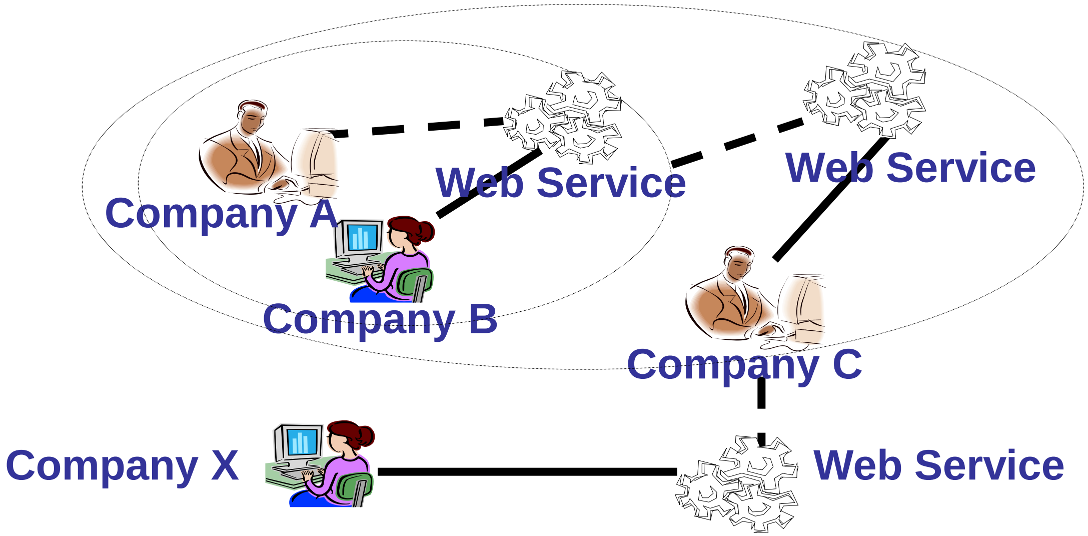
Domain-specificity: Web services

Runtime monitoring and recovery of web service conversations

Domain-specificity: automotive

Dealing with systems of models

Web Services



highly distributed

A software system designed to support interoperable machine-to-machine interaction over a network. – W3C

- Loosely coupled, interaction through standardized interfaces
- Platform- and programming-language independent
- Communicating through XML messaging
- Together, form a Service-Oriented Architecture (SOA)

Support for Quality Web Service Applications: Goals

- Enable automated verification during the development of business process composition
- Ensure reliability and interoperability of the workflow logic representing orchestration of web services
- Determine how to specify behaviors and check if system is consistent with this intended behavior
- Help debug web service-based business processes to determine errors and weaknesses

Challenges of reasoning about web services

- Web services are:
  - Distributed (use different “partners”) + heavy reliance on communication, via “infinite” queues
  - Heterogeneous (written in different languages)
  - Can change at run-time
  - Often “run to completion” rather than having infinite behaviour
  - A service has access to its partners’ interfaces but not code
  - Partners can even be dynamically discovered
- Languages in the web world not very formal
  - … and allow a lot of poorly understood capability
- Notion of correctness?

What is in this talk?

- Choices for web service analysis
  - Static, dynamic
- BPEL – Business process integration language
- Monitoring of web services
  - Properties: safety and liveness
  - Monitoring automata
- Recovery
  - Formalizing BPEL+compensation as a state machine
  - Computation (and ranking) of recovery plans for safety and liveness properties
- Evaluation + some lessons learns
- The bigger picture

What is not in this talk?

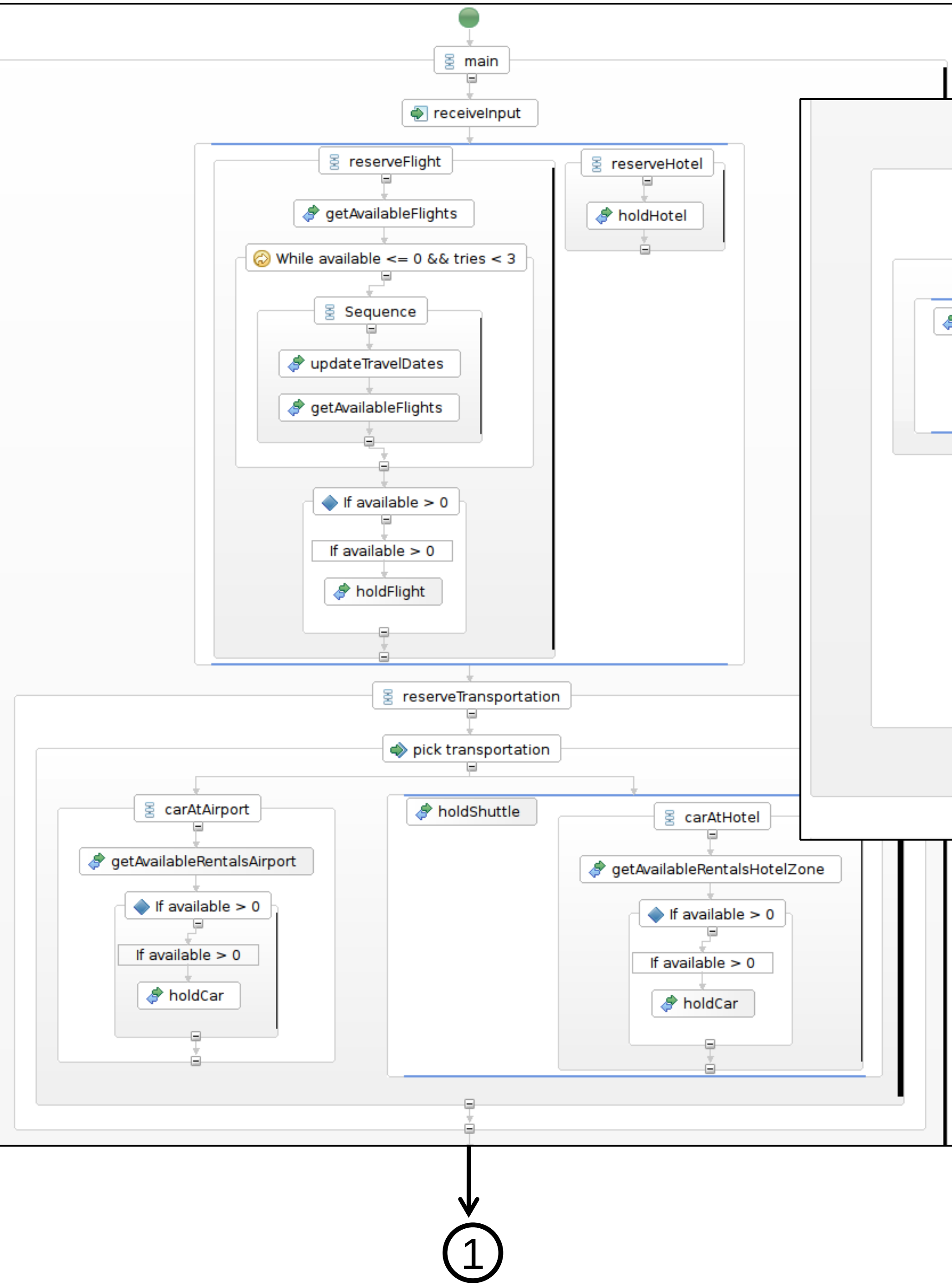
- Language and methodology for specifying properties
- Visualization and explanation of errors
- Helping user identify sources of errors

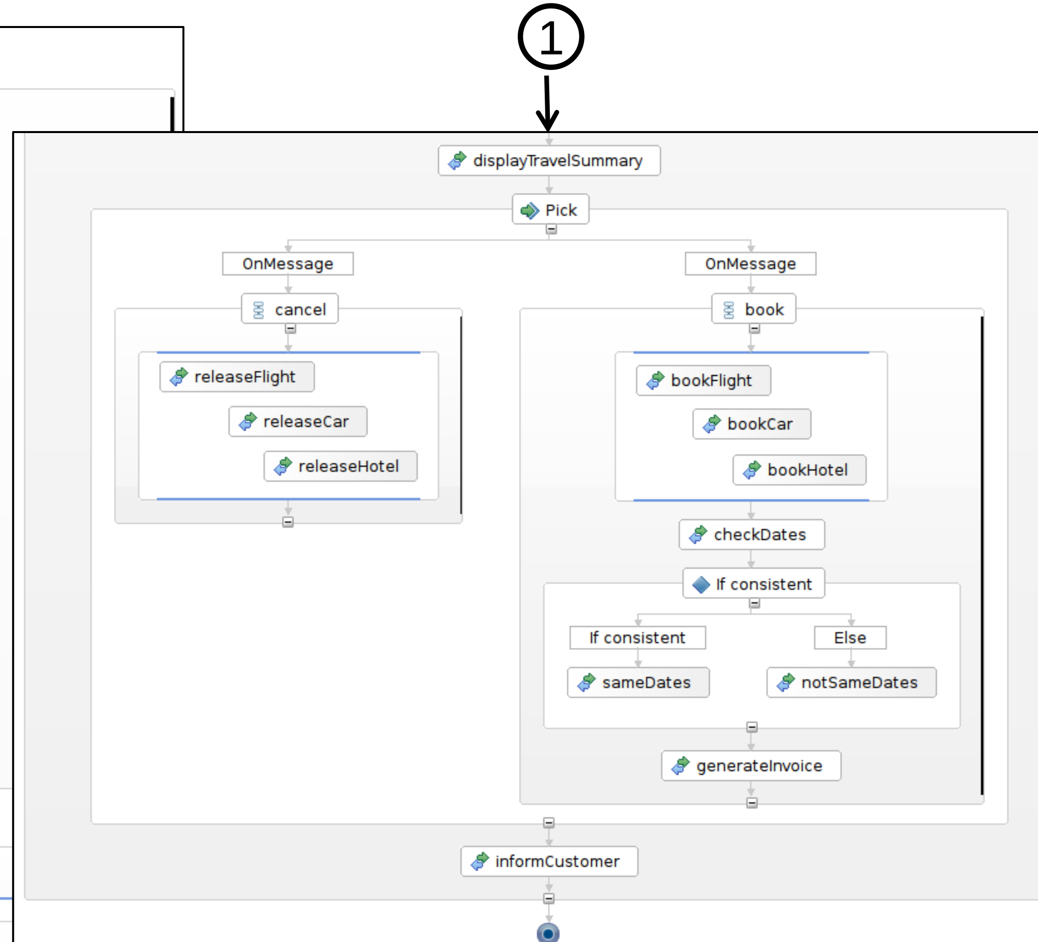
Business Process Execution Language

- BPEL: XML language for defining orchestrations
  - Variable assignment
  - Service invocation (“remote procedure call”)
  - Conditional activities (internal vs. external choice)
  - Sequential and parallel execution of services

Example: Travel Booking System (TBS)

- Customer enters travel request
  - dates, travel location and car rental location (airport or hotel)
- TBS generates proposed itinerary
  - flight, hotel room and car rental
  - also book shuttle to/from hotel if car rental location is hotel
  - no flights available – system prompts user for new travel dates
- Customer books or cancels the itinerary
- Main web service workflow implemented in BPEL





Travel Booking System

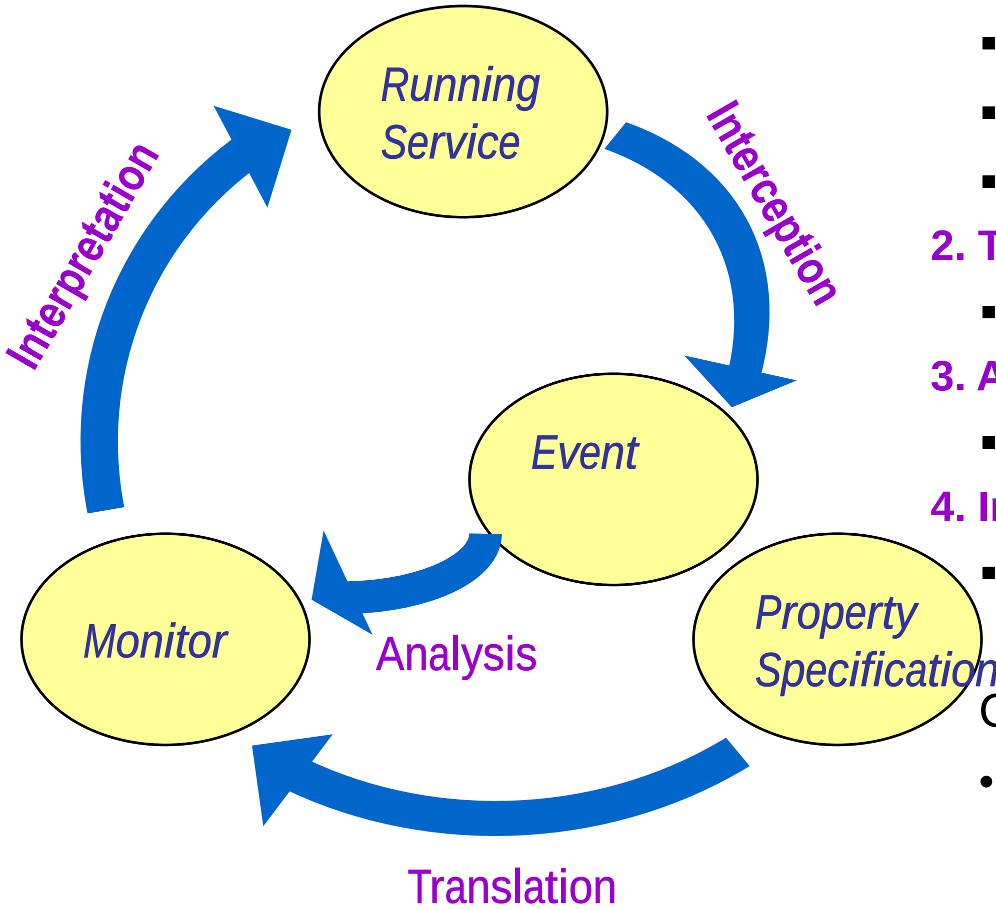
Analyzing Correctness of Web Service Compositions: Statically

- Compose individual web services
- Reason about correctness of the composition
- Problems
  - unbounded message queues
    - undecidable in general [Fu, Bultan, Su ‘04]
  - code may not be available
  - discovery and binding of services is usually dynamic

Analyzing Correctness of Web Service Compositions: Dynamically

- No code - observe finite executions at runtime
- Examine behavioral compatibility
- Pros
  - Can deal with dynamic binding
  - Can be applied to complex systems
- Specifically for Web Services:
  - Interaction is abstracted as a conversation between peers
  - Types of messages
    - method invocations
    - service requests/replies

Our Runtime Monitoring Approach



Implemented on top of IBM WebSphere Process Server

1. Property Specification:
- Sequence Diagrams
- Property Patterns
- Regular Expressions

2. Translation:
- User-specified props to FSAs

3. Analysis:
- Conformance Check

4. Interpretation:
- Visualization of deviations

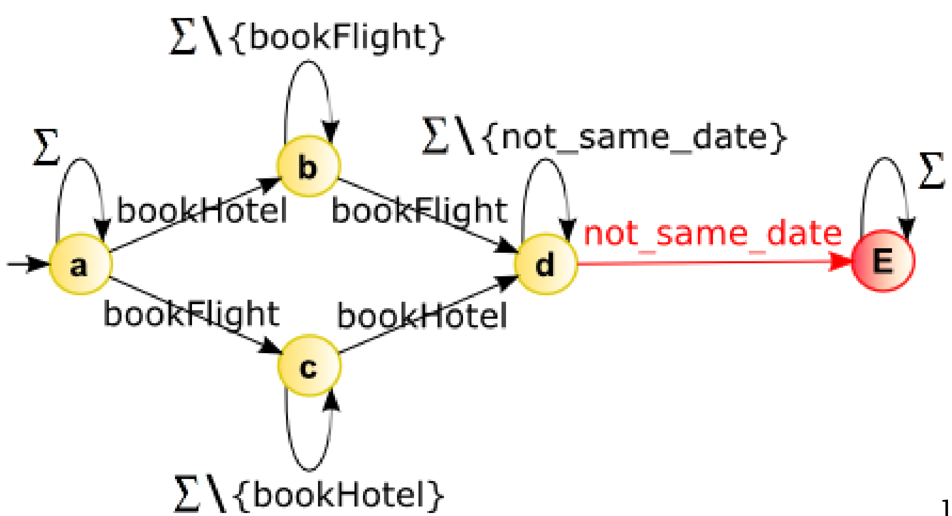
Overall
- non-intrusive framework (application is not aware it is being monitored)
- On-line (monitoring as software runs)

Monitoring Safety Properties

- Safety properties: negative scenarios that the system should not be able to execute.
- Monitorable because they are falsified by a finite prefix of execution trace.

Example:

- "Flight and hotel dates should match"
- **Absence** pattern combined with **After** scope
  - The hotel and flight dates should not be different after the hotel and flight have been booked
- Monitoring Automaton:



Monitoring Liveness Properties

- Liveness properties: positive scenarios that the system should be able to execute.

Example:

- “The car reservation request will eventually be fulfilled regardless of the location chosen”

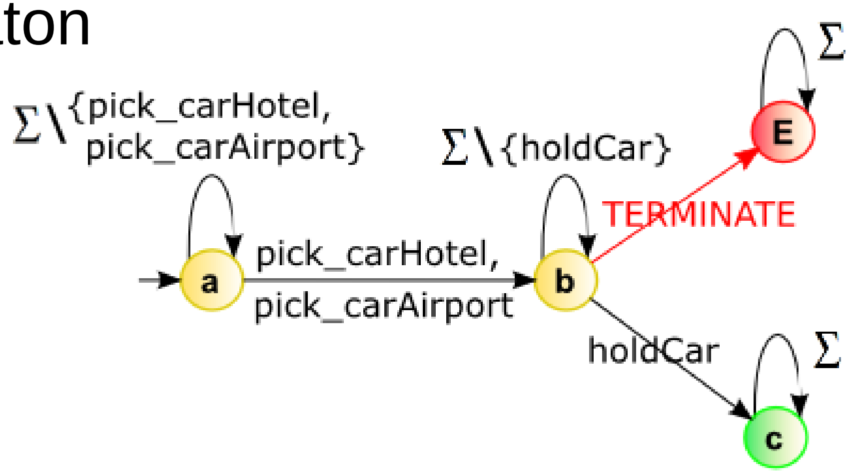
- Not monitorable on finite traces of reactive systems!
- Solution: Finitary Liveness
  - check liveness only for terminating web services
  - a finite trace satisfies a liveness property if it can completely exhibit the liveness behaviour before termination
  - express as a bounded liveness property

Monitoring Liveness Properties

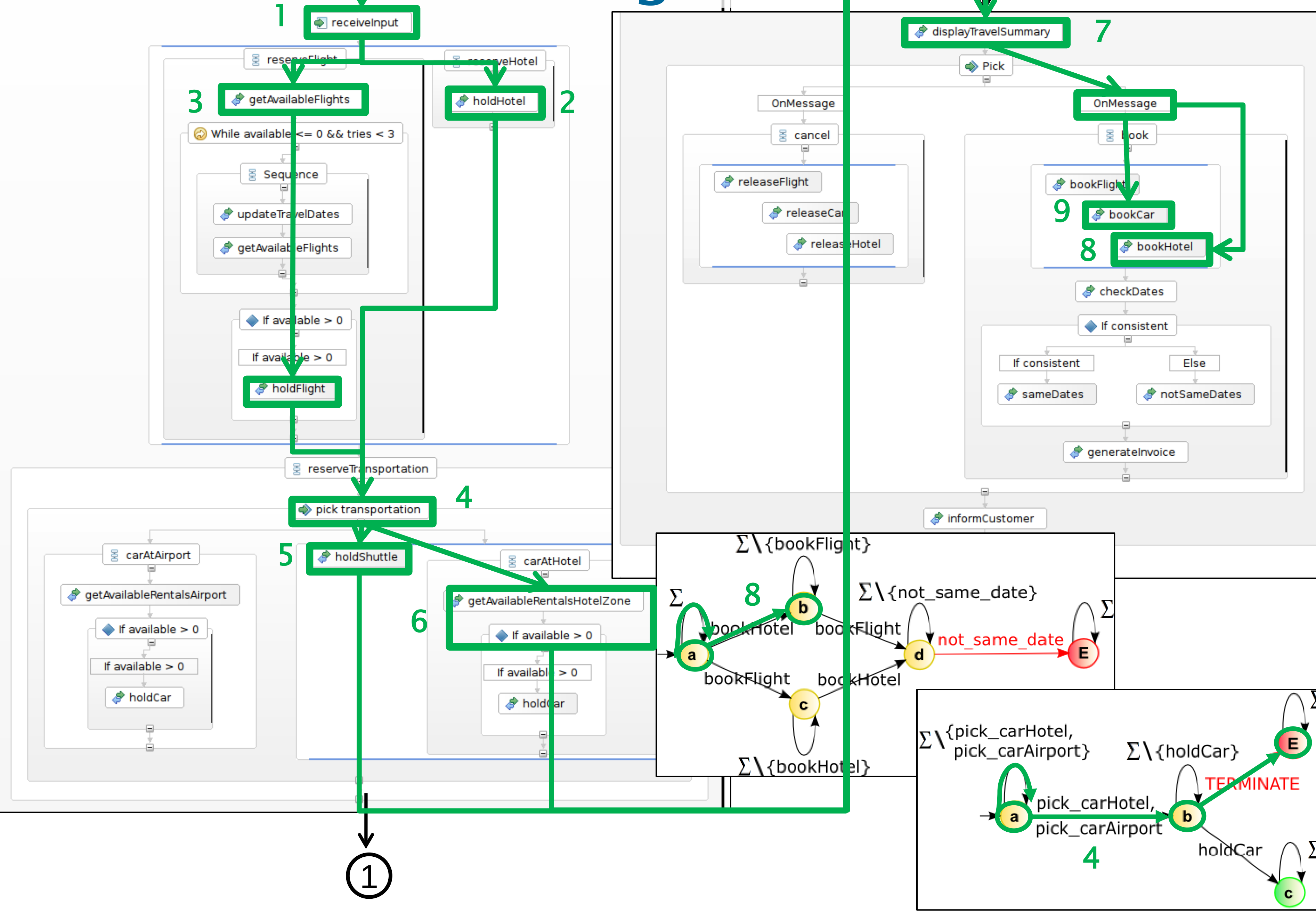
- Liveness properties: positive scenarios that the system should be able to execute.

Example:

- "The car reservation request will be fulfilled regardless of the location chosen"
- **Response** pattern with a **Global** scope
  - A car will be placed on hold, regardless of the rental location picked by the user
- Monitoring Automaton



Violating Scenario

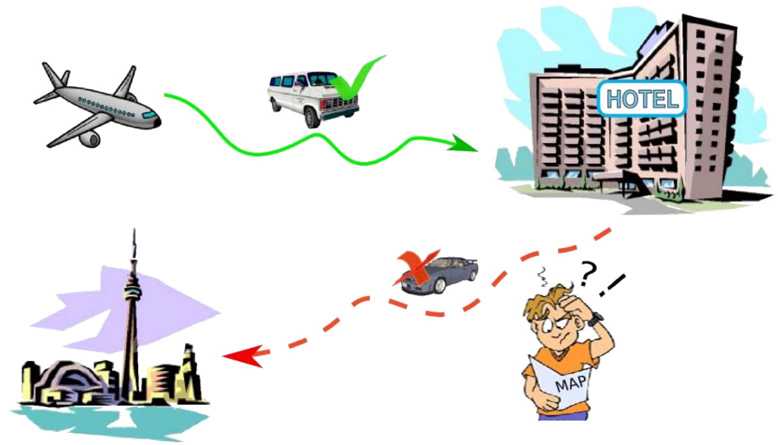


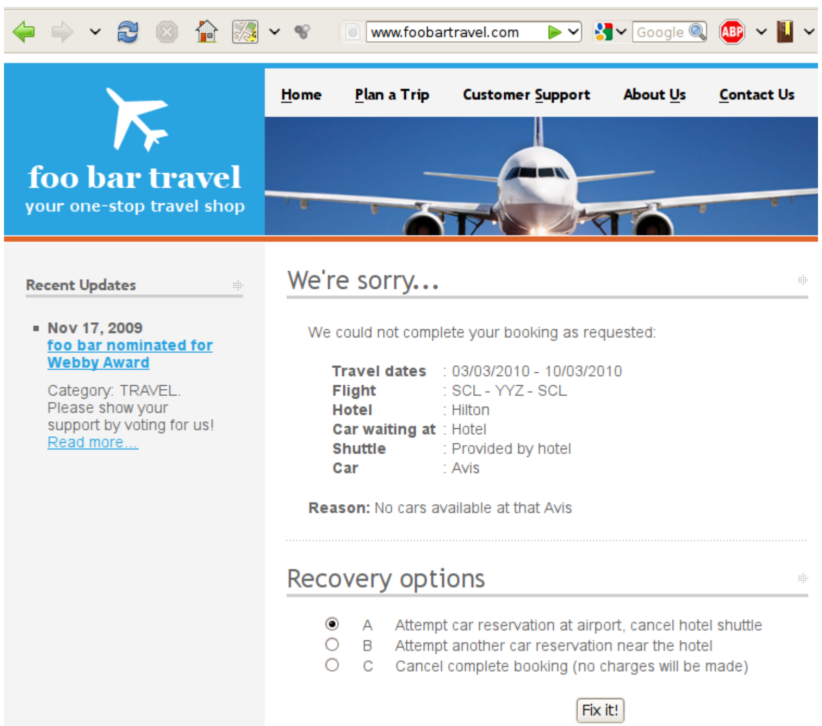
Goal of Recovery

- If a property fails, automatically generate a set of possible recovery plans
  - Exact number and length depend on user preferences
- User picks one
- Apply the plan, reset the monitors, continue

- Now, what is the meaning of recovery here?

Goal


HOTEL
MAP
?


www.foobartravel.com
Google
Home
Plan a Trip
Customer Support
About Us
Contact Us
foo bar travel
your one-stop travel shop
Recent Updates
Nov 17, 2009
foo bar nominated for Webby Award
Category: TRAVEL. Please show your support by voting for us!
Read more...
We're sorry...
We could not complete your booking as requested:
Travel dates : 03/03/2010 - 10/03/2010
Flight : SCL - YYZ - SCL
Hotel : Hilton
Car waiting at : Hotel
Shuttle : Provided by hotel
Car : Avis
Reason: No cars available at that Avis
Recovery options
A Attempt car reservation at airport, cancel hotel shuttle
B Attempt another car reservation near the hotel
C Cancel complete booking (no charges will be made)
Fix it!


HOTEL
Option A


HOTEL
Option B

Meaning of Recovery

- From violations of safety properties:
  - Observed an undesired behaviour
  - “Undo” enough of it so that an alternative behaviour can be taken …
  - … that would not longer be undesired
- From violations of liveness properties:
  - Observed an undesired behaviour
  - “Undo” enough of it so that al alternative behaviour can be taken
  - “Redo” the behaviour so that it becomes successful

- This is only possible if we can undo prev. executed steps – compensation!

Business Process Execution Language

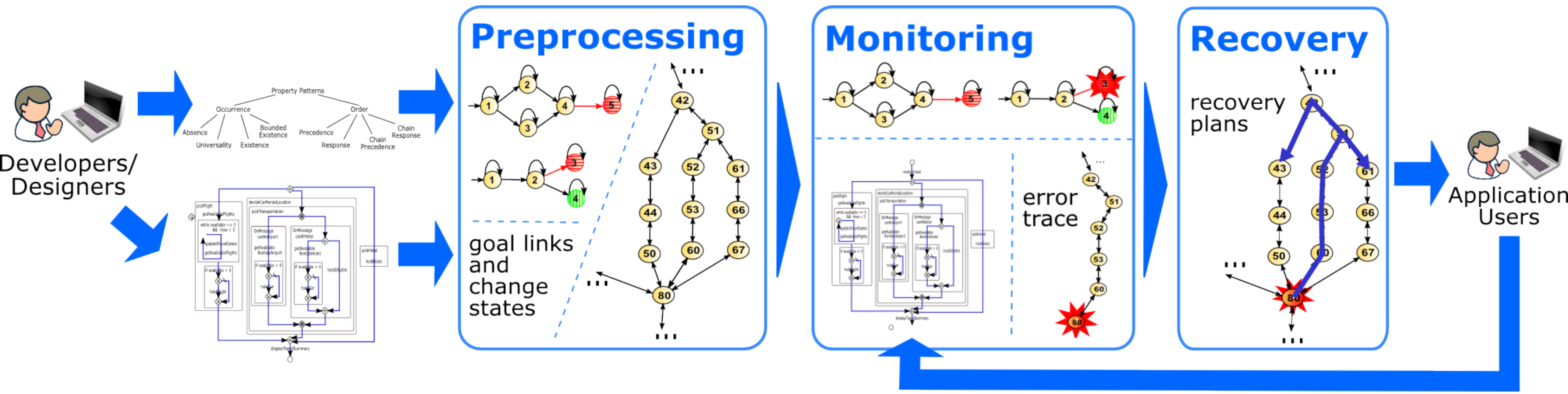
- BPEL: XML language for defining orchestrations
  - Variable assignment
  - Service invocation (“remote procedure call”)
  - Conditional activities (internal vs. external choice)
  - Sequential and parallel execution of services
- Compensation
  - Goal: to reverse effects of previously executed activities
  - Defined per activity and scope
  - Intended to be executed “backwards”:
    - `compensate(a;b) = compensate(b);compensate(a)`
  - Example:

```
<scope name="bf">
   <invoke ... operation = "bf" ... outputVariable = "flightConf" />
   <compensationHandler cost = "9">
       <invoke ... operation = "cancelF" inputVariable = "flightConf"/>
   </compensationHandler>
</scope>
```

What is in this talk?

- Choices for web service analysis
  - Static, dynamic
- BPEL – Business process integration language
- Monitoring of web services
  - Properties: safety and liveness
  - Monitoring automata
- Recovery
  - Formalizing BPEL+compensation as a state machine
  - Computation (and ranking) of recovery plans for safety and liveness properties
- Evaluation + some lessons learns
- The bigger picture

User-Guided Recovery



Preprocessing

- BPEL →LTSA translation:
  LTSA tool + new
- Property translation:
  new (incomplete)
- Goal links, change states:
  python-automata + new

Monitoring

- BPEL engine:
  WebSphere Process Server (WPS)
- Monitoring:
  WPS plugin

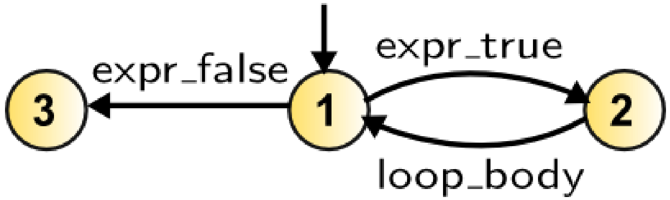
Recovery

- Planner:
  Blackbox
- Generation of multiple plans:
  new, based on SAT-solver
- Plan ranking + Post-Processor:
  new

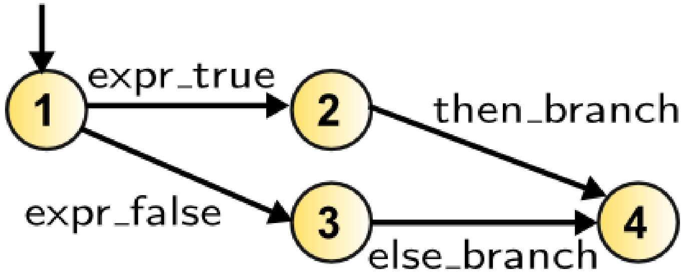
Formalizing WS-BPEL

- Operations formalized [Foster '06]:
  - receive, reply, invoke, sequence, flow, while, if, pick, assign, fault handling
- Modeling language: Labelled Trans. Systems (LTS)
- Tool support: LTSA

```
<while name="while">
  <condition>expr</condition>
  <scope name="loop_body">
    ...
  </scope>
</while>
```

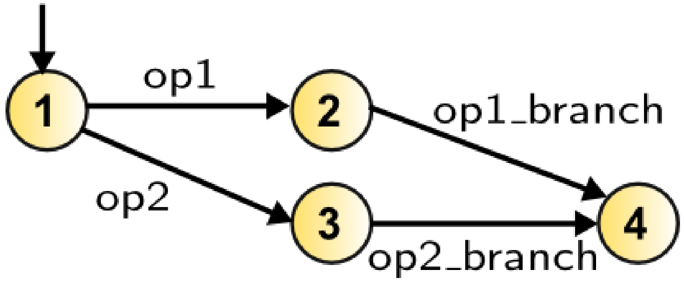



```
<if name="if">
  <condition>expr</condition>
  <scope name="then_branch">
    ...
  </scope>
  <else>
    <scope name="else_branch">
      ...
    </scope>
  </else>
</if>
```

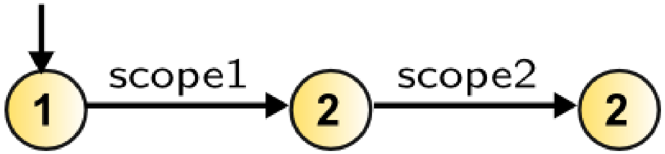


Formalizing WS-BPEL

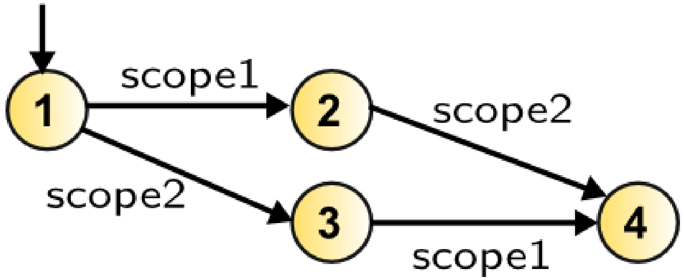
```
<pick name="pick">
  <onMessage operation="op1">
    <scope name="op1_branch">
      ...
    </scope>
  </onMessage>
  <onMessage operation="op2">
    <scope name="op2_branch">
      ...
    </scope>
  </onMessage>
</pick>
```




```
<sequence name="seq">
  <scope name="scope1">
    ...
  </scope>
  <scope name="scope2">
    ...
  </scope>
</sequence>
```

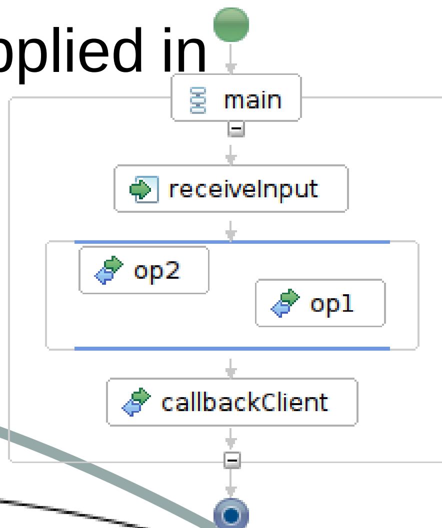


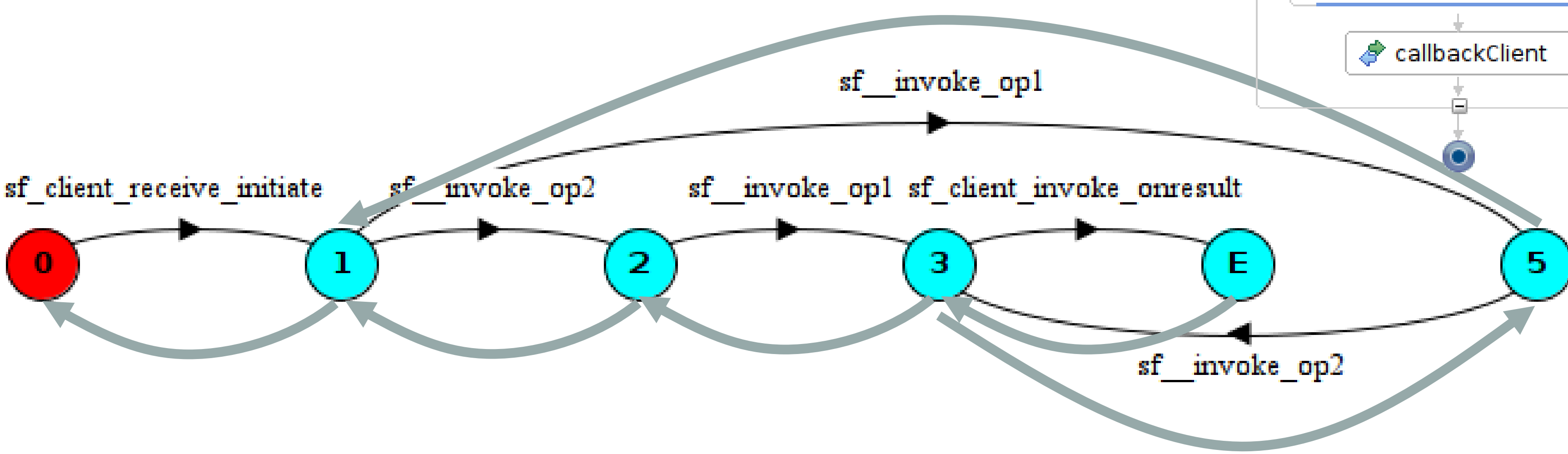

```
<flow name="flow">
  <scope name="scope1">
    ...
  </scope>
  <scope name="scope2">
    ...
  </scope>
</flow>
```



Formalizing BPEL+compensation

- Adding compensation for individual activities
  - Compensation available once activity has been completed successfully
  - Unless specified otherwise, compensation applied in inverse order of execution





Recovery for safety properties

Trace:

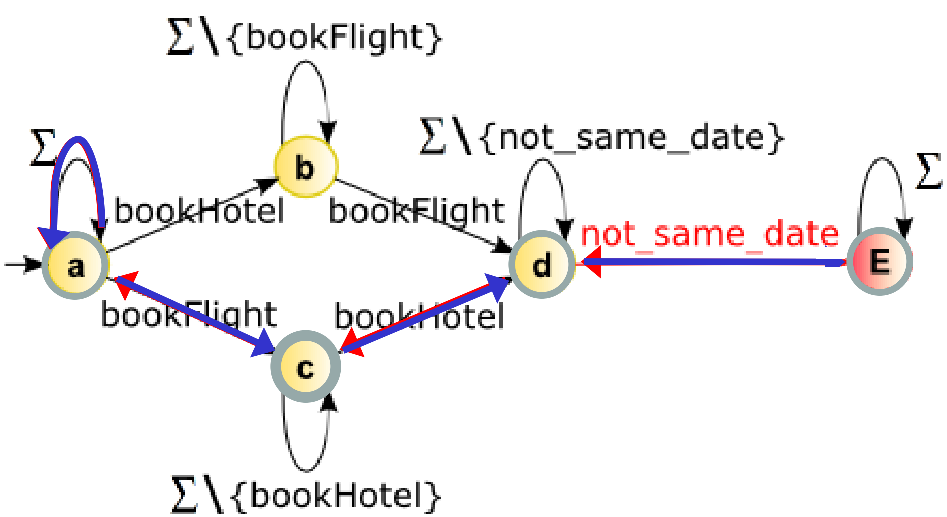
1. Receive input
2. Get car at airport
3. Hold car at airport
4. Hold hotel room
5. Update travel dates and hold flight
6. Display itinerary
7. Book flight
8. Book hotel
9. **Check date consistency**

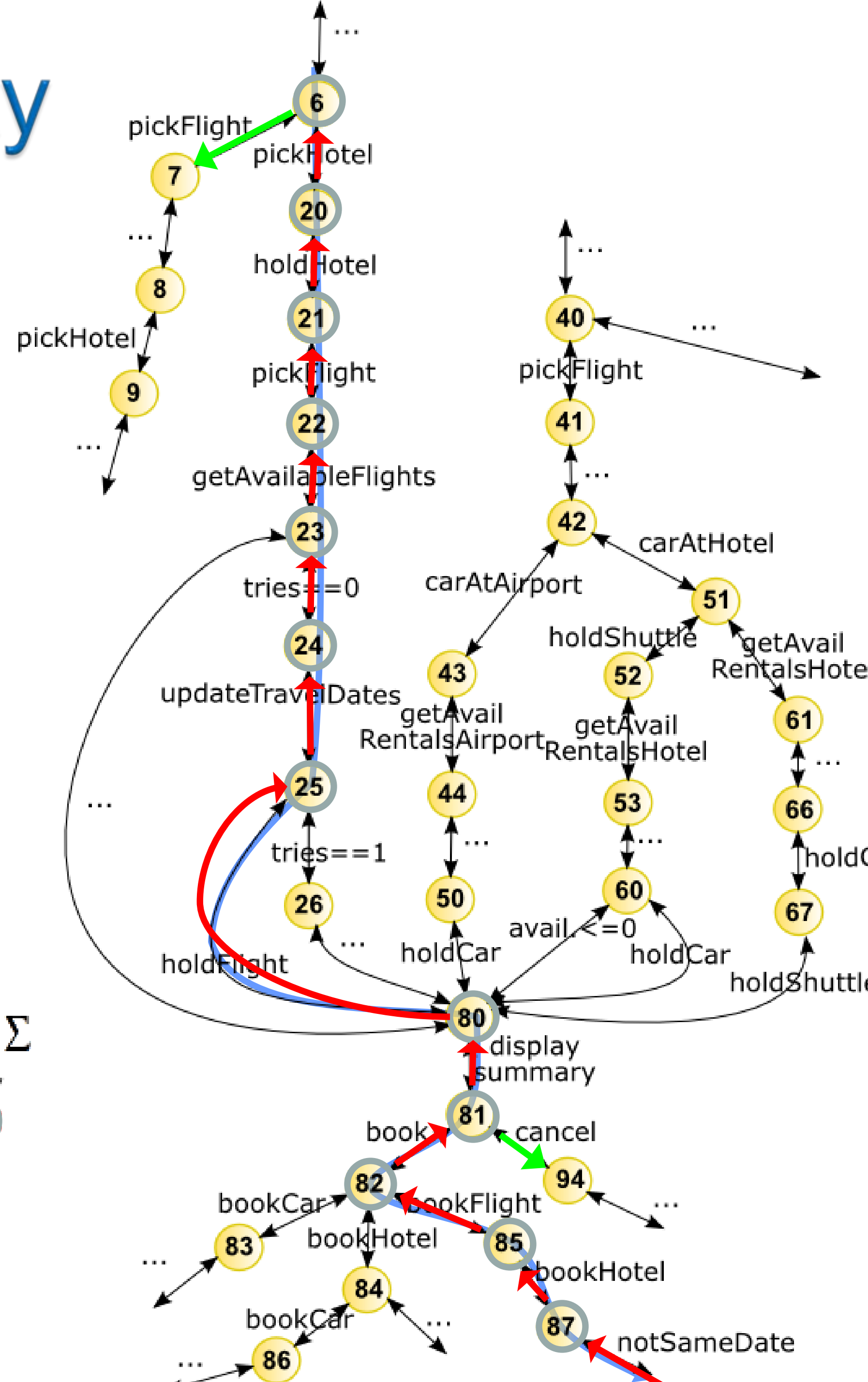
Monitor no longer in error state, but only available event leads to error state

81 – monitor not in error state: option: cancel everything

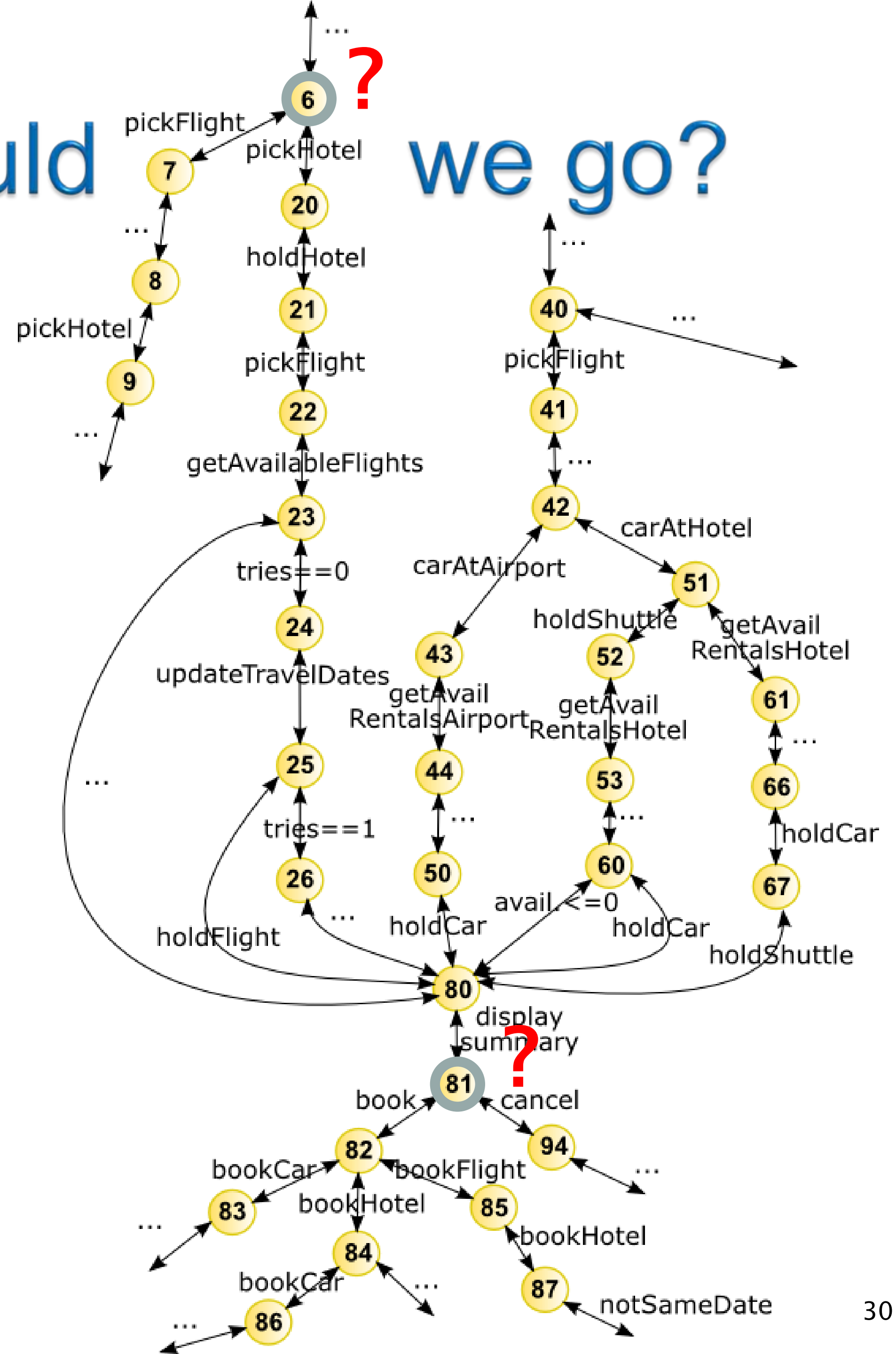
Other option: continue compensation

How far?





How far back should we go?

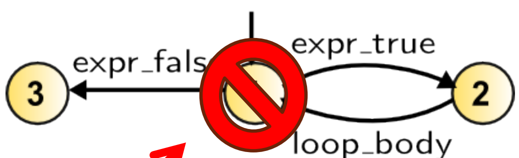


- Goal: it should be possible for the system to avoid executing same error trace!
- Thus: undo error trace till we reach a state from which we can execute an alternative path
- We call these change states

Change States

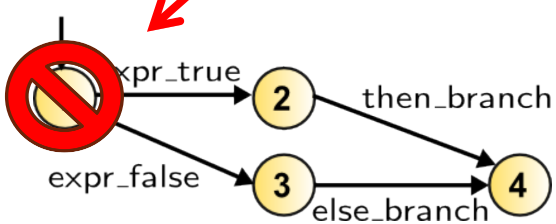
- Definition: a *change state* is a state that can potentially produce a branch in the control flow of the application
- Branching BPEL activities:

while

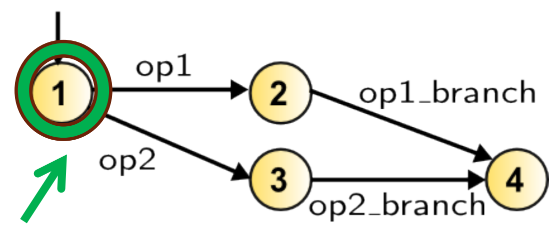


Internal choice, depends on state!

if

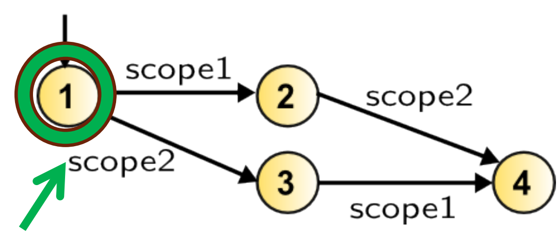


pick



External choice

flow



Alternative execution order

Change States

- How can we affect an internal choice?
  - Idempotent service calls: outcome completely determined by input parameters
    - So executing it twice does not change the outcome
  - Non-idempotent service calls:
    - Executing twice may give a different result
  - Overapproximation: non-idempotent service calls can affect internal choices…
    - … but do not have to!
- So: what are change states?
  - Non-idempotent service calls (user identified), pick and flow activities

Recovery for live-ness properties

Trace:

1. Receive input
2. Hold hotel room
3. Hold flight (no date update)
4. Get car at hotel
5. Hold shuttle
6. No cars available at hotel
7. Display itinerary
8. Book hotel
9. Book car > **TERMINATE**

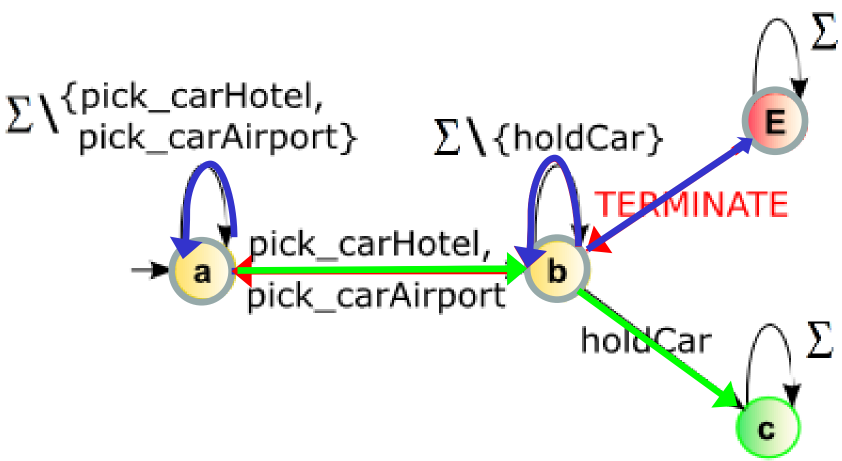
Intercept TERMINATE event

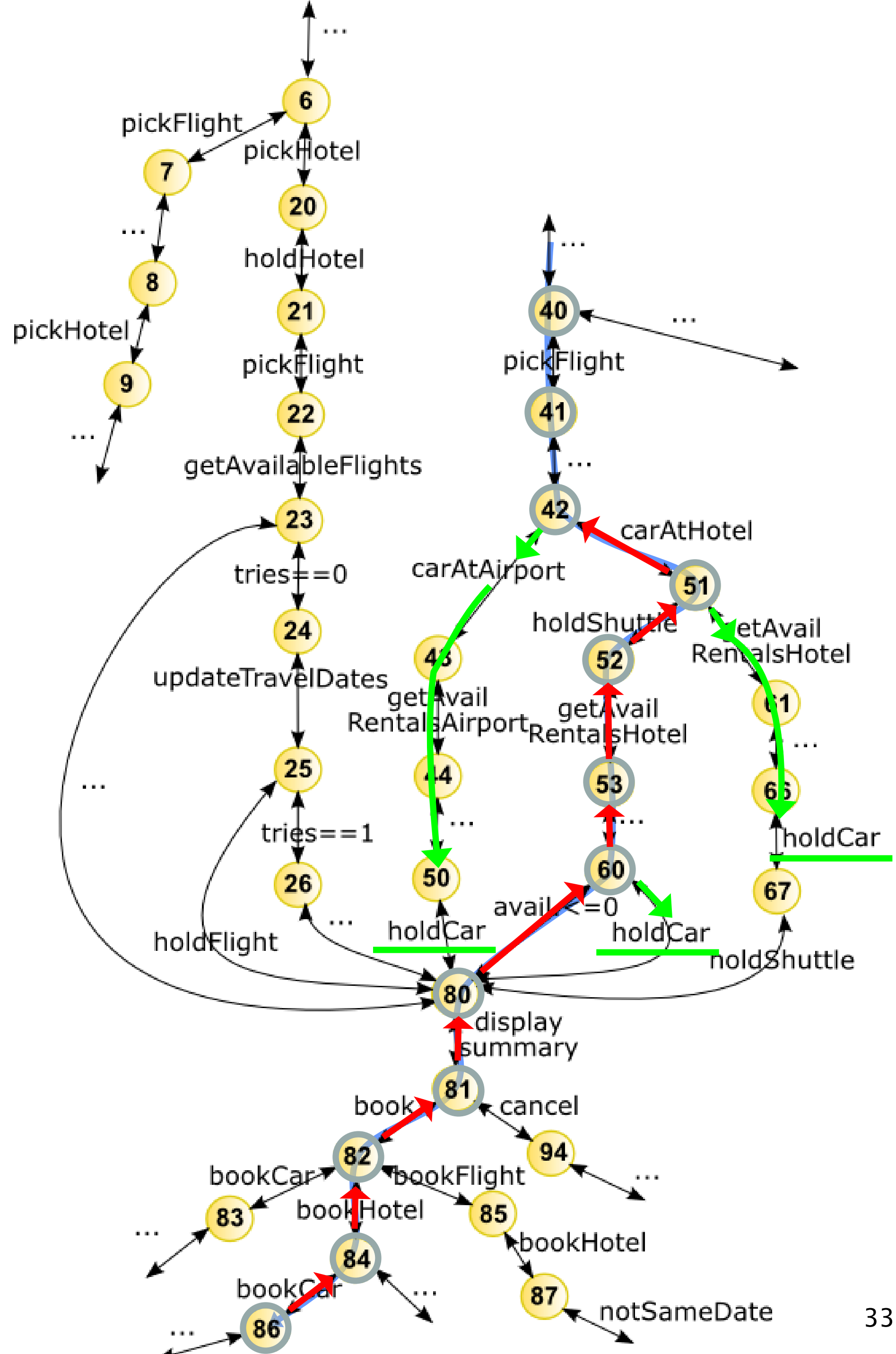
Goal: reach green monitor state

60 – try to get car at hotel again

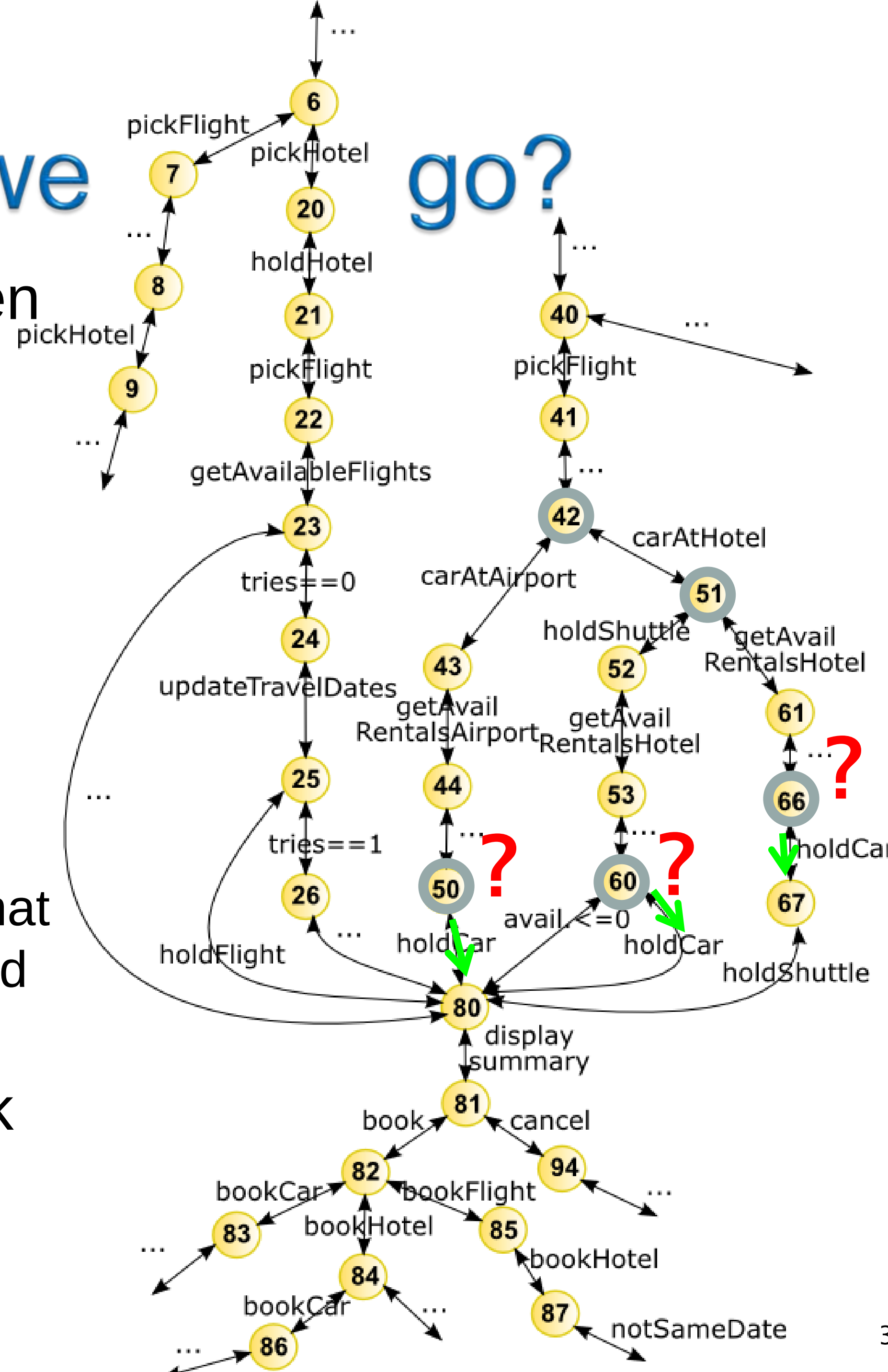
51 – same, new shuttle reservation

42 – try to get car at airport



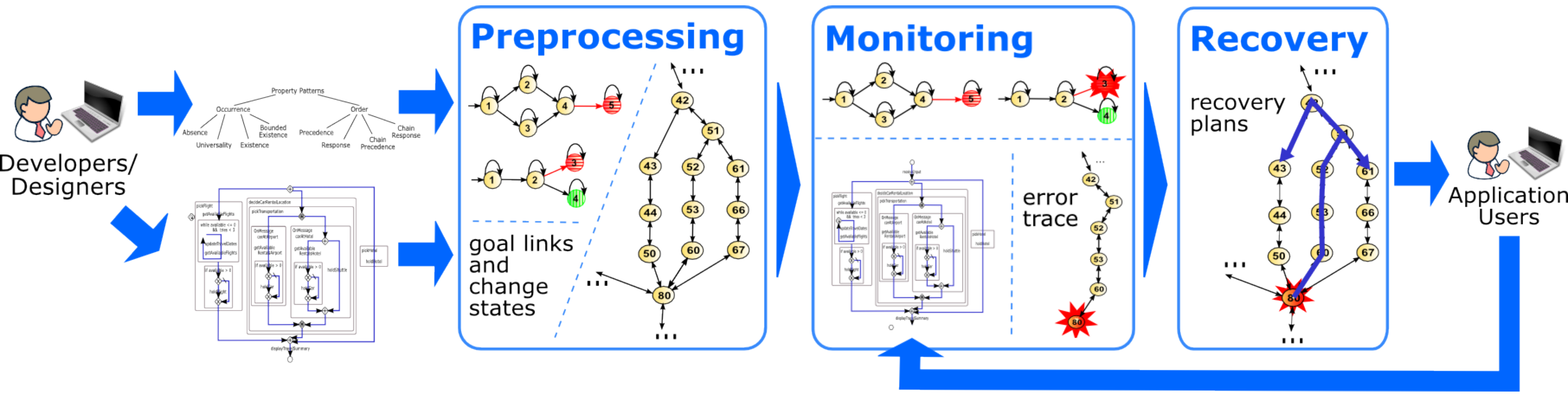


Where should we go?



- Get the monitor into a green state (complete desired behaviour)
- Compute cross-product between application and mixed monitor
- Goal links: cross-product transitions (s, q) $\xrightarrow{a}$ (s', q')
  - (s , q) $\xrightarrow{a}$ (s' , q') means that we have witnessed the desired behaviour
- Moreover, reach a goal link via a change state
  - … to ensure a different execution path

User-Guided Recovery



Preprocessing

- BPEL →LTSA translation:
  LTSA tool + new
- Property translation:
  new (incomplete)
- Goal links, change states:
  python-automata + new

Monitoring

- BPEL engine:
  WebSphere Process
  Server (WPS)
- Monitoring:
  WPS plugin

Recovery

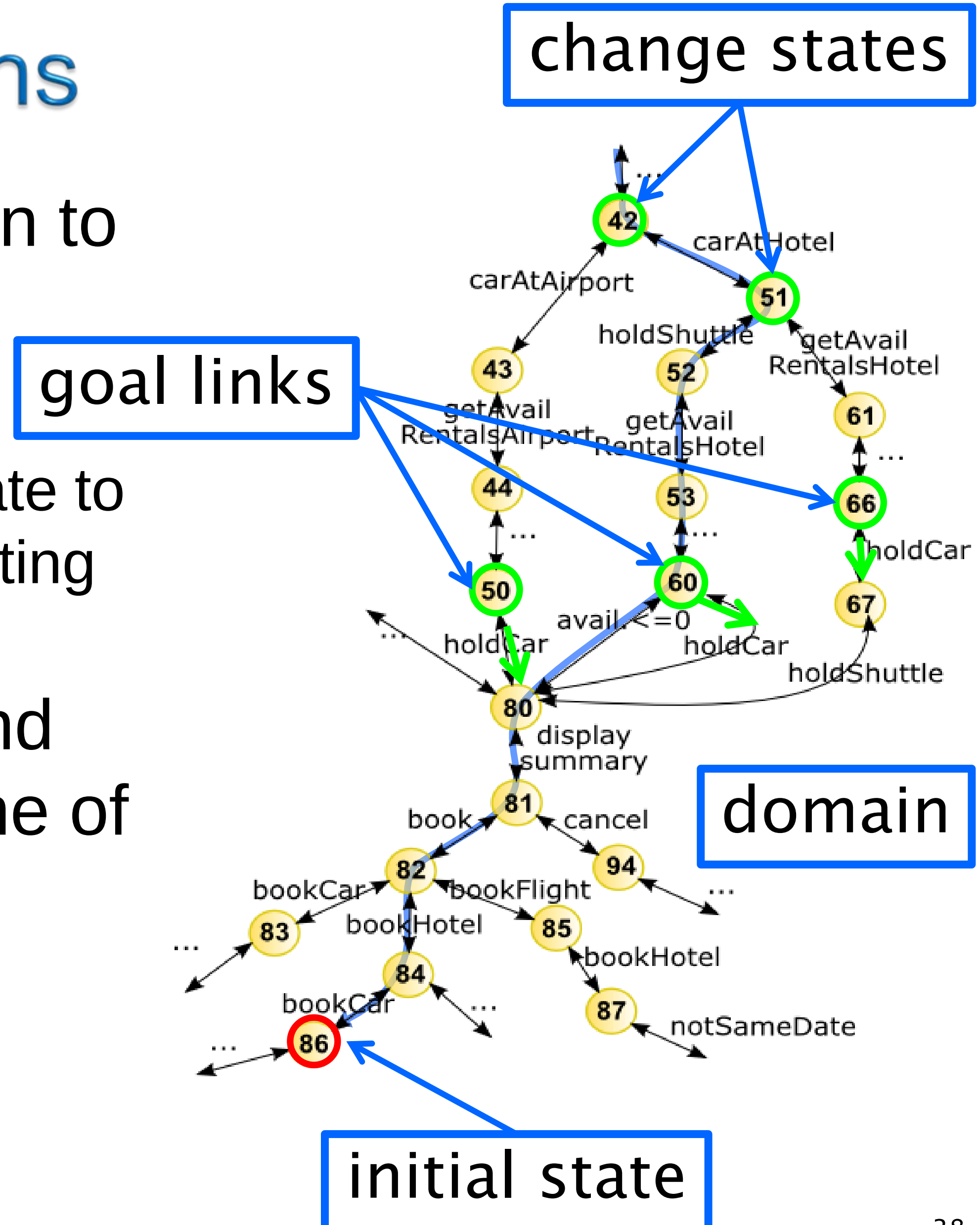
- Planner:
  Blackbox
- Generation of multiple plans:
  new, based on SAT-solver
- Plan ranking +
Post-Processor:
  new

Recovery for web services - outline

- Input:
  - Properties
  - BPEL with recovery mechanism
  - Mechanism for recovery
- Preprocessing
  - Properties -> monitors
  - BPEL -> LTS
  - Computation of goal links, change states
- Recovery
  - Recovery for safety properties
  - Recovery for liveness properties
    - Generating a single plan
    - Generating multiple plans
  - Ranking, displaying, executing plans
- Evaluation
- Related work, conclusion and future work

Generating Plans for Liveness Property Violations

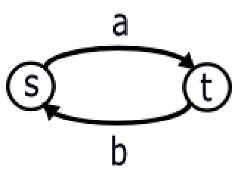
- Convert LTS + violation to a planning problem
- Goal links:
  - go through a change state to better chances of executing an alternative path
- Planner attempts to find the shortest path to one of the goal links

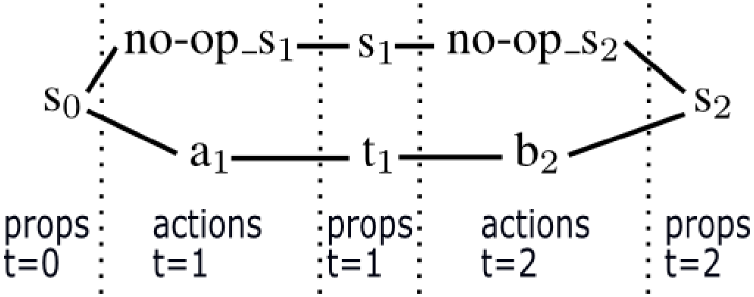


SAT Encoding of Planning Problem

Planning (PSPACE-complete)

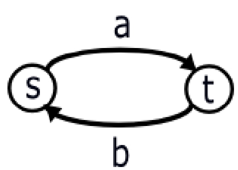
- Planning Graphs [Blum and Furst ‘95]
  - Avoid straightforward exploration of the state space graph
  - Nodes: actions and propositions (arranged into alternate levels)
  - Edges:
    - from a proposition to the actions for which it is a precondition
    - from an action to the propositions it makes true or false

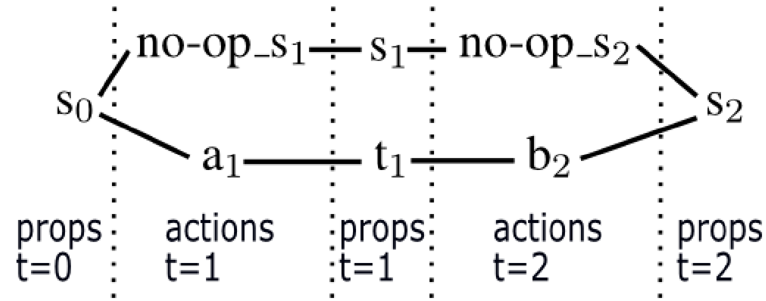




SAT Encoding of Planning Problem

SAT-based planners translate planning graph into CNF





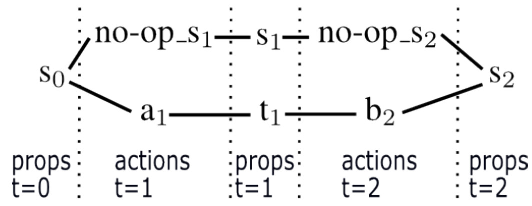
props t=0 → actions t=1	$(\neg\text{no-op\_}s_1 \vee s_0) \wedge (\neg a_1 \vee s_0)$
actions t=1 → props t=1	$\wedge\, (\neg s_1 \vee \text{no-op\_}s_1) \wedge (\neg t_1 \vee a_1)$
etc.	$\wedge\, (\neg b_2 \vee t_1) \wedge (\neg\text{no-op\_}s_2 \vee s_1)$
	$\wedge\, (\neg\text{no-op\_}s_2 \vee s_2) \wedge (\neg b_2 \vee s_2)$
initial state	$\wedge\, (s_0)$
goal state	$\wedge\, (s_2)$

Generating Multiple Plans

- Given a plan to a goal state *g*,
  - Remove *g* from the set of goal states
  - Rerun the planner
- What about other plans to *g*?

Opening the Planner Black Box

Planning domain



Planner

SAT instance

$$(\neg \text{no-op\_}s_1 \vee s_0) \wedge (\neg a_1 \vee s_0) \wedge (\neg \text{no-op\_}s_2 \vee s_1)$$
$$\wedge(\neg b_2 \vee t_1) \wedge (\neg s_1 \vee \text{no-op\_}s_1) \wedge (\neg t_1 \vee a_1)$$
$$\wedge(\neg \text{no-op\_}s_2 \vee s_2) \wedge (\neg b_2 \vee s_2) \wedge (s_0) \wedge (s_2)$$

SAT solver

Satisfying assignment

$$s_0, \neg \text{no-op\_}s_1, \boxed{a_1}, \neg s_1, t_1,$$
$$\neg \text{no-op\_}s_2, \boxed{b_2}, s_2$$

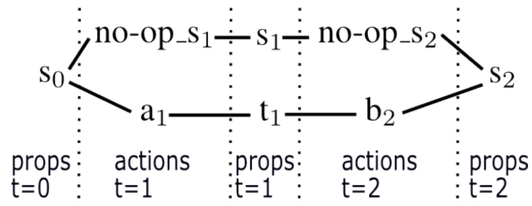
Converter

Plan: (a; b)

Planner used: Blackbox

Generating Multiple Plans

Planning domain



Controlling Max Length of Plans?

Previous plans

Max length k

Planner: expand domain up to k steps

¬prev plan ∧ SAT instance

$\vee s_0) \wedge (\neg a_1 \vee s_0) \wedge (\neg \text{no-op\_}s_2 \vee s_1)$
$\wedge(\neg b_2 \vee t_1) \wedge (\neg s_1 \vee \text{no-op\_}s_1) \wedge (\neg t_1 \vee a_1)$
$\wedge(\neg \text{no-op\_}s_2 \vee s_2) \wedge (\neg b_2 \vee s_2) \wedge (s_0) \wedge (s_2)$

$\bigwedge \neg(a_1 \wedge b_2)$

SAT solver

Satisfying assignment

$s_0, \boxed{\text{no-op\_}s_1}, \neg a_1, s_1, \neg t_1,$
$\boxed{\text{no-op\_}s_2}, \neg b_2, s_2$

Incremental SAT solver

Converter

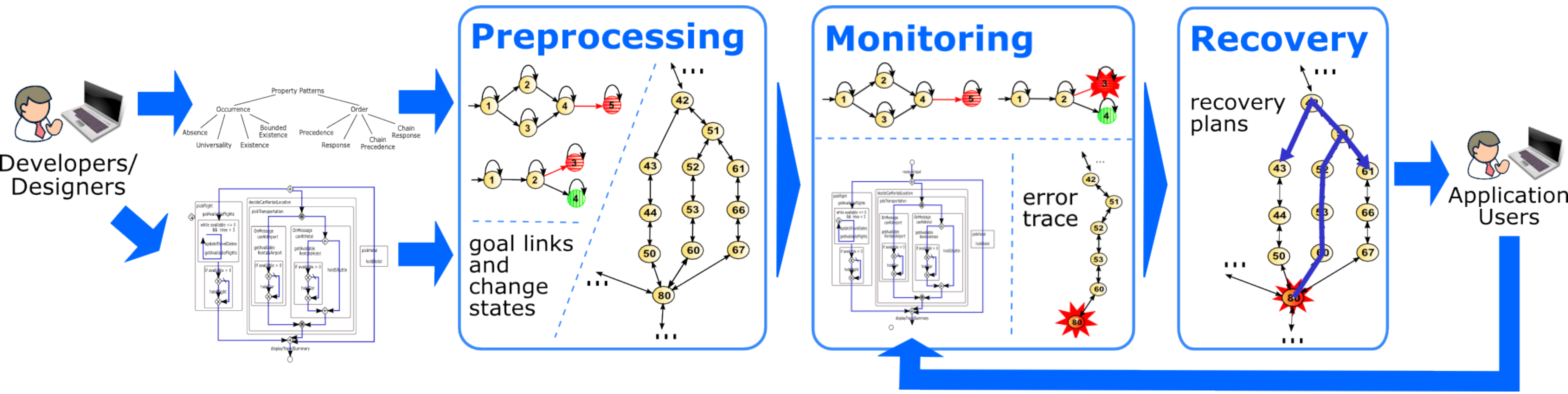
Plan: do nothing (no-op)

Planner used: Blackbox

Ranking Plans + Post Processing

- Ranking plans is based on:
  - Ranking of goal links
  - Length of plans
  - Cost of compensation for each plan
- Post processing:
  - Goal: display plans on the level of BPEL
  - Based on traceability between BPEL and LTS
- Plan execution:
  - When compensation actions are executed, monitors move backwards

Summary: User-Guided Recovery



Preprocessing

- BPEL →LTSA translation:
  LTSA tool + new
- Property translation:
  new (incomplete)
- Goal links, change states:
  python-automata + new

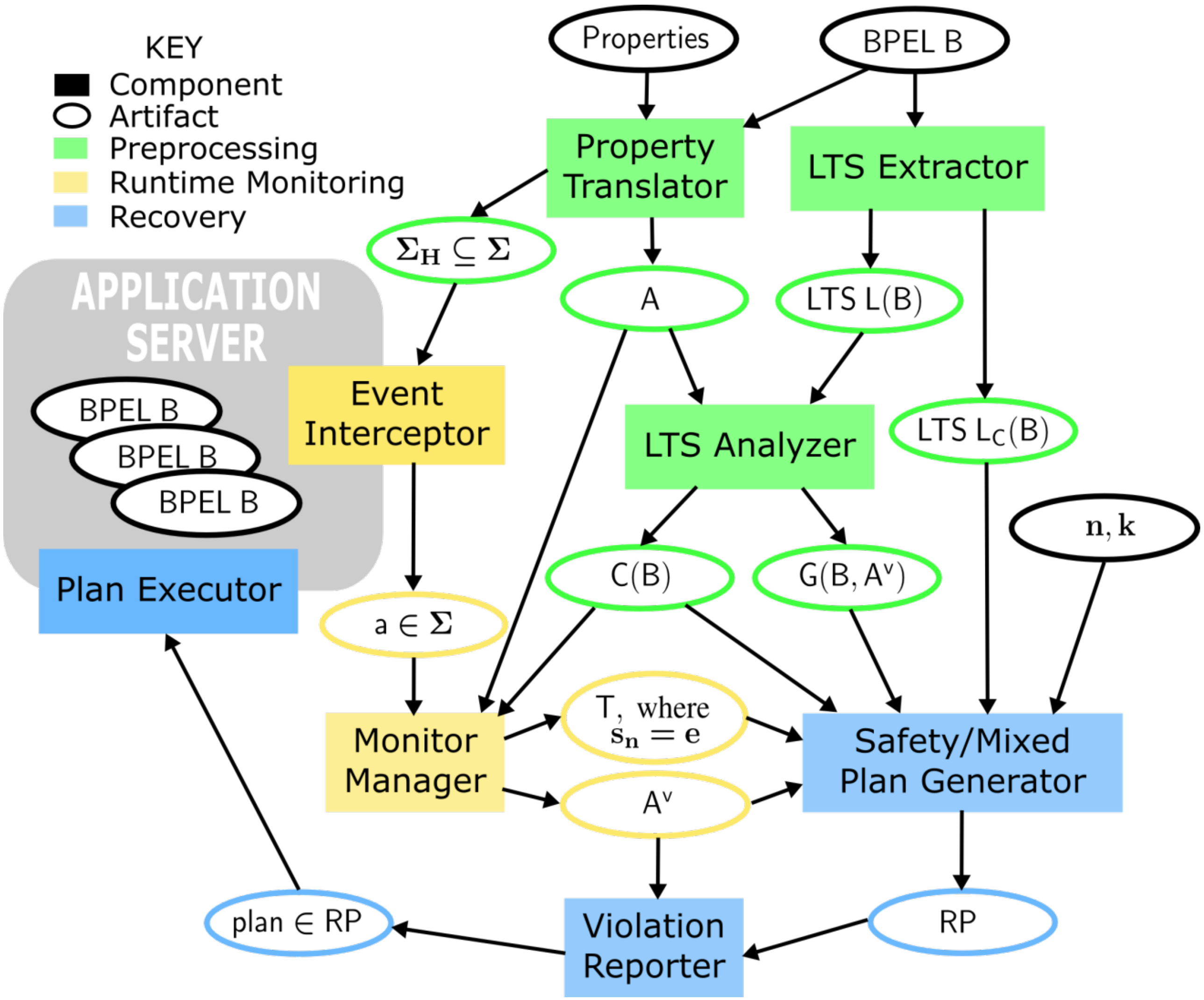
Monitoring

- BPEL engine:
  WebSphere Process
  Server (WPS)
- Monitoring:
  WPS plugin

Recovery

- Planner:
  Blackbox
- Generation of multiple plans:
  new, based on SAT-solver
- Plan ranking +
  Post-Processor:
  new

Our Framework



Evaluation

App.	Our approach					[Carzaniga et al. '08]	
	k	vars	clauses	plans	time (s)	length	plans
`FV`	15	797	16,198	2	0.04	≤ 2	1
	22	1,436	33,954	4	0.74	≤ 3	5
	26	1,804	44,262	8	1.14	≤ 4	13
	42	3,276	85,494	40	3.12	≤ 8	412
`FC`	4	42	159	1	0.01	≤ 1	0
	6	95	592	2	0.02	≤ 2	2
	12	321	3,248	4	0.15	≤ 3	8
	16	554	7,393	5	0.27	≤ 4	22
	20	856	14,427	13	1.38	≤ 8	484

[Carzaniga et al. '08]:

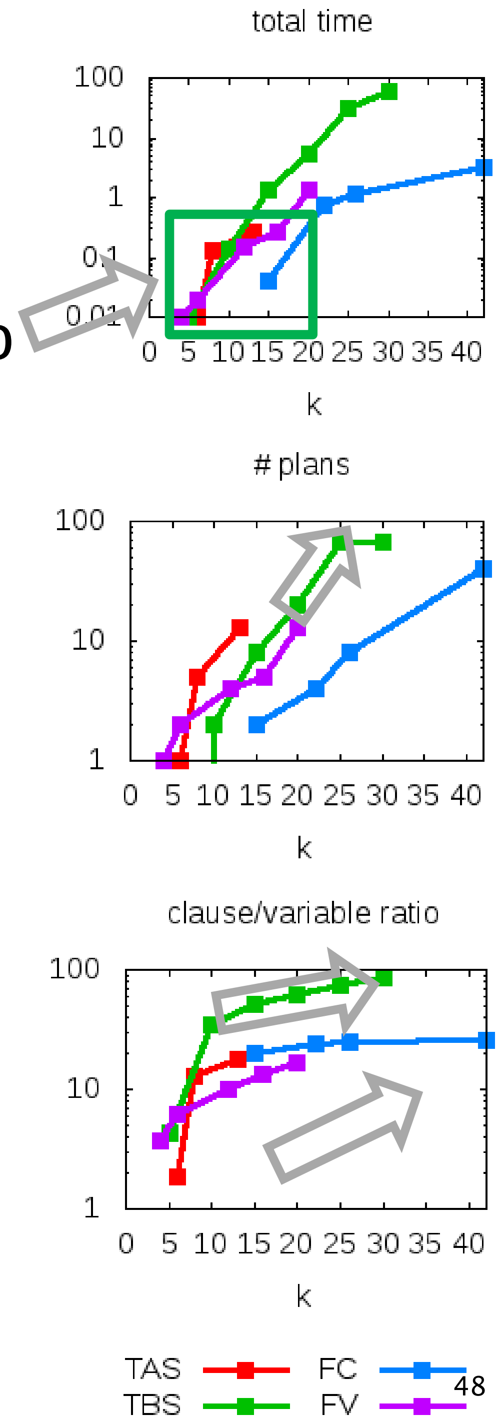
- full state space exploration
- manually created application models
- manually picked goal states

Evaluation

- Expected plans for TBS computed in first two steps
- Steep jump in number of plans caused by exploring alternatives far from the error

  Can we use safety properties to avoid this explosion?
- SAT instances become harder as we increase k, so average time to compute a plan also increases

  Incremental SAT (k → k+1)?
- Scalability?
  - TBS is more complex than other applications
  - … but step k = 30 (68 plans) only took ~ 60 s



Related Work

- Runtime Monitoring – property specification
  - [Mahbub and Spanoudakis ‘04]: event calculus
  - [Baresi and Guinea ‘05]: service pre- and postconditions
  - [Li et al. ‘06]: patterns (without nesting)
  - [Pistore and Traverso ‘07]: global LTL properties
- Recovery mechanisms
  - [Dobson ‘06]: add fault tolerance at compile time
  - [Fugini and Mussi ’06]: predefined fault/repair registry
  - [Ghezzi and Guinea ‘07]: BPEL exception handlers, predefined recovery rules
  - [Carzaniga et al. ‘08]: use existing redundancy

Summary and Challenges

- Success: built a prototype of a user-guided runtime monitoring and recovery framework for web-services expressed in BPEL
  - … Integrated with IBM Web Process Server
- Challenge: Compute fewer plans
  - Use safety properties to decrease the number of “liveness” plans computed
  - Improve precision of change state computation
    - Investigate “relevance” of change states w.r.t. a property
    - Employ static analysis of LTSs
      - “if all paths out a state definitely lead to an error, it is not a change state”
- Challenge: Improve scalability of plan computation
  - Reuse results of SAT solving for plans of length k for k+1

More Challenges

- Coming up with correctness properties
- Modeling data (e.g., `NOT_SAME_DATE`)
  - Can specify “derived events” for monitoring
  - So that monitors can register for them
  - Unclear how to use in recovery
- Modeling compensation
  - We model compensation by back arcs
  - But BPEL compensation is much more general, perhaps moving the system into a completely new state
    - … especially if data is involved
- Developing this framework outside of IBM’s WebSphere, for others to experiment with
  - Dependency: event registry, intercepting events before TERMINATE
  - Chosen plan execution can be implemented using dynamic flows [van der Aalst ‘05]

Lessons Learned

- Application of expected techniques to new domains may lead to unexpected conclusions
- Interesting combination of engineering, software engineering, modeling and verification challenges
- Enables verification experts make a big difference to real state of practice

References

- Our work:
  - [IEEE Transactions on Services Computing ‘09]
  - Recent conference and book chapter submissions
  - Patent being written

[Blum and Furst ‘95] A. Blum and M. Furst. “Fast planning through planning graph analysis”. Artificial Intelligence, 90(1-2):281—300, 2005.

[Carzaniga et al. ‘08] A. Carzaniga, A. Gorla, M. Pezze. “Healing Web Applications through Automatic Workarounds”. *STTT*, 10(6):493--502, 2008.

[Foster ‘06] H. Foster. *A Rigorous Approach to Engineering Web Service Compositions*. Ph.D. thesis, Imperial College London, 2006

[Fu, Bultan, Su ‘04] X. Fu, T. Bultan and J. Su. “Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services”. Theoretical Computer Science, 328(1-2):19--37, 2004.

[van der Aalst ‘05] W. van der Aalst and M. Weske. “Case Handling: a New Paradigm for Business Process Support”. *Data Knowledge Engineering*, 53(2):129--162, 2005.

Acknowledgements

- Web Service runtime monitoring and recovery
  - Jocelyn Simmonds, Shoham Ben-David, Bill O'Farrell (IBM), Yuan Gan, Shiva Nejati
- Model-checking, abstractions, vacuity, counterexample analysis
  - Arie Gurfinkel (SEI CMU), Ou Wei, Aws Albarghouthi, Benet Devereux + many others!
- Model management
  - Sebastian Uchitel (Imperial College + Univ. of Buenos Aires), Shiva Nejati, Mehrdad Sabetzadeh, Rick Salay, Steve Easterbrook, Michalis Famelis, folks at AT&T and General Motors

A simple research map

Multi-Valued logics + Model Checking

Reasoning with partial and inconsistent information

Abstraction

General study of models for representing abstractions

Software Model Checking

Checking behavioral properties of programs

Vacuity Detection

How to trust automated analysis

Temporal Logic Query Checking

Computer-aided model exploration

Understanding Counterexamples

Understanding and exploring results of automated analysis

Model Management

Synthesis, merge, analysis of structural and behavioral models

Domain-specificity: Web services

Runtime monitoring and recovery of web service conversations

Domain-specificity: automotive

Dealing with systems of models

More on Model Management

- Eliminate one of the major verification challenges: coming up with the right level of abstraction for tractable and precise analysis
- Interesting problems:
  - “correct” refinements of models into code
  - Dealing with change propagation, on model and on code level
  - And many other

Thank you!
Questions?

Additional references

[Mahbub and Spanoudakis ‘04] K. Mahbub and G. Spanoudakis. “A Framework for Requirements Monitoring of Service-based Systems”. In ICSOC ’04, 84--93, 2004.

[Baresi and Guinea ’05] L. Baresi and S. Guinea. “Towards Dynamic Monitoring of WS-BPEL Processes”. In ICSOC ‘05, 269--282, 2005.

[Li et al. ‘06]: Z. Li, Y. Jin and J. Han. “A Runtime Monitoring and Validation Framework for Web Service Interactions”. In ASWEC ’06, 70--79, 2006.

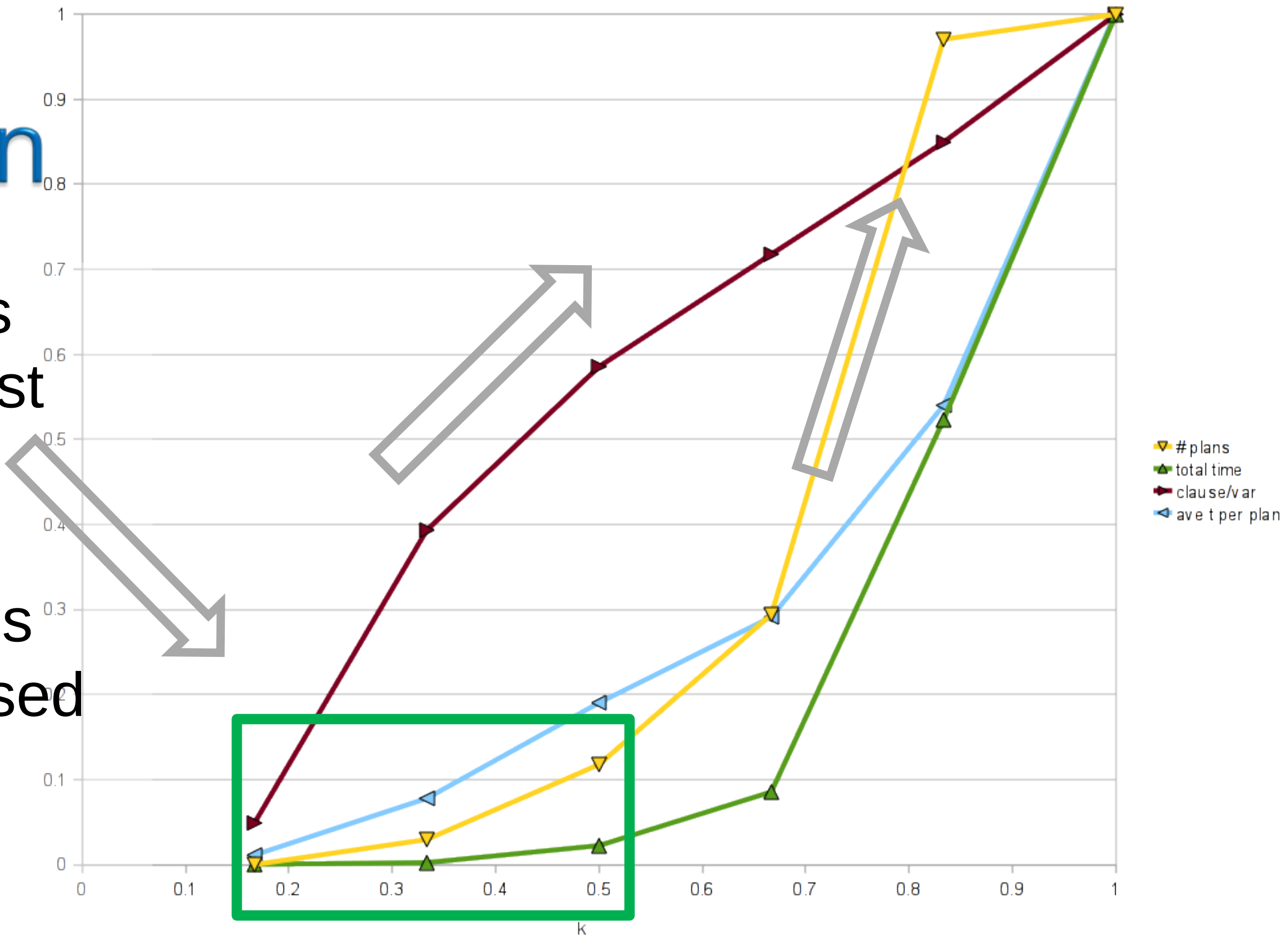
[Dobson ’06] G. Dobson. “Using WS-BPEL to Implement Software Fault Tolerance for Web Services”. In EUROMICRO-SEAA ‘06, 126--133, 2006.

[Fugini and Mussi ’06] M. Fugini and E. Mussi. “Recovery of Faulty Web Applications through Service Discovery”. In SMR-VLDB, 67--80, 2006.

[Pistore and Traverso ’07] M. Pistore and P. Traverso. “Assumption-Based Composition and Monitoring of Web Services”. In Test and Analysis of Web Services, 307--335, 2007.

[Ghezzi and Guinea ’07] C. Ghezzi and S. Guinea. “Run-Time Monitoring in Service-Oriented Architectures”. In Test and Analysis of Web Services, 307--335, 2007.

Evaluation



- Expected plans computed in first two steps
- Steep jump in number of plans generated caused by exploring alternatives far from the error

Can we use safety properties to avoid this explosion?

- SAT instances become harder as we increase k, so average time to compute a plan also increases

Incremental SAT (k → k+1)?