Verification of the IBM System Automation's Expert System

Carsten Sinz, Wolfgang Küchlin

Symbolic Computation Group, WSI, University of Tübingen

Thomas Lumpp

zSeries System Management, IBM Germany Development Lab

Overview

- Introduction IBM System Automation for OS/390
- Presentation of the built-in Expert System
- Consistency Criteria of the Rule Set
- Verification Methodology
- Results
- Conclusion

IBM System Automation (SA)

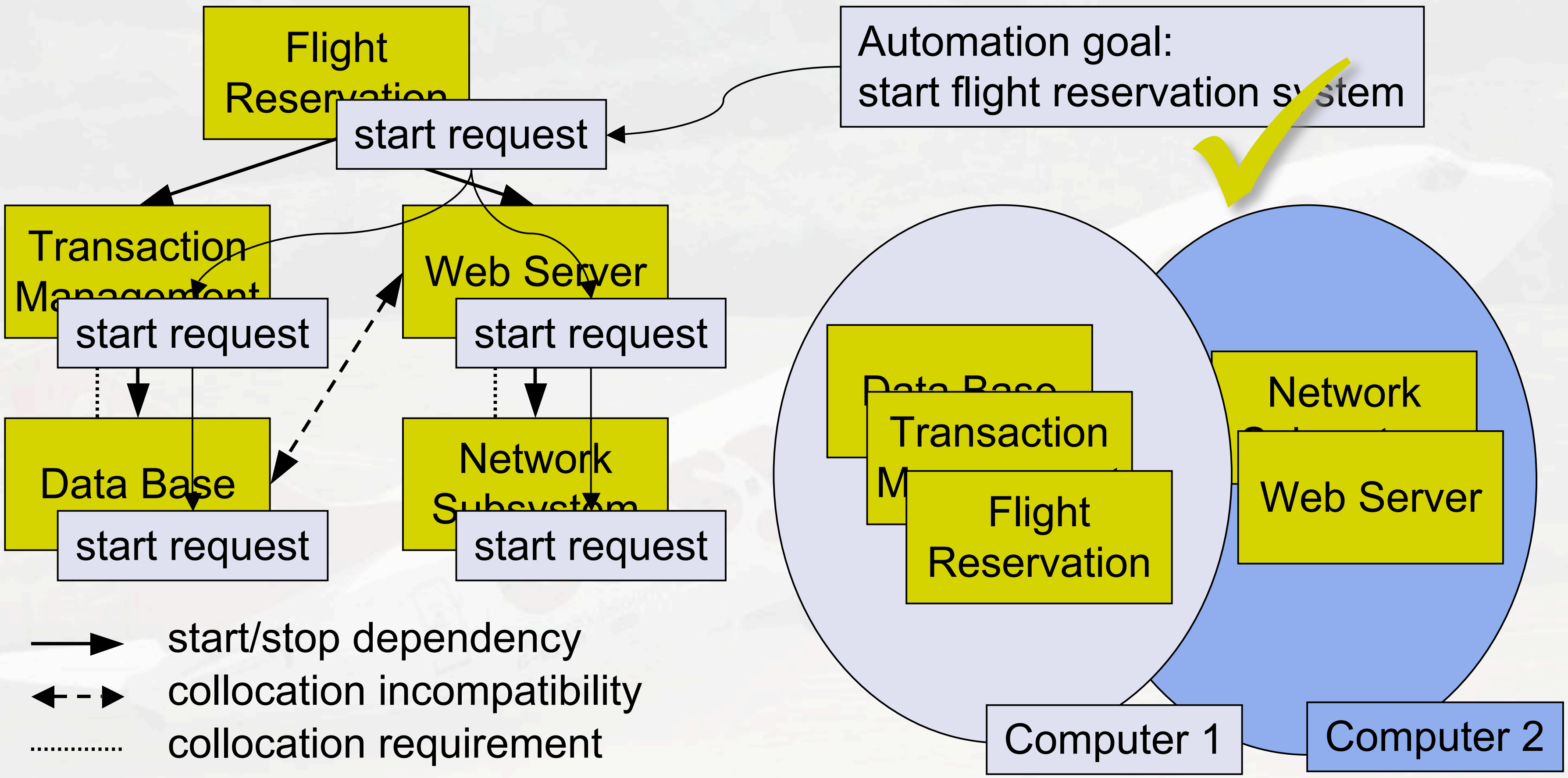
Automates operation of computer centers:

- Starting/stopping of applications
  (taking *dependencies* into account)
- Moving of applications between computers
  (e.g. on failure, for workload balancing)
- Supervision (active monitoring) of applications
  (current status? failure? system's workload?)
- Failure detection and error recovery

IBM System Automation (SA) (cont'd)

- Actions driven by *Automation Goals*, e.g.
  - start application A
  - move application B from S_1 to S_2
- Grouping allows simplified automation of complex applications.
- Plans generated and executed by Automation Manager

SA Example: Flight Reservation System



The Expert System of SA's Automation Manager

- Contains rules for each resource (application, computer system)
- Computes status of resources, propagates start/stop requests
- Situation-action rules (WHEN-THEN) for setting variables

Expert System: Rule Example

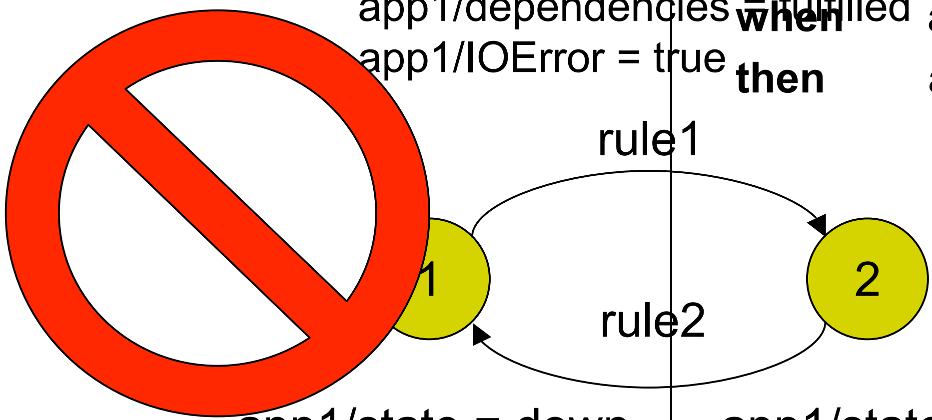
```
CORRELATION set/status/compound/satisfactory:
WHEN   status/compound NOT E {satisfactory}
    AND  status/startable E {yes}
    AND
    (     (        status/observed E {available, wasAvailable}
            AND  status/desired E {available}
            AND  status/automation E {idle, internal}
            AND  correlation/external/stop/failed E {false}
          )
          OR
          (        status/observed E {softDown, standBy}
            AND  status/desired E {unavailable}
            AND  status/automation E {idle, internal}
          )
    )
THEN    SetVariable status/compound = satisfactory
        RecordVariableHistory status/compound
```

SA's Expert System: Example

correlation rule1:
when app1/state = down
and app1/goal = up
and app1/dependencies = fulfilled
then app1/state = up

correlation rule2:
when app1/state = up
and app1/IOError = true
then app1/state = down

correlation rule3:
when app1/IOError = true
then app1/dependencies = pending



Verification Method

- Converting the rules to PDL (propositional dynamic logic)
- Formulating consistency properties in PDL
- Converting consistency properties to BOOL (Boolean or propositional logic)
- Running an Automatic Theorem Prover (ATP)
- Simplifying the result of the ATP

Verification Step 1: Converting Rules to PDL

PDL allows reasoning about programs α,β:

$\alpha;\beta$	consecutive execution
$\alpha\cup\beta$	nondeterministic choice
α^*	finite, nondeterministic repetition
$F?$	test for property (formula) F
$[\alpha]F$	after all terminating executions of α F holds
$\langle\alpha\rangle F$	there is a terminating program run of α after which F holds
$\Delta\alpha$	the program α^* can diverge

Verification Step 1: Converting Rules to PDL (cont'd)

1. Conversion of finite domains

   New propositions $P_{v,d}$ *for each* variable *v* and each possible value *d* of *v.*

2. Introduction of atomic programs

   Atomic programs $\alpha_{v,d}$ for the assignment operation v=d.

3. Translation of rules

   when $F_{v,d}$ **then** $\alpha_{v,d}$ is translated to $(F_{v,d} \wedge \neg P_{v,d})?;\alpha_{v,d}$.

4. Translation of Single Step Program S and Automation Manager Program AM

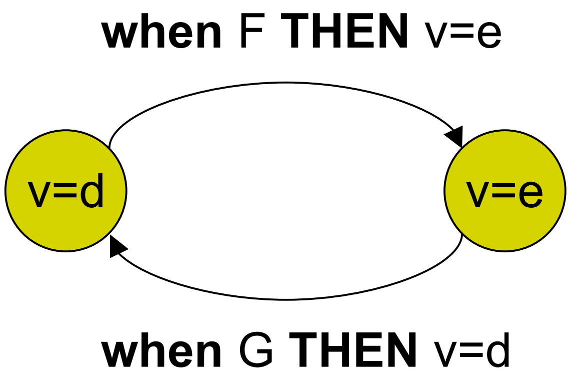
$$\mathrm{S} = \bigcup_{v,d_v} \left(F_{v,d_v} \wedge \neg P_{v,d_v} ?; \alpha_{v,d_v} \right) \quad \mathrm{AM} = \mathrm{S}^*; \bigwedge_{v,d_v} \left(F_{v,d_v} \Rightarrow P_{v,d_v} \right)?$$

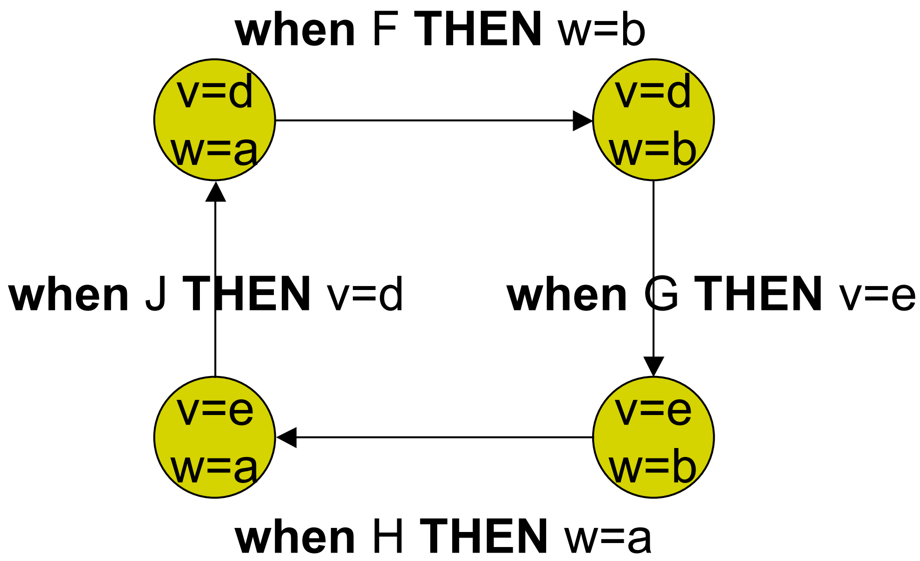
Verification Step 2: Consistency Properties in PDL

- Functionality (unique result of computation):
  $\langle AM \rangle p \Leftrightarrow [AM]p$ (for all propositions p)
- Termination:
  $\neg \Delta S$ (Δ is the divergence operator)
- other consistency criteria, e.g. confluence

Termination / Loops

All non-terminating programs caused by *program loops*, e.g.:





Verification Step 3: Termination Property in BOOL

- Preliminary: *Proper restriction* $F|_{v=d}$

$$P_{w,e}\Big|_{v=d} = \begin{cases} T & \text{if } v = w, d = e \\ \bot & \text{if } v = w, d \neq e \\ P_{w,e} & \text{if } v \neq w \end{cases}$$

allows specification of properties concerning multiple program states:

Let $s_0 \xrightarrow{v=d} s_1$. Then $s_1 \models F$ iff $s_0 \models F\big|_{v=d}$.

Verification Step 3: Termination Property (cont'd)

Example:

- Potential 2-loop: $s_0 \xrightarrow{v=d_1} s_1 \xrightarrow{v=d_0} s_0$
- Corresponding rules: **when** F **then** v=d_1
  when G **then** v=d_0
- Then validity of the formula

$$\neg(P_{v,d_0} \wedge F \wedge G|_{v=d_1})$$

  is a necessary condition for the absence of this 2-loop.
- Actual occurrance of error may depend on rule evaluation order.

Verification Step 3: Termination Property (cont'd)

- In SA ordered evaluation of variables (x<y<z<…), where x<y denotes that x is evaluated before y.
- Extended property indicating absence of 2-loops considering variable evaluation order:

$$\bigwedge_{w<v,d_w} (F_{w,d_w} \Rightarrow P_{w,d_w}) \Rightarrow \neg(P_{v,d_0} \wedge F \wedge G|_{v=d_1})$$

Verification Step 4: Automatic Theorem Prover

- Formulas generated in verification step 3 provide input for standard ATP program, e.g.
  - Davis-Putnam style prover (SAT)
  - BDDs (binary decision diagrams)
- Output is one of:
  - “no error” resp. list of counterexamples (SAT)
  - “no error” resp. formula representing all counterexamples (BDDs)

Verification Step 5: Simplification of Result

- In case of error,

$$\mathrm{EF} := \bigwedge_{w<v,d_w} (F_{w,d_w} \Rightarrow P_{w,d_w}) \Rightarrow \neg(P_{v,d_0} \wedge F \wedge G|_{v=d_1})$$

  is not valid, but formula representing counterexamples may be huge.
- Simplification: remove irrelevant variables (not contained in the 2 rules under consideration) by existential abstraction in EF:

$$\exists \vec{X}.\mathrm{EF}$$

  where $\vec{X}$ contains all irrelevant variables.

Results

- Input Formulas:
  - Computation of resource's *compound status*, 3 errors (rule overlap)
  - 41 rules, 74 variables, ≈1500 symbols
- SAT
  - Runtimes for proving non-looping properties: <1 sec.
  - Formulas for loop errors have relatively large number of models (270-405) representing individual error cases.
- BDD
  - Generation time: 1-2 sec.
  - Generated BDDs have ≈100-200 nodes.
  - Simplification reduces number of error cases to 1-3.

Summary / Conclusion

- Goal:
  - Error detection in Rule-Based Expert Systems
- Method:
  - Conversion of consistency properties to SAT
  - Application of current SAT-checking technology
- Benefits:
  - Correctness assertions possess high quality
  - Compared to testing: covers all possible cases
  - Generates generalized error patterns

Thanks for your attention!

Carsten Sinz
Symbolic Computation Group, WSI
University of Tübingen, Germany

http://www-sr.informatik.uni-tuebingen.de