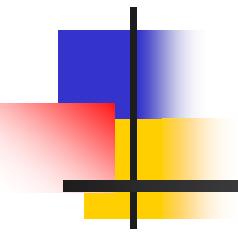
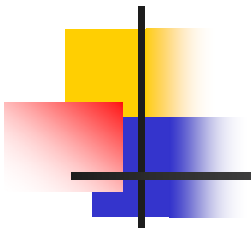


Efficient Rule Testing in Learning By Observation

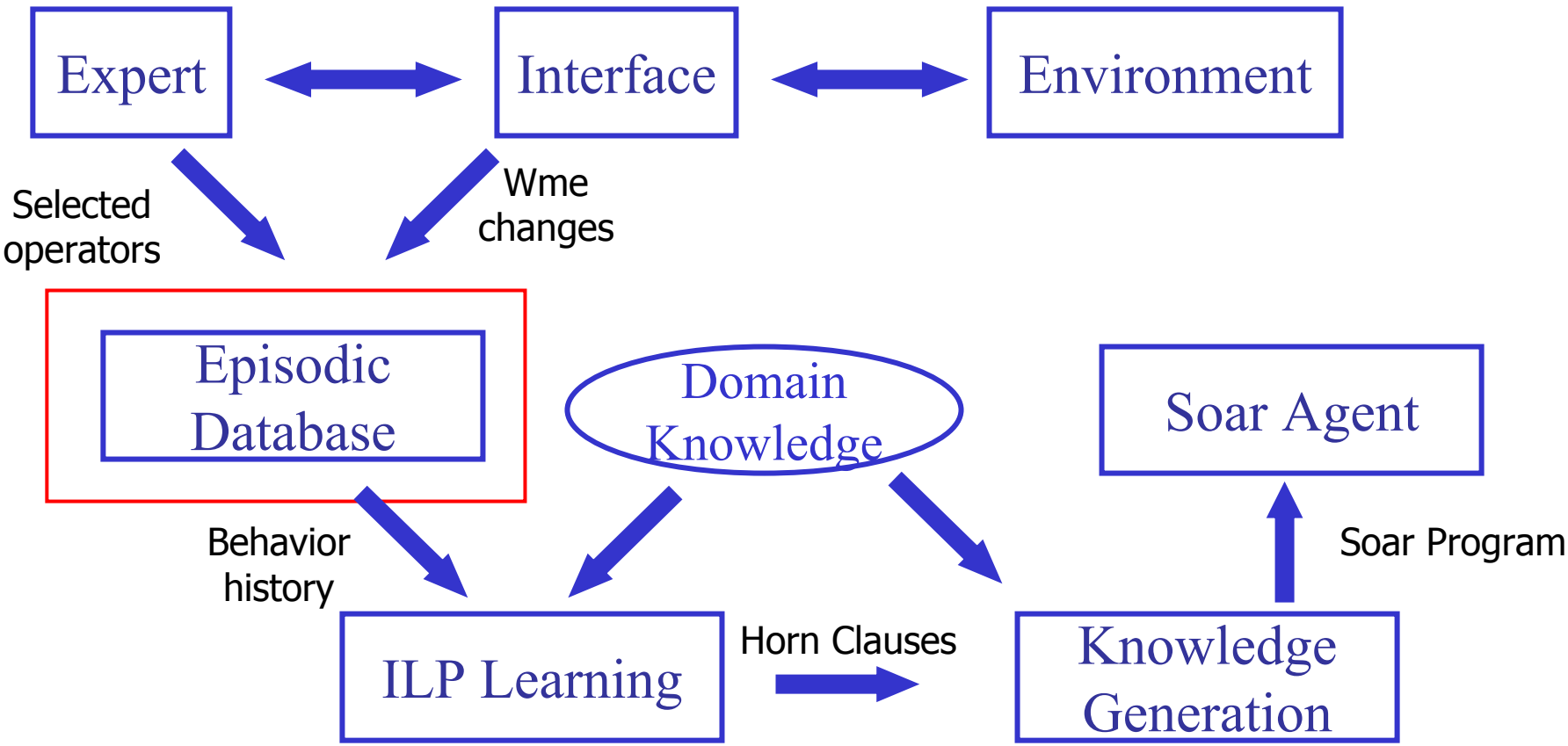
Tolga Konik
University of Michigan



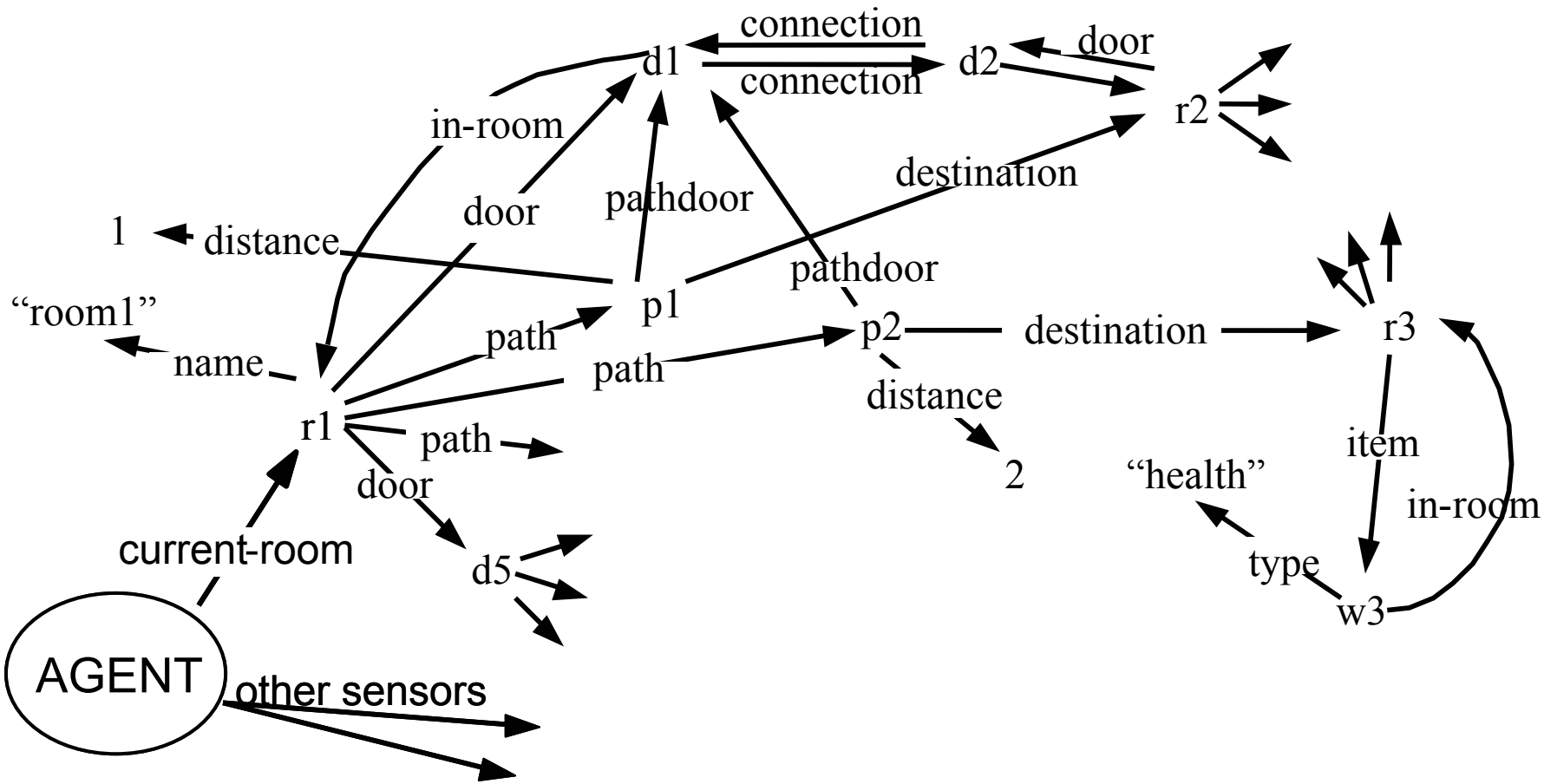
GOAL

- Storing and accessing an agent's past experience efficiently
- Key component of our learning by observation system
- Examining the behavior history of a Soar agent (i.e. for debugging)

Learning By Observation



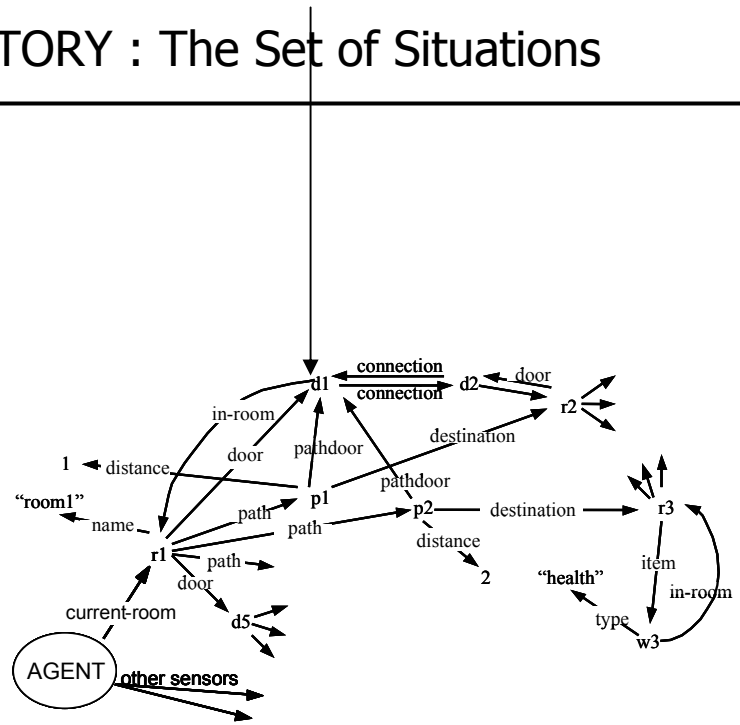
Snapshop of Working Memory



Snapshop of Working Memory

AGENT BEHAVIOR HISTORY : The Set of Situations

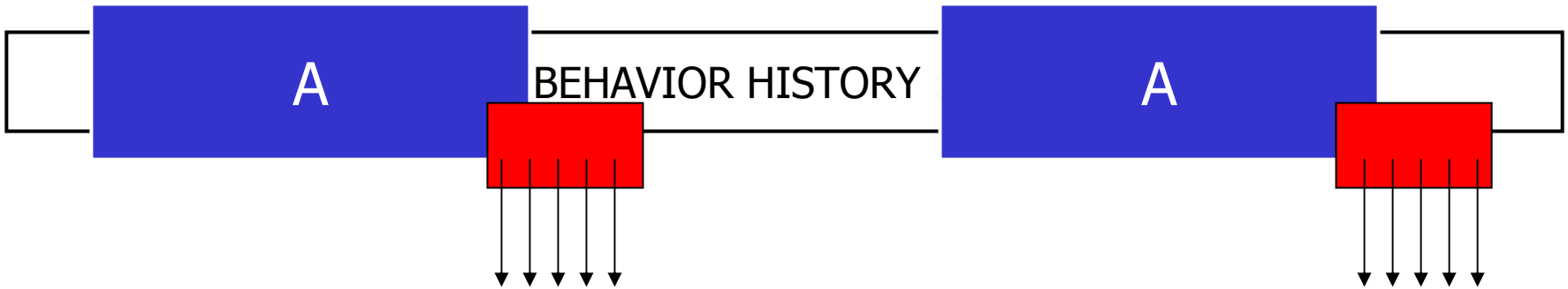
- Situation:
  - a snapshot of the working memory at a time



Example Situation Interval: Positive Examples of Termination

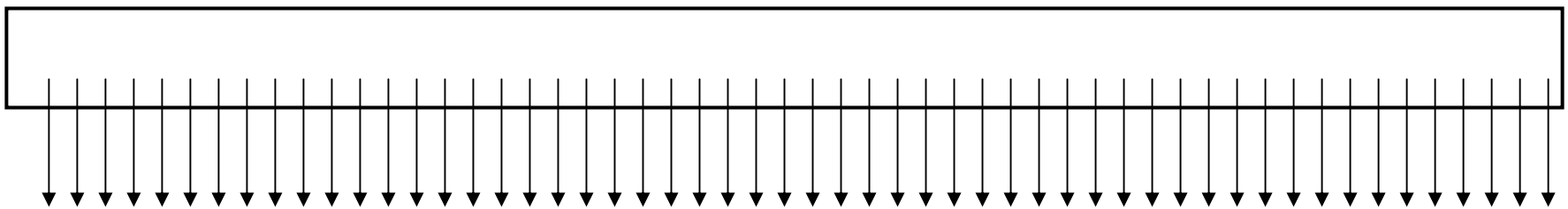
- Regions operator A is selected
- Positive examples of Termination Condition of A

Example Situation Interval: Positive Examples of Termination



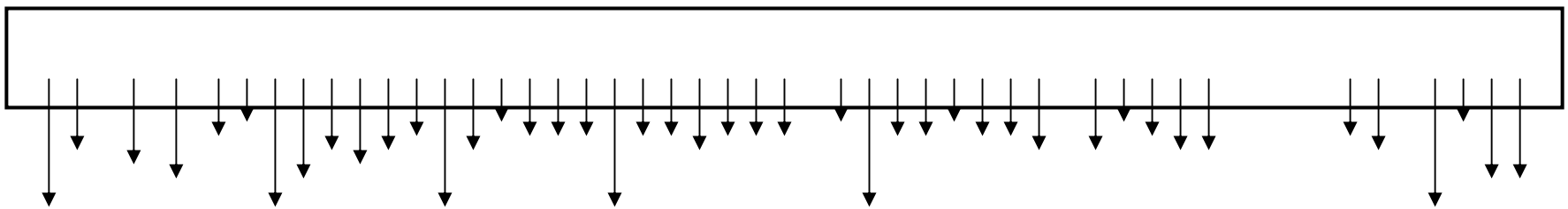
- hypotheses are tested over each "positive situation"

Storing Explicit Situations



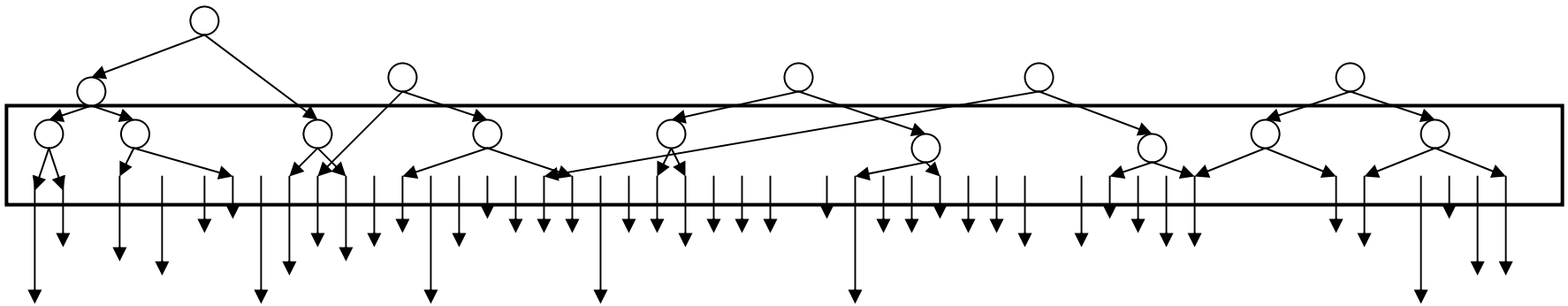
- For each situation, store all WMEs
- Space inefficient

Storing Changes in Situations



- Store only the changes of WMEs
  - i.e. KnoMic (van Lent 2000)
- the database has to be traced forward starting from the initial situation.
  - Difficult to use with learning algorithms that deal with noise

Storing Indexed Changes

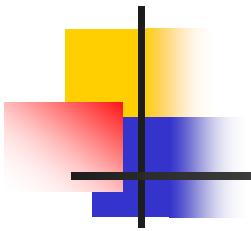


- Store the changes to WMEs and use index mechanism to efficiently access situations

Rule Testing at a Situation

- **Given:**
  - `R1` is a WME id representing a room
  - `S30` is a situation
- **Example Query:**
  - `(R1, ^contains-item, ?VALUE) at S30.`
- **Output:**
  - Return all `VALUE` such that

    `(R1 ^contains-item, VALUE)` holds at `S30`.

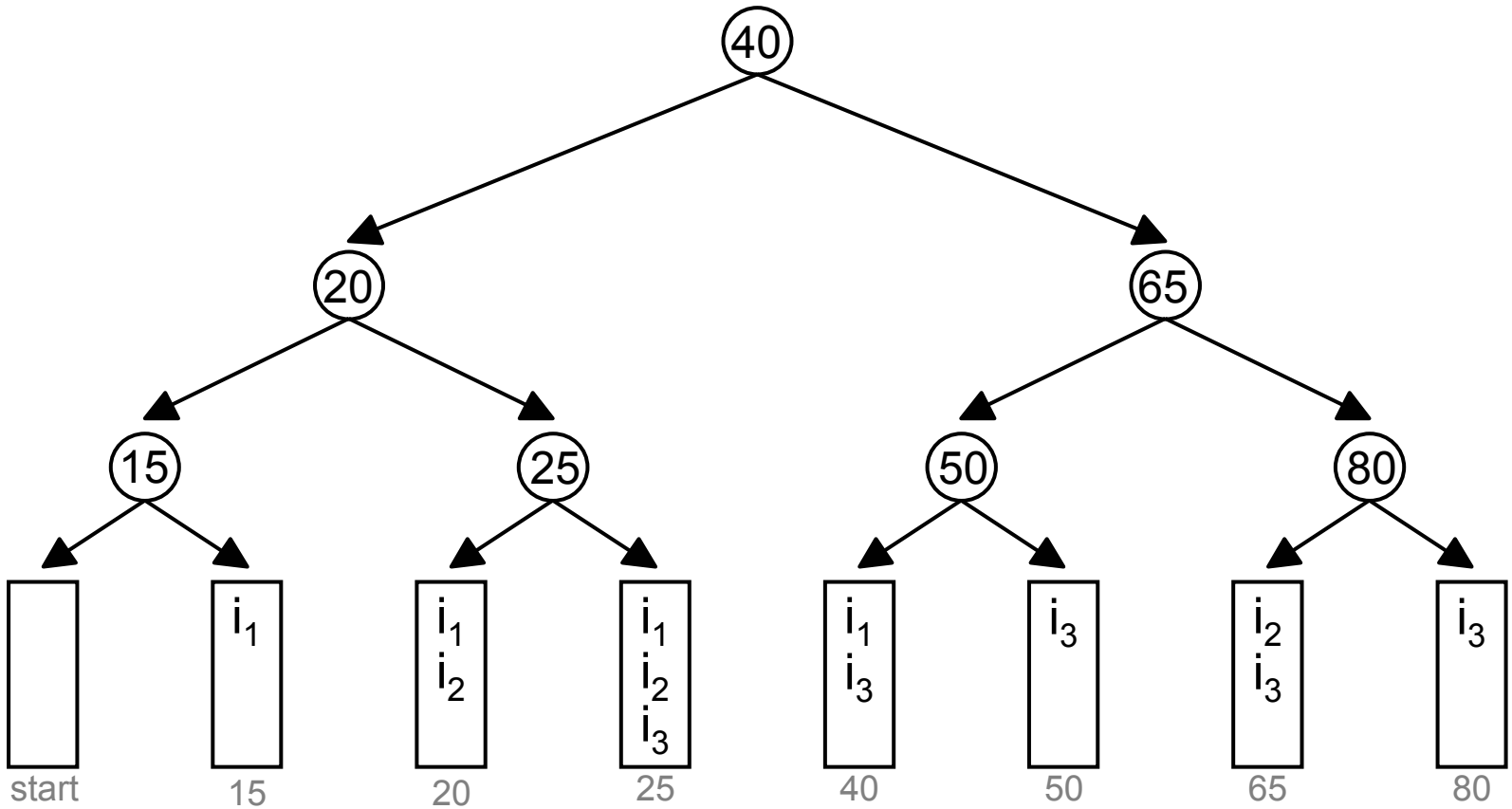


Storing Indexed Changes

- A binary search tree, for each (WME Id, Attribute)
- For each query:
  - Locate the corresponding search tree
  - Find values traversing the tree

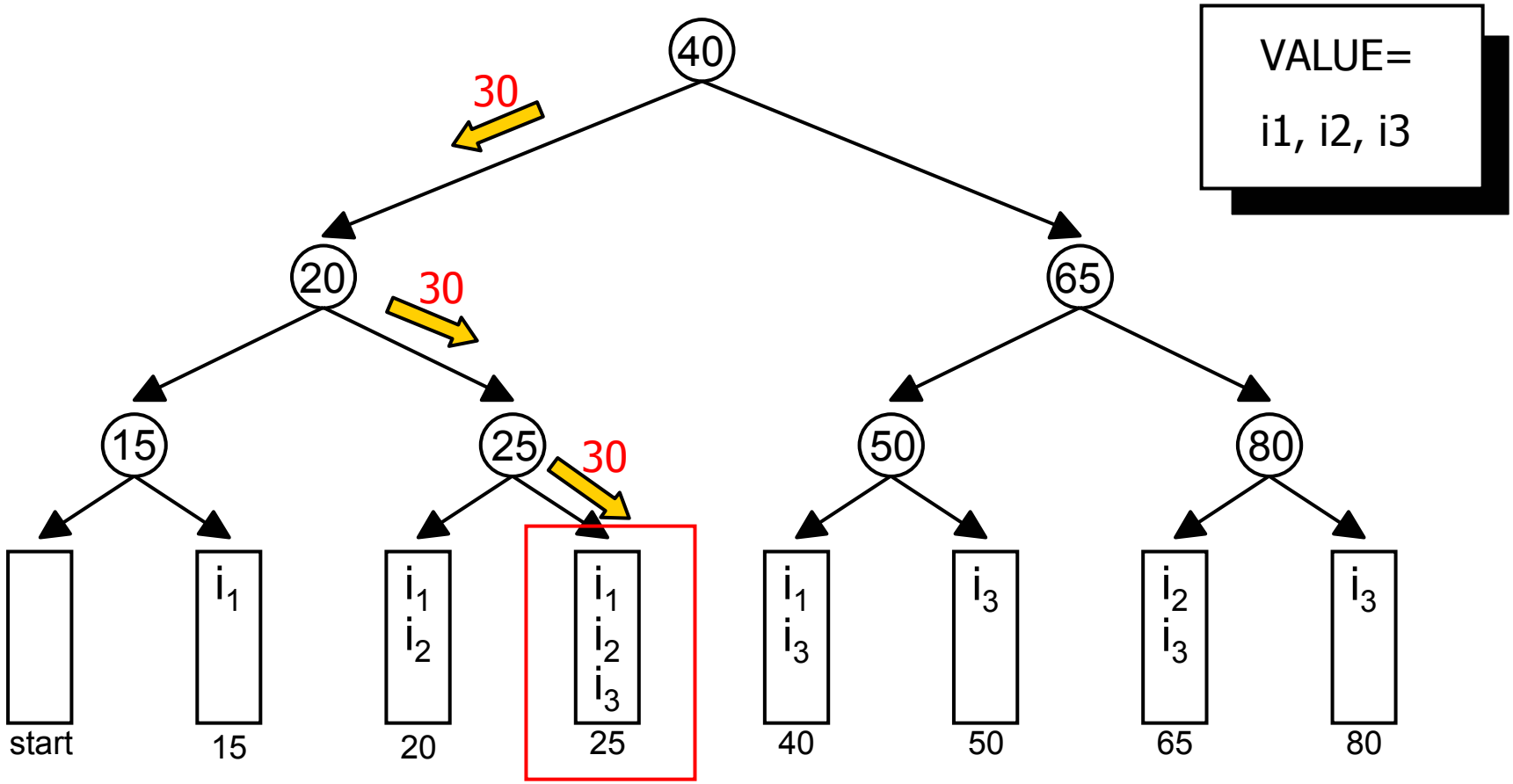
Binary Search Tree Example

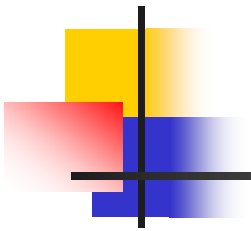
- Search tree of `(R1, contains-item)`



Binary Search Tree Example

Query: `(R1, ^contains-item, ?VALUE) at S30`



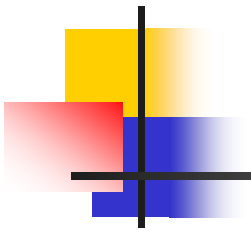


Rule Testing at a Situation

- Using this scheme, a rule can be tested at any situation in the history of an agent.
- This provides space efficiency, while keeping the access times reasonable.

- We want more than that !

Rule Testing over a Range of Situations

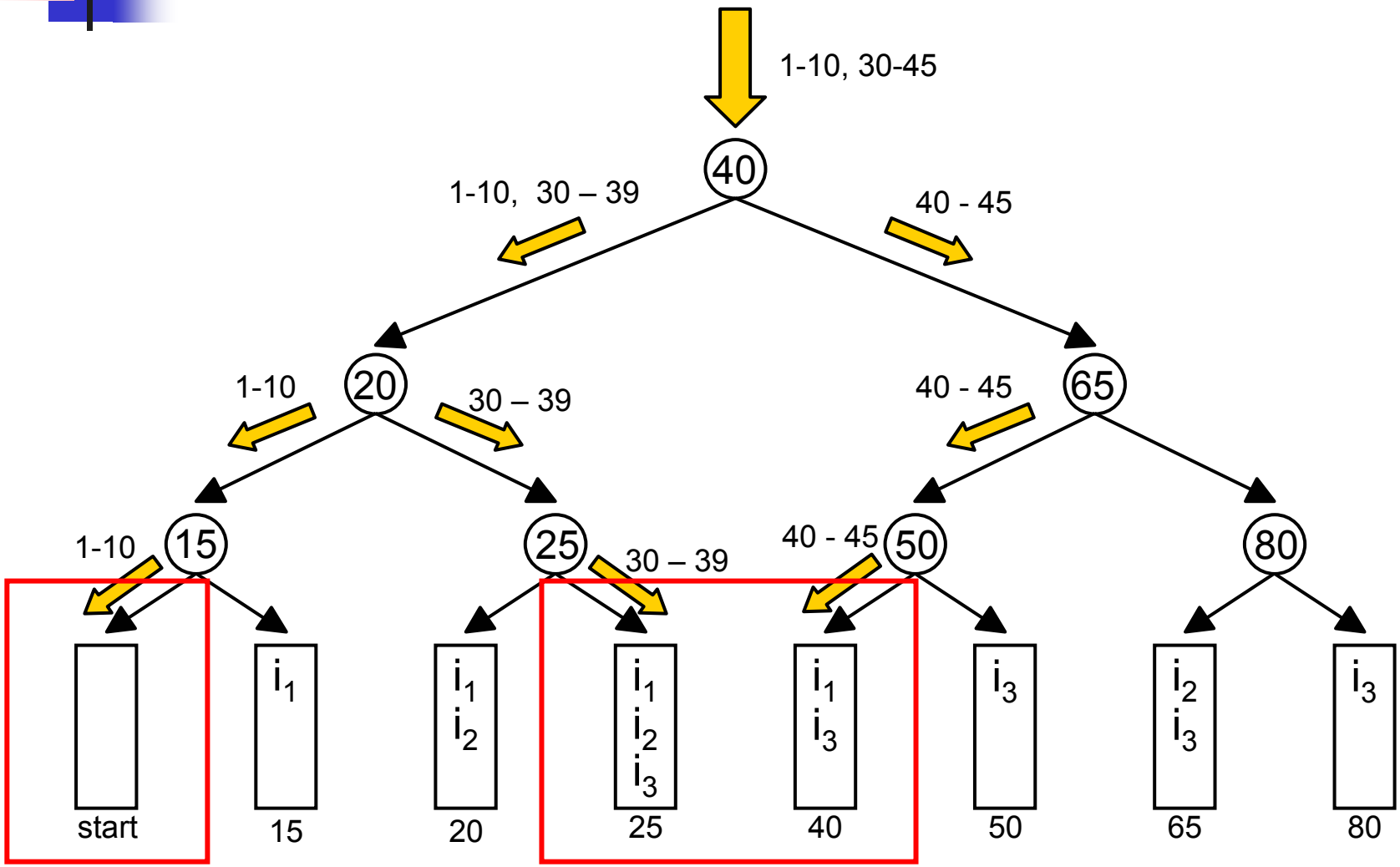


- **GOAL**
  - test a rule over a range of situations at once.
- **MOTIVATION**
  - In consecutive situations, similar conditions hold
- **RESULT**
  - More efficient than testing a rule at each situation individually

Testing a Condition on Multiple States

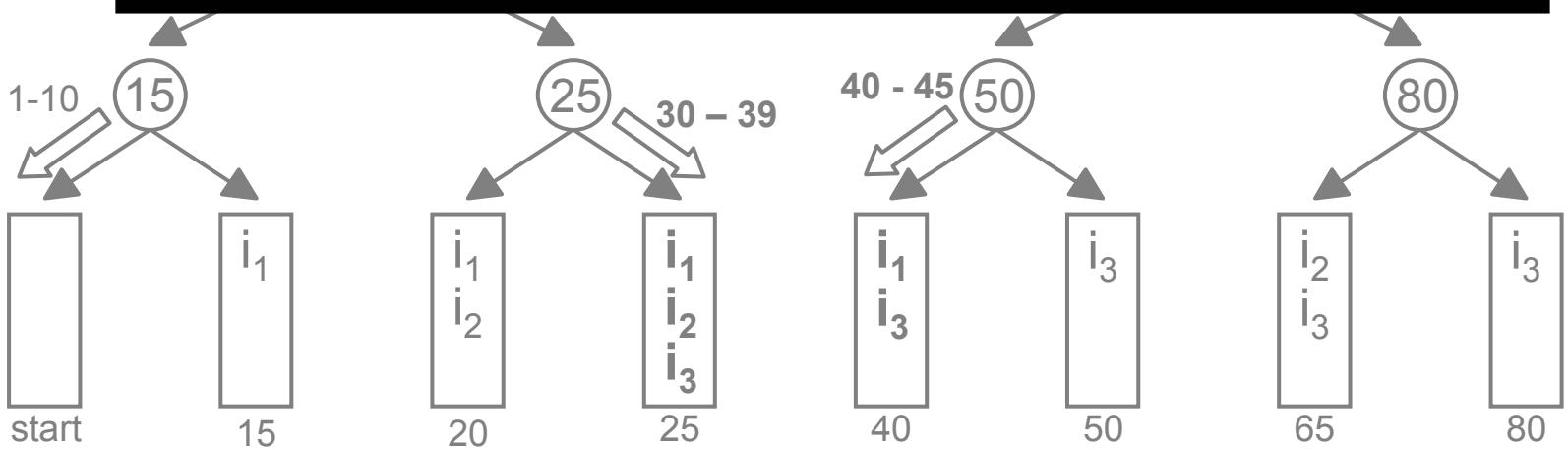
- **Query:**
  - `(R1, contains-item, ?VALUE)`
    `at S1-S10, S30-S45`
- **Output:**
  - A Set of (VALUE, Situation-Range) pairs

Testing a Condition on Multiple States

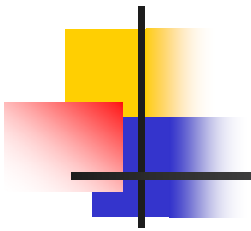


Testing a Condition on Multiple States

- (R1 ^contains-item i_1) at 30-39, 40-45
- (R1 ^contains-item i_2) at 30-39
- (R1 ^contains-item i_3) at 30-39, 40-45



Testing a Rule over a Range of Situations

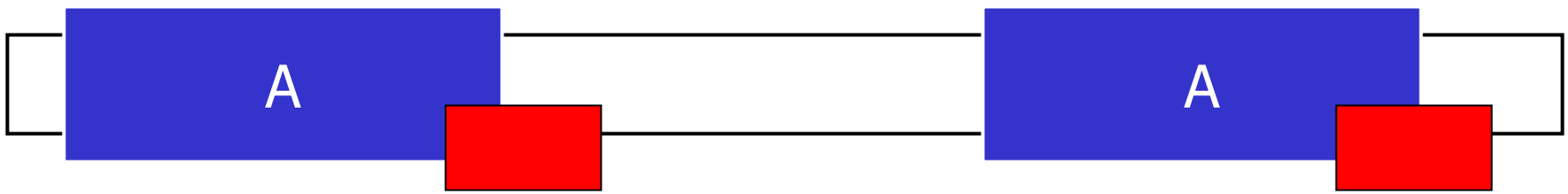


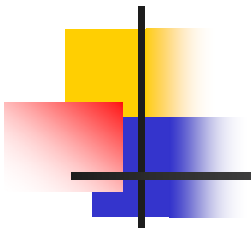
- A Rule contains multiple conditions
  - i.e. C1 **and** C2 …
  - Each Condition Ci propagates:
    - the values
    - the set of situations

    that satisfy Ci and previous conditions
- This idea can be also generalized for rules that contain "**not**" and "**or**"

Why is this useful?

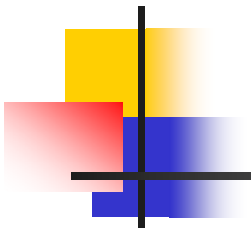
- In Learning by Observation :
  - The rules are checked over continues ranges





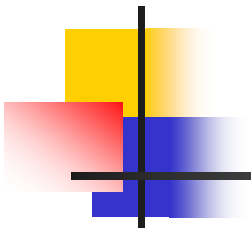
Why is this useful?

- In Soar Programing:
  - It may help to understand the behavior of a Soar agent.
  - i.e. Show me all situations when a Tanksoar agent has perceived a threat but did not fire.



Nuggests

- Range Testing is efficient
  - Space complexity
    - Significantly better than explicit storage
    - Proportional to the number of changes (not situations)
  - Time Complexity:
    - Significantly better than testing at each situation using the index.
    - Can potentially perform better than storing situations explicitly



Coals

- Performance depends on the Environment Representation & Rules
  - Multivalued attributes that change value often cause space inefficiency
  - As the rules gets longer,

    the query mechanism time efficiency decrease
  - multivalued attributes and “or” connections decrease time efficiency.
- No direct connection to Soar in current implementation