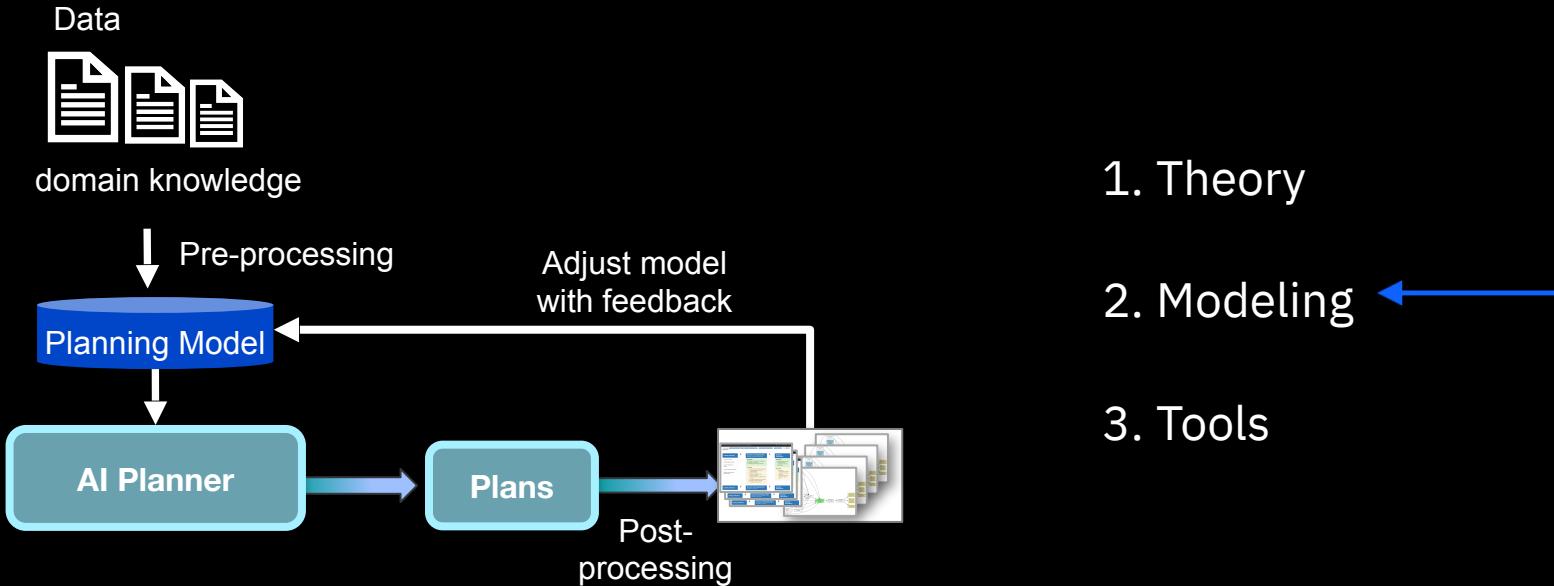


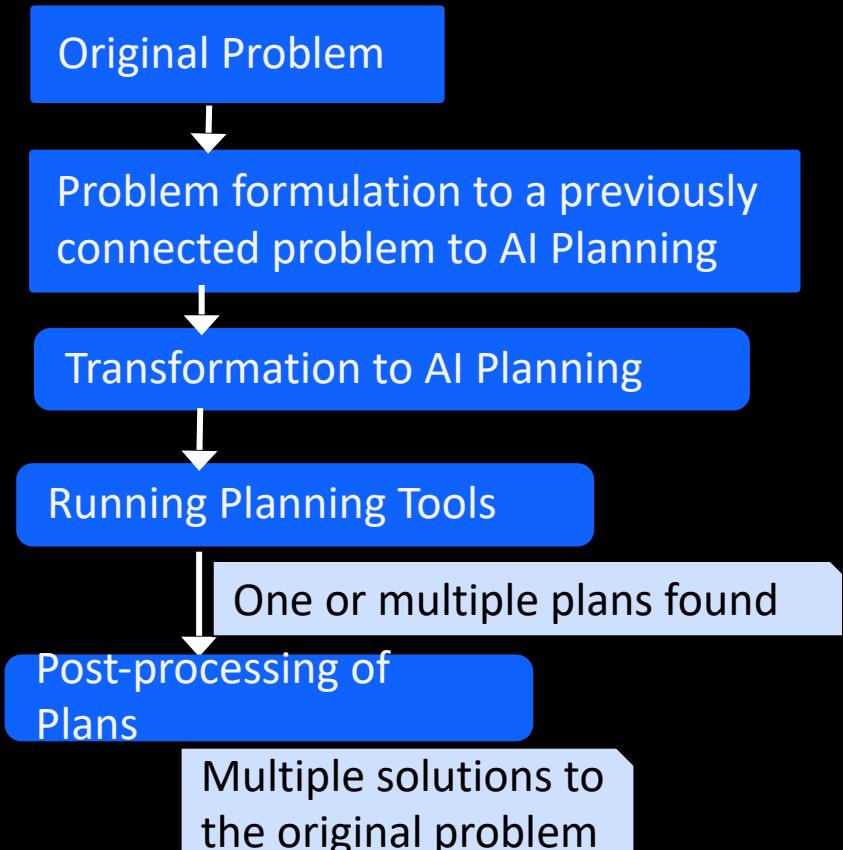
Modeling



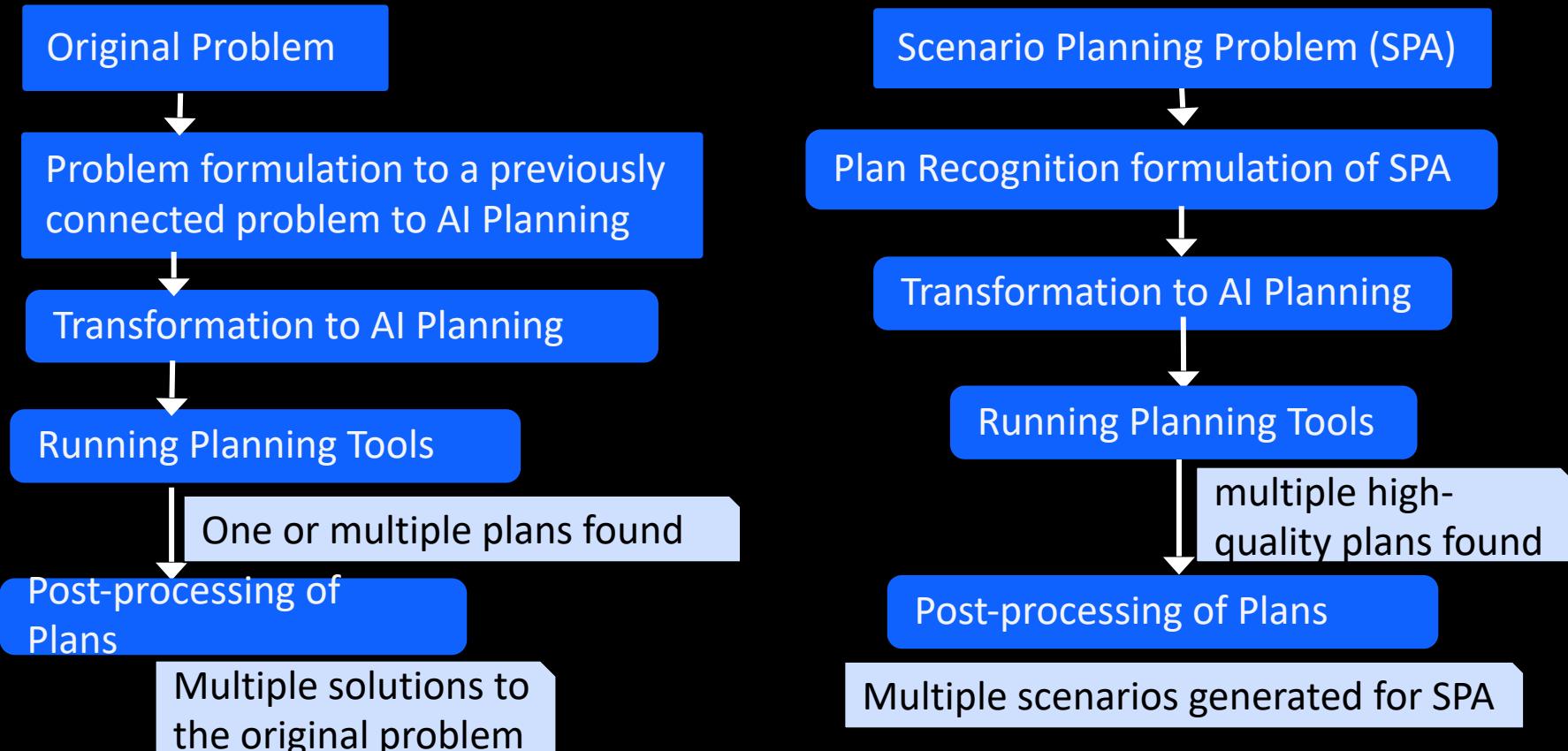
Modeling Challenges

1. Often need to transform the original problem to a problem with known/easier to compute solutions
2. Access to domain experts familiar with input to planners (e.g., PDDL) is rare.
3. Any form of knowledge is practically guaranteed to be incomplete and often inconsistent. Even human validation can be ambiguous
4. Current state-of-art learning approaches may not scale and may have assumptions that may not hold for the practical setting.
Further, the learned knowledge may be not consumable

Relationship to Planning



Relationship to Planning



Established relationships to AI Planning

- Finding excuses (Göbelbecker et al. 2010)
- Diagnosis (Sohrabi et al., 2010)
- Explanation generation (Sohrabi et al., 2011)
- Plan Recognition (Ramirez & Geffner IJCAI'09, AAAI'10, Sohrabi et al., IJCAI'16)

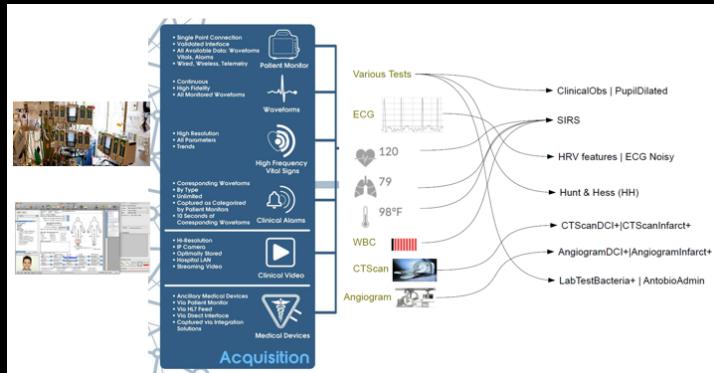
Which in turn helped with:

- Hypothesis Generation and Exploration (Sohrabi et al., 2013, 2015)
- Future State Projection (Sohrabi et al., 2017)
- Multi-agent Plan Recognition (Shvo et al., 2018)
- Scenario Planning (Sohrabi et al., 2018)

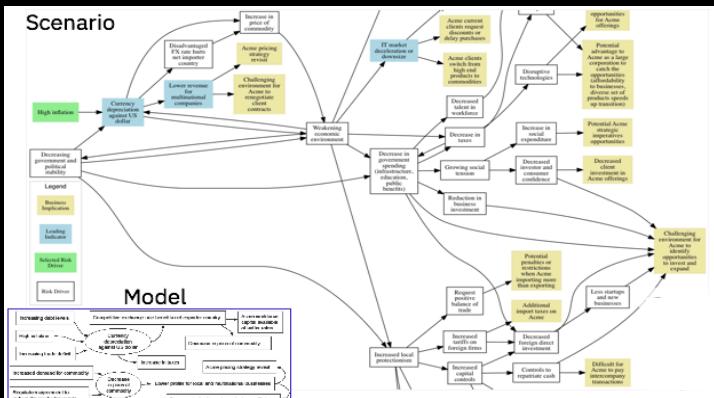
Additional Compilations

- Compiling away soft goals (Keyder and Geffner, 2009)
 - Use any classical planner to obtain a solution for a planning problem with soft goals (i.e., simple preferences)
- Translate HTN Problems into classical planning problems (Alford et al., 2016)
 - Use classical planners (heuristic search) to find solutions for an HTN planning problem.

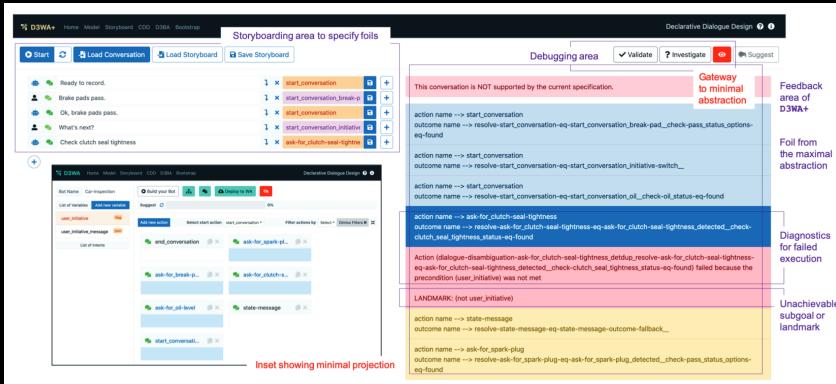
Examples



[Automated large-scale data analysis, ICAPS 2015]



7 [Scenario planning for enterprise risk management, AAAI 2018]

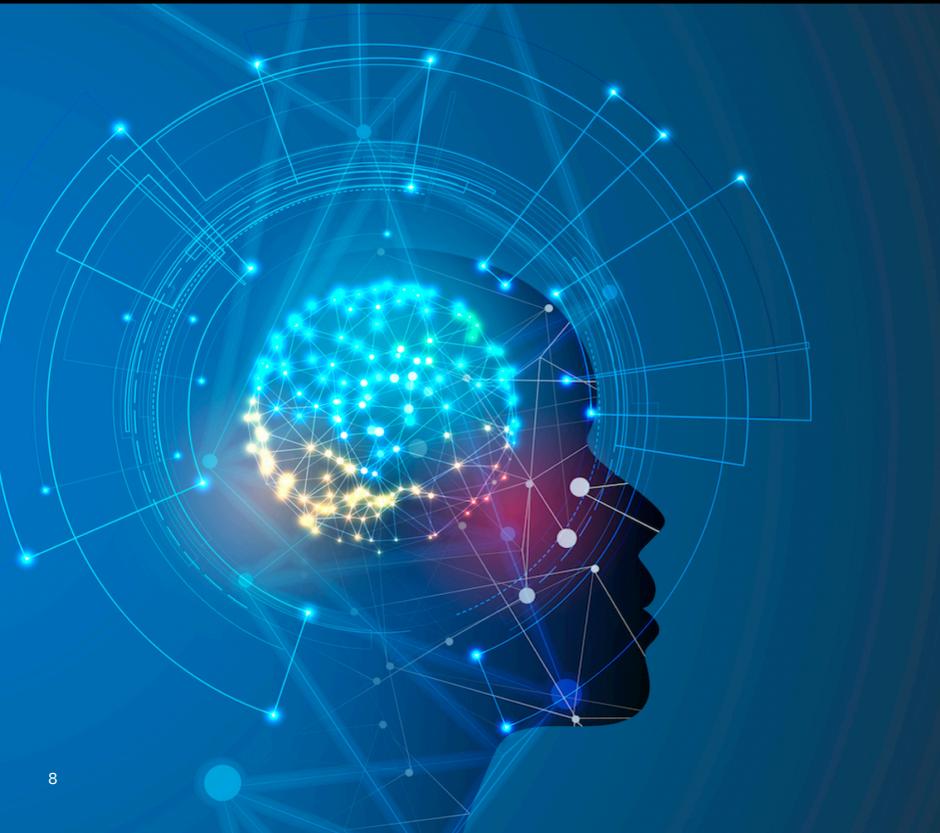


[D3WA+: A Case Study of XAIP in a Model Acquisition Task, ICAPS 2020]



[Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning, ICAPS 2020]

AutoAI: Automating ML Pipeline Generation



Problem

Space of possible pipelines is huge

Humans can explore only a tiny portion of it

Humans are biased towards the pipelines they already are familiar with

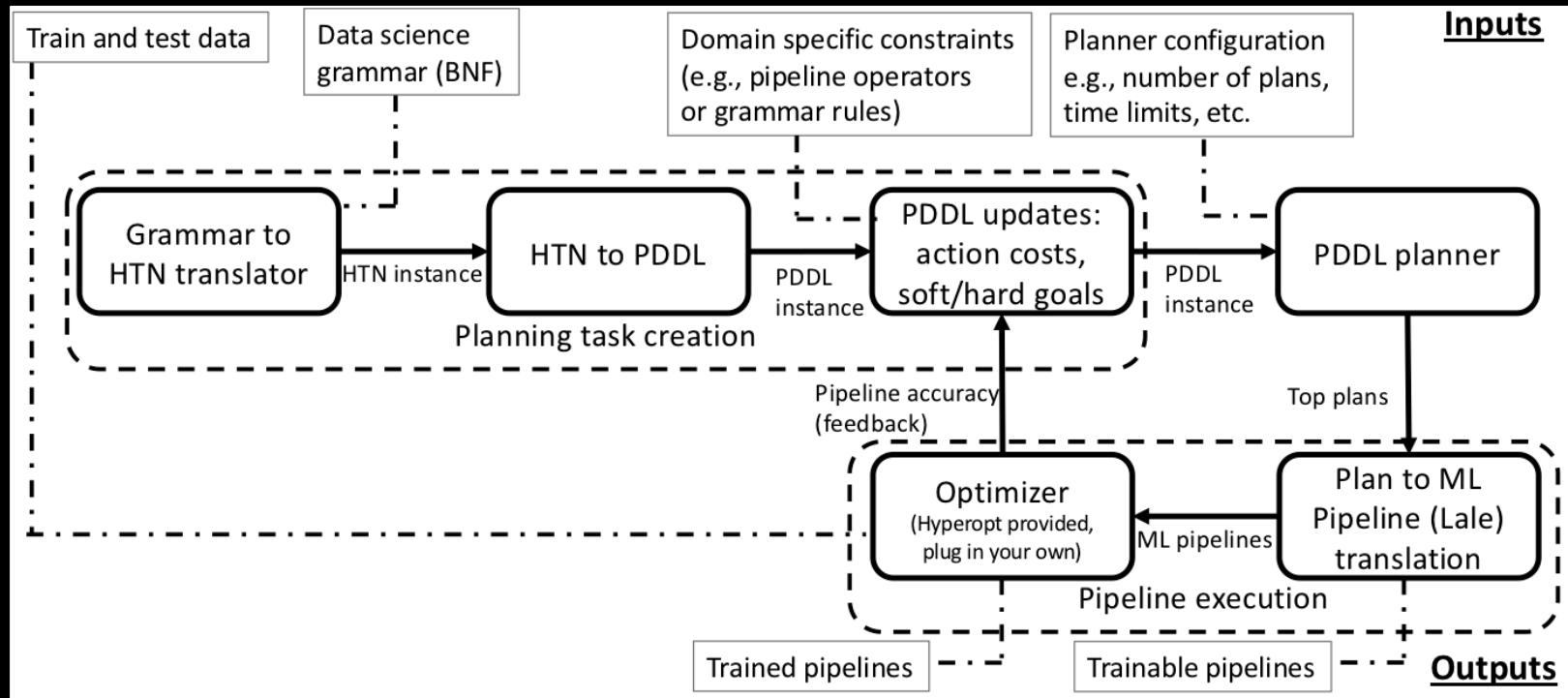
Benefits

- Generation of pipelines of high accuracy automatically
- Reduces the need for human data scientists

Solution

- Use regular grammars to define the possible pipeline compositions
- Translate regular grammars to HTN planning model, and then to classical planning model
- Enrich planning model with user defined constraints
- Use AI Planners to solve the planning model and translate plans to pipelines

Automating ML Pipeline Generation: Solution



User Guided Exploration: Choosing Preferred Symbols

- Non-terminal symbols — rules
- Terminal symbols — specific algorithms and hyper-param values
- Translated to soft goals
- Soft goals compiled away (Keyder & Geffner 2009)

The screenshot shows a user interface element for choosing preferred symbols. At the top right, there is a slider labeled "num_pipeline" with a value of 29. Below it is a dropdown menu titled "Search con...". The dropdown contains a list of machine learning components and hyper-parameters, with several items highlighted in grey, indicating they are selected or recommended. The visible items include:
KNeighborsClassifier
KeepNonNumbers
KeepNumbers
LogisticRegression
MLPClassifier
MinMaxScaler()
NoOp()
Normalizer()
Nystroem
OneHotEncoder
PCA
PolynomialFeatures()
QuadraticDiscriminantAnalysis()
RandomForestClassifier
RobustScaler()
SimpleImputer
StandardScaler()
activation
criterion
dag

At the bottom of the dropdown menu is a button labeled "Get pipelines".

Representing the Knowledge

(Example: PDDL)

```

(define (problem mixed)
  (:domain miconic)
  (:objects p0 p1 p2 p3 f0 f1 f2 f3 f4 f5 f6 f7)

  (:init
    (passenger p0)(passenger p1)(passenger p2)(passenger p3)
    (floor f0)(floor f1)(floor f2)(floor f3)(floor f4)
    (floor f5)(floor f6)(floor f7)

    (above f0 f1)(above f0 f2)(above f0 f3)(above f0 f4)
    (above f0 f5)(above f0 f6)(above f0 f7)(above f1 f2)
    (above f1 f3)(above f1 f4)(above f1 f5)(above f1 f6)
    (above f1 f7)(above f2 f3)(above f2 f4)(above f2 f5)
    (above f2 f6)(above f2 f7)(above f3 f4)(above f3 f5)
    (above f3 f6)(above f3 f7)(above f4 f5)(above f4 f6)
    (above f4 f7)(above f5 f6)(above f5 f7)(above f6 f7)

    (origin p0 f0)(destin p0 f5)(origin p1 f7)(destin p1 f4)
    (origin p2 f0)(destin p2 f7)(origin p3 f1)(destin p3 f6)

    (lift-at f0))

  (:goal (and
    (served p0)(served p1)(served p2)(served p3))))
```

```

(define (domain miconic)
  (:requirements :strips)

  (:predicates
    (origin ?person ?floor)
    (floor ?floor)
    (passenger ?passenger)
    (destin ?person ?floor)
    (above ?floor1 ?floor2)
    (boarded ?person)
    (served ?person)
    (lift-at ?floor))

  (:action board
    :parameters (?f ?p)
    :precondition (and (floor ?f) (passenger ?p)(lift-at ?f) (origin ?p ?f))
    :effect (boarded ?p))

  (:action depart
    :parameters (?f ?p)
    :precondition (and (floor ?f) (passenger ?p) (lift-at ?f) (destin ?p ?f)
                      (boarded ?p))
    :effect (and (not (boarded ?p))
                 (served ?p)))

  (:action up
    :parameters (?f1 ?f2)
    :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1) (above ?f1 ?f2))
    :effect (and (lift-at ?f2) (not (lift-at ?f1)))))

  (:action down
    :parameters (?f1 ?f2)
    :precondition (and (floor ?f1) (floor ?f2) (lift-at ?f1) (above ?f2 ?f1))
    :effect (and (lift-at ?f2) (not (lift-at ?f1)))))
```

(up fo f1)

(up f1 f7)

(board f7 p1)

(down f7 f4)

(depart f4 p1)

(down f4 fo)

(board fo po)

(up fo f5)

(depart f5 po)

(up f5 f6)

(down f6 fo)

(board fo p2)

(up fo f7)

(depart f7 p2)

(down f7 f1)

(board f1 p3)

(up f1 f6)

(depart f6 p3)

Modeling Examples

The screenshot shows a code editor window for a file named `components.fpl.tl`. The code defines a composite pattern with several components:

- PIG component**: A component with annotations like `@type "pig"` and `@title "Top K By Count"`.
- JAQL component**: A component with annotations like `@type "jaql"` and `@title "Count Flows by Destination Host"`.
- SPL component**: A component with annotations like `@type "spl"` and `@title "Live DNS Data"`.
- Platform specific code for the component**: Annotations like `@type "platform"` and `@title "LiveDNS"`.
- Semantic annotations (tags) on the component ports**: Annotations like `@tag "osiris0"` and `@tag "ohh1"`.
- Pattern (composite) with variability points**: Annotations like `@variability "Main (top-level) composite for this pattern"` and `@variability "MyMain(output final)"`.
- Cascade is used by data scientists/analytic**: A small cartoon character icon.

Cascade (Ranganathan et al., 2009)

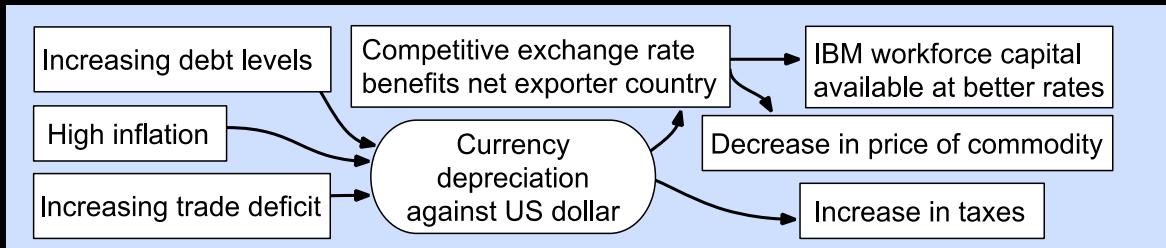
The screenshot shows a state transition graph in the LTS++ tool. The states are labeled with medical conditions and transitions between them:

```

1 default-class bad_state
2 LIFE_CYCLE = (start <good_state> -> ADMITTED)
3 ADMITTED = (admitted {OADMITTED} -> LOWRISK | admitted -> HIGHRISK)
4 LOWRISK = (lowrisk {good_state} OSIRS0 OHH1) -> HIGHRISK
5 HIGHRISK = (highrisk {OSIRS2 OHH3 OHH4 OHRVL OCTSCANDCINEGATIVE}
6           -> PRECOMPLICATION | highrisk -> LOWRISK)
7 PRECOMPLICATION = (precomp {OSIRS2 OHH3 OHRVL} -> SUSPECTEDDCI |
8                     precomp -> SUSPECTEDINFECTION)
9 SUSPECTEDDCI = (suspecteddcii {OSIMDCI OVISUALDCI} -> PREDCI)
10 PREDCI = (predci {OPREDDCI OVISUALDCI OSIMDCI} -> DCI)
11 DCI = (dci {OANGIOGRAMDCIPOSITIVE} -> HIGHRISK |
12        dci -> PRECOMPLICATION | dci {OFLAT} -> ICUDEATH)
13 INFECTION = (infection {OINFECTIONPOSITIVE} -> HIGHRISK |
14                 infection {OANTIBIOTICSADMINISTERED} -> PRECOMPLICATION |
15                 infection {OFLAT} -> ICUDEATH)
16 ICUDEATH = (icudeath {OFLAT})
17 DISCHARGE = (discharge {OPATIENTDISCHARGED} -> discharge)
18 starting start
  
```

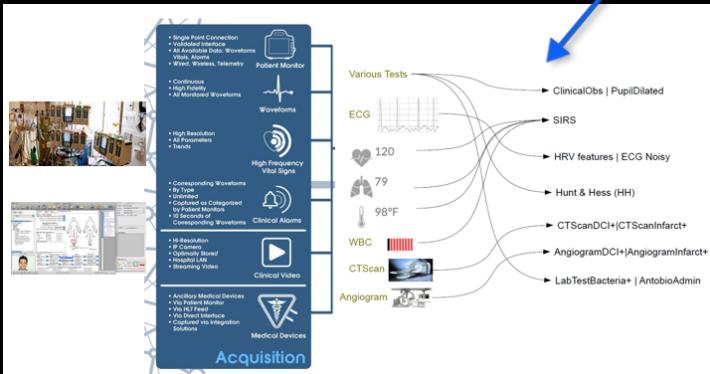
A warning message at the bottom left states: "Line 1:1: The state transition graph contains a cycle with states: dci,predci,suspecteddcii,suspectedinfection,highrisk,lowrisk".

LTS++ (Sohrabi et al., 2015, 2020)

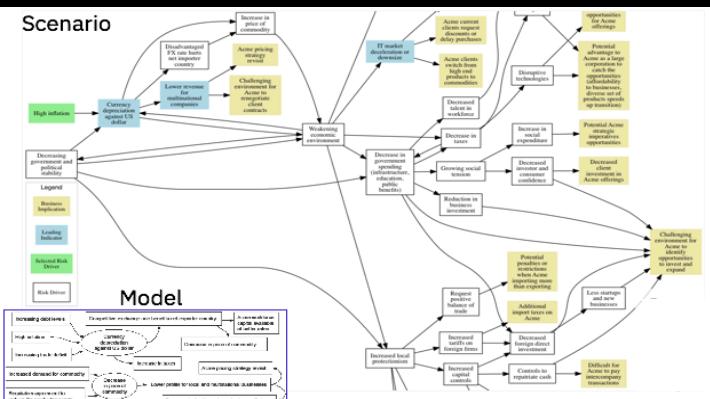


MindMaps
(Sohrabi et al., 2018)

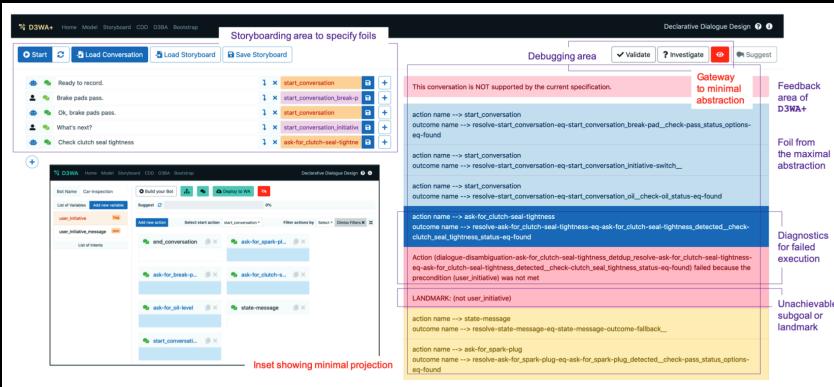
Examples



[Automated large-scale data analysis, ICAPS 2015]



¹³ [Scenario planning for enterprise risk management, AAAI 2018]

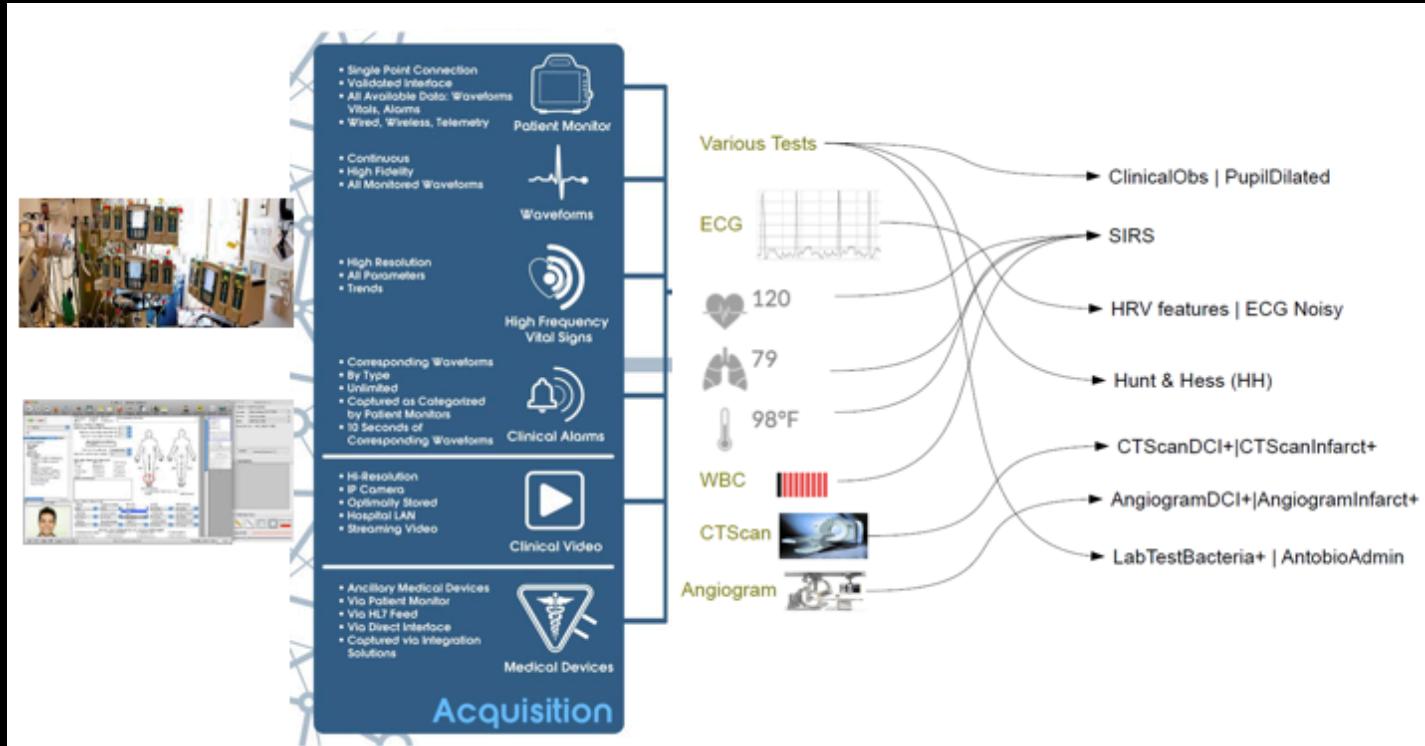


[D3WA+: A Case Study of XAIP in a Model Acquisition Task, ICAPS 2020]

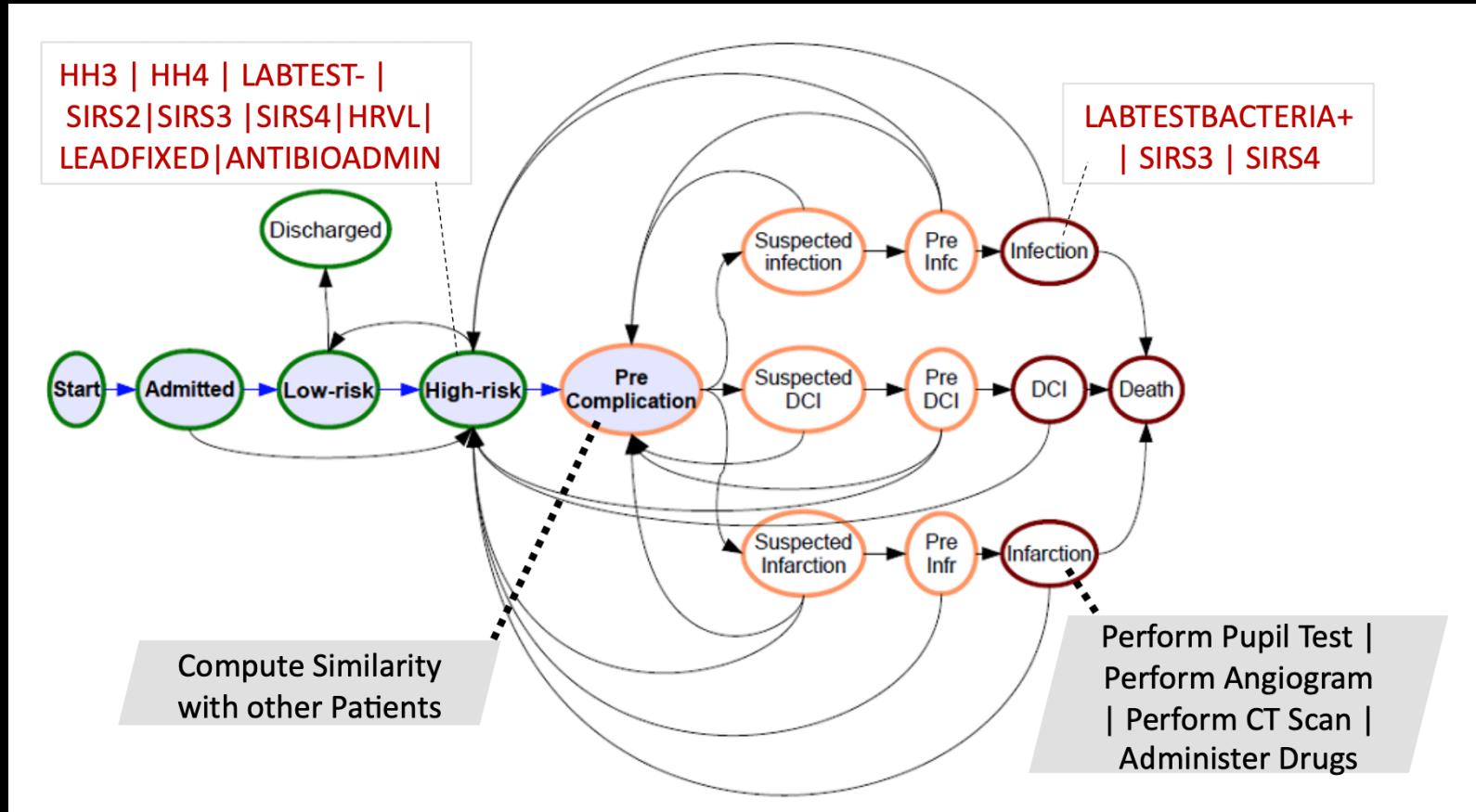


[Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning, ICAPS 2020]

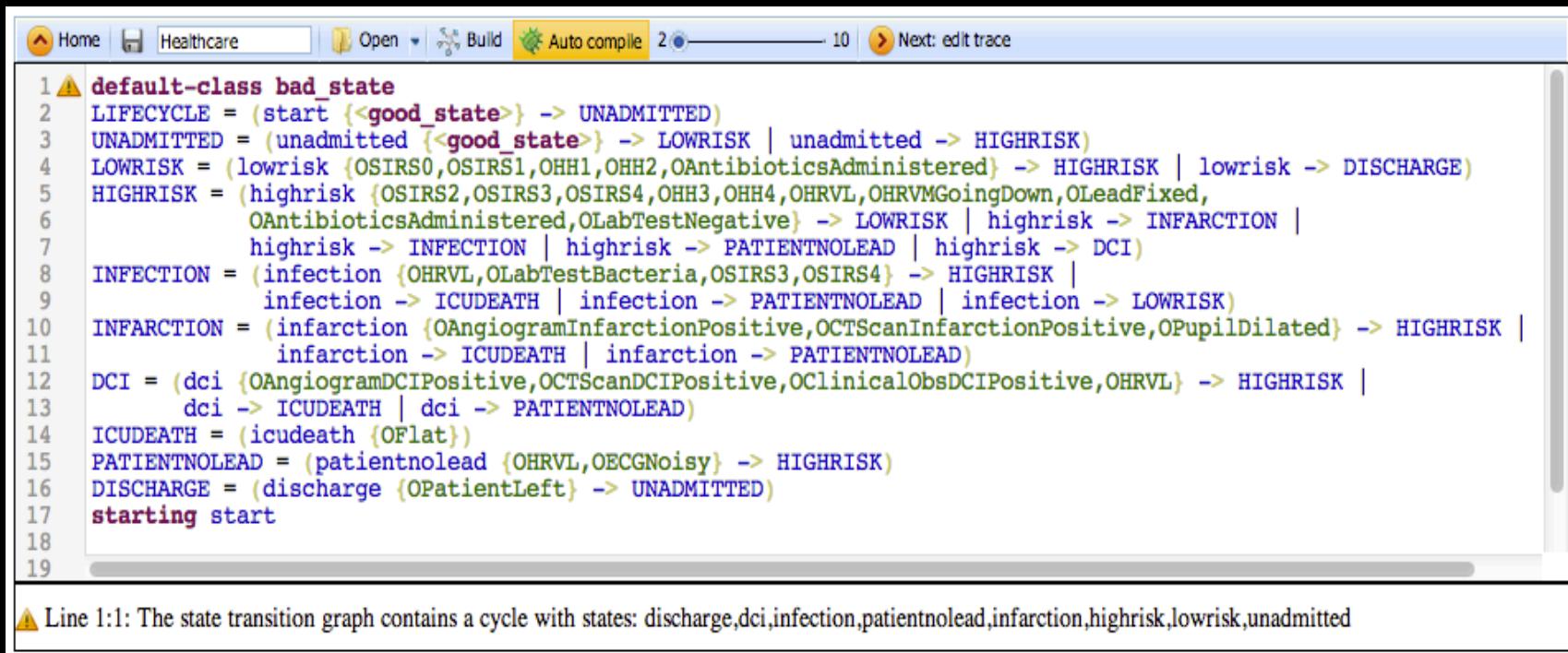
Hypothesis Generation



Patient Complication Detection



LTS++



The screenshot shows the LTS++ software interface with the following details:

- Toolbar:** Home, Healthcare, Open, Build, Auto compile, Next: edit trace.
- Code Editor:** Displays a state transition graph definition in LTS++ syntax. The code includes:
 - States: bad_state, UNADMITTED, LOWRISK, HIGHRISK, INFARCTION, INFECTION, ICUDEATH, PATIENTNOLEAD, DCI.
 - Transitions: LIFECYCLE, UNADMITTED, LOWRISK, HIGHRISK, INFARCTION, INFECTION, ICUDEATH, PATIENTNOLEAD, DCI.
 - Observations: OSIRS0, OSIRS1, OHH1, OHH2, OAntibioticsAdministered, OSIRS2, OSIRS3, OSIRS4, OHH3, OHH4, OHRVL, OHRVMGoingDown, OLeadFixed, OAntibioticsAdministered, OLabTestNegative, OAngiogramInfarctionPositive, OCTScanInfarctionPositive, OPupilDilated, OAngiogramDCIPositive, OCTScanDCIPositive, OClinicalObsDCIPositive, OHRVL.
 - Initial State: starting start.
- Message Bar:** A warning message: "Line 1:1: The state transition graph contains a cycle with states: discharge,dci,infection,patientnolead,infarction,highrisk,lowrisk,unadmitted".

- Derived from an existing language, Labeled Transition System (LTS)
- Can associate an observation with a state, can specify class type

Partial encoding of the sample example

```
(:action explain-observation
:parameters(?x - state ?obs1 - obs ?obs2 - obs ?cat - obs-type)
:precondition (and (is-next-obs ?obs1 ?obs2) (matches ?obs1 ?cat)
(explains ?x ?cat) (at-state ?x) (at-obs ?obs1))
:effect (and (not (at-obs ?obs1)) (at-obs ?obs2) (ready)
(increase (total-cost) 0)))

(:action discard-observation
:parameters(?x - state ?obs1 - obs ?obs2 - obs)
:precondition (and (is-next-obs ?obs1 ?obs2) (at-state ?x) (at-obs ?obs1))
:effect (and (not (at-obs ?obs1)) (at-obs ?obs2) (ready)
(increase (total-cost) 2000)))

(:action state-change
:parameters(?x - state ?y - state ?obs - obs)
:precondition (and (is-next-state ?x ?y) (at-obs ?obs) (at-state ?x) (ready))
:effect (and (not (at-state ?x)) (not (ready)) (entering-state ?y)
(increase (total-cost) 0)))

(:action enter-state-good
:parameters(?y - state ?obs - obs)
:precondition (and (at-obs ?obs) (entering-state ?y) (good-state ?y))
:effect (and (at-state ?y) (not (entering-state ?y))
(increase (total-cost) 1)))

(:action enter-state-bad
:parameters(?y - state ?obs - obs)
:precondition (and (at-obs ?obs) (entering-state ?y) (bad-state ?y))
:effect (and (at-state ?y) (not (entering-state ?y))
(increase (total-cost) 10)))

(:action allow-unobserved
:parameters(?x - state ?obs - obs)
:precondition (and (at-obs ?obs) (at-state ?x))
:effect (and (ready) (increase (total-cost) 1100)))
```

Partial encoding of the sample example

```
(:init
  (at-state admitted) (at-obs o_1) (ready)
```

```
(matches o_1 OHH1) (matches o_2 OSIRSO) (matches o_3 OSIRS2)
```

```
(explains lowrisk OSIRSO) (explains highrisk OSIRS2)
(explains precomp OSIRS2) (explains lowrisk OHH1)
(explains dci OANGIOGRAMDCIPOSITIVE)
(explains highrisk OHRVL) (explains precomp OHRVL)
```

```
(is-next-state admitted highrisk) (is-next-state admitted lowrisk)
(is-next-state lowrisk highrisk) (is-next-state highrisk lowrisk)
(is-next-state highrisk precomp) (is-next-state dci highrisk)
(is-next-state dci icudeath) (is-next-state dci precomp)
```

```
(bad-state dci) (bad-state highrisk) (good-state lowrisk)
```

```
(is-next-obs o_1 o_2) (is-next-obs o_2 o_3) (is-next-obs o_3 o_end))
```

```
(:goal (and (at-obs o_end) (ready)))
```

```
(:action explain-observation
  :parameters (?x - state ?obs1 - obs ?obs2 - obs ?cat - obs-type)
  :precondition (and (is-next-obs ?obs1 ?obs2) (matches ?obs1 ?cat)
    (explains ?x ?cat) (at-state ?x) (at-obs ?obs1))
  :effect (and (not (at-obs ?obs1)) (at-obs ?obs2) (ready)
    (increase (total-cost) 0)))

(:action discard-observation
  :parameters (?x - state ?obs1 - obs ?obs2 - obs)
  :precondition (and (is-next-obs ?obs1 ?obs2) (at-state ?x) (at-obs ?obs1)
    (bs ?obs1)) (at-obs ?obs2) (ready)
    se (total-cost) 2000))

(:action enter-state
  :parameters (?y - state ?obs - obs)
  :precondition (and (next-state ?x ?y) (at-obs ?obs) (at-state ?x) (ready)
    state ?x)) (not (ready)) (entering-state ?y)
  :effect (and (not (ready)) (entering-state ?y)
    ease (total-cost) 0))

(:action leave-state
  :parameters (?y - state ?obs - obs)
  :precondition (and (next-state ?x ?y) (at-obs ?obs) (at-state ?x) (ready)
    state ?x)) (not (entering-state ?y))
  :effect (and (not (entering-state ?y)) (ready)
    leave (total-cost) 0))

(:action good-state
  :parameters (?obs - obs)
  :precondition (and (obs ?obs) (entering-state ?y) (good-state ?y))
  :effect (and (good-state ?y) (not (entering-state ?y))
    (total-cost) 1))

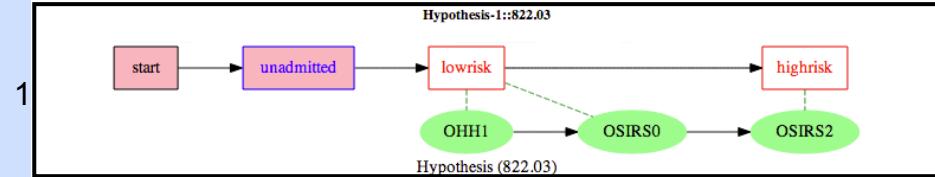
(:action bad-state
  :parameters (?obs - obs)
  :precondition (and (obs ?obs) (entering-state ?y) (bad-state ?y))
  :effect (and (bad-state ?y) (not (entering-state ?y))
    (total-cost) 10))

(:action allow-unobserved
  :parameters (?x - state ?obs - obs)
  :precondition (and (at-obs ?obs) (at-state ?x))
  :effect (and (ready) (increase (total-cost) 1100)))
```

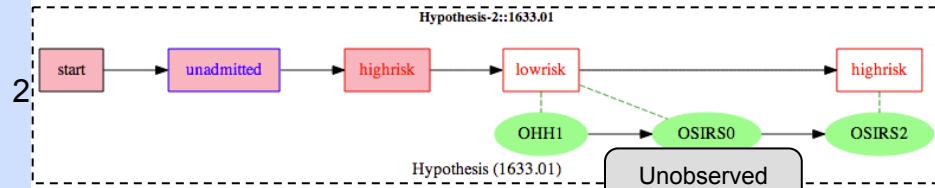
Generated hypotheses for the critical care application

Hypothesis Generation Results

[Home](#) [Edit model](#) [New trace](#) [Detailed graphs](#) [Next page](#)

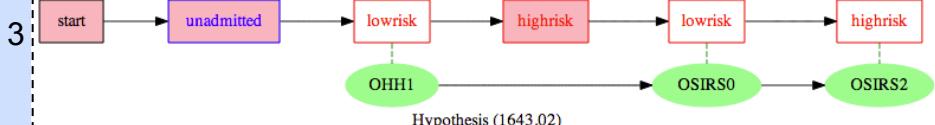
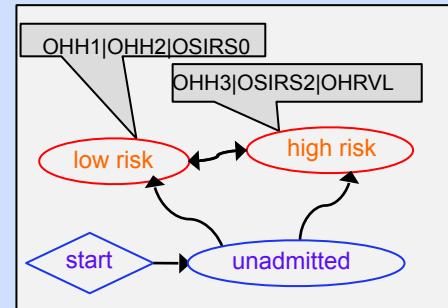


Both OHH1 and SIRS0
are explained by lowrisk

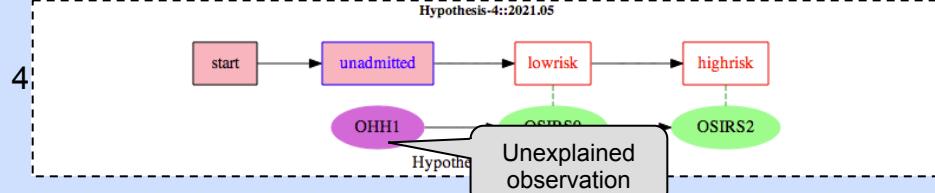


Each hypothesis is shown as
sequence of states matched
to observed event sequence

State Transition Diagram

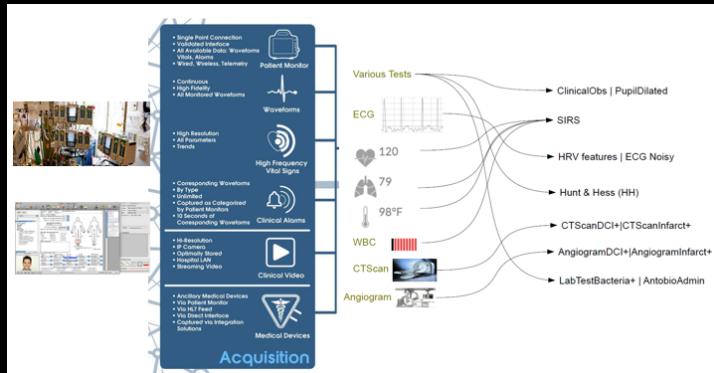


Less plausible
hypothesis

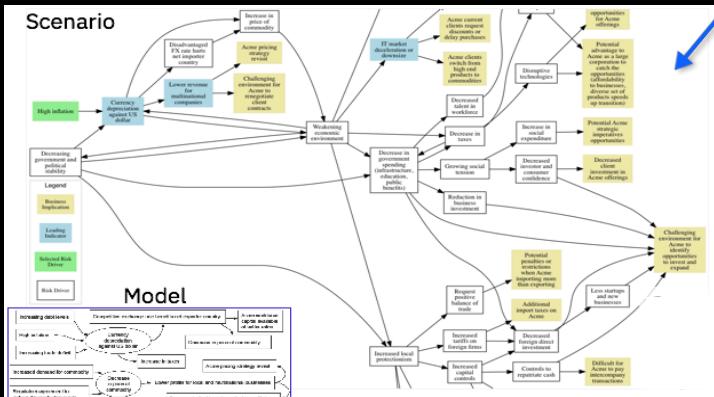


Unexplained
observation

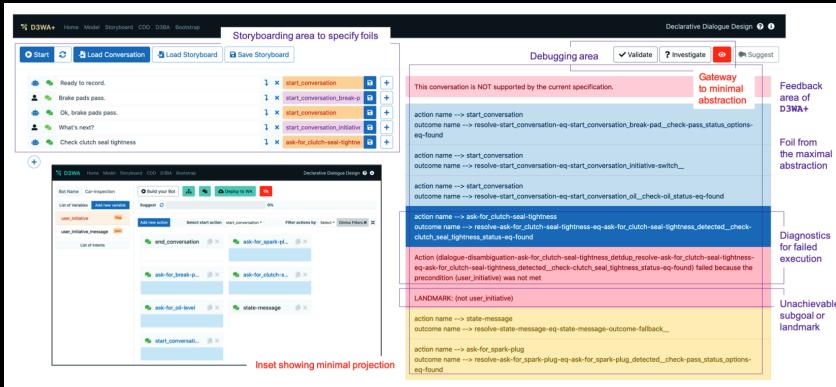
Examples



[Automated large-scale data analysis, ICAPS 2015]



20 [Scenario planning for enterprise risk management, AAAI 2018]



[D3WA+: A Case Study of XAIP in a Model Acquisition Task, ICAPS 2020]

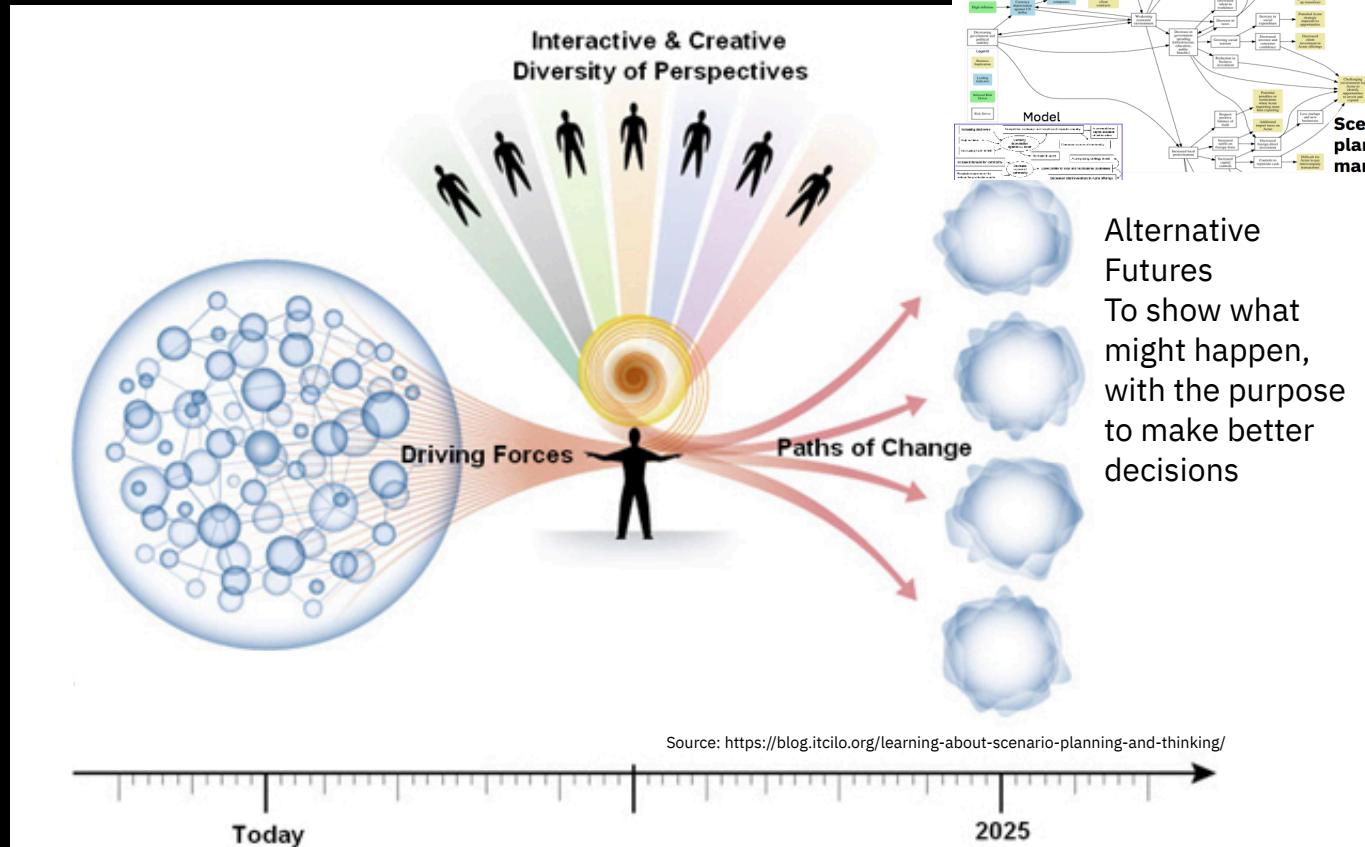


[Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning, ICAPS 2020]

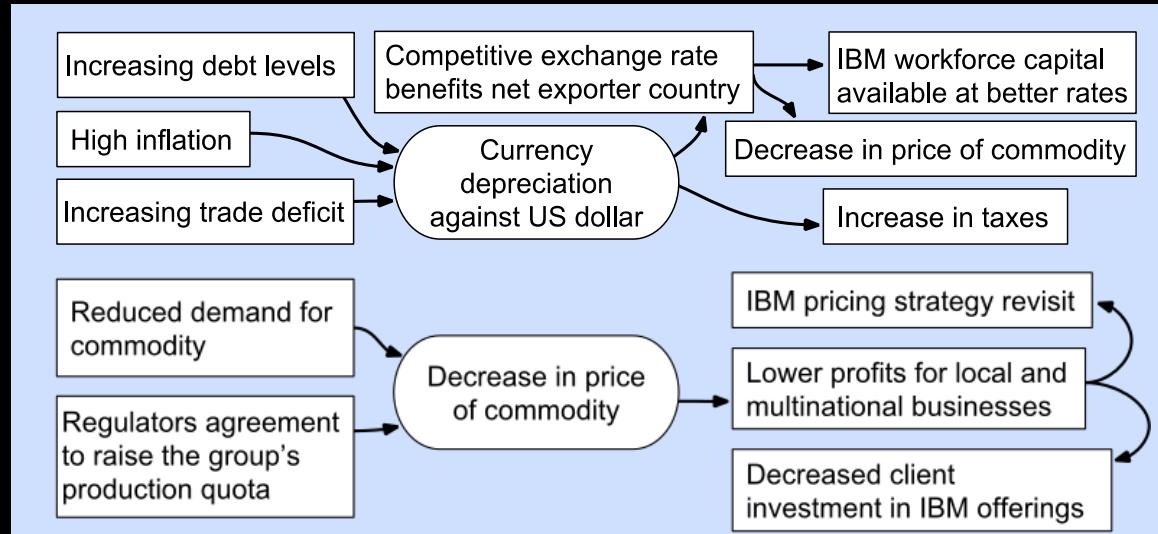
Scenario Planning for Enterprise Risk Management

Main forces:

- Economic environment
- Technology
- Currency
- Social order/unrest
- Corruption
- Natural disaster
- Market disruptors
- Government Stability



Mindmaps (in the context of Scenario Planning Advisor)



Question 1 of 4

How likely are any of the following to lead to currency depreciation against US dollar?

High inflation

Increasing trade deficit

Likelihood

High

Medium

Planning Models for Scenario Planning – How Acquired?

Manual creation of scenario planning models is impracticable.

Automated extraction of causal models (Hassanzadeh et al. 2019, Bhandari et al., 2021)

- use learning-based natural language understanding techniques to identify risk drivers and extract causal pairs
- AI QA for automated reading comprehension
- with causal questions (“what causes X”) no supervision is needed
- authoritative documents that are rich in causal content (10-K, NATO SFA)
- with open-ended questions based on seed sets of risks, discover new candidate risks, bootstrap

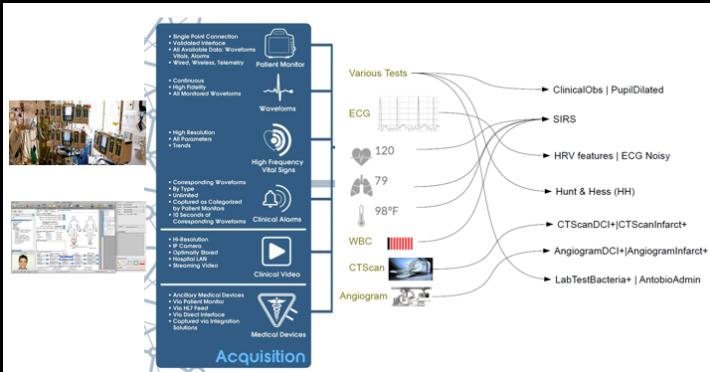
Causal Extraction in Support of Planning Model Generation

CEFAS – Causal Extraction From Authoritative Sources

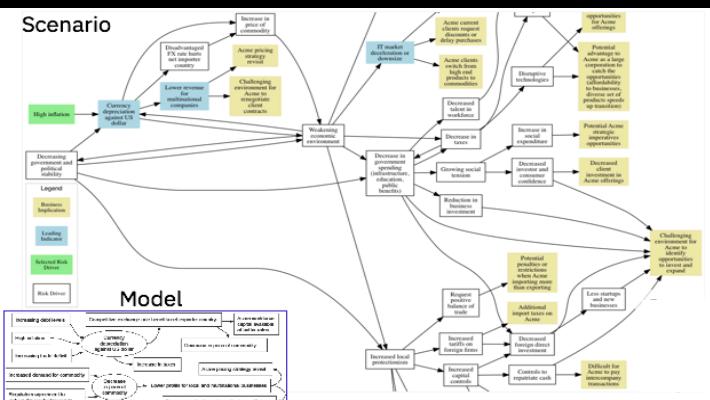
- uses AI Question Answering to answer causal questions about risk drivers...
- relies on relatively small set of authoritative documents, e.g.,
 - SEC Form 10-K – Item 1A Risk Factors
 - NATO Strategic Foresight Analysis doc, 2017
- CEFAS processes all input documents, asking of each paragraph of each document and for each known risk driver:

What causes D?
What does D cause?

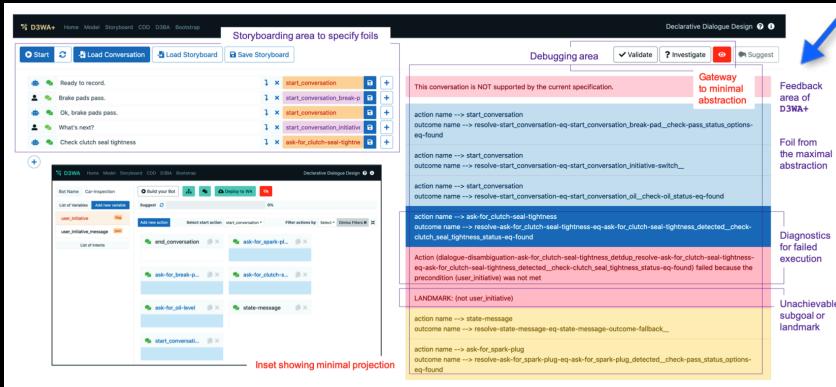
Examples



[Automated large-scale data analysis, ICAPS 2015]



²⁵ [Scenario planning for enterprise risk management, AAAI 2018]



[D3WA+: A Case Study of XAIP in a Model Acquisition Task, ICAPS 2020]



[Exploring Context-Free Languages via Planning: The Case for Automating Machine Learning, ICAPS 2020]

State of Conversational Agents Today

- **Players:** Google's DialogueFlow, IBM's Watson Assistant, Apple's Siri, Amazon's Alexa, Microsoft's Cortana, etc.

Agent: Hello, how can I help you?

User: I'd like a trip to Toronto please.

Agent: Ok, a trip. Where to?

User: Ugh, never mind.

Agent: Hi. What seems to be the issue?

User: All of my emails are gone! Help!!

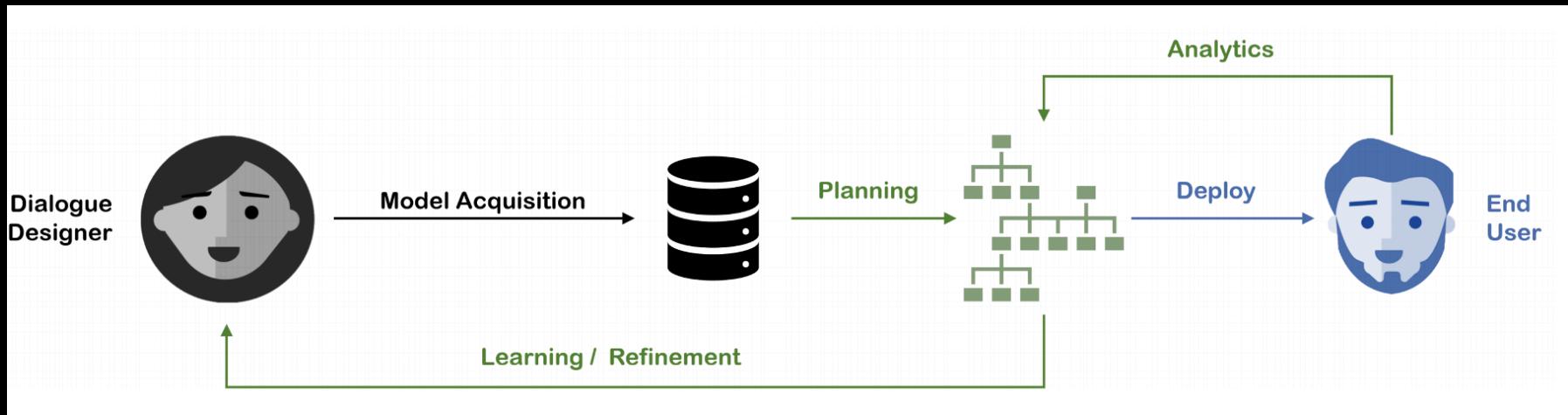
Agent: I understand entirely how frustrating this can be. We know exactly what happened, and will have it resolved in an hour. If you'd like to check the status, your ticket is #123.

User: That's a relief, thanks!

Dialog as Planning

Setting : Multi-turn goal-oriented dialogue

Challenge: How to scale beyond explicit trees



Muise et al. 2019. Planning for Goal-Oriented Dialogue Systems.

D3WA Exponential scale-up of sophistication of composed conversations

Exponential scale-up means

Increased sophistication of dialogue tree with same size of specification

Decreased size of specification for same sophistication of dialogue tree

The screenshot shows the D3WA interface with two main sections. On the left, a list of actions is shown: start_conversation, ask-for_oil-level, ask-for_clutch-s..., ask-for_break-p..., ask-for_spark-pl..., end_conversation, state-message. A red arrow points from this list to a larger graph below. The graph is highly complex, with many nodes and numerous connections between them, representing the compiled dialogue logic. A large black arrow points downwards from the list to the graph. To the right of the graph, there is a legend: a speech bubble icon labeled "speech actions", a cloud icon labeled "API calls", and a lightbulb icon labeled "logical inferences". The title "Abstraction for skills" is centered above the graph.

7 actions automatically compiled into the graph

Compose



The screenshot shows the D3WA interface with several panels. At the top, a purple box says "Beta features" and "Build • Visualize • Chat • Deploy". Below it, a "Variables (collapsed)" section lists variables like spark_plug_status, break_pad_status, user_initiative_message, clutch_seal_tightness_status, pars_status_options, oil_status, user_initiative, and message. A red arrow points from the "Actions (collapsed)" section to the "expanded" section, which shows a sequence of actions: state-message, start_conversation, ask-for_oil-level, ask-for_break-pad, ask-for_spark-pl..., and end_conversation. Another red arrow points from the "NEEDs or preconditions" section to the "Compose" area. The bottom section is titled "Modeling Enhancements" and "Non-deterministic OUTCOMES", showing various configuration options for outcomes and updates. A sidebar on the right is titled "In-house chat" and contains a list of messages related to the dialogue process.

Related Work

- ItSimple (Vaquero et al., 2007)
- PDDL Editor <http://editor.planning.domains/>
- PDDL in Python <https://github.com/IBM/pddl-in-python>
- Planimation <https://github.com/AI-Planning/planimation>
- PDDL plugin for VSCode
- Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) @ ICAPS
- PDDL book titled “*An Introduction to the Planning Domain Definition Language*”
(Haslum et al., 2020)
- Extensive literature on planning and learning (Aineto et al., AIJ 2019)

Modeling Summary

- Modeling is an important aspect of use and adaptation of AI Planning
- Investing in transformation/compilation techniques pays off
- Learning consumable planning models or even enhancing a planning model automatically is important