PSL: An Expert System to Evaluate Degree Plans

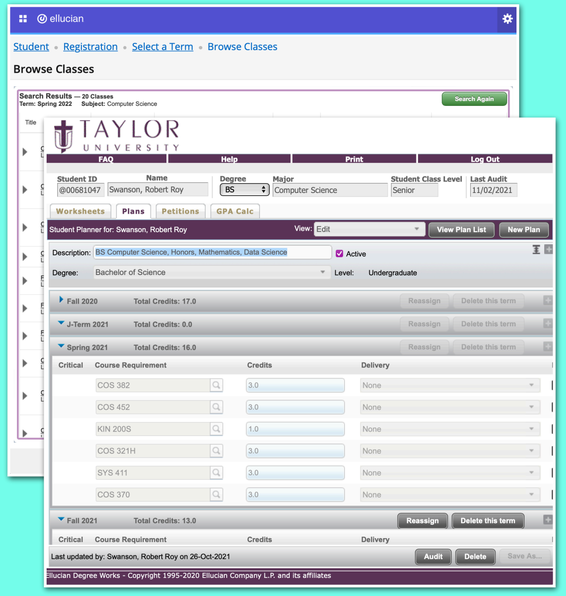
Robert Swanson

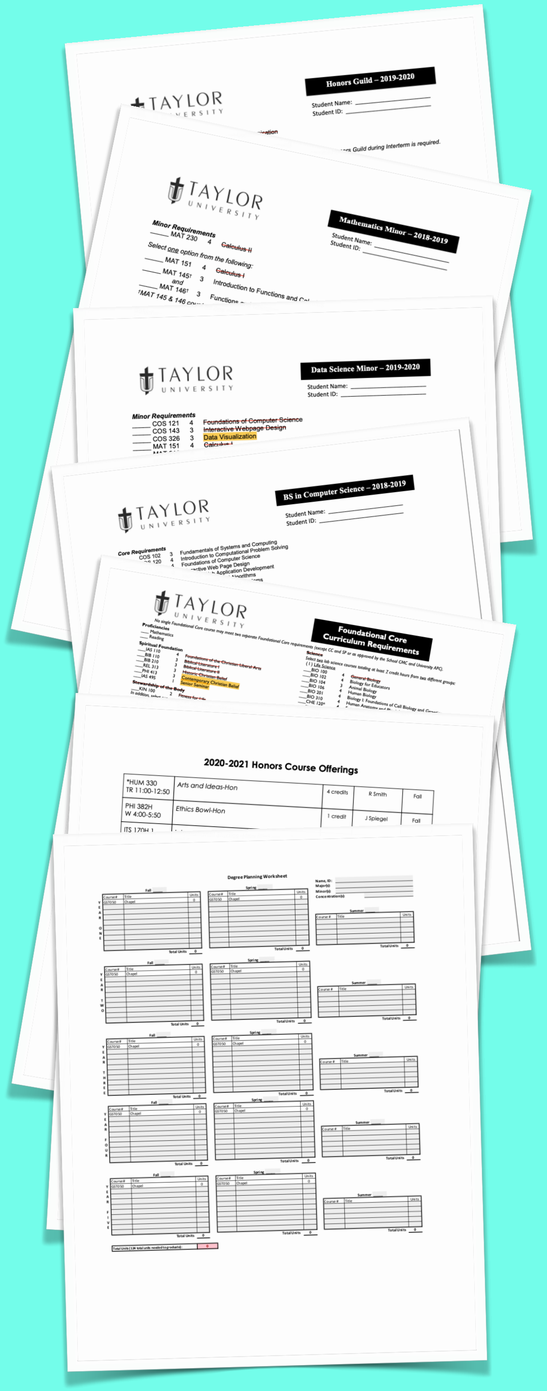
Taylor University-Research I&II

Context

The Challenge of Degree Planning

- Information overload
  - Yet hard to find all the info you need
  - Eg. When courses will be offered
- Inflexible
  - Boolean audit
  - Can’t add your own preferences
- Hard Optimization Problem
  - Advisors must help multiple students
  - Hard for each student to consider degree possibilities



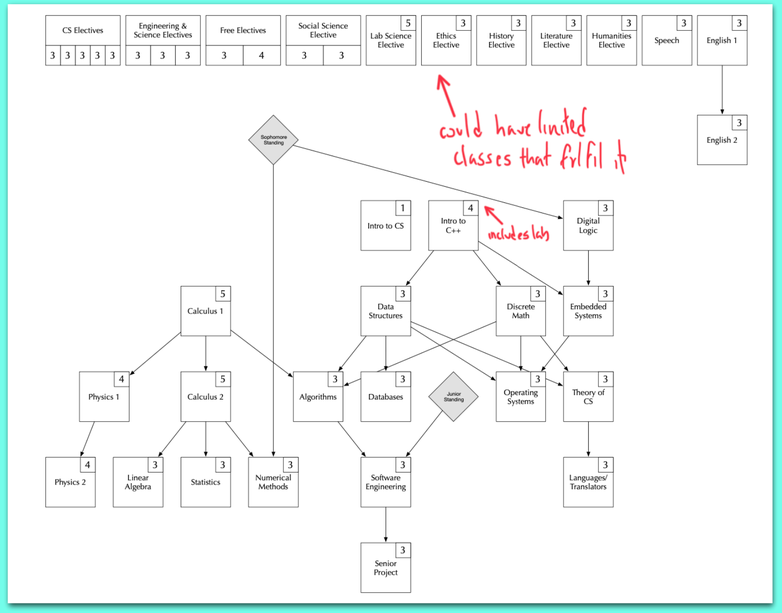


Objective: Automated Planning

- Benefits
  - Convenience for students and advisors
  - Saving time and financial cost of bad scheduling
  - Answer “what if” questions quickly
- Features
  - Expression of requirements and preferences (the specification) for a degree plan
  - Given a degree plan, evaluate how well it meets the specification
  - Generate one or more plans that are optimized to the specification

Existing Work

- Genetic Algorithm
  - Encodes a degree plan into a chromosome
  - Defines a fitness function based on a limited set of hard coded preferences
- Percepolis
  - Defines a directed graph expressing dependencies between requirements
  - Implements an algorithm to choose and schedule classes to semesters
- Tarot
  - Prerequisites, requirements, and course offering times are all specified as prolog facts
  - Uses prolog's backtracking engine to generate a plan that fulfills user-specified constraints



2018 Fall	cinf401	csci311	csci321	math142
2019 Spring	csci331	csci431	gen. ed.	gen. ed.
2019 Fall	csci498	phys141	gen. ed.	gen. ed.
2020 Spring	csci301	csci499	phys142	gen. ed.

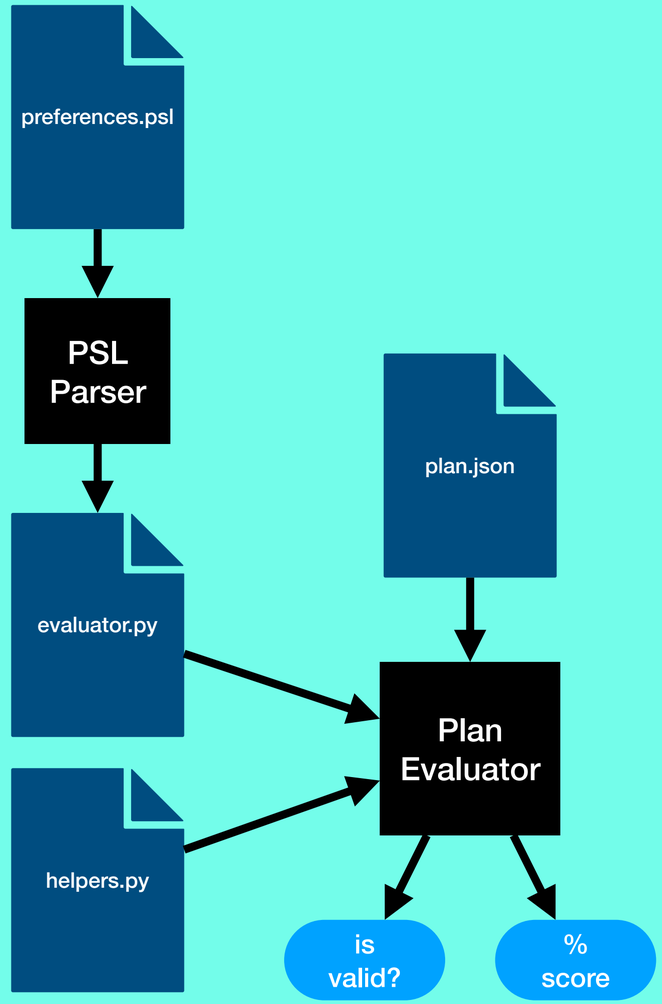
Game Plan

- Improve on existing systems
  - They made considerable assumptions about how students want and need their plans optimized
  - Most presented no explanation for the result
- Develop a preference specification language
  - Emphasis on expressiveness over performance
  - Distinguish between requirements (which invalidate a plan) and preferences
- Use the PSL system to generate an optimized plan
  - Didn’t get this far

PSL Prototype

System Design

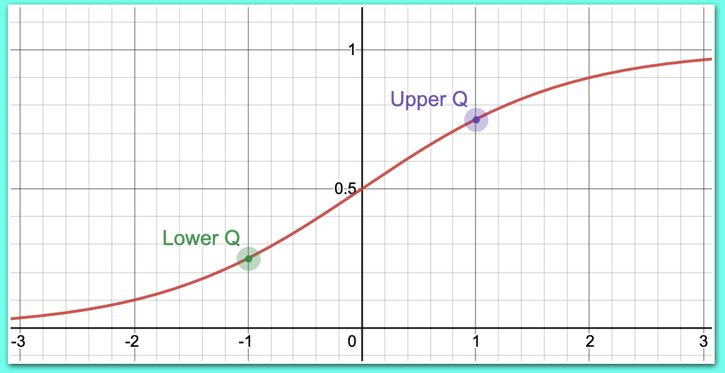
- Informal survey to collect kinds of student preferences
  - Found large diversity in datapoints
  - Decided to limit to factors that a registration system would know
- Organized preferences into a grammar
  - Basic format: `<quantifier> <value> <feature>`
  - Eg.: `at least 15 credits`
- Designed Python/Java Prototype
  - Java PSL parser takes in a pal file and produces a python file
  - This python file combined with a helper file creates the plan evaluator
  - The plan evaluator takes a plan file and produces a boolean and percent score



Preference Scoring

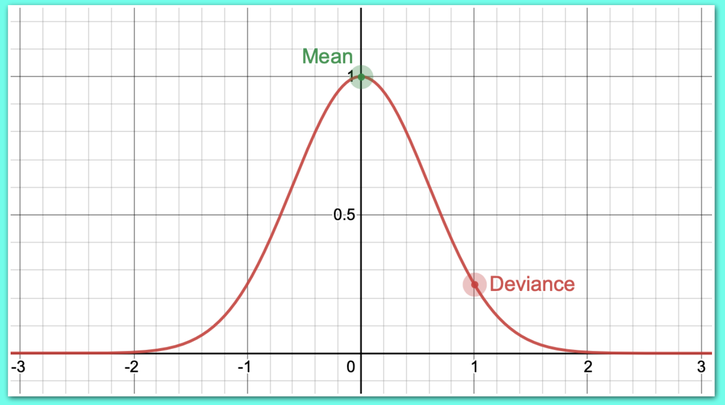
- Need a way to scale preference values
  - Normalizes a domain into a value between 0% and 100%
  - Which function depends on quantifier
  - x depends on the plan feature
  - Function parameters depend on the kind of feature
- Sigmoid Function
  - Optimizes to more or less without limit (eg. `at least 15 credits`)
  - Parameters set lower and higher quadrant, and can be flipped
- Optimization Function
  - Optimizes towards a central value (eg. `15 credits`)
  - Parameters set optimal (mean) value and deviance

$$s(x) = \frac{1}{1 + 9^{-x}} \qquad n_1(x, l, u) = \frac{x - l}{u - l}$$



$$y = s(n_1(x, -1, 1))$$

$$o(x) = (\frac{1}{4})^{x^2} \qquad n_2(x, m, d) = \frac{x - m}{d}$$



$$y = o(n_2(x, -1, 1))$$

Prototype Weaknesses

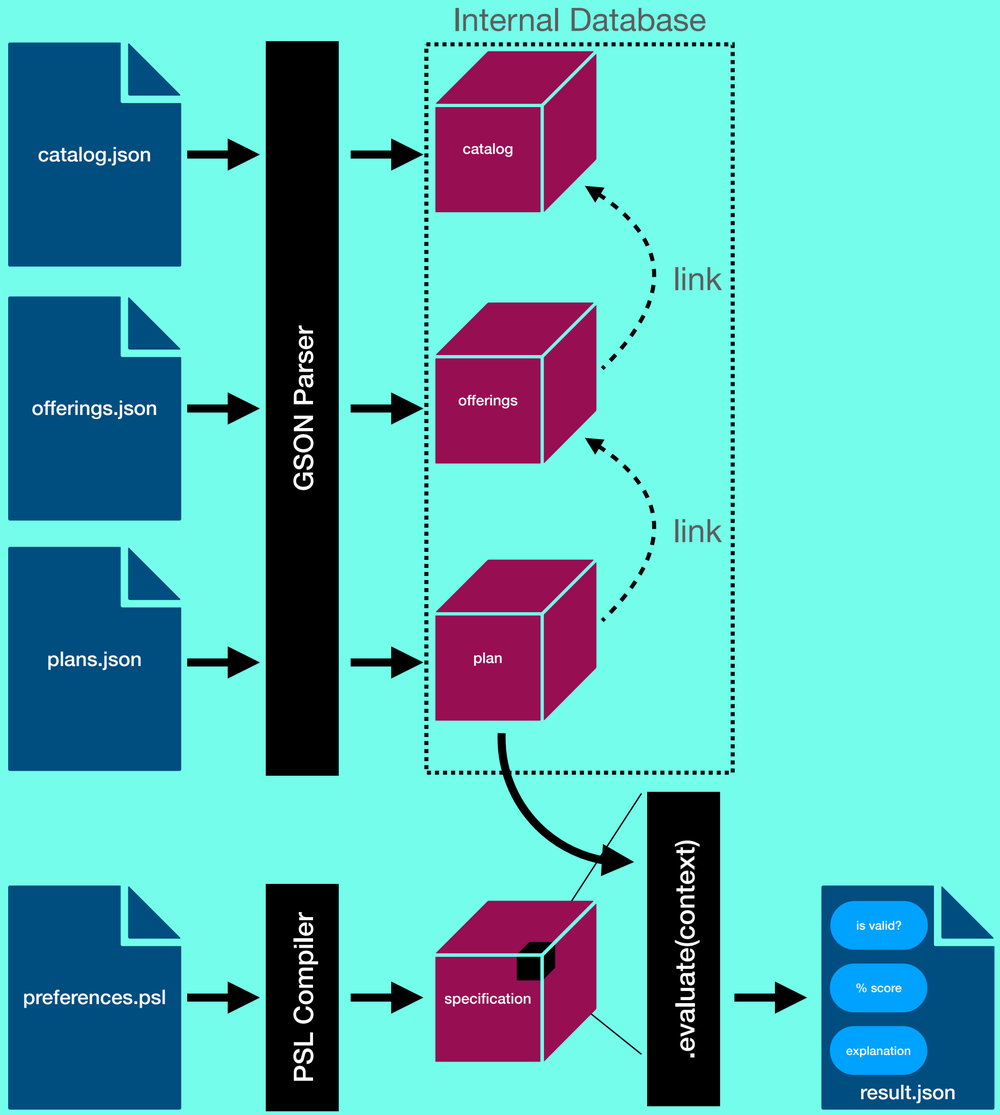
- Split Java/Python implementation
  - Requires new python interpreter process for each evaluation
  - Requires double implementation of evaluation engine
- Grammar limitations
  - Individual definition for each preference

PSL

The Real Deal

System Design

- Inputs
  - Catalog: general course info (eg. Course name)
  - Offerings: term-specific info (eg. Offering time)
  - Plan: terms and course offerings within each to express a degree plan
  - Preferences: the student's requirements and preferences
- File Parsing
  - GSON parses json into java objects
  - PSL Compiler parses pal into a `Specification` object
- Plan Evaluation
  - `Specification.evaluate()` takes in the plan and produces a result containing the validity, score, and explanation



PSL Grammar

PSL Grammar: Specification

A PSL file lists out a number of specifications:

- **Requirement Specification**: represent a constraint that must be fulfilled for a valid plan
- **Preference Specification**: add a priority and represent constraints that impact the score of the plan
- **Specification List**: contains a curly brace surrounded list of specifications
- **Conditional Specification**: defines a set of condition-specification pairs, where only specifications of fulfilled conditions will be evaluated
- **Contextual Specification**: defines a context level and a condition that will serve as a context filter for the contained specification

Specification Segment of the PSL ANTLR Grammar

```
start: (block)+ EOF;
block: NAME priorityList? '{' specification+ '}';

specification:
    requirementSpecification |
    preferenceSpecification |
    specificationList |
    conditionalSpecification |
    contextualSpecification;

requirementSpecification: REQUIRE NOT? requirableConstraint DOT;
preferenceSpecification: PREFER NAME? NOT? constraint DOT;
specificationList: ('{' specification* '}');
conditionalSpecification: IF condition THEN specification (OTHERWISE_IF condition THEN
↪ specification)* (OTHERWISE specification)?;
contextualSpecification: FOR (contextLevel WHERE condition | termYearList | weekdayList)
↪ specification;
```

Example psl File

```
student_preferences (moderately=2.0, strongly=10.0) {
    require plan starting in fall 2018.
    require plan ending on or before spring 2022.

    prefer strongly starting at or after 9:00 AM.
    prefer strongly more courses.
    prefer strongly taking course "COS-121".

    if taking course "COS-120" then {
        prefer moderately taking course "COS-120" before fall 2019.
    } otherwise if taking course "SYS-120" then {
        prefer moderately taking course "SYS-120" before fall 2019.
    }

    for terms where less than 16 credits {
        prefer not meeting at 12:00 PM - 12:50 PM.

        for days where less than 120 meeting minutes {
            prefer ending before 1:00 PM.
        }
    }

    for days where (at least 2 courses or not meeting at 12:00 PM - 12:50 PM) {
        prefer meeting at 11:00 AM - 11:50 AM.
    }

    for thursdays prefer strongly not meeting at 2:00 PM - 4:00 PM.
}
```

PSL Grammar: Constraint

Constraints specify a restriction on the degree plan and come in two forms:

- **Requirable Constraints**: specifies a restriction that supports boolean evaluation (for requirements)
  - **Quantifiers (=>≥<≤):** Specifies an evaluator for the left side and a constant value for the right side
  - **Boolean Constraint:** Specifies an boolean evaluator
- **Constraints**: (includes all requirable constraints) specifies a restriction that supports numeric evaluation (for preferences)
  - **More/Less Constraints:** Specifies only an evaluator that should be maximized or minimized

Constraint Segment of the PSL ANTLR Grammar

```
requirableConstraint:
    equalConstraint |
    greaterThanConstraint |
    greaterThanOrEqualConstraint |
    lessThanConstraint |
    lessThanOrEqualConstraint |
    booleanConstraint;
equalConstraint: (INT numericEvaluator) | (timeEvaluators AT time) | (termYearEvaluators IN
↪ termYear);
booleanConstraint: booleanEvaluators;
constraint: requirableConstraint | moreConstraint | lessConstraint;
moreConstraint: MORE_ numericEvaluator | timeEvaluators LATER | termYearEvaluators LATER;
```

Example psl File

```
student_preferences (moderately=2.0, strongly=10.0) {
    require plan starting in fall 2018.
    require plan ending on or before spring 2022.

    prefer strongly starting at or after 9:00 AM.
    prefer strongly more courses.
    prefer strongly taking course "COS-121".

    if taking course "COS-120" then {
        prefer moderately taking course "COS-120" before fall 2019.
    } otherwise if taking course "SYS-120" then {
        prefer moderately taking course "SYS-120" before fall 2019.
    }

    for terms where less than 16 credits {
        prefer not meeting at 12:00 PM - 12:50 PM.

        for days where less than 120 meeting minutes {
            prefer ending before 1:00 PM.
        }
    }

    for days where (at least 2 courses or not meeting at 12:00 PM - 12:50 PM) {
        prefer meeting at 11:00 AM - 11:50 AM.
    }

    for thursdays prefer strongly not meeting at 2:00 PM - 4:00 PM.
}
```

PSL Grammar: Evaluator

Evaluators define a portion of the degree plan to be constrained

They can return 4 different value types:

1. Numeric Evaluators
2. Term Year Evaluators
3. Time Evaluators
4. Boolean Evaluators

Evaluator Segment of the PSL ANTLR Grammar

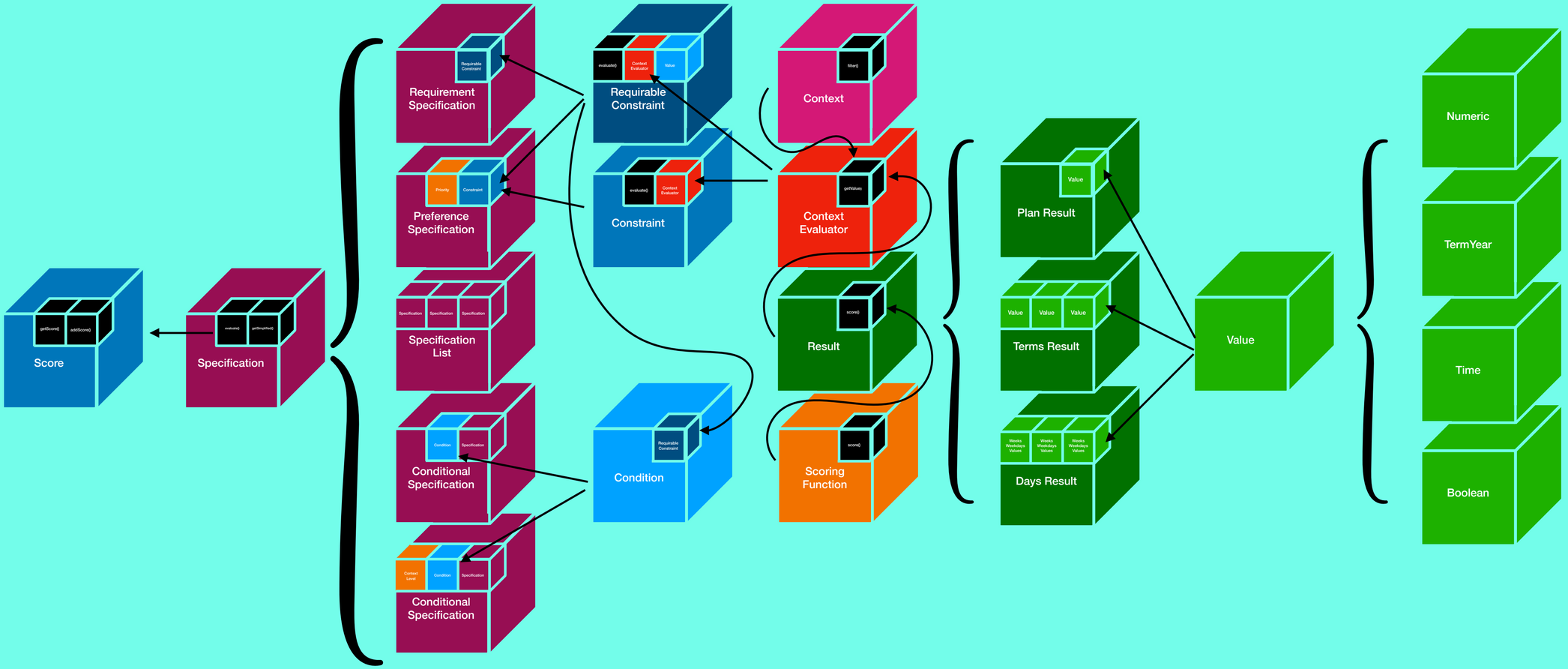
```
numericEvaluator:
    totalCredits | totalCreditsFromSet | upperDivisionCredits | totalCourses |
    totalCoursesFromSet | upperDivisionCourses | meetingMinutes |
    numCoursesWithProfessor | numTimeBlocks | termsInPlan ;
termYearEvaluators: courseTermYear | planStart | planEnd;
timeEvaluators: dayStarting | dayEnding | courseStart | courseEnd;
booleanEvaluators: meetingAtTimeRange | courseBeforeCourse | coursesInSameTerm | termExists;
```

Example psl File

```
student_preferences (moderately=2.0, strongly=10.0) {
    require plan starting in fall 2018.
    require plan ending on or before spring 2022.

    prefer strongly starting at or after 9:00 AM.
    prefer strongly more courses.
    prefer strongly taking course "COS-121".

    if taking course "COS-120" then {
        prefer moderately taking course "COS-120" before fall 2019.
    } otherwise if taking course "SYS-120" then {
        prefer moderately taking course "SYS-120" before fall 2019.
    }

    for terms where less than 16 credits {
        prefer not meeting at 12:00 PM - 12:50 PM.

        for days where less than 120 meeting minutes {
            prefer ending before 1:00 PM.
        }
    }

    for days where (at least 2 courses or not meeting at 12:00 PM - 12:50 PM) {
        prefer meeting at 11:00 AM - 11:50 AM.
    }

    for thursdays prefer strongly not meeting at 2:00 PM - 4:00 PM.
}
```

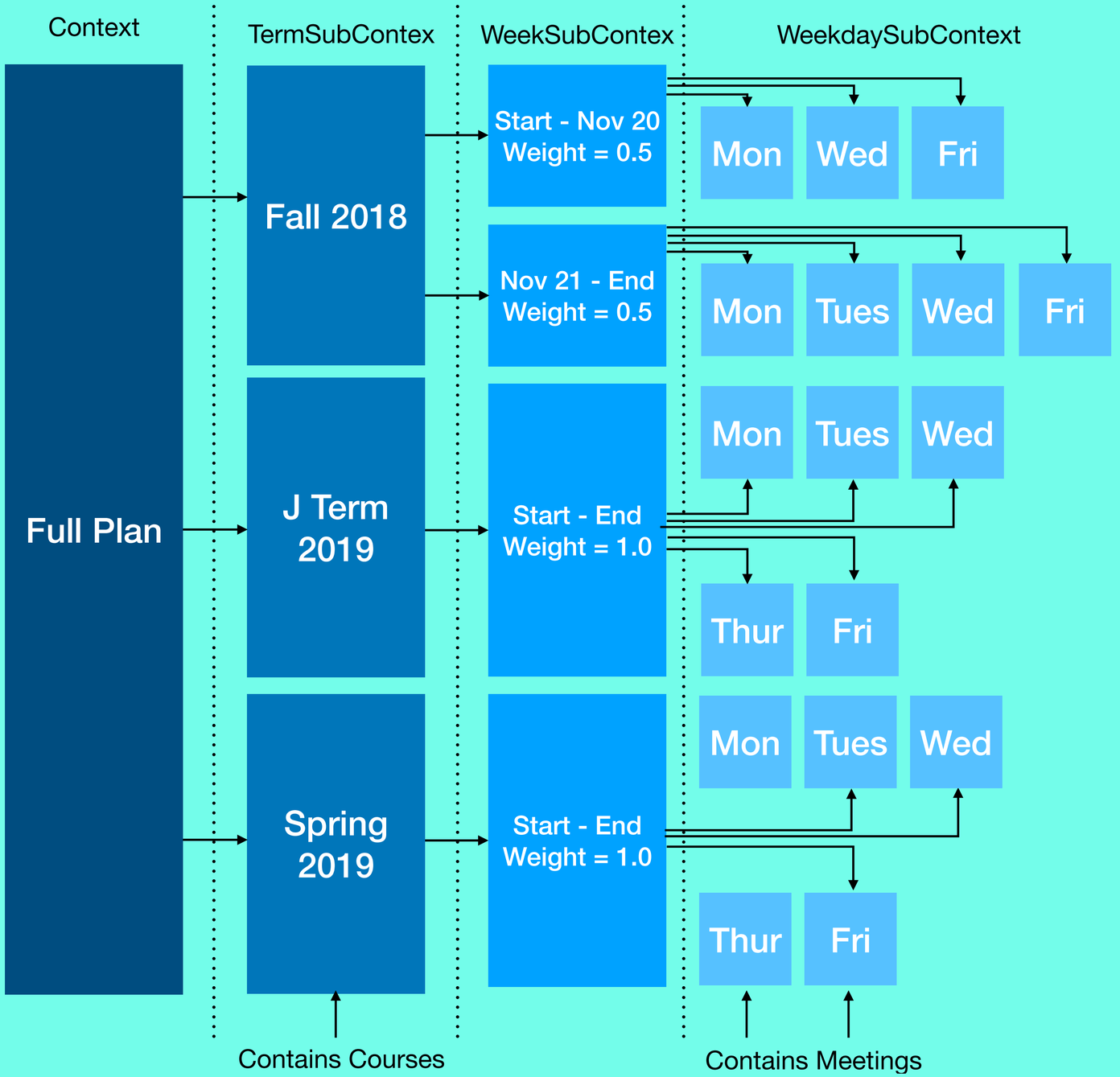
PSL Evaluation Engine

Engine Design



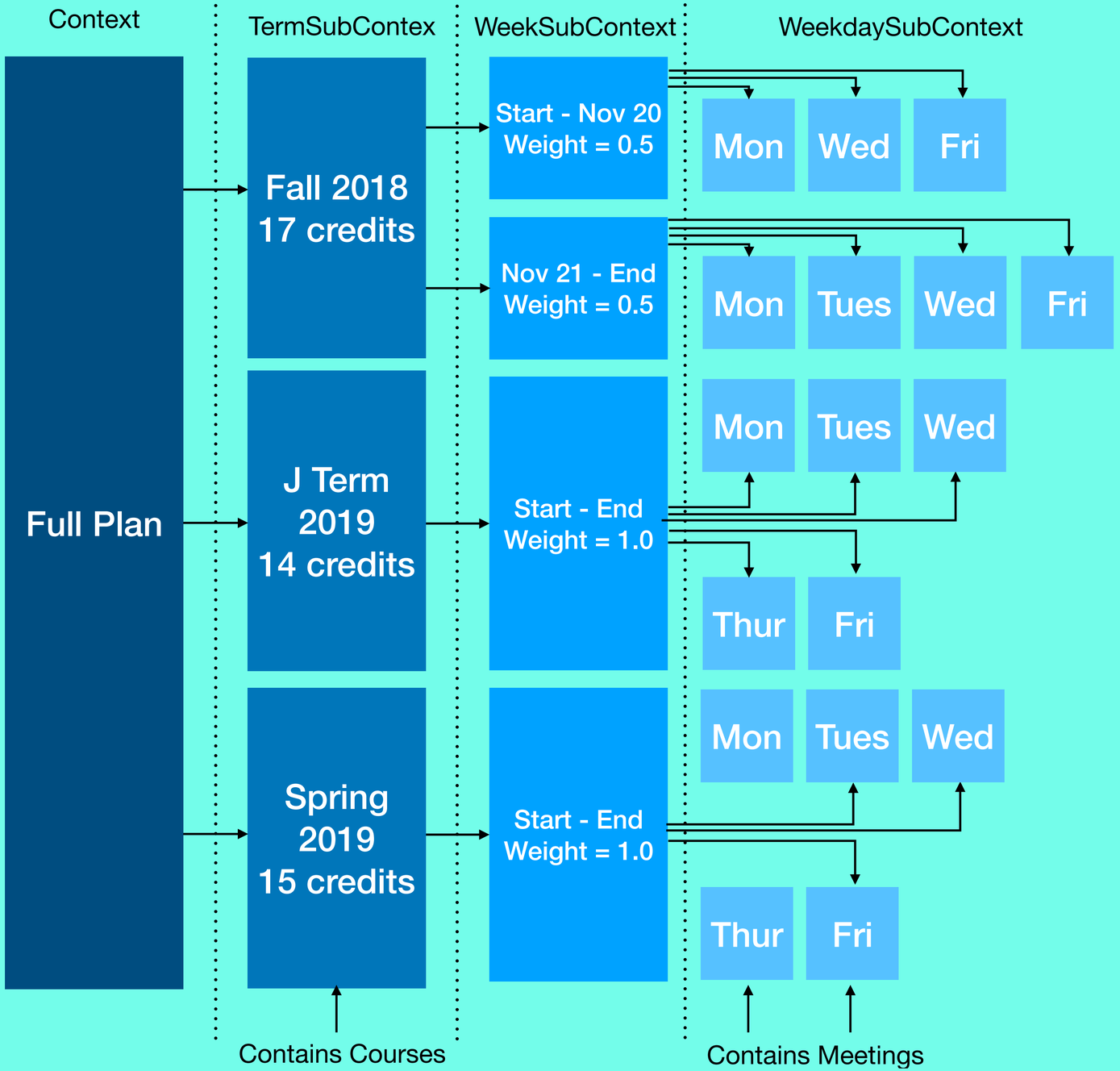
Context

- The Context class serves two purposes
    1. Container that can focus on certain time frames
    2. Maintain a `ContextLevel` that specifies what kind of `Result` that `ContextEvaluators` should return
- 4 context levels:
    1. **FullPlan**: Top level containing everything
    2. **TermSubContext**: second level listing course offerings
    3. **WeekSubContext**: third level to divide terms that have classes that don't run for the full term (not exposed to PSL user)
    4. **WeekdaySubContext**: fourth level listing meetings in a weekday



Context

- The context can be filtered by passing in a condition and new context level (which must be smaller or equal to the current level)
- The condition is applied to each context at the new level
- Eg. `for each term where at least 15 credits`
- Eg. `for each day starting after 9:00 AM` (lets say only Fridays starts after 9:00 AM)



Explanation System

- Every component of the evaluation engine can produce an explanation for its last result at any time
- Useful for debugging the system and PSL files
- Can be used with a GUI to visualize a specification and its evaluation

Explanation.json

Root	Object	
planContext	Object	
specification	Object	
specifications	Array	
Item[0]	Object	
constraint	Object	
evaluator	Object	
result	String	fall 2018
description	String	first term
description	String	first term equals fall 2018
score	String	valid
description	String	require first term equals fall 2018
Item[1]	Object	
Item[2]	Object	
Item[3]	Object	
Item[4]	Object	
Item[5]	Object	
Item[6]	Object	
condition	Object	
context	Object	
specification	Object	
score	String	valid (89%)
description	String	for terms where (credits less than 16.00)
Item[7]	Object	
Item[8]	Object	
score	String	valid (61%)
description	String	Specification List
score	String	valid (61%)
description	String	test.psl

Now What?

Future Work: Developing PSL

- Expand supported preferences by adding more context evaluators
- Robust debugging
- Find better way to tune scoring functions to context evaluators and context
  - Maybe extract to configuration file
  - Maybe calculate such values rather than hard-code them

Future Work: Plan Generation

- Complete the game plan by using PSL to generate optimized plans
- Start with brute force solution: GA
  - Use the PSL Specification object as the fitness function
  - Determine a manner to encode the plan in a chromosome
- Investigate constraint-based searching

Conclusion

- The PSL system demonstrates how student preferences for their degree plan can be expressed as a series of specifications
- Demonstrates how a fairly straightforward syntax can express complex preferences
- Because the system is well-implemented, it demonstrates how new kinds of preferences can easily be implemented by adding evaluators and constraints
- Demonstrates how a plan’s adherence to a set of preferences can be mathematically defined
- The PSL system would be far more useful if paired with a degree plan generation system

Questions?

If I left time for that