VRIJE
UNIVERSITEIT
BRUSSEL

MUHAI RECIPE EXECUTION BENCHMARK

A Benchmark for Recipe Understanding in Autonomous Agents

Robin De Haes

January 10, 2023

Science and Bio-Engineering Sciences

Contents

Chapter 1

Introduction

This benchmark for recipe understanding aims to support progressing the domain of natural language understanding by providing a setting in which performance can be measured on the everyday human activity of cooking. Showing deep understanding of such an activity requires both linguistic and extralinguistic skills, including reasoning with general domain knowledge. For this goal, the benchmark provides a number of recipes written in natural (human) English that should be converted to a procedural semantic network of cooking operations from MUHAI Cooking Language which could then be interpreted and executed by autonomous agents. To further support transferable performance measures, an evaluation suite with a symbolic simulator and custom metrics is included in the benchmark.

First and foremost, chapter 2 explains how to actually run and use the benchmark's evaluation tools. Chapter 3 shortly introduces MUHAI Cooking Language and gives an overview of all the cooking operations that are available in it. Furthermore, it elucidates their interpretation and use within the benchmark's simulated environment. In chapter 4 an overview is given of the initial state of the kitchen from which simulation is started. Knowing which ingredients and tools are actually available can be important information required for correct recipe understanding. Chapter 5 delves further into the ideas behind the gold standard solutions that are provided with the benchmark and why they are considered the ideal recipe execution sequence based on the given recipe texts. Recipe execution times for these gold standard solutions are given as well in this chapter. Finally, the provided metrics are briefly explained in chapter 6 with some explanatory examples that can aid in correct interpretation of evaluation results.

More in-depth information about the benchmark's design can be found in the master's thesis "A Benchmark for Recipe Understanding in Autonomous Agents" by Robin De Haes (2023).

Chapter 2

Simulation & Evaluation

2.1 Executable

The benchmark simulator, with its built-in evaluation tools, is made available in the form of a standalone executable that allows one-click runs without needing prior installation. This executable can either be called via command-line with the appropriate arguments or it can simply be run directly in which case a graphical interface is shown requesting the user to specify required arguments.

2.1.1 Dependencies

The executable is completely standalone in the sense that running it is possible without installing any additional software. However, in case Smatch Score computations are requested a Python version of at least 3.5 should be installed and the path to the Python Smatch library should be specified. This library is bundled together with the simulator in the provided benchmark, but is also available as a standalone tool[1].

Furthermore, for visualizing semantic networks in more detail the Graphviz[2] tool can be used. However, interactive detailed visualizations are already possible without GraphViz. The only difference is that the implicit semantic networks will not be added as an image to the web visualization without GraphViz being installed (and also added to PATH in Windows).

2.1.2 Command-line

Command-line runs with the executable have two mandatory arguments that specify an input and output file path, namely *-input* and *-output* respectively. Additionally there are multiple optional arguments to further fine-tune the evaluation system. An overview

[1]`https://github.com/RobinDHVUB/smatch`

[2]`https://graphviz.org/download`

of all options is given in Table 2.1. An execution call that includes all these possible options for an evaluation run on Windows would look as follows:

```
cookingbot-evaluator.exe -input "input_file_path/predicted.solution"
                         -output "output_file_path/results.csv"
                         -show-output true
                         -metrics smatch-score
                                  goal-condition-success
                                  dish-approximation-score
                                  execution-time
                         -lib-dir "smatch_dir_path"
```

Argument	**Description**
-input	Path to a .solution file with predicted semantic networks
-output	Path to a .csv file to write evaluation results to
-show-output	If *true* or *t* the simulation process is visualized in the browser
-metrics	Evaluation metrics to use: *smatch-score*, *goal-condition-success*, *dish-approximation-score*, *execution-time* or *none*
-lib-dir	Path to the Smatch library directory (required for *smatch-score*)

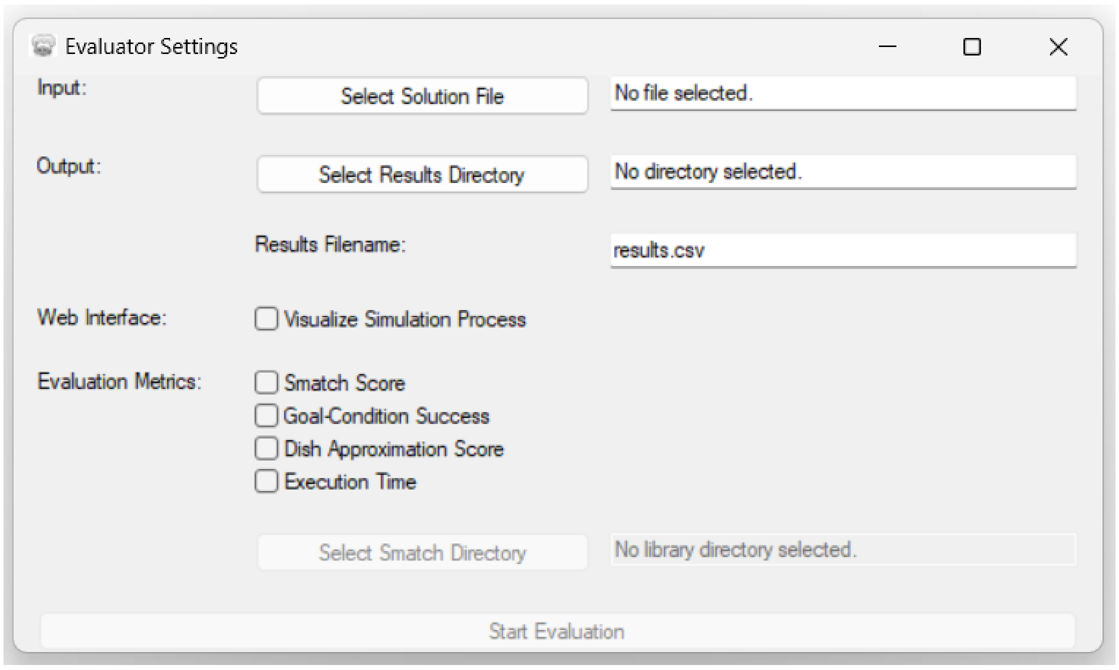
Table 2.1: An overview of all command-line arguments that are available for a cookingbot-evaluator run.

The input file path specified by *-input* should be a path to a .solution file containing one or more semantic network predictions that should be evaluated. Each semantic network should be prefixed by its recipe ID, which has been provided via the ID field in the structured recipe texts in the *data* directory. Example solution files can be found in the *documentation/examples/evaluation* directory.

Evaluation results are written away to the output CSV file specified by *-output*. The metrics to use for evaluation are given via the optional argument *-metrics*, which defaults to only using the simulation-based metrics. Additionally, in case the boolean argument *-show-output* is true Babel's interactive web visualization will be used to visualize the predicted network's execution for further analysis.

2.1.3 Graphical User Interface

Alternatively the executable can also be run directly in which case a graphical user interface is shown to the user requesting the same information as would otherwise be given via command-line. As shown in figure 2.1 starting evaluation is disabled until at least the mandatory arguments, i.e., the input and output paths, are provided.



No file selected.

No directory selected.

Results Filename: results.csv

- [ ] Visualize Simulation Process
- [ ] Smatch Score
- [ ] Goal-Condition Success
- [ ] Dish Approximation Score
- [ ] Execution Time

No library directory selected.

Figure 2.1: Graphical user interface shown when directly starting up the simulator executable. The 'Start Evaluation' button is disabled until input and output paths are specified. Smatch library selection is disabled unless Smatch Score evaluation is requested

2.2 Extensions via Babel Toolkit

In addition to the standalone executable, the benchmark has been made available as part of the Babel toolkit (Loetzsch, Wellens, De Beule, Bleys, & van Trijp, 2008). This is an open source toolkit containing modules for implementing and running language- and communication-related experiments in simulated environments or real-world environments using physical robots. In contrast to the executable approach, Babel toolkit installation and setup is not a one-click process but it has the advantage of providing other language-related development tools as well as benchmark extensibility. More information about the Babel Toolkit and its installation process can be found in its online documentation[3].

The main advantage of using the Babel toolkit instead of the executable is the possibility of extending the evaluation tools. Just like other Babel modules the simulator's modules are open source components, which can be found in the applications package *muhai-cookingbot*. Possible extensions include the addition of new metrics, new initial kitchen states and even new primitive operations. Since the simulator is built in IRL, such extensions will have to be made via IRL. For this, we refer to the Babel toolkit manual (Loetzsch et al., 2008) and the additional documentation that is accompanying the Babel codebase.

[3]`https://emergent-languages.org`

Chapter 3

MUHAI Cooking Language

The MUHAI Cooking Language is a representation language that provides a way of translating a sequence of natural language sentences into a machine-readable semantic network of executable predicates. Through argument sharing a notion of dependencies between steps is encoded as well in this predicate-argument structure. Such dependencies are derivable from the network by realizing that a shared argument used as input in one predicate is only available once it is provided as output in the other predicate.

3.1 Primitives

The primitives that are currently supported in MUHAI Cooking Language, their intended meaning and simulator-specific implementation choices will be listed and described alphabetically in the following paragraphs, with the number after / indicating the number of arguments the predicate takes.

bake/9

- Arguments:
  ?baked-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-bake*, *?oven*, *?time-value*, *?time-unit*, *?temperature-value*, *?temperature-unit*
- Intended Meaning:
  Obtain *?baked-thing* by baking *?thing-to-bake* in *?oven* at the temperature specified by *?temperature-value* and *?temperature-unit* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?oven* defaults to the closest unused oven in the kitchen

  - *?temperature-value* and *?temperature-unit* default to the temperature of *?oven* (only possible in case *?oven* is specified)

- Constant Arguments:
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*
  - *?temperature-unit* can be *degrees-celsius*

boil/8

- Arguments:
  ?boiled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-boil*, *?stove*, *?heating-setting*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?boiled-thing* by boiling *?thing-to-boil* on the *?stove* at the heating setting specified by *?heating-setting* for the duration specified by *?time-value* and *?time-unit*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?stove* defaults to the closest unused stove in the kitchen
  - *?heating-setting* defaults to *medium-heat*
  - *?time-value* and *?time-unit* default to 30 minutes
- Constant Arguments:
  - *?heating-setting* can be *low-heat*, *medium-heat*, *medium-high-heat*, *high-heat*
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

beat/5

- Arguments:
  ?beaten-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-beat*, *?beating-tool*
- Intended Meaning:
  Obtain *?beaten-thing* by beating *?thing-to-beat* using *?beating-tool*. Beating can be seen as a more intense form of mixing which adds some air bubbles during the combination process. This form of mixing also leads to a homogeneous result as individual components are not kept intact. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:
  - *?beating-tool* defaults to the closest unused whisk in the kitchen

bring-to-temperature/6

- Arguments:
  ?thing-at-desired-temperature, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-bring-to-temperature*, *?temperature-value*, *?temperature-unit*
- Intended Meaning:
  Obtain *?thing-at-desired-temperature* at the temperature specified by *?temperature-value* and *?temperature-unit* by waiting for *?thing-to-bring-to-temperature* to cool off or warm up by advancing towards the ambient temperature. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?temperature-value* and *?temperature-unit* default to the current room temperature of the kitchen, which is around 18 °C.
- Constant Arguments:
  - *?temperature-value* can be a numerical value
  - *?temperature-unit* can be *degrees-celsius*

cover/5

- Arguments:
  ?covered-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cover*, *?cover*
- Intended Meaning:
  Obtain *?covered-thing* by covering *?thing-to-cover* with the specified *?cover.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?cover* defaults to an appropriate cover for the given *?thing-to-cover*, i.e., a *bowl-lid* for a *bowl*, a *jar-lid* for a *jar* or *plastic-wrap* for anything else.

cut/6

- Arguments:
  ?cut-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cut*, *?cutting-pattern*, *?cutting-tool*

- Intended Meaning:
  Obtain *?cut-thing* by using *?cutting-tool* to cut *?thing-to-cut* according to the specified *?cutting-pattern*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?cutting-tool* defaults to the closest unused knife in the kitchen
- Constant Arguments:
  - *?cutting-pattern* can be *chopped*, *finely-chopped*, *slices*, *fine-slices*, *squares*, *two-cm-cubes*, *halved*, *shredded*, *minced*, *diced*

crack/5

- Arguments:
  ?container-with-whole-eggs, *?kitchen-state-out*, *?kitchen-state-in*, *?eggs-to-crack*, *?target-container-for-whole-eggs*
- Intended Meaning:
  Obtain *?container-with-whole-eggs* by cracking *?eggs-to-crack*, i.e., removing the egg shell from *?eggs-to-crack*, and dropping the egg contents in the container specified by *?target-container-for-whole-eggs*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?target-container-for-whole-eggs* defaults to the closest unused medium bowl in the kitchen

dip/5

- Arguments:
  ?dipped-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-dip*, *?dip*
- Intended Meaning:
  Obtain *?dipped-thing* by dipping *?thing-to-dip* into *?dip*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

drain/6

- Arguments:
  ?drained-thing, *?remaining-liquid*, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-drain*, *?draining-tool*

- Intended Meaning:
  Obtain *?drained-thing* by draining *?thing-to-drain* using *?draining-tool* leaving the remaining liquid in *?remaining-liquid*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?draining-tool* defaults to the closest unused colander in the kitchen

fetch/5

- Arguments:
  ?fetched-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-fetch*, *?quantity-to-fetch*
- Intended Meaning:
  Obtain *?fetched-thing* by locating one or more *?thing-to-fetch* objects in the kitchen and bringing it to a common work area such as a kitchen countertop. The exact number of objects to fetch is specified by *?quantity-to-fetch*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:
  - *?quantity-to-fetch* can be a numerical value
  - *?thing-to-fetch* can be any transferable container or cooking utensil available in the kitchen environment. These can currently all be found in the *kitchen-cabinet* of the initial kitchen state shown in Figure 4.1.

fetch-and-proportion/7

- Arguments:
  ?fetched-and-proportioned-ingredient, *?kitchen-state-out*, *?kitchen-state-in*, *?target-container-for-proportioned-ingredient*, *?mach-and-proportion*, *?proportion-value*, *?proportion-unit*
- Intended Meaning:
  Obtain an amount of the food product *?fetched-and-proportioned-ingredient* by fetching an *?ingredient-to-fetch-and-proportion*, taking a portion from it specified by *?proportion-value* and *?proportion-unit* and placing this portion inside the container specified by *?target-container-for-proportioned-ingredient*. Ingredient leftovers are returned to their original location. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:

  - *?ingredient-to-fetch-and-proportion* can be any ingredient that is mentioned in the ingredient list of a supported recipe. Ingredients should be specified by replacing all spaces in an ingredient name with the minus sign (-), e.g., 'ground black pepper' would become *ground-black-pepper.* A list of all ingredients can be found by combining the contents of the *fridge*, *freezer* and *pantry* of the initial kitchen state shown in Figure 4.1.
  - *?proportion-value* can be a numerical value
  - *?proportion-unit* can be *piece*, *g*, *teaspoon*, *tablespoon*, *l* or *ml*

flatten/5

- Arguments:
  ?flattened-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-flatten*, *?flattening-tool*
- Intended Meaning:
  Obtain *?flattened-thing* by flattening *?thing-to-flatten* using *?flattening-tool.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?flattening-tool* defaults to the closest unused rolling pin in the kitchen

flour/5

- Arguments:
  ?floured-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-flour*, *?flour*
- Intended Meaning:
  Obtain *?floured-thing* by flouring *?thing-to-flour* with *?flour.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?flour* defaults to 10 grams of all-purpose flour taken from the closest container with all-purpose flour

fry/8

- Arguments:
  ?fried-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-fry*, *?stove*, *?heating-setting*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?fried-thing* by frying *?thing-to-fry* on the *?stove* at the heating setting

specified by *?heating-setting* for the duration specified by *?time-value* and *?time-unit.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:
  - *?stove* defaults to the closest unused stove in the kitchen
  - *?heating-setting* defaults to *medium-heat*
  - *?time-value* and *?time-unit* default to 30 minutes
- Constant Arguments:
  - *?heating-setting* can be *low-heat*, *medium-heat*, *medium-high-heat*, *high-heat*
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

get-kitchen/1

- Arguments:
  ?initial-kitchen-state
- Intended Meaning:
  Obtain the initial state of the kitchen *?initial-kitchen-state.* This is expected to provide access to an environment model of the kitchen to provide contextual information needed for executing a recipe.
- Default Values:
  - *?initial-kitchen-state* defaults to the initial kitchen state in which a recipe will be executed. This argument is expected to be left to its default value in which case this primitive functions as a 'getter'.

grease/5

- Arguments:
  ?greased-thing, *?kitchen-state-out*, *?kitchen-state-in*,
  ?thing-to-grease, *?thing-to-grease*, *?grease*
- Intended Meaning:
  Obtain *?greased-thing* by greasing *?thing-to-grease* with *?grease.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?grease* defaults to 10 grams of butter taken from the closest container with butter

grind/5

- Arguments:
  ?ground-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-grind*, *?grinding-tool*
- Intended Meaning:
  Obtain *?ground-thing* by grinding *?thing-to-grind* using *?grinding-tool.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?grinding-tool* defaults to the closest unused food-processor in the kitchen

leave-for-time/6

- Arguments:
  ?cooled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-cool*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?cooled-thing* by waiting for the duration specified by *?time-value* and *?time-unit* to let *?thing-to-cool* cool off towards the ambient temperature. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:
  - *?time-value* and *?temperature-value* can be numerical values
  - *?time-unit* can be *hour* or *minute*

line/5

- Arguments:
  ?lined-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-line*, *?lining*
- Intended Meaning:
  Obtain *?lined-thing* by lining *?thing-to-line* with *?lining*, e.g., lining a baking tray with some baking paper or lining muffin tins with paper baking cups. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?lining* defaults to the closest unused sheet of baking paper
- Constant Arguments:
  - *?lining* can be *baking-paper* or *paper-baking-cups*

  - *?thing-to-line* can be *baking-tray*, *cookie-sheet*, *pan* or *muffin-tins*

mash/5

- Arguments:
  ?mashed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mash*, *?mashing-tool*
- Intended Meaning:
  Obtain *?mashed-thing* by mashing up *?thing-to-mash* using *?mashing-tool.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?mashing-tool* defaults to the closest unused fork in the kitchen

melt/5

- Arguments:
  ?melted-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-melt*, *?melting-tool*
- Intended Meaning:
  Obtain *?melted-thing* by melting *?thing-to-melt* using *?melting-tool.* This melting tool could be any kind of heating appliance in the kitchen, ranging from a pan on the stove to a microwave. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?melting-tool* defaults to the closest unused microwave in the kitchen

mingle/5

- Arguments:
  ?mingled-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mingle*, *?mingling-tool*
- Intended Meaning:
  Obtain *?mingled-thing* by mingling *?thing-to-mingle* using *?mingling-tool.* Mingling can be seen as a softer form of mixing in which the individual components are still kept intact during the combination process. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?mingling-tool* defaults to the closest unused wooden spoon in the kitchen

mix/5

- Arguments:
  ?mixed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-mix*, *?mixing-tool*
- Intended Meaning:
  Obtain *?mixed-thing* by mixing *?thing-to-mix* using *?mixing-tool.* Mixing can be seen as a form of mixing that is intense enough to achieve a homogeneous mixture without being so intense that air bubbles are added during the mixing process. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?mixing-tool* defaults to the closest unused whisk in the kitchen

peel/6

- Arguments:
  ?peeled-thing, *?peel*, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-peel*, *?peeling-tool*
- Intended Meaning:
  Obtain *?peeled-thing* and its *?peel* by peeling *?thing-to-peel* using *?peeling-tool.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?peeling-tool* defaults to the closest unused knife in the kitchen

portion-and-arrange/8

- Arguments:
  ?portions, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-portion*, *?portion-size-value*, *?portion-size-unit*, *?placement-pattern*, *?container-for-portions*
- Intended Meaning:
  Obtain *?portions* by portioning *?thing-to-portion* into portions that each have a size specified by *?portion-size-value* and *?portion-size-unit.* These portions are placed onto the container *?container-for-portions* following the *?placement-pattern.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?placement-pattern* defaults to a pattern in which all portions are evenly spread out over the container
  - *?container-for-portions* defaults to the countertop of the kitchen

  - *?portion-size-value* and *?portion-size-unit* default to portion sizes that cause an equal division over the available tins (only possible in case *?container-for-portions* are muffin tins)

- Constant Arguments:
  - *?placement-pattern* can be *side-to-side*, *evenly-spread*, *5-cm-apart*

preheat-oven/6

- Arguments:
  ?preheated-oven, *?kitchen-state-out*, *?kitchen-state-in*, *?oven*, *?temperature-value*, *?temperature-unit*
- Intended Meaning:
  Obtain *?preheated-oven* by changing the settings of the *?oven* to reach the temperature specified by *?temperature-value* and *?temperature-unit.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?oven* defaults to the closest unused oven in the kitchen
- Constant Arguments:
  - *?temperature-value* can be a numerical value
  - *?temperature-unit* can be *degrees-celsius*

refrigerate/7

- Arguments:
  ?refrigerated-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-refrigerate*, *?refrigerator*, *?time-value*, *?time-unit*
- Intended Meaning:
  Obtain *?refrigerated-thing* by putting *?thing-to-refrigerate* inside *?refrigerator* for the duration specified by *?time-value* and *?time-unit.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?refrigerator* defaults to the closest unused fridge in the kitchen
  - *?time-value* and *?time-unit* default to one hour
- Constant Arguments:
  - *?time-value* can be a numerical value

  - *?time-unit* can be *minute* or *hour*

seed/6

- Arguments:
  ?seeded-thing, *?seed*, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-seed*, *?seeded-tool*
- Intended Meaning:
  Obtain *?seeded-thing* and its *?seed* by seeding *?thing-to-seed* using *?seeding-tool.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?seeding-tool* defaults to the closest unused knife in the kitchen

separate-eggs/8

- Arguments:
  ?egg-yolks, *egg-whites*, *?kitchen-state-out*, *?kitchen-state-in*, *?eggs*, *?container-for-yolks*, *?container-for-whites*, *?egg-separator*
- Intended Meaning:
  Obtain *?egg-yolks* and *egg-whites* by using an *?egg-separator* to separate separating the whole *?eggs* into the *?container-for-yolks* and *?container-for-whites* respectively. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?container-for-yolks* defaults to the closest unused stove in the kitchen
  - *?container-for-whites* defaults to the closest unused medium bowl in the kitchen (excluding the one found for *?container-for-yolks*)
  - *?egg-separator* defaults the closest unused egg separator in the kitchen

shake/4

- Arguments:
  ?shaken-thing, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-shake*
- Intended Meaning:
  Obtain *?shaken-thing* by shaking *?thing-to-shake* to mix its contents, which are generally liquids, until a homogeneous mixture is reached. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

shape/5

- Arguments:
  ?shaped-thing, *?kitchen-state-out* *?kitchen-state-in*, *?thing-to-shape*, *?shape*
- Intended Meaning:
  Obtain *?shaped-thing* by shaping *?thing-to-shape* into the shape specified by *?shape*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Constant Arguments:
  - *?shape* can be *ball-shape* or *crescent-shape*

sift/6

- Arguments:
  ?sifted-thing, *?kitchen-state-out* *?kitchen-state-in*, *?container-to-sift-into*, *?thing-to-sift*, *?sift*
- Intended Meaning:
  Obtain *?sifted-thing* by using *?sift* to sift *?thing-to-sift* into the container specified by *?container-to-sift-into*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?container-to-sift-into* defaults to the closest unused large bowl in the kitchen
  - *?sift* defaults to the closest unused sift in the kitchen

spread/6

- Arguments:
  ?thing-with-spread-on, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-spread-on*, *?thing-to-spread*, *?spreading-tool*
- Intended Meaning:
  Obtain *?thing-with-spread* by spreading *?thing-to-spread* on *?thing-to-spread-on* using *spreading-tool*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?spreading-tool* defaults to the closest unused spatula in the kitchen

sprinkle/5

- Arguments:
  ?thing-with-sprinkles-on, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-sprinkle-on*, *?sprinkles*
- Intended Meaning:
  Obtain *?thing-with-sprinkles-on* by sprinkling *?sprinkles* onto *?thing-to-sprinkle-on.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

transfer-contents/8

- Arguments:
  ?container-with-transferred-contents, *?container-with-rest-of-contents*, *?kitchen-state-out*, *?kitchen-state-in*, *?container-to-transfer-contents-to*, *?container-with-contents-to-transfer*, *?value-of-transfer-amount*, *?unit-of-transfer-amount*
- Intended Meaning:
  Obtain *?container-with-transferred-contents* and *?container-with-rest-of-contents* by transferring an amount (specified by *?value-of-transfer-amount* and *?unit-of-transfer-amount*) of the container *?container-with-contents-to-transfer*'s contents into *?container-to-transfer-contents-to* leaving the remaining contents in *?container-with-rest-of-contents.* The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.
- Default Values:
  - *?container-to-transfer-contents-to* defaults to the closest unused large bowl in the kitchen
  - *value-of-transfer-amount* and *unit-of-transfer-amount* default to an amount for which all contents are transferred, effectively emptying the original container
- Constant Arguments:
  - *?value-of-transfer-amount* can be a numerical value
  - *?unit-of-transfer-amount* can be *piece*, *g*, *teaspoon*, *tablespoon*, *ml* or *percent*

transfer-items/6

- Arguments:
  ?transferred-items, *?kitchen-state-out*, *?kitchen-state-in*, *?items-to-transfer*, *?placement-pattern*, *?destination*
- Intended Meaning:
  Obtain *?transferred-items* by carefully transferring all items from *?items-to-transfer*

to *?destination* and placing them there according to the pattern specified by *?placement-pattern*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

- Default Values:
  - *?placement-pattern* defaults to a pattern in which the available location is filled up from side to side by creating rows of items one at a time in which items are placed next to each other.

uncover/5

- Arguments:
  ?uncovered-thing, *?cover* *?kitchen-state-out*, *?kitchen-state-in*, *?covered-thing*
- Intended Meaning:
  Obtain *?uncovered-thing* and its prior *?cover* by removing the cover from *?covered-thing*. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

wash/4

- Arguments:
  ?washed-thing, *?kitchen-state-out*, *?kitchen-state-in*, *?thing-to-wash*
- Intended Meaning:
  Obtain *?washed-thing* by rinsing off or washing *?thing-to-wash* with water. The arguments *?kitchen-state-in* and *?kitchen-state-out* represent the contextual situation before and after execution of this predicate.

Chapter 4

Initial Kitchen State

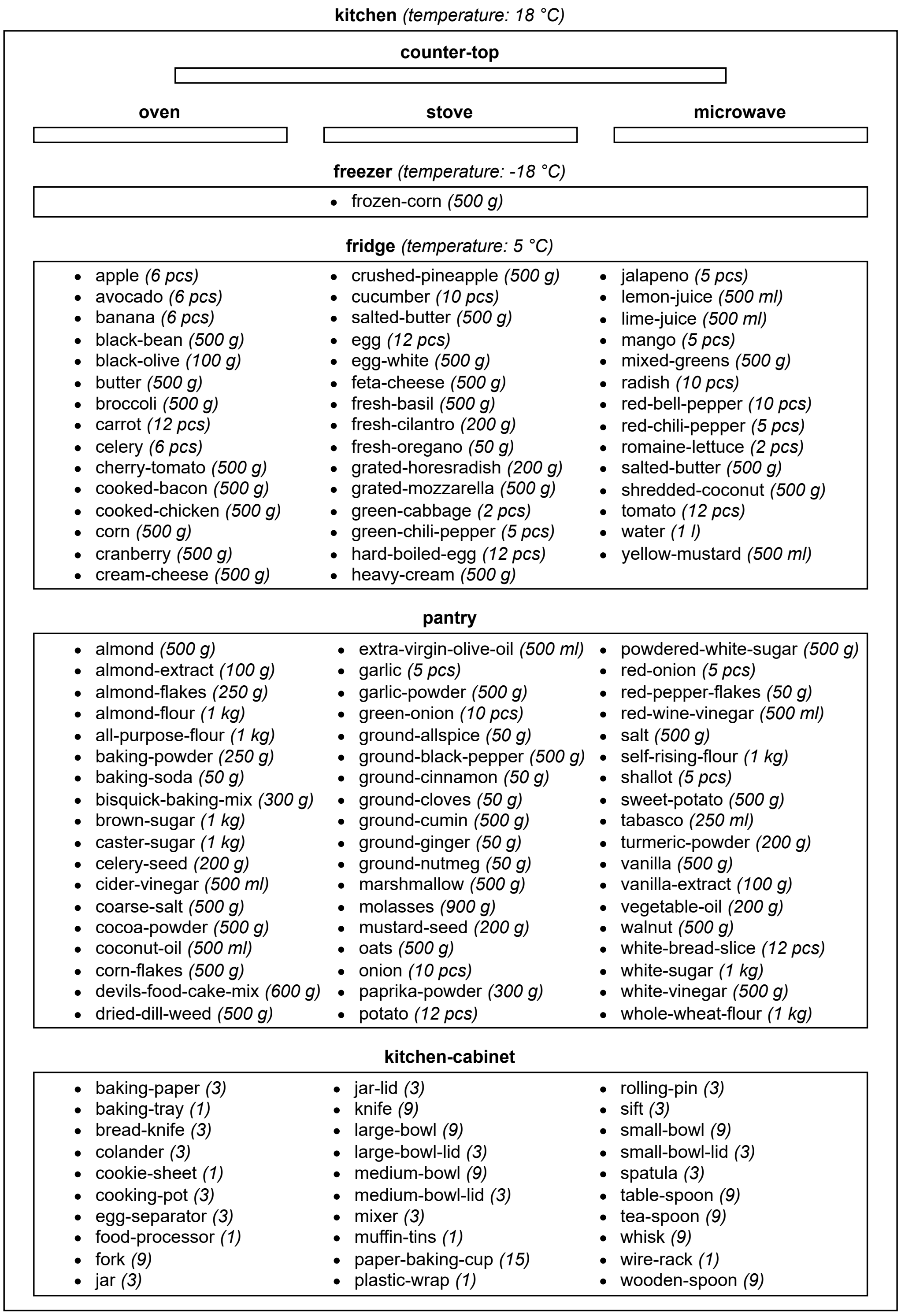
This chapter describes the composition of the initial kitchen state from which recipe execution will be started in the simulator for any of the available test recipes. The initial kitchen state provides the initial context to start from and gets loaded in by the *get-kitchen* operation, which should be present in every semantic network.

The contextual information provided by knowing the initial kitchen state could influence the actual interpretation of recipe texts, since it could alter what is possible and thus what the semantic network should look like. Therefore, the composition of the initial kitchen state should be taken into account during model development and evaluation.

4.1 full-kitchen

There is currently only one initial kitchen state made available in the benchmark which is a very full kitchen containing all kitchen commodities, tools, appliances and ingredients that could be needed for executing any of the recipes in the benchmark. All ingredients are made available in medium bowls, which could be located in the fridge, freezer or pantry based on the ingredient's storage requirements.

Figure 4.1 gives a detailed overview of the supported initial kitchen state *full-kitchen*.



kitchen *(temperature: 18 °C)*

counter-top

oven

stove

microwave

freezer *(temperature: -18 °C)*

- frozen-corn *(500 g)*

fridge *(temperature: 5 °C)*

- apple *(6 pcs)*
- avocado *(6 pcs)*
- banana *(6 pcs)*
- black-bean *(500 g)*
- black-olive *(100 g)*
- butter *(500 g)*
- broccoli *(500 g)*
- carrot *(12 pcs)*
- celery *(6 pcs)*
- cherry-tomato *(500 g)*
- cooked-bacon *(500 g)*
- cooked-chicken *(500 g)*
- corn *(500 g)*
- cranberry *(500 g)*
- cream-cheese *(500 g)*
- crushed-pineapple *(500 g)*
- cucumber *(10 pcs)*
- salted-butter *(500 g)*
- egg *(12 pcs)*
- egg-white *(500 g)*
- feta-cheese *(500 g)*
- fresh-basil *(500 g)*
- fresh-cilantro *(200 g)*
- fresh-oregano *(50 g)*
- grated-horesradish *(200 g)*
- grated-mozzarella *(500 g)*
- green-cabbage *(2 pcs)*
- green-chili-pepper *(5 pcs)*
- hard-boiled-egg *(12 pcs)*
- heavy-cream *(500 g)*
- jalapeno *(5 pcs)*
- lemon-juice *(500 ml)*
- lime-juice *(500 ml)*
- mango *(5 pcs)*
- mixed-greens *(500 g)*
- radish *(10 pcs)*
- red-bell-pepper *(10 pcs)*
- red-chili-pepper *(5 pcs)*
- romaine-lettuce *(2 pcs)*
- salted-butter *(500 g)*
- shredded-coconut *(500 g)*
- tomato *(12 pcs)*
- water *(1 l)*
- yellow-mustard *(500 ml)*

pantry

- almond *(500 g)*
- almond-extract *(100 g)*
- almond-flakes *(250 g)*
- almond-flour *(1 kg)*
- all-purpose-flour *(1 kg)*
- baking-powder *(250 g)*
- baking-soda *(50 g)*
- bisquick-baking-mix *(300 g)*
- brown-sugar *(1 kg)*
- caster-sugar *(1 kg)*
- celery-seed *(200 g)*
- cider-vinegar *(500 ml)*
- coarse-salt *(500 g)*
- cocoa-powder *(500 g)*
- coconut-oil *(500 ml)*
- corn-flakes *(500 g)*
- devils-food-cake-mix *(600 g)*
- dried-dill-weed *(500 g)*
- extra-virgin-olive-oil *(500 ml)*
- garlic *(5 pcs)*
- garlic-powder *(500 g)*
- green-onion *(10 pcs)*
- ground-allspice *(50 g)*
- ground-black-pepper *(500 g)*
- ground-cinnamon *(50 g)*
- ground-cloves *(50 g)*
- ground-cumin *(500 g)*
- ground-ginger *(50 g)*
- ground-nutmeg *(50 g)*
- marshmallow *(500 g)*
- molasses *(900 g)*
- mustard-seed *(200 g)*
- oats *(500 g)*
- onion *(10 pcs)*
- paprika-powder *(300 g)*
- potato *(12 pcs)*
- powdered-white-sugar *(500 g)*
- red-onion *(5 pcs)*
- red-pepper-flakes *(50 g)*
- red-wine-vinegar *(500 ml)*
- salt *(500 g)*
- self-rising-flour *(1 kg)*
- shallot *(5 pcs)*
- sweet-potato *(500 g)*
- tabasco *(250 ml)*
- turmeric-powder *(200 g)*
- vanilla *(500 g)*
- vanilla-extract *(100 g)*
- vegetable-oil *(200 g)*
- walnut *(500 g)*
- white-bread-slice *(12 pcs)*
- white-sugar *(1 kg)*
- white-vinegar *(500 g)*
- whole-wheat-flour *(1 kg)*

kitchen-cabinet

- baking-paper *(3)*
- baking-tray *(1)*
- bread-knife *(3)*
- colander *(3)*
- cookie-sheet *(1)*
- cooking-pot *(3)*
- egg-separator *(3)*
- food-processor *(1)*
- fork *(9)*
- jar *(3)*
- jar-lid *(3)*
- knife *(9)*
- large-bowl *(9)*
- large-bowl-lid *(3)*
- medium-bowl *(9)*
- medium-bowl-lid *(3)*
- mixer *(3)*
- muffin-tins *(1)*
- paper-baking-cup *(15)*
- plastic-wrap *(1)*
- rolling-pin *(3)*
- sift *(3)*
- small-bowl *(9)*
- small-bowl-lid *(3)*
- spatula *(3)*
- table-spoon *(9)*
- tea-spoon *(9)*
- whisk *(9)*
- wire-rack *(1)*
- wooden-spoon *(9)*

Figure 4.1: Visualization of the simulator's initial kitchen state *full-kitchen.*

Chapter 5

Gold Standard Solution

The simulator has gold standard solutions built-in. These solutions are used in the computation of all evaluation results, except recipe execution time. For completeness sake, the execution time for each gold standard solution is given in section 5.2 although execution time differences should generally not be interpreted in a standalone manner by comparing them to the gold standard execution time as will be explained in the section 5.2.

In addition to the built-in solutions, gold standard solution files have also been provided in the *data* directory of this benchmark. Due to common recipe idiosyncrasies such as missing steps, ellipses and contextual references semantic networks for recipes are most meaningful when viewed as a whole. Therefore, one complete gold standard annotation is given for each entire recipe instead of for each separate sentence. The data subdirectory *meaning-only* contains the full network and can for example be used for Smatch score computations without the use of the provided simulator if desired. The data subdirectory *utterance and meaning* contains XMLs that mention per sentence to which primitive operation it explicitly or implicitly led. These XMLs are mostly meant to aid interpretation, analysis and troubleshooting during development.

Lastly, a subset of gold standard solutions are given in the folder *documentation/examples/gold standard annotations*. These are the regular solutions from the *meaning-only* subdirectory, but with ample comments being provided in the file to fully understand the process of generating solutions from recipe text which will be helpful when curating new training data yourself.

5.1 Characteristics

Since gold standard solutions are used in computation of evaluation results, some insight is required into what makes a solution a gold standard solution in order to understand obtained results.

First, each recipe network in the MUHAI Cooking Language should start with *get-kitchen*. This primitive operation loads in the initial kitchen state, which provides the initial context to start from. Without this primitive operation simulation will not be possible.

Secondly, the gold standard solution will go over the recipe text in a sequential manner and add primitive operations based on the order of sentences in the text. This is important to keep in mind when comparing the more traditional Smatch score, as this score directly compares network composition and not execution. If some instructions can happen in parallel or in a permuted order, simulation-based scores will still be equal to the gold standard scores but the traditional Smatch Score will not.

Thirdly, the gold standard solution aims to execute recipes as efficiently as possible. This means tools are reused as much as possible instead of constantly fetching a new tool, which would lead to more required cleaning and lost time. If a recipe states to 'cut the tomato and the cucumber' for example, the same knife will be used for both in the gold standard solution even though using two different knives would lead to the same outcome. However, in the case of inefficient but correct solutions only Smatch score and recipe execution time should be impacted.

5.2 Recipe Execution Time

This section contains an overview of the recipe execution time for the gold standard solutions of each recipe, given the specified initial kitchen state. It should be noted that a lower execution time than this baseline does not automatically mean better performance as it could be caused by inadequate comprehension of the recipe and performing less or incorrect operations. Recipe execution time should thus always be interpreted in the context of other evaluation results.

An overview of the specified initial kitchen states' composition can be found in chapter 4.

Recipe ID	Kitchen State ID	Execution Time
afghan-biscuits	full-kitchen	2735
almond-crescent-cookies	full-kitchen	2600
almond-crescent-cookies-2	full-kitchen	3190
almond-crescent-cookies-3	full-kitchen	2485
almond-crescent-cookies-4	full-kitchen	2055
almond-crescent-cookies-5	full-kitchen	3475
avocado-chicken-salad	full-kitchen	4920
basic-chicken-salad	full-kitchen	2280
best-brownies	full-kitchen	2475
bisquick-shortcake-biscuits	full-kitchen	1145
black-bean-and-sweet-potato-salad	full-kitchen	2795
black-bean-salad-2	full-kitchen	1950
black-bean-salad-3	full-kitchen	4210
black-bean-salad-4	full-kitchen	1100
black-bean-salad-5	full-kitchen	4770
broccoli-salad	full-kitchen	2074280
chocolate-cream-cheese-cupcakes	full-kitchen	2660
chocolate-fudge-cookies	full-kitchen	1650
classic-greek-salad	full-kitchen	1640
classic-potato-salad	full-kitchen	3290
coconut-tuiles	full-kitchen	1680
cole-slaw	full-kitchen	6360
cranberry-fluff-salad	full-kitchen	230920
croutons-vinegar-salad	full-kitchen	4955
cucumber-slices-with-dill	full-kitchen	15050
easy-banana-bread	full-kitchen	4210
easy-cherry-tomato-corn-salad	full-kitchen	4880
easy-oatmeal-cookies	full-kitchen	1845
mexican-wedding-cookies	full-kitchen	1925
whole-wheat-ginger-snaps	full-kitchen	2320

Table 5.1: An overview of the recipe execution times for simulations in which the gold standard solution of the recipes are executed for the given initial kitchen state. The execution time is expressed in simulation time steps which can be interpreted as approximated seconds that are needed for real-world execution.

Chapter 6

Metrics

The simulator supports four different evaluation metrics, allowing multiperspective performance estimates which enhances transferability of results to real-world utility. These metrics are the following:

- **Smatch Score**: A commonly used semantic graph comparison tool that measures overlap between semantic structures (Cai & Knight, 2013).
- **Goal-Condition Success**: The ratio of required goal-conditions that have been successfully reached during recipe execution to the ones that were not reached. An example goal-condition could be having a large bowl with dough on the kitchen countertop.
- **Dish Approximation Score**: A custom similarity estimate of two prepared food products. Pseudo-code explaining the score computation is given in section 6.2.
- **Recipe Execution Time**: An efficiency measure that tracks how many time steps would be needed to execute a recipe following the given solution.

It should be noted that it is highly recommended to use at least all simulation-based metrics to adequately measure performance. Examples demonstrating the usefulness of combining metrics and how to effectively interpret their results are given in section 6.1.

6.1 Examples of Result Interpretations

This section will guide you through a number of examples aimed at providing a better understanding of how to evaluate a solution and interpret its metric results. Moreover, it further accentuates the importance of combining multiple metrics. The examples that are used are different possible solutions for the recipe "Almond Crescent Cookies" (recipe ID: almond-crescent-cookies), which all have varying degrees of correctness. The solution files can be found in the folder *documentation/examples/evaluation.* Additional comments have been added in each file to clearly indicate deviations from the perfect solution.

6.1.1 perfect

The file perfect.solution contains a perfect solution for almond-crescent-cookies. The implicit steps of warming up butter and fetching baking utensils have been deduced and the same mixing tool is reused for maximum efficiency. Table 6.1 shows the scores obtained for this perfect solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
1.00	1.00	1.00	2600

Table 6.1: An overview of the evaluation results for the file perfect.solution.

Since all mandatory steps and no unneeded steps are included, the same implicit graph is obtained as in the gold standard solution. This is the only case in which Smatch is expected to detect full propositional overlap and a Smatch Score of 1 is achieved. As the exact same implicit graph is obtained goal-condition success and the dish approximation score are also 1. The recipe execution time is the minimum that is possible and coincides with the environments recipe execution time, which is 2600 seconds.

6.1.2 perfect-permuted-sequence

The file perfect-permuted-sequence.solution contains a perfect solution for almond-crescent-cookies, but the actual sequence of primitives in the file are randomly permuted. Table 6.2 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
1.00	1.00	1.00	2600

Table 6.2: An overview of the evaluation results for the file perfect-permuted-sequence.solution.

The sequence of the primitives in the file itself are unimportant, since all metrics are based on the implicit execution graph that is formed through argument sharing. This implicit graph is still the same as the implicit graph of the gold standard solution, meaning all results are still optimal.

6.1.3 perfect-switched-operations

The file perfect-switched-operations.solution contains a near-perfect solution for almond-crescent-cookies, but some ingredients were added to the bowl in a different order than specified by the steps in the recipe. In Table 6.3 shows the scores obtained for this near-perfect solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.99	0.92	1.00	2600

Table 6.3: An overview of the evaluation results for the file perfect-switched-operations.solution.

Since operational steps have been switched in the implicit graph, there is no complete propositional overlap with the gold standard graph and thus the Smatch Score is slightly lowered. Additionally, goal-condition success is lower as well. If two goal-conditions are to have a bowl with 'ingredient1' and then to have a bowl with 'ingredient1' and 'ingredient2', then adding 'ingredient2' before 'ingredient1' leads to the first goal-condition never being reached. Nevertheless, just adding ingredients in a bowl in a different order before mixing them should not have an impact on the taste which is the reason the dish approximation score is still maximal. Furthermore, adding ingredients in a different order should not lead to a change in recipe execution time either which is the reason recipe execution time is still the same as for the perfect solution.

6.1.4 missing-tool-reuse

The file missing-tool-reuse.solution contains a correct solution for almond-crescent-cookies that contains all steps, but does not always reuse tools whenever possible. In Table 6.4 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.93	1.00	1.00	2660

Table 6.4: An overview of the evaluation results for the file missing-tool-reuse.solution.

Since tools are not reused, this means that strictly speaking some unnecessary steps have been executed. Therefore, the implicit graph is different from the gold standard solution which results in a lower Smatch Score. Furthermore, fetching new tools also increases the recipe execution time as tool reuse is more efficient.

Important to note here is that all gold standard goal-conditions are still reached and the final dish is completely correct, which means the goal-condition success and the dish approximation score are still maximal. The recipe will still be successfully executed, but it will simply happen in a more inefficient way. Therefore, from a simulation-based perspective adding the recipe execution time is actually quite informative in this particular case.

6.1.5 missing-minor-implicit

The file missing-implicits.solution contains an incorrect solution for almond-crescent-cookies. The cooking agent failed to deduce the implicit step of letting the butter warm up before mixing it, which is an inconvenience for the later mixing process. Table 6.5 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.94	0.38	0.99	1980

Table 6.5: An overview of the evaluation results for the file missing-implicits.solution.

Since a mandatory step is missing, the implicit graph is different from the gold standard solution which results in a lower Smatch Score as there is no complete propositional overlap. Furthermore, warming up the butter was a required step and thus a goal-condition that is missed.

It is important to note here that this missed goal-condition occurs quite early in the process leading to many subsequent goal-condition failures and a low goal-condition success score overall. Every goal-condition that directly or indirectly requires this warmed up butter will now fail. The evaluation of goal-conditions is very strict, with a goal-condition either being completely fulfilled or being unsuccessful. This is a clear example of why the addition of the dish approximation score is useful. Not warming up the butter will not have a large influence on the taste of the final dish which is represented in a high dish approximation score.

A second important note here is that the recipe execution time is actually lower than the recipe execution time of a perfect solution. This is normal as less steps are executed and therefore execution is finished quicker. Therefore, recipe execution time should generally not be interpreted in isolation as it is only informative in combination with the other metrics.

6.1.6 partial-failure

The file partial-failure.solution contains an incorrect solution for almond-crescent-cookies. The agent failed to deduce to fetch a baking tray and baking paper, which led to not executing any of the later steps. These steps include baking the cookies and sprinkling some powdered sugar on them. Table 6.6 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.84	0.77	0.82	1320

Table 6.6: An overview of the evaluation results for the file partial-failure.solution.

Since some mandatory steps are missing, the implicit graph is different from the gold standard solution which results in a lower Smatch Score as there is no complete propositional overlap. Furthermore, all goal-conditions after fetching the baking tray are missing which leads to a significant lowering of goal-condition-success as well. Since the mistake happens fairly late in the process, there are less goal-conditions impacted by the mistake compared to forgetting an operation early on even if the early operation would have been less significant. Forgetting to bake cookies at the end will have a lower impact than forgetting to warm up butter at the start of cooking, since goal-condition success considers all goal-conditions to be equally important.

The dish approximation score has been lowered as well. It should be noted, however, that this lowering is mainly caused by the fact that an ingredient is missing due to the absence of a sprinkling step. When it comes to taste the impact of not baking the dough might be equally high in practice, but the dish score mostly biases correctness of ingredient composition. This seems defensible as it is easy for a human to still intervene and go from portioned dough to baked cookies, while human intervention would not easily allow turning incorrectly made baked cookies into correctly made cookies.

Since the baking operation is the most time-consuming step in this recipe, recipe execution time is much lower than the perfect solution. This example thus demonstrates again that this metric is not very informative in absence of the other metrics.

6.1.7 wrong-ingredient

The file wrong-ingredient.solution contains an incorrect solution for almond-crescent-cookies. The agent mistakenly switched one of the ingredients with another, namely it switched white sugar with cocoa powder. Table 6.7 shows the scores obtained for this solution.

Smatch	**Goal-Condition Success**	**Dish Score**	**Execution Time**
0.99	0.42	0.76	2600

Table 6.7: An overview of the evaluation results for the file wrong-ingredient.solution.

Since a mandatory step has changed, the implicit graph is different from the gold standard solution which results in a lower Smatch Score as there is no complete propositional overlap. It is important to note here that changing a base ingredient does not lead to a significant lowering of the Smatch Score, although it can be expected to have a significant impact on the final dish. The achieved Smatch score is even comparable to a setting in which only a minor step is forgotten, such as letting butter warm up before adding it. This further highlights the importance of including other metrics than Smatch score.

Both goal-condition success and the dish approximation score have been heavily affected by the ingredient mistake. The missed goal-conditions of fetching and adding white sugar happened early on in the process, which led to many subsequent goal-condition failures and thus a low goal-condition success score overall. Every goal-condition directly or indirectly requiring the sugar ingredient have failed.

In addition, this particular mistake is also expected to have a big influence on the taste of the final dish. Therefore, the dish approximation score is substantially lowered as well, resulting in a score that is justifiably lower than a case in which a minor step such as warming up butter is forgotten.

However, recipe execution time has remained optimal as both ingredients should take equally long to be fetched and added to a bowl. This examples thus demonstrates once more that recipe execution time is not informative in isolation.

6.1.8 additional-side-dish

The file additional-side-dish.solution contains a solution for almond-crescent-cookies in which an additional side dip is created after the main dish is made. This side dip preparation was not included in the original recipe. Table 6.8 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.90	1.00	1.00	2740

Table 6.8: An overview of the evaluation results for the file additional-side-dish.solution.

Since a surplus of steps is present due to the side dish preparation, the implicit graph is different from the gold standard solution which results in a lower Smatch Score. Additionally, recipe execution time has increased due to the extra steps that have been executed.

However, both goal-condition success and the dish approximation score are still maximal. Even if more steps are executed than needed for the main dish, all required goal-conditions have been achieved leading to perfect goal-condition success. Furthermore, the main dish that needed to be made is available in perfect condition at the end leading to a perfect dish approximation score.

It is important to note that the dish approximation score is not based on the side dish, even though this is the last dish that was prepared during recipe execution. The dish approximation score will always be computed on the dish that maximally approximates the gold standard dish irrespective of the timing of its availability. Therefore, this is another case in which recipe execution time is useful to distinguish efficient from inefficient time use as both other simulation-based metrics do not detect this.

6.1.9 extended-main-dish

The file extended-main-dish.solution contains a solution for almond-crescent-cookies in which an additional side dip is created after the main dish is made. The cookies from the main dish are then also dipped in this side dip, effectively altering the main dish. The steps related to the side dip were not included in the original recipe. Table 6.9 shows the scores obtained for this solution.

Since a surplus of steps is present due to the dip preparation and dipping operation, the implicit graph is different from the gold standard solution which results in a lower

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.92	1.00	0.87	2790

Table 6.9: An overview of the evaluation results for the file extended-main-dish.solution.

Smatch Score. Additionally, recipe execution time has increased due to the extra steps that have been executed.

Furthermore, the dish approximation score has been affected as well. Dipping the main dish's cookies in chocolate alters their composition. Even though the main dish was available in perfect condition at some point during execution, the final dish that can be served is not perfectly comparable with the dish that needed to be created leading to a lower dish approximation score.

However, it is important to note that goal-condition success is still maximal in this case. More steps are executed than needed to reach a state in which the main dish is available, but all required goal-conditions have still been achieved at some point leading to perfect goal-condition success. Goal-condition success is thus independent of the fact that some goal-conditions might be undone at a later point by altering the main dish as long as the goal-conditions have been reached once. This example thus demonstrates why combining multiple metrics can be important in order to obtain a reliable view on performance.

6.1.10 no-cooking

The file no-cooking.solution contains an incorrect solution for almond-crescent-cookies in which no actual cooking takes place. The agent only recognized two fetch operations, namely fetching the baking tray and baking paper. Table 6.10 shows the scores obtained for this solution.

Smatch	Goal-Condition Success	Dish Score	Execution Time
0.12	0.08	0.00	60

Table 6.10: An overview of the evaluation results for the file no-cooking.solution.

Since many mandatory steps are missing, the implicit graph is very different from the standard solution which results in a low Smatch score. However, it should be noted that a score of 0.12 for an execution network that performs no actual cooking is still relatively high. Moreover, since fetching a baking tray and baking paper were actual goal-conditions, goal-condition success is also not 0 and even relatively high for a recipe execution network that performs no actual cooking. These results again highlight the importance of using a combination of metrics as they provide multiple perspectives on the same result. The dish approximation score for example is effectively 0 which clearly indicates no useful food product has been made prepared.

Recipe execution time is very low as not many cooking operations have actually taken place. This emphasizes once more that recipe execution time is only informative in combination with the other metrics. Optimizing solely for recipe execution time could lead to results in which no cooking takes place.

6.2 Pseudo-Algorithm for Dish Approximation Score

DishApproximationScore(GoldStandardDish, PredictedOutputs)

parameters
GoldStandardDish - the main output food product of the gold standard solution
PredictedOutputs - the set of final output food products from the prediction

implementation:

```
$MaxScore \leftarrow 0$
for every $Output$ in $PredictedOutputs$ do
    $ContainerScore \leftarrow 0$
    if $SameLocation(Output, GoldStandardDish)$ then
        $IncreaseScore(ContainerScore)$
    else
        $DecreaseScore(ContainerScore)$
    end if
    for every $Property$ in $PropsOf(ContainerOf(GoldStandardDish))$ do
        if $hasProperty(ContainerOf(Output), Property)$ then
            $IncreaseScore(ContainerScore)$
        else
            $DecreaseScore(ContainerScore)$
        end if
    end for

    $PredIngredients \leftarrow Unfold(Contents(Output), None)$
    $GSIngredients \leftarrow Unfold(Contents(GoldStandardDish), None)$
    $ContentsScores \leftarrow \emptyset$
    for every $GSIngredient$ in $GSIngredients$ do
        $MaxBaseScore \leftarrow 0$
        $MaxBaseIngredient \leftarrow \emptyset$
        for every $PredIngredient$ in $PredIngredients$ do
            $IngScore \leftarrow PropertyOverlap(PredIngredient, GSIngredient)$
            $SeqScore \leftarrow HierarchyOverlap(PredIngredient, GSIngredient)$
            $BaseScore \leftarrow 0.60 \times IngScore + 0.40 \times SeqScore$
            if $BaseScore > MaxBaseScore$ then
                $MaxBaseScore \leftarrow BaseScore$
                $MaxBaseIngredient \leftarrow PredIngredient$
            end if
        end for
        if $MaxIngScore > 0$ then
            $GSIngredients \leftarrow GSIngredients \setminus \{GSIngredient\}$
            $PredIngredients \leftarrow PredIngredients \setminus \{MaxBaseIngredient\}$
        end if
        $ContentsScores \leftarrow ContentsScores \cup \{MaxBaseScore\}$
    end for
    for $Length(GSIngredients) + Length(PredIngredients)$ times do
        $ContentsScores \leftarrow ContentsScores \cup \{0\}$
    end for

    $CurrentScore \leftarrow 0.02 \times ContainerScore + 0.98 \times AVG(ContentsScores)$
    if $CurrentScore > MaxScore$ then
        $MaxScore \leftarrow CurrentScore$
    end if
end for
return $MaxScore$
```

Unfold(Dish)

parameters:
Dish - the dish whose contents should be unfolded into base ingredients

implementation:

```
$Unfolded \leftarrow \emptyset$
for every $Portion$ in $Contents(Dish)$ do
    $Unfolded \leftarrow Unfolded \cup UnfoldIngredient(Portion, [])$
end for

$Merged \leftarrow \emptyset$
for every $BaseIng1$ in $Unfolded$ do
    for every $BaseIng2$ in $Unfolded \setminus \{BaseIng1\}$ do
        if $Similar(BaseIng1, BaseIng2)$ then
            $Amount(BaseIng1) \leftarrow Amount(BaseIng1) + Amount(BaseIng2)$
            $Unfolded \leftarrow Unfolded \setminus \{BaseIng2\}$
        end if
    end for
    $Merged \leftarrow Merged \cup \{BaseIng1\}$
    $Unfolded \leftarrow Unfolded \setminus \{BaseIng1\}$
end for

return $Merged$
```

UnfoldIngredient(Ingredients, Hierarchy)

parameters:
Ingredient - a single ingredient which might need additional unfolding (in case it is an aggregate food product)
Hierarchy - an ordered list of aggregate food products in which *Ingredient* is used

implementation:

```
if $Ingredient$ is an aggregate food product then
    return $UnfoldIngredient(Ingredient,\ [Ingredient]\ +\ Hierarchy)$
else
    return $(Ingredient, Hierarchy)$
end if
```

References

Cai, S., & Knight, K. (2013, 08). Smatch: an Evaluation Metric for Semantic Feature Structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 748–752). Sofia, Bulgaria.

De Haes, R. (2023). *A benchmark for recipe understanding in autonomous agents* (Unpublished master's thesis). Vrije Universiteit Brussel, Brussels, Belgium.

Loetzsch, M., Wellens, P., De Beule, J., Bleys, J., & van Trijp, R. (2008). *The Babel2 Manual* (Tech. Rep. No. AI-Memo 01-08). Brussels, Belgium: AI-Lab VUB.