Strategies

Notes related to design problems and strategies.

Z3

Z3 looks for models that satisfy a set of specified constraints.

1. Input parameters determine constraints on frames, models, and worlds.
2. The semantics for the language will impose a number of further constraints.
3. Each sentence of the language being interpreted will determine an additional constraint on models and worlds.
4. Z3 models that satisfy the general constraints and sentence constraints may then be stored in an output file.

Input Parameters

Variables to be specified by the user include:

1. Let `N` be the number of atomic states under consideration.
2. Let `Atoms` be a list of sentence letters, e.g., `{A,B,C,D}`.
3. Let `Gamma` be a list of infix sentences with sentence letters in `Atoms`.

Glossary

These definitions indicate the intended meanings of the elements employed below.

1. `State(x)` includes the predicate `State` whose extension is to be determined by Z3 given the constraints below.
2. `Or(x,y)` indicates the bitwise operation which takes the greatest value for each index of the bit vectors `x` and `y`.
3. `Possible(x)` includes the predicate `Possible` whose extension is to be determined by Z3 given the constraints below.
4. `Proposition(X)` is a set of Z3 constraints that require `X` to be a proposition.
5. `Semantics(w,X)` is a set of Z3 constraints that require `X` to be true in `w`.
6. `Alternative(u,w,X)` is a set of constraints that require `u` to be a (perhaps non-unique) result of minimally changing `w` to include a verifier for `X`.
7. `Prefix(X)` and `Infix(X)` will translate between prefix and infix notations.

Frame Constraints

The following constraints only depend on `N`, and are always to be included.

1. If `State(x)`, then `BitVec(x,N)` i.e., 'x is a bitvector of length N'.
2. If `State(x)` and `State(y)`, then `State(Or(x,y))`.
3. If `Possible(x)`, then `State(x)`.
4. If `Possible(x)`, `State(y)`, and `Or(x,y) = x`, then `Possible(y)`.

Model Constraints

The following constraints depend on the functions `Proposition()`, `Semantics()`, and `Prefix()` defined in the following section for Python.

1. For each sentence letter `X` in `Atoms`, include the output of `Proposition(X)`.
2. `Possible(w)` and `Or(x,w) = w` whenever `Possible(x)` and `Possible(Or(x,w))`.
3. For each sentence `X` in `Gamma`, include the output of `Semantics(w,Prefix(X))`.

Whereas (1) assigns sentence letters to a propositions, (2) requires `w` to be a world, and (3) requires each sentence included in `Gamma` to be true at `w`.

Python

Python will provide translations between readable sentences and Z3 model constraints as well as between Z3 models and readable model structures.

Syntax

Given a set of sentence letters `Atoms` and set of operators `Opts`, we may define the set of infix sentences of the language.

1. To avoid further translation, use LaTeX commands for the operators `\neg`, `\wedge`, `\vee`, `\rightarrow`, `\Box`, and `\boxarrow` in `Opts`.
2. Every sentence letter `X` in `Atoms` is an infix sentence.
3. If `X` and `Y` are infix sentences, then `\neg X`, `X \wedge Y`, `X \vee Y`, `X \rightarrow Y`, `\Box X`, and `X \boxright Y` are also infix sentences.

Infix/Prefix Translators

Define translation functions to convert between infix and prefix notation.

1. Given a sentence `A` in infix notation (e.g., `(A \wedge B) \boxright C`), the `Prefix(A)` function will output a unique sentences in prefix notation using lists (e.g., `[\boxright, [\wedge, A, B]]`).
2. Given a sentence `A` in prefix notation, the `Infix(A)` function will output a unique sentence in infix notation.
3. For any sentence `A` in infix notation, `A = Infix(Prefix(A))`.

Proposition Constraints

Given any sentence letter `X` in `Atoms`, the function `Proposition(X)` yields the following Z3 constraints as outputs:

1. If `Verify(x,X)` or `Falsify(x,X)`, then `State(x)`.
2. If `Verify(x,X)` and `Verify(y,X)`, then `Verify(Or(x,y,X))`.
3. If `Falsify(x,X)` and `Falsify(y,X)`, then `Falsify(Or(x,y,X))`.
4. If `Verify(x,X)` and `Falsify(y,X)`, then not `Possible(Or(x,y))`.
5. If `Possible(x)`, then `Possible(Or(x,y))` where either `Verify(y,X)` or `Falsify(y,X)`.

The constraints on `Verify` and `Falsify` may be extended to complex prefix sentences by way of the following general constraints:

1. If `Verify(z,[\neg, X])`, then `Falsify(z,X)`.
2. If `Falsify(z,[\neg, X])`, then `Verify(z,X)`.
3. If `Verify(z,[\wedge, X,Y])`, then `z = Or(x,y)` where `Verify(x,X)` and `Verify(y,Y)`.
4. If `Falsify(z,[\wedge, X,Y])`, then `Falsify(z,X)` or `Falsify(z,Y)` (or both).
5. If `Verify(z,[\vee, X,Y])`, then `Verify(z,X)` or `Verify(z,Y)` (or both).
6. If `Falsify(z,[\vee, X,Y])`, then `z = Or(x,y)` where `Falsify(x,X)` and `Falsify(y,Y)`.
7. If `Verify(z,[\rightarrow, X,Y])`, then `Falsify(z,X)` or `Verify(z,Y)` (or both).
8. If `Falsify(z,[\rightarrow, X,Y])`, then `z = Or(x,y)` where `Verify(x,X)` and `Falsify(y,Y)`.

World Alternatives

Given a world `w` and sentence `X`, the function `Alternatives(u,w,X)` yields the following Z3 constraints:

1. `State(x)` where `Verify(x,X)` and `Or(x,u) = u`.
2. `State(y)` where `Or(y,w) = w` and `Or(y,u) = u`.
3. If `State(z)`, `Or(z,w) = w`, `Possible(Or(x,z))`, and `Or(y,z) = z`, then `y = z`.

Semantic Constraints

The function `Semantics(w,X)` may be defined recursively as follows:

1. If `X` is in `Atoms`, then `Semantics(w,X)` is: `Verify(x,X)` and `Or(x,w)=w`.
2. If `X = [\neg,Y]`, then `Semantics(w,X)` is: not `Semantics(w,Y)`.
3. If `X = [\wedge,Y,Z]`, then `Semantics(w,X)` is: `Semantics(w,Y)` and `Semantics(w,Z)`.
4. If `X = [\vee,Y,Z]`, then `Semantics(w,X)` is: `Semantics(w,Y)` or `Semantics(w,Z)`.
5. If `X = [\Box,Y]` and neither `\Box` nor `\boxright` occur in `Y`, then `Semantics(w,X)` is: `Semantics(u,Y)` whenever `World(u)`.
6. If `X = [\boxright,Y,Z]` and neither `\Box` nor `\boxright` occur in `Y`, then `Semantics(w,X)` is: `Semantics(u,Z)` whenever `Alternative(u,w,Y)` and `World(u)`.

Model Builder

Z3 models will be translated back into a natural representation.

1. The python representation of a Z3 model should specify the states, possible

states, worlds, and the propositions assigned to the sentence letters in question where further visualization can be added later.

2. A function should specify the propositions assigned to all subsentences of the sentences under evaluation.
3. These details may then be stored in an output file, prompting the user whether to search for another model to add to the file.