JAIST Repository
https://dspace.jaist.ac.jp/

Title	CoqによるBBSLの形式化と検証
Author(s)	宇田, 拓馬
Citation	
Issue Date	2021-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/17083
Rights	
Description	Supervisor:青木 利晃, 先端科学技術研究科, 修士（情報科学）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修士論文

Coq による BBSL の形式化と検証

宇田　拓馬

主指導教員 青木　利晃

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和３年３月

Abstract

In recent years, technological development aimed at the practical application of autonomous driving has been actively carried out. Safety evaluation has become an important issue in autonomous driving systems. In addition, it is difficult to define safety requirements because it is large-scale and complicated, and various driving environments are assumed. Therefore, research is being conducted by the ministries and agencies of each country to define safety standards by systematizing autonomous vehicles and their surrounding environments. However, in those studies, the specifications are described using figures and natural language, which causes ambiguity in the content. Therefore, the meaning of the specification is not uniquely determined, and it is difficult to verify the safety. Formal specification is a method for describing specifications strictly without ambiguity. Since the meaning of the specification is strictly defined in the formal specification, its unambiguity can be guaranteed. However, in general formal specification languages such as Z and VDM, abstract description using figures and natural language is difficult. Therefore, the Bounding Box Specification Language (BBSL) has been proposed by previous research as a formal specification language for images in autonomous driving systems. BBSL is an original extension of the interval arithmetic system, and the focus is on formally describing the positional relationship of objects on an image using a Bounding box represented by a two-dimensional interval.

The purpose of this study is to improve the quality of the specifications described in BBSL by formalizing BBSL using Coq and verifying its language specifications. Since high safety is required for autonomous driving systems, high quality is required for BBSL, which is the language that describes the specifications. By formalizing BBSL, it becomes possible to describe reliable specifications. Coq, a theorem proving support system, is used to operate BBSL on a computer.

A language can be formalized by giving formal semantics to the syntax of the language. In this study, Coq is used to formalize BBSL. In other words, it is necessary to express BBSL on Coq. There are shallow embedding and deep embedding as methods for implementing a language on another language. In shallow embedding, the target language is evaluated by the semantics of the implementation language. Implementation is easy as an advantage, but the target language may not be implemented due to the limitation of the semantics of the implementation language. In deep embed-

ding, evaluation is performed by implementing semantics in the abstract syntax of the target language. The advantage is that the semantics of the target language can be freely given, but the disadvantage is that the implementation becomes complicated. In this study, deep embedding is adopted to give BBSL a formal semantics. To formalize BBSL with deep embedding using Coq, first define the abstract syntax of BBSL. Second, we define a semantic function that gives semantics by associating mathematical objects with abstract syntax. In addition, since BBSL extends the interval system independently, it is necessary to define mathematical objects as well. Third, implement these on Coq.

Evaluation experiments will be conducted on the formalized BBSL from the following three perspectives. The first is to test the formalized BBSL relationships / functions to confirm that they are defined as intended. Mathematical properties are adopted for the test policy. In other words, if it is an inclusion relationship of an interval, the property of half order is proved. The second is confirmation of the descriptive ability of formalized BBSL. The BBSL study described the specifications compiled by NHTSA to confirm descriptive ability. By describing the same specifications in the formalized BBSL, make sure that it has the same description capability as the original BBSL. The third evaluates the proof ability of formalized BBSL. By proving the practical property of case block completeness, we confirm the proof ability of formalized BBSL.

From the experiments, all the properties that should be satisfied for the relations and functions of BBSL were proved. This allowed the formalization of BBSL to be defined as intended. In addition, most of the proofs were done in a small number of steps, around 10 steps. However, it took 61 steps to prove completeness, which is a practical property. The basic properties could be proved efficiently, but the practical properties were not easy to prove. It is considered necessary to review the definitions of relationships and functions. Regarding the description experiment of BBSL, it was found that the formalized BBSL has the same description ability as the original BBSL. In addition, from the experimental results, the amount of description in BBSL and the amount of description in formalized BBSL were almost clear. However, what is written in formalized BBSL is an abstract syntax, and the amount of description should be small. Therefore, it is considered that the amount of description is increasing overall. In the experiment to prove the completeness, the completeness was described as a

mathematical formula and then described using the formalized BBSL. However, due to the convenience of implementation by Coq, there was a large discrepancy between the mathematical formula and the description in Coq. Therefore, the content was not such that the completeness could be described intuitively. A future issue is the implementation of a parser. This is because it is not realistic to directly describe the abstract school branch. This can be automatically generated by implementing a parser. Another problem is that it is not easy to describe the properties verified for the specifications. It is thought that this problem can be solved by defining a dedicated assertion language that describes the verified properties. In addition, by proposing a method for automatically verifying the properties described in the expression language, it is possible to verify the specifications using BBSL. It will be easier and will lead to higher quality of description specifications.

目次

図目次

表目次

第 1 章

はじめに

1.1 背景

近年、自動運転の実用化を目指した技術開発が盛んに行われている。自動運転システムは人命に直接的に関わるため、安全性の評価が重要課題となる。また、大規模かつ複雑であり、様々な走行環境が想定されることから安全性に関する要件定義が難しい。そこで、各国の監督省庁によって自動運転車両とその周辺環境を体系化することで安全性の基準を定義する研究がなされている。米国の国家道路交通安全局（以降 NHTSA）による研究 [2] ドイツ経済エネルギー省が主導するペガサスプロジェクト [3]、日本自動車研究所（以降 JARI）の研究 [4] が上げられる。これらの研究では自動運転システムの仕様は自動運転車両とその走行環境に基づき図や自然言語を用いて記述されている。

しかし、先に述べたように走行環境は種々様々なためあらゆる状況を詳細に記述することは非常に困難となる。そのため、自動運転システムの仕様は適切な抽象度で記述される必要がある。また、図と自然言語を用いて記述された仕様は内容に曖昧性が発生するため、仕様の意味が一意に定まらず、安全性の検証が困難となる。仕様を曖昧さなく厳密に記述するための手法として形式仕様記述が上げられる。形式仕様記述では仕様の意味を厳密に定めるため、その非曖昧性が保証できる。また、記述した仕様に対して条件の網羅性や排他性といった重要な性質の判定に利用することができる。しかし、Z 言語 [5] や VDM[6] に代表される一般的な仕様記述言語では図と自然言語を用いた抽象的な仕様の記述が困難となっている。そこで、自動運転システムにおける画像を対象とした形式仕様記述言語である Bounding Box Specification Language（以降 BBSL）が先行研究 [7] によって提案されている。BBSL では区間演算の体系 [8] を独自に拡張しており、２次元の区間で表現される bounding box を用いて画像上のオブジェクト同士の位置関係を形式的

に記述することに主眼が置かれている。

BBSL で記述された仕様が信頼できるものであるためには、BBSL の言語仕様に問題があってはならない。BBSL の言語仕様に問題がないことを確かめるため、BBSL を形式化する必要がある。つまり、BBSL に形式的な意味論を定義して与える必要がある。

1.2 目的

本研究では Coq[1] を用いて BBSL の形式化を行いその言語仕様を検証することで、BBSL で記述された仕様の品質を向上させることを目的とする。自動運転システムでは高い安全性が求められるため、その仕様を記述する言語である BBSL には高い品質が求められる。また、BBSL の品質を保証することで BBSL で記述された仕様の信頼性を向上させることができる。BBSL を形式化し厳密な意味論を与えることで、BBSL の品質を保証することができる。また、BBSL を形式化することで BBSL で記述した仕様の品質を評価することができるようになる。本研究では BBSL の形式化および検証を行うため、定理証明支援系の Coq を用いる。Coq を用いることにより、豊富な Coq の資産を BBSL の検証に活用することができる。Coq を用いて BBSL を実装することで、BBSL を Coq のシステムの上で正しく形式化することができる。また、BBSL の処理系の実装を得られる。処理系は BBSL で記述した仕様の検証に利用できるため、仕様の品質を向上させることに繋がる。

第 2 章

関連研究

G. Melquiond らの研究 [8] では区間演算の体系が Coq を用いて形式化されている。一般的に区間は上界・下界を計算し、実数値の不等式を証明するために利用される。上界・下界を計算することで、証明項のサイズを効率的に縮小することができる。この研究では浮動小数点演算に基づいて、自動微分と区間演算を組み合わせた効率的で信頼されたソルバーが提案されている。自動微分は基本的な演算と関数を用いて偏導関数を計算する手法で、小さい計算量で自動的に求めることができる。また、ソルバーを Coq のライブラリとして実装し、不等式の証明のためのタクティクを提供している。タクティクとは証明項を自動解決する Coq の機能である。これによって、Coq で記述された実数値の不等式の命題について証明項のサイズを自動的に縮小しより簡単に証明することができるようになる。BBSL では区間が扱われるが、既存の区間演算が実数値計算に利用されるのに対し、BBSL では物体同士の位置関係を表現するために利用されている。そのためにいくつかの演算が追加されている。よって BBSL の形式化では区間の形式化について既存研究との差異が出てくる。

物体の位置関係を表現する論理体系として空間様相論理が挙げられる。空間様相論理では空間的な関係を表現する様相演算子が導入されている。$S4_u$ は様相論理の公理系 $S4$ を位相的意味論で解釈したもので、原子命題を移送空間内の図形、開核作用素と閉包作用素を洋装演算子として解釈する。先行研究 [9] では空間について推論するための論理として Region Connection Calsulus (以下 RCC) が提案されている。RCC では領域とその結合関係を用いて空間的関係を記述する。領域とは有限次元のベクトル空間の開部分集合で連結なもののことをいう。RCC には既存の空間論理と比較していくつかの利点がある。一つは通常の領域と、半開部分集合でかつ連結な半開領域、閉部分集合でかつ連結な閉領域を明示的な区別なく扱うことができることである。二つ目は、定義された述語が少なく、

公理が少ないことである。三つ目は、同じ定理の場合より小さい式での記述が可能なことである。RCC-8[10] は RCC の拡張であり、空間的関係として表 2.1 のものが定義されている。また、RCC-8 ではユークリッド空間だけではなく位相空間上の空間的関係を扱うことができる。

これらの研究ではユークリッド空間または位相空間上での位置関係について記述できる論理体型が定義されている。一方で、BBSL では物体の位置関係に限らない画像上の仕様を表現することができる。BBSL ではオブジェクトは 2 次元の区間で表現される Bounding box によって記述され, また BBSL で扱われる区間は位置関係を記述するため独自の拡張を行っているため既存の区間演算の体系 [8] とは異なる。さらに、BBSL ではオブジェクトとして区間と Bounding box の集合も扱うことができる。そのため、これらのオブジェクト間の関係も記述できる必要がある。

記法	説明
$DC(X, X)$	X と Y は離れている (disconnected)
$EC(X, X)$	X は Y と外部で接している (externally connected)
$PO(X, X)$	X は Y と部分的に重なっている (partially overlapping)
$EQ(X, X)$	X と Y は等しい (identical)
$TPP(X, X)$	X は Y と内部で接している (tangential proper part)
$TPP^{-1}(X, X)$	Y は X と内部で接している
$NTPP(X, X)$	X は Y の内部にあり、接していない (nontangential proper part)
$NTPP^{-1}(X, X)$	Y は X の内部にあり、接していない

表 2.1　RCC-8 の結合関係

第 3 章

準備

本研究では Coq を用いて BBSL を形式化する。BBSL では区間を独自に拡張した体系が用いられるため、まず一般的な区間とその演算の定義について説明する。BBSL で扱われる区間は閉区間のみであるため、閉区間についてのみ説明を行う。次に、BBSL の文法について仕様の記述例を用いながら説明する。最後に、定理証明支援系である Coq ついて説明する。

3.1 区間

■定義 閉区間 X は実数の集合として式 (3.1) のように与えられる。また、閉区間の左、右端点をそれぞれ $\underline{X}$、$\overline{X}$ と表記する。以下、単に区間と言った場合閉区間を指すものとする。$\underline{X} > \overline{X}$ のとき空区間として扱う。

$$X = [\underline{X}, \overline{X}] = \{x \in \mathbb{R} : \underline{X} \leq x \leq \overline{X}\} \tag{3.1}$$

■共通部分、合併、Interval Hull 二つの区間 X と Y の共通部分は、式 (3.2) のように定義される。共通部分は $\overline{Y} < \underline{X}$ もしくは $\overline{X} < \underline{Y}$ のとき空集合となる。

$$\begin{aligned} X \cap Y &= \{z : z \in X \wedge z \in Y\} \\ &= [\max\{\underline{X}, \underline{Y}\}, \min\{\overline{X}, \overline{Y}\}] \end{aligned} \tag{3.2}$$

区間 X と Y の合併は式 (3.3) のように定義される。

$$X \cup Y = \{z : z \in X \vee z \in Y\} \tag{3.3}$$

一般的に区間同士の合併は区間にはならない。しかし、式 (3.4) で定義される Interval Hull はいつでも区間を返す。

$$X \underline{\cup} Y = [\min\{\underline{X}, \underline{Y}\}, \max\{\overline{X}, \overline{Y}\}] \tag{3.4}$$

また、二つの区間 X, Y について式 (3.5) が成り立つ。

$$X \cup Y \subseteq X \underline{\cup} Y \tag{3.5}$$

■幅、絶対値、中間点　区間 X の幅は式 (3.6) のように記述され、定義される。

$$w(X) = \overline{X} - \underline{X} \tag{3.6}$$

区間 X の絶対値は $|X|$ のように記述され、端点の絶対値のうち最大のものとして式 (3.7) のように定義される。

$$|X| = max\{|\underline{X}|, |\overline{X}|\} \tag{3.7}$$

区間 X の中間点は式 (3.8) のように与えられる。

$$m(X) = \frac{1}{2}(\underline{X} + \overline{X}) \tag{3.8}$$

■順序関係、同値関係　区間 X と Y の順序関係は実数の順序関係と同じく記号 $<$ で記述し、式 (3.9) のように定義される。

$$X < Y = \overline{X} < \underline{Y} \tag{3.9}$$

区間の順序関係では実数の順序関係と同じく式 (3.10) のように推移律が成り立つ。

$$A < B \wedge B < C \Rightarrow A < C \tag{3.10}$$

また、式 (3.11) で与えられる集合の包含関係もまた区間の順序関係となる。

$$X \subseteq Y \Leftrightarrow \underline{Y} \leq \underline{X} \wedge \overline{X} \leq \overline{Y} \tag{3.11}$$

区間 X と Y の同値関係は式 (3.12) のように定義される。

$$X = Y \Leftrightarrow \underline{X} = \underline{Y} \wedge \overline{X} = \overline{Y} \tag{3.12}$$

■演算　二つの区間 X、Y の和、差、積、商はそれぞれ式 (3.13)、(3.14)、(3.15)、(3.16) のように与えられる。商については $0 \in Y$ とする。

$$X + Y = \{x + y : x \in X, y \in Y\} \tag{3.13}$$

$$X - Y = \{x - y : x \in X, y \in Y\} \tag{3.14}$$

$$X \times Y = \{x \times y : x \in X, y \in Y\} \tag{3.15}$$

$$X/Y = \{x/y : x \in X, y \in Y\} \tag{3.16}$$

■多次元の区間　X_i（$i = 1, ..., n$）を区間として、多次元区間 X は式 (3.17) のように定義される。

$$X = (X_1, ..., X_n) \tag{3.17}$$

また、区間の各関数と演算は多次元の区間についても定義することができる。所属、共通部分、包含関係、幅、中間点、絶対値は実数ベクトルを $x = (x_1, ..., x_n)$ としてそれぞれ式 (3.18)、(3.19)、(3.20)、(3.21)、(3.22)、(3.23) のように与えられる。

$$x \in X \Leftrightarrow \text{任意の } i \text{ について } x_i \in X_i \tag{3.18}$$

$$X \cap Y = (X_1 \cap Y_1, ..., X_n \cap Y_n) \tag{3.19}$$

$$X \subseteq Y \Leftrightarrow \text{任意の } i \text{ について } X_i \subseteq Y_i \tag{3.20}$$

$$w(X) = max\{w(X_1), ..., w(X_n)\} \tag{3.21}$$

$$m(X) = (m(X_1), ..., m(X_n)) \tag{3.22}$$

$$||X|| = max\{|X_1|, ..., |X_n|\} \tag{3.23}$$

3.2　BBSL

BBSL[7] では自動運転車両の走行環境を区間や Bounding box を用いて記述し、それらに対して自動運転車が取るべき行動とその条件を記述する。例えば、「車線を左に変更する」という行動に「左車線に他車が存在しない」という条件を記述する。BBSL による仕様の記述例として、図 3.1 の前方カメラの画像について、図 3.2 のような仕様を考える。図 3.2 は前方車両が減速区間より近ければ停止し、遠ければ停止しないという仕様を図と自然言語を用いて記述したものである。これらの仕様を BBSL で記述したものが図 3.3 である。以下では BBSL の文法について、この例を用いて説明を行う。

BBSL はいくつかの型と関係・関数を持っている。また、BBSL の構文は外部関数ブロック、前提条件ブロック、ケースブロックの 3 つのブロックに分けられる。第一に、BBSL のもつ型について例を用いて説明する。第二に、BBSL の基本的な関係・関数について例を用いて説明する。第三に、BBSL の構文について例を用いて説明する。

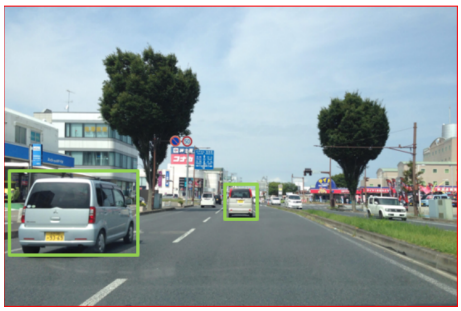

図 3.1　前方カメラからの画像

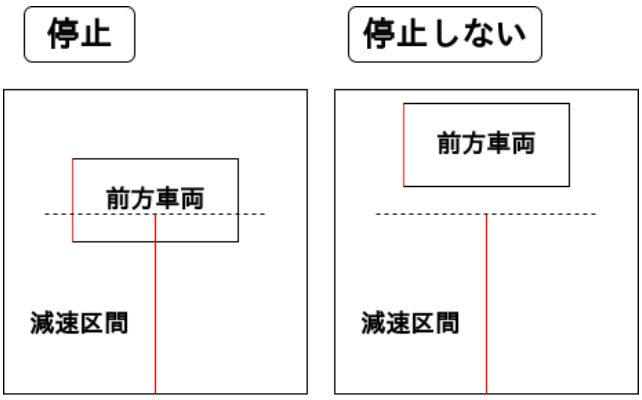

図 3.2　仕様の例

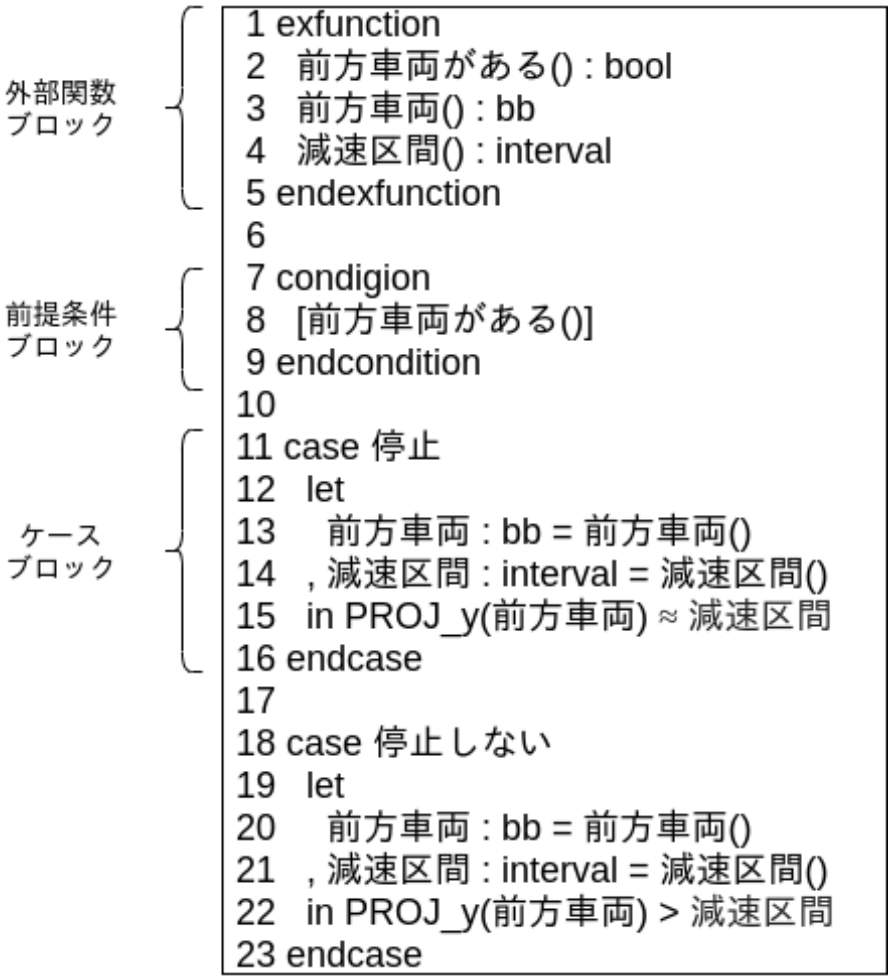

図 3.3　BBSL での仕様の記述例

3.2.1　型

BBSL は表 3.1 のような型をもっている。

実数型とブール型については一般的なものである。区間の定義は式 (3.1) である。Bounding box は、2 次元の区間として定義される。多次元区間の定義 (3.17) より、2 つの区間 X、Y を用いて式 (3.24) のように表される。

$$A = (X, Y) \tag{3.24}$$

BBSL では画像上のオブジェクトを区間や Bounding box を用いて表現する。例として、図 3.3 の 2 行目では、関数「前方車両がある ()」の型をブール値型 bool としている。

型	説明
real	実数型
bool	ブール値型
interval	区間型
bb	Bounding box 型
setBB	Bounding box の集合型

表 3.1　BBSL の基本的な型

同じく 3 行目では関数「前方車両 ()」の型を Bounding box 型 bb としている。つまり、図 3.2 の前方車両を表す長方形を Bounding box として表現している。4 行目では「減速区間 ()」の型を区間型 interval としている。これは図 3.2 で自動運転車両と前方車両との車間距離について減速するべき範囲を示した直線があったが、これを区間として表現している。

3.2.2　基本的な関係・関数

BBSL には表 3.2 のような関係・関数が用意されている。

■区間の関数　区間の関数として id9、id10 がある。id9 は区間の幅を実数値として求める関数である。区間演算で用いられるものと同じで、式 (3.6) で定義される。図 3.3 の 22 行目では、図 3.2 の前方車両を表す長方形の y 軸方向の区間の方が減速区間を表す区間より大きい、つまり上側にあることを記述している。id10 は 2 つの区間の共通部分を求める関数である。区間演算で用いられるものと同じで、式 (3.2) で定義される。例として自動運転車の前方カメラからの画像を簡略化した図 3.4 を考える。これは自動運転車の前方に障害物があるときの図である。この図において自動運転車が停止するべき条件は、BBSL では図 3.5 のように記述できる。記述では Bounding box である障害物検知領域と障害物の共通部分が空集合ではないことを停止する条件としている。

■Bounding box の関数　Bounding box の関数として id4、id4' がある。id4 は Bounding box から区間を取り出す射影関数である。Bounding box は図形的には長方形を表す。長方形の x 軸方向の区間を取り出す関数が $PROJ_x$、y 軸方向の区間と取り出す関数が $PROJ_y$ である。$PROJ_x$、$PROJ_y$ はそれぞれ式 (3.25)、(3.26) で定義される。A は Bounding box、X と Y は区間とする。図 3.3 の 15 行目では、前方車両を表す Bounding

id	関係・関数	記法	型
1	区間の比較関係	$<, >, =$	$interval * interval \rightarrow bool$
2	区間の重なり	$\approx$	$interval * interval \rightarrow bool$
3	区間の包含関係	$\subset, \subset, supset$	$interval * interval \rightarrow bool$
4	Bounding box から区間への射影	$PROJ_i$	$bb \rightarrow interval\ (i \in \{x, y\})$
4'	Boundig box から実数への射影	$PROJ_i$	$bb \rightarrow \mathbb{R}\ (i \in \{\underline{x}, \overline{x}, \underline{y}, \overline{y}\})$
5	Bounding box の重なり	$\approx$	$bb * bb \rightarrow bool$
6	否定	not	$bool \rightarrow bool$
6'	論理積、論理和	and, or	$bool * bool \rightarrow bool$
7	RAT 関数	RAT	$setBB * setBB \rightarrow real$
8	実数の比較関係	$<, >, =$	$real * real \rightarrow real$
9	区間の幅	w	$interval \rightarrow real$
10	区間の共通部分	cap	$interval \rightarrow real$
11	Bounding box の集合の共通部分、合併	cap, cup	$setBB * setBB \rightarrow setBB$

表 3.2　BBSL の関係・関数

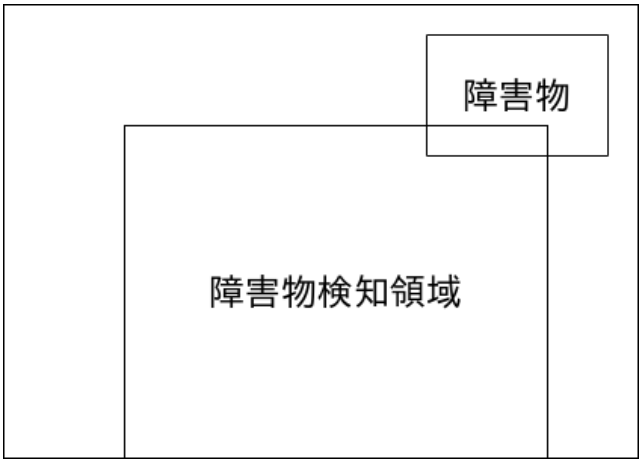

図 3.4　仕様の例　障害物の検知

box の y 軸方向の区間を取り出して、減速区間との重なりを判定している。

$$\begin{aligned} PROJ_x(A) &= PROJ_x((X, Y)) \\ &= X \end{aligned} \tag{3.25}$$

```
1 case 停止する
2   let
3     障害物検知領域: bb = 障害物検知領域()
4   , 障害物 : bb = 障害物()
5   in not(障害物検知領域 ∩ 障害物 = {})
6 endcase
```

図 3.5　仕様の例　障害物の検知の BBSL による記述

$$\begin{aligned} PROJ_x(A) &= PROJ_x((X,Y)) \\ &= Y \end{aligned} \tag{3.26}$$

id4' は Bounding box の x 軸、y 軸方向の区間の端点を取り出す射影関数である。$PROJ_{\underline{x}}$、$PROJ_{\overline{x}}$、$PROJ_{\underline{y}}$、$PROJ_{\overline{x}}$ はぞれぞれ x 軸方向の区間の左端点、x 軸方向の区間の右端点、y 軸方向の区間の左端点、y 軸方向の区間の右端点を取り出す。それぞれ式 (3.27)、(3.28)、(3.29)、(3.30) で定義される。

$$\begin{aligned} PROJ_{\underline{x}}(A) &= PROJ_x l((X,Y)) \\ &= PROJ_x l(([\underline{X},\overline{X}],Y)) \\ &= \underline{X} \end{aligned} \tag{3.27}$$

$$\begin{aligned} PROJ_{\overline{x}}(A) &= PROJ_x l((X,Y)) \\ &= PROJ_x l(([\underline{X},\overline{X}],Y)) \\ &= \overline{X} \end{aligned} \tag{3.28}$$

$$\begin{aligned} PROJ_{\underline{y}}(A) &= PROJ_x l((X,Y)) \\ &= PROJ_x l((X,[\underline{Y},\overline{Y}])) \\ &= \underline{Y} \end{aligned} \tag{3.29}$$

$$\begin{aligned} PROJ_{\overline{y}}(A) &= PROJ_x l((X,Y)) \\ &= PROJ_x l((X,[\underline{Y},\overline{Y}])) \\ &= \overline{Y} \end{aligned} \tag{3.30}$$

■Bounding box の集合の関数　Bounding box の集合の関数として id7、id11 がある。id7 は BBSL に特有の関数で、２つの Bounding box の集合の総面積の比を求める。式 (3.31) のように定義されている。A、B は Bounding box の集合である。Bounding box の集合の総面積を求める方法については定義されていない。

$$RAT(A, B) = (\text{A が占める総面積})/(\text{B が占める総面積}) \tag{3.31}$$

id11 は２つの Bounding box の集合の共通部分と合併である。合併については一般的な集合と同様に、式 (3.32) のように定義する。

$$A \cup B = \{X | X \in A \vee X \in B\} \tag{3.32}$$

共通部分については BBSL 特有の定義がなされており、式 (3.33) で表される。

$$A \cap B = \{x \cap y | x \in X, y \in Y\} \backslash \{\emptyset\} \tag{3.33}$$

■ブール値の関数　id6、id6' は一般的なブール値の演算である。id6 は否定、id7 は論理積、論理和を表す。

■区間の関係　区間同士の位置関係を記述する関係として id1、id2、id3 がある。id1 は順序関係と同値関係で、区間 X、Y について $X < Y$ のとき、x 軸方向なら X は Y より左側、y 軸方向なら X は Y より下側にあることを表現する。また、$X = Y$ のとき同じ区間であることを表す。順序関係と同値関係はそれぞれ式 (3.9)、(3.12) で定義される。図 3.3 の 22 行目では、前方車両の y 軸方向の区間より減速区間の方が大きい、つまり上側にあることを停止しない条件として記述している。id2 は BBSL 特有の関係であり、区間同士が重なっていることを判定する関数である。式 (3.34) で定義される。つまり、２つの区間の共通部分が空であることを判定している。図 3.3 の 15 行目では、前方車両の y 軸方向の区間と減速区間が重なっていることを停止する条件として記述している。

$$X \approx Y \Leftrightarrow X \cap Y \neq \emptyset \tag{3.34}$$

id3 は区間の包含関係を表現する関係である。式 (3.11) で定義される。区間 X と Y について、$X \subset Y$ なら区間 X は区間 Y の中に収まっている。

■Bounding box の関係　Bounding box 同士の位置関係を記述する関係として、id5 がある。id5 は 2 つの Bounding box の重なりを判定する関数であり、区間の重なりを判定する関係を用いて式 (3.35) のように定義される。Bounding box は 2 つの区間によって構成される長方形であるので、2 つの Bounding box の x 軸方向、y 軸方向の区間がそれぞれ重なっているなら Bounding box が重なっていると定義できる。

$$\begin{aligned} A \approx B &\Leftrightarrow (A_1, A_2) \approx (B_1, B_2) \\ &\Leftrightarrow A_1 \approx B_1 \wedge A_2 \approx B_2 \end{aligned} \tag{3.35}$$

■実数の関係　実数の関係として id8 がある。id8 は一般的な実数の比較関係と同値関係である。

3.2.3　構文

BBSL の構文は外部関数ブロック、前提条件ブロック、ケースブロックの 3 つのブロックに分けられる。BBSL の構文を拡張 BNF で記述したものをソースコード 3.1 に示す。image-spec が BBSL で記述する仕様全体を示す。external-function は外部関数ブロックを示し、図 3.3 では 1 行目から 5 行目に当たる。senario-condition は前提条件ブロックを示し、図 3.3 では 7 行目から 9 行目に当たる。case+ はケースブロックを示し、図 3.3 では 11 行目から 23 行目に当たる。ケースブロックは 1 つ以上のケースから構成される。

ソースコード 3.1　BBSL の構文の BNF

```
1 image-spec ::= external-function senario-condition case+
```

以下では外部関数ブロック、前提条件ブロック、ケースブロックについてそれぞれ図 3.3 を例に使いながら構文とその構成要素を説明する。

■外部関数ブロック　BBSL では抽象的な記述をするため、具体的な値は仕様には記述せず外部から受け取る形で記述する。仕様の外部から値を取得する関数を記述するのが外部関数ブロックである。値自体は外部から取得するため、関数の実態は記述せず、関数名と型のみを記述する。外部関数ブロックは記述仕様に現れる自由変数を予め宣言してお

くための機能と見ることもできる。外部関数ブロックの構文を拡張 BNF で記述したものをソースコード 3.2 に示す。external-function は外部関数ブロックを示す。外部関数ブロックは 0 個以上の関数定義から構成される。function-definition は関数定義を示す。関数定義は関数名、0 個以上の引数の型、返り値の型から構成される。function-name は関数名で任意の文字列が使える。type は型名であり、ブール値型の bool、実数型の real、区間型の interval、Bounding box 型の bb、Bounding box の集合型の setBB が使える。

ソースコード 3.2　外部関数ブロックの構文の BNF

```
1 external-function ::= "exfunction" function-definition* "endexfunction
      "
2 function-definition ::= function-name "(" type* ")" ":" type
3 type ::= "real" | "bool" | "interval" | "bb" | "setBB"
```

図 3.3 の 2 行目では、「前方車両がある」という bool 型の値を返す関数を定義している。図 3.3 の 3 行目では、前方車両を表す長方形、Bounding box を取得する関数を定義している。図 3.3 の 4 行目では、減速区間を表す区間を取得する関数を定義している。

■前提条件ブロック　前提条件ブロックでは、すべてのケースで満たされるべき条件を記述する。条件は bool 型の式として記述する。条件がない場合は none と記述する。前提条件ブロックの構文を拡張 BNF で記述したものをソースコード 3.3 に示す。senario-condition は前提条件ブロックを示す。前提条件ブロックは前提条件がない場合は none、ある場合はブール値の式となる。bexp はブール値の式である。

ソースコード 3.3　前提条件ブロックの構文の BNF

```
1 senario-condition ::= "condition" "[" condition-definition "]" "
      endcondition"
2 condition-definition ::= "none" | bexp
```

また、ブール値型の式の構文を拡張 BNF で記述したものをソースコード 3.4 に示す。ブール値の式はそれぞれの型の関係と、変数、関数呼び出し、論理和、論理積、否定、全称

量化、存在量化から構成される。var-name は変数を表す。value は値を表す。値は変数である var-name、実数リテラルである real-number、関数呼び出し function-call、二項演算 value binop value から構成される。function-call は関数呼び出しである。0 個以上の値を受け取り、何らかの値を返す。外部関数ブロックで定義した関数と、各種射影関数、RAT 関数、区間の幅を求める w 関数がある。構文上ではブール値型の式で呼び出された関数がブール値型の値を返すことを保証していない。condition-com はそれぞれの型の関係を表す。value conditino-com value は関係に値を適用した形であり、ブール値型の値を返す。関係の型と適用される値の型が一致することは構文上では保証されていない。

ソースコード 3.4　ブール値の式の構文の BNF

```
1  bexp ::= var-name
2    | function-call
3    | value condition-com value
4    | "not" condition-definition
5    | condition-definition "and" condition-definition
6    | condition-definition "or" condition-definition
7    | "forall" var-name "∈" value "(" condition-definition ")"
8    | "exists" var-name "∈" value "(" condition-definition ")"
9  conditoin-com ::= "=" | "<" | ">" | "≈" | "⊂" | "⊃" | "⊆" | "⊇"
10 function-call ::= function-name "(" value* ")"
11 function-name ::= "PROJ" proj-index | "RAT" | "w" | string
12 proj-index ::= "x" | "y" | "x̲" | "x̄" | "y̲" | "ȳ"
13 value ::= var-name
14   | real-number
15   | function-call
16   | (value binop value)
17 binop ::= "∩" | "∪"
```

図 3.3 では、7 行目から 9 行目で前提条件ブロックが記述されている。8 行目ではブール値型の値を返す関数「前方車両がある」が前提条件として呼び出されている。

■ケースブロック　ケースブロックは複数のケースから構成される。ケースの１つ１つは自動運転車が取るべき行動と、その行動が起こる条件を記述する。ケースブロックの構

文を拡張 BNF で記述したものをソースコード 3.5 に示す。case は 1 つのケースを表す。ケースはケースラベルである case-name と条件からなる。条件では let を用いて 1 個以上の変数を宣言することができる。変数宣言は変数名、変数の型名、割り当てる値から構成される。

ソースコード 3.5　ケースブロックの構文の BNF

```
1 case ::= "case" case-name case-definition "end-case"
2 case-definition ::= let-definition condition-definition
3 let-definition ::= "let" let-expr ("," let-expr)* "in"
4 let-expr ::= var-name ":" type "=" value
```

図 3.3 では 11 行目から 16 行目に停止のケース、18 行目から 23 行目に停止しないケース記述されている。つまり、この仕様では前方車両があるという前提で、ある条件を満たすときは自動運転車が停止し、ある条件を満たすときは停止しないということを記述している。15 行目の停止する条件は $PROJ_y$(前方車両) $\approx$ 減速区間 となっている。これは前方車両の y 軸方向の区間と減速区間が重なっていることを表現している。つまり、自動運転車が前方車両に近づきすぎていることを意味している。22 行目の停止しない条件は $PROJ_y$(前方車両) $>$ 減速区間 となっている。この式では前方車両との間に十分な車間距離が保たれていることを表現している。

3.3　Coq

Coq[1] はカリー＝ハワード同型対応を利用した定理証明支援系であり、プログラムに対して命題を記述し、証明を行うことができる。Coq はコア言語である Gallina と主に証明を記述する言語である Vernacular とに大きく分けられる。まず Gallina でプログラムを記述し、Vernacular で命題とその証明を記述していく形になる。Gallina は非常に協力な型システムをもった言語で、基本的に停止性のある関数しか記述することができない。そのため、再帰関数の扱いに関して非常にセンシティブである。

通常の関数を定義するためには Definition を利用する。Definition の記述例をソースコード 3.6 に示す。例は実数の加算である。

ソースコード 3.6　Definition の記述例

```
1 Definition add (a b : R) : R := a + b.
```

Coq で再帰関数を記述する場合、大きく 2 種類の方法が挙げられる。帰納的な関数と、余機能的な関数である。帰納的な関数は帰納的なデータ構造に対して記述され、余帰納的な関数は余帰納的なデータ構造に対して記述される。帰納的なデータ構造を定義するためには Inductive を利用する。帰納的な関数を定義するには Fixpoint を利用する。Inductive と Fixpoint の記述例をソースコード 3.7 に示す。例は自然数のリストの定義と、リスト同士を結合する帰納的関数の実装例である。

ソースコード 3.7　Inductive と Fixpoint の記述例

```
1 Inductive natlist :=
2   | Nil
3   | Cons (x : nat) (xs : natlist).
4
5 Fixpoint append (xs ys : natlist) :=
6   match xs with
7   | Nil => ys
8   | Cons x xs' => append xs' (Cons x ys)
9   end.
```

Coq で部分関数を扱う方法として、option を使う方法が挙げられる。option は型を一つ受け取って型を返す高カインド型で、定義はソースコード 3.8 のようになる。option は任意の型に例外的な値 None を導入するものと見ることができる。関数が定義されている場合は Some で結果を返し、未定義の場合は None を返すことで部分関数を表現する。

ソースコード 3.8　option の定義

```
1 Inductive option (A:Type) : Type :=
2   | Some : A -> option A
3   | None : option A.
```

Coq での証明の例として、$\forall AB, A \wedge B \to B \wedge A$ を証明する。Coq での命題とその証明の記述をソースコード 3.9 に示す。まず、論理積である and を定義している。and は命題を 2 つ受け取って命題を返すコンストラクタ conj をもっている。where 以下は糖衣構文の定義である。命題は Lemma コマンドを用いて記述する。and_comm が命題の名前で、コロン以下が命題の内容となる。Lemma 以外に Proposition や Theorem、Remark、Fact、Corollary などがあるが、機能的には同じである。Proof. 以下ではタクティクと呼ばれるコマンドを用いて証明を記述していく。例えば、destruct は仮定にある論理積を 2 つに分けることができる。つまり、論理和の除去が行える。intros は包含の導入が行える。

ソースコード 3.9　Coq による証明の例

```
1  Inductive and (A B:Prop) : Prop :=
2    conj : A -> B -> A /\ B
3  where "A /\ B" := (and A B) : type_scope.
4
5  Lemma and_comm : forall A B, A /\ B -> B /\ A.
6  Proof.
7    intros A B HAandB.
8    destruct HAandB as (HA & HB).
9    apply (conj HB HA).
10 Qed.
```

第 4 章

Coq による BBSL の形式化

言語は文字列によって構成される構文をもつ。言語の構文がどのように評価されるかを定めるものを意味論と呼ぶ。構文に形式的な意味論を与えることでその言語を形式化することができる。本研究では BBSL で記述された自動運転システムの仕様の品質を向上させるため、BBSL の形式化を行う。また、BBSL を計算機上で動作させるために定理証明支援系である Coq を用いる。

言語を他言語上で実装する手法として、shallow embedding と deep embedding がある。shallow embedding では対象言語の構文を実装言語の意味論で評価を行う。つまり、実装言語の値と関数、その他の言語機能を用いて評価を行う。shallow embedding の利点は実装が容易であることと、実装言語の関数やライブラリをそのまま評価に使えることである。欠点として意味論が実装言語のものに固定されてしまう。そのため、実装言語の意味論の制限が対象言語の意味論の制限より強い場合、別の言語で実装する必要がある。deep embedding では対象言語の抽象構文に対する簡約を与えることで評価する。つまり、意味論を抽象構文に対する簡約として任意に与えることができる。deep embedding の利点として対象言語の意味論を自由に与えることができる。欠点としては構文と意味論をそれぞれ実装する必要があり、実装難易度が高いことが挙げられる。本研究では BBSL に形式的な意味論を与えることを目的としているため deep embedding を採用している。

BBSL の構文は外部関数ブロック、前提条件ブロック、ケースブロックの 3 つに分けられ、またケースブロックは複数のケースから構成される。外部関数ブロックでは仕様外部の値を受け取り、前提条件ブロックではすべてのケースで満たされるべき条件を記述し、ケースブロックでは自動運転車が取るべき行動とその条件をケースとして複数記述する。よって、BBSL を形式化したものは、記述した仕様と画像上の仕様外部の値をパラメータとして受け取り、その画像がどのケースに当てはまり、どのケースに当てはまらないかを

返すような関数になる。図 3.3 の例なら、入力画像に前方車両があるかどうかと、前方車両の Bounding box、減速区間のそれぞれ具体的な値を受け取って、その画像が停止ケースに当てはまるかと、停止しないケースに当てはまるかをそれぞれ返す。よって、BBSL の意味を解釈するこの関数の型は式 (4.1) のようになる。*Spec* は BBSL の仕様、Σ は仕様外部のパラメータをもつ変数環境、*label* はケースラベル、*bool* はそのケースに当てはまるか否かを表すブール値である。変数環境とは、変数とその変数に割り当てられる値の組の集合である。また、BBSL の仕様 *Spec* は BBSL の抽象構文として表現する。

$$C_{spec} : Spec \to (\Sigma \to \mathbb{P}(label \times bool)) \tag{4.1}$$

この章では、Coq を用いて deep embedding で BBSL に形式的な意味論を与える。まず BBSL の構文から抽象構文を定義し、それらを Coq で実装する。次に、定義した抽象構文に対してそれを解釈する意味関数を定義し、それらを Coq で実装する。また、意味関数を実装するにあたって、解釈に用いるデータ構造を定義する。BBSL では区間が用いられるが、区間演算では誤差付きの実数値を計算するために演算が定義されている。それに対し、BBSL では物体同士の位置関係を表現するためにいくつかの関係を新たに導入している。そのため、区間の Coq での実装には既存のライブラリを用いず新たに実装を行っている。

4.1 抽象構文の定義と Coq による実装

BBSL の構文の全体について、BBSL の構文 3.1 から、BBSL の抽象構文を拡張 BNF で定義したものをソースコード 4.1 に示す。BBSL の構文全体である spec は、前提条件ブロック cond とケースブロック cases から構成される。BBSL の構文は外部関数ブロック、前提条件ブロック、ケースブロックの 3 つのブロックから構成されるが、この中で外部関数ブロックは記述仕様に現れる自由変数を宣言するだけのものであるため、意味論を与えていない。そのため、抽象構文としても定義していない。

ソースコード 4.1　BBSL の抽象構文

```
1 spec ::= "(" cond "," case+ ")"
```

BBSL の構文は帰納的に定義される。Coq には帰納的なデータ構造を定義できる機能があるので、それを用いて BBSL の構文の実装を行う。BBSL の抽象構文の Coq での実装をソースコード 4.2 に示す。Spec は仕様全体、Cond は前提条件ブロック、Case が 1 つのケースを表す。ケースブロックはケースのリストとして実装している。ケース数は有限であるので、リストで表現することができる。また、リストを用いることで写像の処理や畳込みの処理が簡単に実装できる利点がある。意味関数の実装や証明を行う中でこれらの処理は多用されると考えられる。

ソースコード 4.2　抽象構文の Coq での実装

```
1 Definition Spec : Set := Cond * list Case.
```

以下では、まず前提条件ブロックとケースブロックについて抽象構文を定義し、Coq で実装を行う。また、抽象構文の例と Coq での記述例を用いて説明する。次に、前提条件ブロックとケースブロックで扱われる式について抽象構文を定義し、Coq で実装を行う。BBSL の構文 3.4 では式はブール値型 bexp とそれ以外の値 value として定義されているが、本研究では値をさらに型ごとに細分化して抽象構文を定義する。また、同様に例を用いて説明する。

4.1.1　ブロックの抽象構文

■前提条件ブロック　前提条件ブロックの構文 3.3 から、前提条件ブロックの抽象構文を拡張 BNF で定義したものをソースコード 4.3 に示す。bexp はブール値型の式である。

ソースコード 4.3　前提条件ブロックの抽象構文

```
1 cond ::= "none" | bexp
```

前提条件ブロックの抽象構文の Coq での実装をソースコード 4.4 に示す。前提条件ブロックの実装は前提条件がある場合とない場合で分けている。前提条件がある場合はブール値の式 Bexp と一致する。

ソースコード 4.4　前提条件ブロックの抽象構文の Coq での実装

```
1 Inductive Cond : Set :=
2   | CND_None
3   | CND (b : Bexp).
```

例として図 3.3 の 7 行目から 9 行目の前提条件ブロックを抽象構文で記述したものをソースコード 4.5 に示す。

ソースコード 4.5　前提条件ブロックの抽象構文による記述例

```
1 前方車両がある()
```

また、Coq での記述例をソースコード 4.6 に示す。

ソースコード 4.6　前提条件ブロックの Coq による記述例

```
1 CND (EXP_Bvar "前方車両がある")
```

EXP_Bvar はブール値の式で、変数であることを示している。

■ケースブロック　ケースブロックの構文 3.5 から、ケースブロックの抽象構文を拡張 BNF で定義したものをソースコード 4.7 に示す。ケースブロックは複数のケースから構成される。label はケースラベルを表す任意の文字列である。ケースブロックでは let によって変数を定義することができる。def は変数定義である。value には任意の型の式が入る。

ソースコード 4.7　ケースブロックの抽象構文

```
1 case ::= "(" label "," "[" def ("," def)* "]" "," bexp ")"
2 def ::= "(" var "," value ")"
```

```
3 value ::= bexp | qexp | iexp | bbexp | sbbexp
```

ケースブロックの抽象構文の Coq での実装をソースコード 4.8 に示す。Case はケース、Def は変数定義を表す。ケースラベルは文字列で表す。ケースブロックは string * list Def * Bexp では文字列と変数定義の集合とブール値の式のペアを表す。

ソースコード 4.8　ケースブロックの抽象構文の Coq での実装

```
1 Inductive Def : Set :=
2   | DEF_SBB (x : string) (sbb : SBBexp)
3   | DEF_BB (x : string) (bb : BBexp)
4   | DEF_I (x : string) (i : Iexp)
5   | DEF_Q (x : string) (q : Qexp)
6   | DEF_B (x : string) (b : Bexp).
7
8 Definition Case : Set := string * list Def * Bexp.
```

例として図 3.3 の 11 行目から 16 行目の停止するケース、18 行目から 23 行目の停止しないケースについて抽象構文で記述したものをそれぞれソースコード 4.9、4.9 に示す。

ソースコード 4.9　停止するケースの抽象構文による記述

```
1 ( 停止
2 , [ (前方車両, 前方車両 ())
3   , (減速区間, 減速区間 ())
4   ]
5 , PROJ_y(前方車両) ≈ 減速区間
6 )
```

ソースコード 4.10　停止しないケースの抽象構文による記述

```
1 ( 停止しない
2 , [ (前方車両, 前方車両 ())
3   , (減速区間, 減速区間 ())
4   ]
5 , PROJ_y(前方車両) > 減速区間
6 )
```

また、それぞれについて Coq での記述例をソースコード 4.11、4.12 に示す。

ソースコード 4.11　停止するケースの Coq での記述例

```
1 ( "停止"
2 , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
3   ; DEF_I "減速区間" (EXP_Ivar "減速区間")
4   ]
5 , EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区
      間")
6 )
```

ソースコード 4.12　停止しないケースの Coq での記述例

```
1 ( "停止しない"
2 , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
3   ; DEF_I "減速区間" (EXP_Ivar "減速区間")
4   ]
5 , EXP_Igt (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区間")
6 )
```

4.1.2 式の抽象構文

式は型ごとに定義される。BBSL の型にはブール値型、実数型、区間型、Bounding box 型、Bounding box の集合型がある。それぞれについて抽象構文を定義し、Coq での実装を行う。Coq での実装では実数ではなく有理数を用いる。理由については後述の評価にて説明する。

■Bounding box の集合型　Bounding box の集合型の式の抽象構文を BNF で記述したものをソースコード 4.13 に示す。sbbexp は Bounding box の集合の式、bbexp は Bounding box の式である。Bounding box の集合型の式には変数、共通部分、合併、Bounding box の列挙による構成がある。

ソースコード 4.13　Bounding box の集合型の式の BNF による記述

```
1 sbbexp ::= var
2   | "sbbintersection" "(" sbbexp "," sbbexp ")"
3   | "sbbunion" "(" sbbexp "," sbbepx ")"
4   | "{" bbexp ("," bbexp)* "}"
```

Bounding box の集合型の式の抽象構文の Coq での実装をソースコード 4.14 に示す。SBBexp は Bounding box の集合の式、BBexp は Bounding box の式、EXP_SBBvar は変数、EXP_SBBintersection は共通部分、EXP_SBBunion は合併、EXP_makeSBB は Bounding box の列挙による構成を表す。SBBexp は BBexp と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.14　Bounding box の集合型の式の抽象構文の Coq での実装

```
1 Inductive SBBexp : Set :=
2   | EXP_SBBvar (x : string)
3   | EXP_SBBintersection (sbb0 sbb1 : SBBexp) | EXP_SBBunion (sbb0 sbb1
        : SBBexp)
4   | EXP_makeSBB (bbs : list BBexp).
```

例として、ソースコード 4.15 のような BBSL の記述を考える。これは車線変更時の仕様の一部である。

ソースコード 4.15　BBSL の記述例

```
1  24 case 右の車線に車線変更
2  25 let
3  26 割り込み車両:bb = 割り込み車両 ()
4  27 , 他車両集合:setBB = 他車両集合 ()
5  28 , 自車線左区間集合:setBB = 自車線左区間集合 ()
6  29 , 自車線右区間集合:setBB = 自車線右区間集合 ()
7  30 , 右車線変更区間集合:setBB = 右車線変更区間集合 ()
8  31 in 右車線存在確認 () and
9  32 forall x ∈ 他車両集合,exists y ∈ 右車線変更区間集合.
10 33 (not(PROJx(x)≈PROJx(y) and PROJy(x)≈PROJy(y)))
11 34 and
12 35 RAT(自車線左区間集合 ∩{割り込み車両}, 自車線右区間集合 ∩{割り込み車
       両}) > 1.0
13 36 endcase
```

この仕様の 35 行目の条件の抽象構文を記述したものをソースコード 4.16 に示す。qlt は有理数の比較関係である。

ソースコード 4.16　Bounding box の集合型の式の抽象構文の記述例

```
1  qlt(RAT(sbbintersection(自動車線左区間集合, {割り込み車両})
       sbbintersection(自動車線右区間集合, {割り込み車両})), 1.0)
```

また、Coq での記述をソースコード 4.17 に示す。EXP_Qlt は有理数の比較演算、

EXP_BBvar は Bounding box 型の変数である。

ソースコード 4.17　Bounding box の集合型の式の抽象構文の Coq による記述例

```
1 (EXP_Qgt
2   (EXP_RAT
3     (EXP_SBBintersection (EXP_SBBvar "自車線左区間集合") (EXP_makeSBB [
          EXP_BBvar "割り込み車両" ]))
4     (EXP_SBBintersection (EXP_SBBvar "自車線右区間集合") (EXP_makeSBB [
          EXP_BBvar "割り込み車両" ])))
5   (EXP_Q 1.0))
```

■Bounding box 型　Bounding box 型の式の抽象構文を BNF で記述したものをソースコード 4.18 に示す。bbexp は Bounding box の式、iexp は区間の式である。Bounding box の集合型の式には変数、区間の列挙による構成がある。また、画像全体の Bounding box を取得する特別な変数 img を定義する。

ソースコード 4.18　Bounding box 型の式の BNF による記述

```
1 bbexp ::= var | "{" iexp ("," iexp)* "}" | img
```

Bounding box 型の式の抽象構文の Coq での実装をソースコード 4.19 に示す。BBexp は Bounding box の集合の式、EXP_BBvar は変数、EXP_makeBB は 2 つの区間による構成、EXP_BBimg は画像全体を表す Bounding box を表す。BBexp は SBBexp と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.19　Bounding box 型の式の抽象構文の Coq での実装

```
1 Inductive BBexp : Set :=
2   | EXP_BBvar (x : string)
```

```
3    | EXP_makeBB (x y : Iexp)
4    (* 画像全体のBB *)
5    | EXP_BBimg.
```

例として図 3.3 の 15 行目の条件の抽象構文を記述したものをソースコード 4.20 に記述する。Ioverlap は区間の重なりを表す関数、$PORJ_y$ は Bounding box の y 軸方向の区間を取り出す射影関数である。

ソースコード 4.20　Bounding box の式の抽象構文の記述例

```
1 Ioverlap(PROJ_y(前方車両), 減速区間)
```

また、Coq での記述をソースコード 4.21 に示す。EXP_Qlt は有理数の比較演算、EXP_BBvar は Bounding box 型の変数である。

ソースコード 4.21　Bounding box の式の Coq による記述例

```
1 EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区間")
```

■区間型　区間型の式の抽象構文を BNF で記述したものをソースコード 4.22 に示す。iexp は区間の式、qexp は有理数の式である。Bounding box の集合型の式には変数、区間の列挙による構成がある。$PROJ_x$ は区間の下限を取得する射影関数、$PROJ_y$ は上限を取得する射影関数、iintersection は共通部分を表す。

ソースコード 4.22　区間型の式の抽象構文の BNF による記述

```
1 iexp ::= var | "PROJ_x" "(" bbexp ")" | "PROJ_y" "(" bbexp ")" |
      iintersection "(" iexp "," iexp ")" | "{" qexp ("," qexp)* "}"
```

区間型の式の抽象構文の Coq での実装をソースコード 4.23 に示す。Iexp は区間の式、EXP_Ivar は変数、EXP_Iintersection は共通部分、EXP_makeI は 2 つの有理数による構成を表す。Iexp は BBexp と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.23　区間型の式の抽象構文の Coq での実装

```
1 Inductive Iexp : Set :=
2   | EXP_Ivar (x : string)
3   | EXP_projx (bb : BBexp) | EXP_projy (bb : BBexp)
4   | EXP_Iintersection (i0 i1 : Iexp)
5   | EXP_makeI (l u : Qexp).
```

例として図 3.3 の 22 行目の条件の抽象構文を記述したものをソースコード 4.24 に記述する。Igt は区間の比較関係、$PORJ_y$ は Bounding box の y 軸方向の区間を取り出す射影関数である。

ソースコード 4.24　区間の式の抽象構文の記述例

```
1 Igt(PROJ_y(前方車両), 減速区間)
```

また、Coq での記述をソースコード 4.25 に示す。EXP_Igt は区間の比較演算、EXP_BBvar は Bounding box 型の変数、EXP_Ivar は区間型の変数である。

ソースコード 4.25　区間の式の Coq による記述例

```
1 EXP_Igt (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区間")
```

■有理数型　有理数型の式の抽象構文を BNF で記述したものをソースコード 4.26 に示す。qexp は区間の式である。Bounding box の集合型の式には変数、区間の列挙による構成がある。a は有理数リテラル、X は変数、”RAT”は Bounding box の集合同士の面積比、PROJ_l, u は区間の下限、上限を取得する射影関数、PROJ_xl, xu, yl, yu は Bounding box の x、y 軸方向の区間の上限、下限を取得する射影関数を表す。projx は区間の下限を取得する射影関数、projy は上限を取得する射影関数を表す。

ソースコード 4.26　有理数の式の抽象構文の BNF による記述

```
1 qexp ::= literal | var | "w" "(" iexp ")" | "RAT" "(" sbbexp ","
      sbbexp ")" | "PROJ_l" "(" iexp ")" | "PROJ_u" "(" iexp ")"
2     | "PROJ_xl" "(" bbexp ")" | "PROJ_xu" "(" bbexp ")" | "PROJ_yl"
        "(" bbexp ")" | "PROJ_yu" "(" bbexp ")"
```

有理数型の式の抽象構文の Coq での実装をソースコード 4.27 に示す。Iexp は区間の式、EXP_Q は有理数リテラル、EXP_Qvar は変数、EXP_width は区間の幅、EXP_RAT は Bounding box の集合同士の面積比、EXP_projl, u は区間の下限、上限を取得する射影関数、EXP_projxl, xu, yl, yu は Bounding box の x、y 軸方向の区間の上限、下限を取得する射影関数表す。Qexp は Iexp と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.27　有理数型の式の抽象構文の Coq での実装

```
1 Inductive Qexp : Set :=
2   | EXP_Q (a: Q)
3   | EXP_Qvar (x : string)
4   | EXP_width (i : Iexp) | EXP_RAT (sbb0 sbb1 : SBBexp)
5   | EXP_projl (i : Iexp) | EXP_proju (i : Iexp)
6   | EXP_projxl (bb : BBexp) | EXP_projxu (bb : BBexp)
7   | EXP_projyl (bb : BBexp) | EXP_projyu (bb : BBexp).
```

例として図 4.15 の 35 行目の条件の抽象構文を記述したものをソースコード 4.28 に記述する。qgt は有理数の比較関係である。

ソースコード 4.28　有理数の式の抽象構文の記述例

```
1 qgt(RAT(sbbintersection(自動車線左区間集合，{割り込み車両})
      sbbintersection(自動車線右区間集合，{割り込み車両}))，1.0)
```

また、Coq での記述をソースコード 4.29 に示す。EXP_Qlt は有理数の比較演算、EXP_BBvar は Bounding box 型の変数である。

ソースコード 4.29　有理数の式の抽象構文の Coq による記述例

```
1 (EXP_Qgt
2   (EXP_RAT
3     (EXP_SBBintersection (EXP_SBBvar "自車線左区間集合") (EXP_makeSBB [
          EXP_BBvar "割り込み車両" ]))
4     (EXP_SBBintersection (EXP_SBBvar "自車線右区間集合") (EXP_makeSBB [
          EXP_BBvar "割り込み車両" ])))
5   (EXP_Q 1.0))
```

■ブール値型　ブール値型の式の抽象構文を BNF で記述したものをソースコード 4.30 に示す。b はブール値の式である。ブール値型の式には否定、論理和、論理積、forall、exists がある。また、有理数、区間、Bounding box それぞれの等号と比較演算、区間、Bounding box の集合演算（所属、部分集合）がある。forall と exists は Bounding box の集合型についてのみ定義されている。

ソースコード 4.30　ブール値型の式の抽象の BNF による記述

```
1 bexp ::= var | "not" "(" bexp ")" | "and" "(" bexp "," bexp ")" | "
      or" "(" bexp "," bexp ")"
```

```
2     | "beq" "(" bexp "," bexp ")" | "boverlap" "(" bexp "," bexp ")"
3     | "bbsubset" "(" bbexp "," bbexp ")" | "bbsupset" "(" bbexp ","
          bbexp ")"
4     | "bbsubseteq" "(" bbexp "," bbexp ")" | "bbsupseteq" "(" bbexp
          "," bbexp ")"
5     | "ilt" "(" iexp "," iexp ")" | "igt" "(" iexp "," iexp ")" | "
          ieq" "(" iexp "," iexp ")"
6     | "ioverlap" "(" iexp "," iexp ")"
7     | "iin" "(" qexp "," iexp ")" | "iinrev" "(" iexp "," qexp ")"
8     | "isubset" "(" iexp "," iexp ")" | "isupset" "(" iexp "," iexp
          ")"
9     | "qlt" "(" qexp "," qexp ")" | "qgt" "(" qexp "," qexp ")" | "
          qeq" "(" qexp "," qexp ")"
10    | "qle" "(" qexp "," qexp ")" | "qge" "(" qexp "," qexp ")"
11    | "forall" "(" var "," sbbexp "," bexp ")" | "exists" "(" var
          "," sbb "," b ")"
```

ブール値型の式の抽象構文の Coq での実装をソースコード 4.31 に示す。eq は equal、lt は less than、le は less than or equal、gt は greater than、ge は greater than or equal の略である。

ソースコード 4.31　ブール値型の式の抽象構文の Coq での実装

```
1 Inductive Bexp : Set :=
2   | EXP_Bvar (x : string)
3   | EXP_not (b : Bexp) | EXP_and (b0 b1 : Bexp) | EXP_or (b0 b1 :
        Bexp)
4   | EXP_BBeq (bb0 bb1 : BBexp)
5   | EXP_BBoverlap (bb0 bb1 : BBexp)
6   | EXP_BBsubset (bb0 bb1 : BBexp) | EXP_BBsupset (bb0 bb1 : BBexp)
7   | EXP_BBsubseteq (bb0 bb1 : BBexp) | EXP_BBsupseteq (bb0 bb1 : BBexp
        )
8   | EXP_Ilt (i0 i1 : Iexp) | EXP_Igt (i0 i1 : Iexp) | EXP_Ieq (i0 i1
        : Iexp)
9   | EXP_Ioverlap (i0 i1 : Iexp)
```

```
10   | EXP_Iin (q : Qexp) (i : Iexp) | EXP_Iinrev (i : Iexp) (q : Qexp)
11   | EXP_Isubset (i0 i1 : Iexp) | EXP_Isupset (i0 i1 : Iexp)
12   | EXP_Isubseteq (i0 i1 : Iexp) | EXP_Isupseteq (i0 i1 : Iexp)
13   | EXP_Qlt (q0 q1 : Qexp) | EXP_Qgt (q0 q1 : Qexp)
14   | EXP_Qeq (q0 q1 : Qexp)
15   | EXP_Qle (q0 q1 : Qexp) | EXP_Qge (q0 q1 : Qexp)
16   | EXP_forall (bound : string) (sbb : SBBexp) (b : Bexp)
17   | EXP_exists (bound : string) (sbb : SBBexp) (b : Bexp).
```

例として図 3.3 の 15 行目の条件の抽象構文を記述したものをソースコード 4.32 に記述する。Ioverlap は区間の重なりを表す関数、$PORJ_y$ は Bounding box の y 軸方向の区間を取り出す射影関数である。

ソースコード 4.32　ブール値の式の抽象構文の記述例

```
1 Ioverlap(PROJ_y(前方車両), 減速区間)
```

また、Coq での記述をソースコード 4.33 に示す。EXP_Qlt は有理数の比較演算、EXP_BBvar は Bounding box 型の変数である。

ソースコード 4.33　ブール値の式の Coq による記述例

```
1 EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区間")
```

4.2　区間、Bounding box、Bounding box の集合の Coq による実装

本研究では構文の意味論として数学的オブジェクトを与える表示的意味論を採用している。以下では BBSL の区間、Bounding box、Bounding box の集合の意味論に対応する

数学的オブジェクトを定義し、Coq で実装を行う。

■区間　BBSL での区間は式 (3.1) で定義されている。

区間の Coq での実装をソースコード 4.34 に示す。BBSL の定義では実数を用いて定義されているが、本研究では有理数を用いて、その直積集合として実装する。つまり、区間を上限と下限の組としている。Iin は有理数が区間に含まれていることを判定する関数、upper、lower は上限、下限を取得する関数、Iempty、Inempty は区間が空であること、空でないことを判定する関数である。i を区間としたとき、$lower i > upper i$ のときは区間は空であるとして実装している。

ソースコード 4.34　区間の Coq での実装

```
1  Definition Interval : Type := Q * Q.
2
3  Definition lower (i : Interval) : Q :=
4    match i with
5    | (l, _) => l
6    end.
7
8  Definition upper (i : Interval) : Q :=
9    match i with
10   | (_, u) => u
11   end.
12
13 Definition Iempty (i : Interval) : Prop :=
14   lower i > upper i.
15
16 Definition Inempty (i : Interval) : Prop :=
17   lower i <= upper i.
18
19 Definition Iin (v : Q) (i : Interval) : Prop :=
20   (lower i <= v /\ v <= upper i)%Q.
```

区間上の関数として下限、上限を取得する関数、比較演算、幅、集合演算などが定義されている。BBSL 独自の関数として、区間の重なりを判定する関数がある。区間の重なり

を判定する関数は X、Y を区間として式 (4.2) のように定義される。つまり、区間が重なっていることは２つの区間の共通部分がないこととしている。

$$X \approx Y = X \cap Y \neq \emptyset \tag{4.2}$$

区間の重なりを判定する関数の Coq での実装をソースコード 4.35 に示す。Ioverlap は重なりを判定する関数、Iempty は区間が空であることを判定する関数、Iintersection は共通部分を表す関数である。

ソースコード 4.35　区間の重なりを判定する関数の Coq での実装

```
1 Definition Ioverlap (i0 i1 : Interval) : Prop :=
2   ~Iempty (Iintersection i0 i1).
```

■Bounding box　BBSL で Bounding box は２次元の区間として定義される。つまり、A を Bounding box、X、Y を区間として式 (4.3) のように定義されている。

$$A = (X, Y) \tag{4.3}$$

Bounding box の Coq での実装をソースコード 4.36 に示す。Bounding box である BB は区間である Interval の直積として実装している。projx、projy は x 軸、y 軸方向の区間を取り出す射影関数である。projxl、projxu、projyu、projyu は x、y 軸方向の区間の上限、下限を取り出す関数である。また、Bounding box が空でないことを、x、y 軸方向の区間がどちらも空でないこととして実装している。

ソースコード 4.36　Bounding box の Coq での実装

```
1 Definition BB : Type := Interval * Interval.
2
3 Definition projx (bb : BB) : Interval :=
4   match bb with
5   | (x, _) => x
6   end.
7
```

```
 8 Definition projy (bb : BB) : Interval :=
 9   match bb with
10   | (_, y) => y
11   end.
12
13 Definition projxl (bb : BB) : Q :=
14   lower (projx bb).
15
16 Definition projxu (bb : BB) : Q :=
17   upper (projx bb).
18
19 Definition projyl (bb : BB) : Q :=
20   lower (projy bb).
21
22 Definition projyu (bb : BB) : Q :=
23   upper (projy bb).
24
25 Definition BBempty (bb : BB) : Prop :=
26   Iempty (projx bb) /\ Iempty (projy bb).
27
28 Definition BBnempty (bb : BB) : Prop :=
29   Inempty (projx bb) /\ Inempty (projy bb).
```

Bounding box の関数として比較演算、集合演算が定義されている。BBSL 独自の関数として、Bounding box の重なりを判定する関数がある。Bounding box の重なりを判定する関数は x、y 軸方向の区間がどちらも重なることを表現する。つまり、A、B を Bounding box、$PROJ_x$、$PROJ_y$ を x、y軸方向の区間を取得する関数として、式 (4.4) のように定義されている。

$$A \approx B = PROJ_x(A) \approx PROJ_x(B) \wedge PROJ_y(A) \approx PROJ_y(B) \tag{4.4}$$

Bounding box の重なりを判定する関数の Coq での実装をソースコード 4.37 に示す。BBoverlap はは Bounding box の重なりを判定する関数である。

ソースコード 4.37　Bounding box の重なりを判定する関数の Coq での実装

```
1 Definition BBoverlap (bb0 bb1 : BB) : Prop :=
2   Ioverlap (projx bb0) (projx bb1) /\ Ioverlap (projy bb0) (projy bb1
      ).
```

■Bounding box の集合　Bounding box の集合はそのまま Bounding box の集合として定義されている。Bounding box の集合の Coq での実装をソースコード 4.38 に示す。SetBB は Bounding box の集合を表す。Coq での実装では Bounding box の集合を Bounding box のリストとして実装している。BBSL は画像上のオブジェクトとその位置関係を Bounding box を用いて記述するための形式仕様記述言語であるため、無限の Bounding box を扱うことは考えなくてよい。そのため集合の変わりにリストを実装に用いることができる。

ソースコード 4.38　Bounding box の集合の Coq での実装

```
1 Definition SetBB : Type := list BB.
```

Bounding box の集合の関数として共通部分と合併が定義されている。Bounding box の共通部分と合併の Coq での実装をソースコード 4.39 に示す。合併は単純に和を取ることで実装する。++ はリストの和をとる関数である。共通部分は、Bounding box の共通部分をすべての要素に対して繰り返し適用することで実装している。

ソースコード 4.39　Bounding box の集合の共通部分と合併の Coq での実装

```
1 Fixpoint _BB_SBBintersection (bb : BB) (sbb accum : SetBB) : SetBB :=
2   match sbb with
3   | nil => accum
4   | cons bb' sbb' => _BB_SBBintersection bb sbb' (cons (BBintersection
        bb bb') accum)
5   end.
6
```

```
7 Fixpoint _SBBintersection (sbb0 sbb1 accum : SetBB) : SetBB :=
8   match sbb0 with
9   | nil => accum
10  | cons bb sbb => _SBBintersection sbb sbb1 (_BB_SBBintersection bb
        sbb1 nil ++ accum)
11  end.
12
13 Definition SBBintersection (sbb0 sbb1 : SetBB) : SetBB :=
14  _SBBintersection sbb0 sbb1 nil.
15
16 Definition SBBunion (sbb0 sbb1 : SetBB) : SetBB :=
17  sbb0 ++ sbb1.
```

Bounding box の集合の BBSL に特有の関数として RAT 関数がある。RAT 関数は 2 つの Bounding box の集合の面積比を計算する。RAT 関数は BBSL では面積比としか定義されておらず、明確な定義を与えられていない。ここでは、Bounding box の集合に総面積を計算する関数を追加し、それを使って RAT 関数を実装する。RAT 関数の Coq での実装をソースコード 4.40 に示す。BBarea は Bounding box の面積を求める関数である。区間は幅が取れるので、x 軸方向と y 軸方向の区間の幅を掛けることで Bounding box の面積を得られる。SetBBarea は Bounding box の集合の総面積を求める関数である。Bounding box の集合に含まれるすべての Bounding box の面積を足しながら、共通部分の面積を引くことで実装している。RAT 関数は 2 つの Bounding box の集合の総面積をそれぞれ計算したあと、それらで除算を行っている。

ソースコード 4.40　Bounding box の集合の RAT 関数の Coq での実装

```
1 Definition BBarea (bb : BB) : Q :=
2   width (projx bb) * width (projy bb).
3
4 Fixpoint _SetBBarea (sbb accum : SetBB) (area : Q) : Q :=
5   match sbb with
6   | nil => area
7   | cons bb sbb' =>
8     let sbb'' := _BB_SBBintersection bb accum nil in
```

```
9      let sbb''area := List.fold_right Qplus 0 (List.map BBarea sbb'')
           in
10     _SetBBarea sbb' (cons bb accum) (area + BBarea bb - sbb''area)
11   end.
12
13 Definition SetBBarea (sbb : SetBB) : Q :=
14   _SetBBarea sbb nil 0.
15
16 Definition RAT (sbb0 sbb1 : SetBB) : Q :=
17   SetBBarea sbb0 / SetBBarea sbb1.
```

4.3 意味関数の定義と Coq による実装

定義した抽象構文を用いて、意味関数を定義する。意味関数は、抽象構文を定義した数学的オブジェクトに対応させる。BBSL の構文全体の意味関数は、それぞれのブロックの意味関数に依存する。また、ブロックの意味関数はそれぞれの型の式の意味関数に依存する。よって、以下ではブロックと式についてそれぞれ意味関数を定義し、Coq で実装を行っていく。意味関数の一覧とその型を表 4.1 に示す。以下ではそれぞれの意味関数について定義と Coq による実装を行っている。

抽象構文	意味関数の型
仕様全体	$C_{spec} : Spec \to (\Sigma \to \mathbb{P}(label \times bool))$
前提条件ブロック	$C_{cond} : Cond \to (\Sigma \to Prop)$
ケースブロック	$C_{cases} : \mathbb{P}(Case) \to (\Sigma \to \mathbb{P}(label * Prop))$
Bounding box の集合型	$A_{sbb} : SBBexp \to (\Sigma \to SetBB)$
Bounding box 型	$A_{bb} : BBexp \to (\Sigma \to BB)$
区間型	$A_i : Iexp \to (\Sigma \to Interval)$
有理数型	$A_q : Iexp \to (\Sigma \to Q)$
ブール値型	$B : Bexp \to (\Sigma \to Prop)$

表 4.1　意味関数

■仕様全体の意味関数　BBSL の仕様全体の意味は変数環境、つまり外部で定義したパラメータを受け取り、定義したそれぞれのケースが真になるか否かを返す関数となる。直感的には、画像一枚分のパラメータを与えると、その画像がどのケースに当てはまりどのケースには当てはまらないかを判定する。BBSL で記された述仕様を解釈する意味関数の型を式 (4.5) に示す。Σ は変数環境の集合、*label* はケースラベルの集合、*bool* は真理値の集合である。変数環境とは、変数と束縛される値の組の集合である。

$$C_{spec} : Spec \to (\Sigma \to \mathbb{P}(label \times bool)) \tag{4.5}$$

BBSL で記述された仕様を解釈する意味関数の Coq での実装をソースコード 4.41 に示す。仕様全体の抽象構文である Spec 型の spec と変数環境である Env 型の env を受け取る。返り値が option なのは意味関数が部分関数で、仕様の記述内容によっては意味の解釈に失敗するためである。ケースラベルは文字列、Prop は命題を表す型である。ケースブロックに記述されるケースは有限であるため、返り値の集合は list で表現している。よって返り値 list (string * Prop) はケースラベルとそのケースの条件となる命題の組の集合となる。

仕様全体の意味関数 Cspec の中では前提条件の意味関数 Ccond とケースブロックの意味関数 Ccases が使われている。前提条件はすべてのケースで満たされるべき性質なので、すべてのケースについて意味関数で解釈した結果と前提条件を解釈した結果とで論理積を取っている。

ソースコード 4.41　仕様全体の意味関数の Coq での実装

```
1  Definition Cspec (spec : Spec) (env : Env) : option (list (string *
       Prop)) :=
2    match spec with
3    | (cond, cases) =>
4      match Ccond cond env, Ccases cases env nil with
5      | Some b, Some lbs => Some (List.map
6            (fun lb => match lb with (l, b') => (l, b /\ b') end)
7            lbs)
8      | _, _ => None
9      end
10   end.
```

■前提条件ブロックの意味関数　前提条件ブロックではすべてのケースで満たされるべき条件を記述する。よって、前提条件ブロックを形式化したものはブール値の式を形式化したものとほぼ一致する。ただし、前提条件ブロックでは前提条件がない場合 none を記述できる。前提条件はすべてのケースで満たされるべき条件なので、none のとき、つまり前提条件がないときは真を返せばよい。よって、前提条件ブロックの意味関数の型は式 (4.6) のようになる。$Cond$ は先に定義した前提条件ブロックの抽象構文である。

$$C_{cond} : Cond \to (\Sigma \to Prop) \tag{4.6}$$

前提条件ブロックの意味関数の Coq での実装をソースコード 4.42 に示す。前提条件がない none のときは True を返し、前提条件があるときはブール値の式の意味関数 B を用いて解釈を行う。B はブール値の式の意味関数である。

ソースコード 4.42　前提条件ブロックの意味関数の Coq での実装

```
1 Definition Ccond (cond : Cond) (env : Env) : option Prop :=
2   match cond with
3   | CND_None => Some True
4   | CND b => B b env
5   end.
```

例として、図 3.3 の 7 行目から 9 行目の前提条件ブロックの部分の抽象構文を意味関数で評価する式を式 (4.43) に示す。前提条件ブロックの抽象構文はソースコード 4.5 のように記述できた。C_{cond} は前提条件ブロックの意味関数である。

ソースコード 4.43　前提条件ブロックの意味関数の記述例

```
1   C_cond⟦ 前方車両がある() ⟧
```

Coq での記述は式 (4.44) のようになる。前提条件ブロックの抽象構文は Coq を用いてソースコード 4.6 のように記述できた。Ccond は前提条件ブロックの意味関数である。

ソースコード 4.44　前提条件ブロックの意味関数の Coq による記述例

```
1   Ccond (CND (EXP_Bvar "前方車両がある"))
```

■ケースブロックの意味関数　ケースブロックはケースの集まりである。よってケースブロックを解釈した結果もケースを解釈した結果の集合となる。ケースではケースラベルとその条件を記述する。よって、ケースを形式化したものはケースラベルと条件を表す命題の組となる。ケースブロックの意味関数の型を式 (4.7) に示す。$Case$ はケースの抽象構文、$label$ はケースラベルを表す。よって $\mathbb{P}(Case)$ はケースブロックの抽象構文、返り値 $\mathbb{P}(label * Prop)$ はケースラベルとその条件の組の集合となる。

$$C_{cases} : \mathbb{P}(Case) \to (\Sigma \to \mathbb{P}(label * Prop)) \tag{4.7}$$

ケースブロックの意味関数の Coq での実装をソースコード 4.45 に示す。ケースでは let で変数を定義できる。変数定義 Def を解釈する意味関数 Cdef では、変数環境にその変数を追加した新しい変数環境を返す。変数に代入する値は型ごとに用意された意味関数を用いて解釈し、その結果を変数環境に追加する。Asbb は Bounding box の集合の式の意味関数、Abb は Bounding box の式の意味関数、Ai は区間の式の意味関数、Aq は有理数の式の意味関数、B はブール値の式の意味関数である。Cdefs はすべての変数定義に対して Cdef を適用する。ケースの意味関数では、変数定義を解釈した結果の変数環境を用いて、ケースの条件を解釈する。解釈結果はケースラベルと組にして string * Prop という形で返す。ケースブロックを解釈した結果はケースを解釈した結果の集合になるが、ケース数は有限なので list を用いて list (string * Prop) という型で実装する。

ソースコード 4.45　ケースブロックの意味関数の Coq での実装

```
1 Definition Cdef (def : Def) (env : Env) : option Env :=
2   match def with
3   | DEF_SBB s sbb_expr =>
4     match Asbb sbb_expr env with
5     | Some sbb => Some (add s (Vsbb sbb) env)
6     | _ => None
7     end
8   | DEF_BB s bb_expr =>
```

```
9       match Abb bb_expr env with
10      | Some bb => Some (add s (Vbb bb) env)
11      | _ => None
12      end
13    | DEF_I s i_expr =>
14      match Ai i_expr env with
15      | Some i => Some (add s (Vi i) env)
16      | _ => None
17      end
18    | DEF_Q s q_expr =>
19      match Aq q_expr env with
20      | Some q => Some (add s (Vq q) env)
21      | _ => None
22      end
23    | DEF_B s b_expr =>
24      match B b_expr env with
25      | Some b => Some (add s (Vb b) env)
26      | _ => None
27      end
28    end.
29
30  Fixpoint Cdefs (defs : list Def) (env : Env) : option Env :=
31    match defs with
32    | nil => Some env
33    | cons def defs' =>
34      match Cdef def env with
35      | Some env' => Cdefs defs' env'
36      | _ => None
37      end
38    end.
39
40  Definition Ccase (case : Case) (env : Env) : option (string * Prop)
        :=
41    match case with
42    | (l, defs, b_expr) =>
43      match Cdefs defs env with
44      | Some env' =>
45        match B b_expr env' with
```

```
46        | Some b => Some (l, b)
47        | _ => None
48        end
49      | _ => None
50      end
51    end.
52
53  Fixpoint Ccases (cases : list Case) (env : Env) (accum : list (string
        * Prop)) : option (list (string * Prop)) :=
54    match cases with
55    | nil => Some accum
56    | cons case cases' =>
57      match Ccase case env with
58      | Some lb => Ccases cases' env (cons lb accum)
59      | _ => None
60      end
61    end.
```

例として、図 3.3 の 11 行目から 16 行目のケースの部分の抽象構文を意味関数で評価する式を式 (4.46) に示す。ケースブロックの抽象構文はソースコード 4.9 のように記述できた。C_{case} はケースブロックの意味関数である。

ソースコード 4.46　ケースの意味関数の記述例

```
1  C_case⟦
2      ( "停止"
3    , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
4      ; DEF_I "減速区間" (EXP_Ivar "減速区間")
5      ]
6    , EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区
         間")
7    ) ⟧
```

Coq での記述は式 (4.47) のようになる。ケースブロックの抽象構文は Coq を用いてソースコード 4.11 のように記述できた。Ccase はケースブロックの意味関数である。

ソースコード 4.47　ケースの意味関数の Coq による記述例

```
1 Ccase
2   ( "停止"
3   , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
4     ; DEF_I "減速区間" (EXP_Ivar "減速区間")
5     ]
6   , EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区
        間")
7   )
```

■Bounding box の集合の式の意味関数　BBSL 上の Bounding box の集合の式を形式化したものは、4.2 節で定義した Bounding box の集合となる。Bounding box の集合の式の意味関数の型を式 (4.8) に示す。$SBBexp$ は Bounding box の集合の式の抽象構文、$SetBB$ は Bounding box の集合である。

$$A_{sbb} : SBBexp \to (\Sigma \to SetBB) \tag{4.8}$$

Bounding box の集合の式の意味関数の Coq での実装をソースコード 4.48 に示す。EXP_SBBvar は変数で、解釈としては変数環境から一致する変数名に対応する値を取得する。変数環境に変数が存在しない場合、解釈は失敗する。EXP_SBBintersection は共通部分、EXP_SBBunion は合併はそれぞれ Bounding box の集合の実装で実装した関数をそのまま解釈に用いる。EXP_makeSBB は Bounding box を複数列挙することで Bounding box の集合を構成する構文であり、解釈は Bounding box のリストを受け取り、すべてを Bounding box の式の意味関数 Abb で解釈した結果を list にする。ここで、Bounding box の集合の定義は Bounding box の list であったことを思い出すと、SetBB を返すことになる。Asbb は Abb と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.48　Bounding box の集合の式の意味関数の Coq での実装

```
1  Fixpoint Asbb (expr : SBBexp) (env : Env) : option SetBB :=
2    match expr with
3    | EXP_SBBvar s =>
4      match find s env with
5      | Some (Vsbb sbb) => Some sbb
6      | _ => None
7      end
8    | EXP_SBBintersection sbb_expr0 sbb_expr1 =>
9      match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
10     | Some sbb0, Some sbb1 => Some (SBBintersection sbb0 sbb1)
11     | _, _ => None
12     end
13   | EXP_SBBunion sbb_expr0 sbb_expr1 =>
14     match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
15     | Some sbb0, Some sbb1 => Some (SBBunion sbb0 sbb1)
16     | _, _ => None
17     end
18   | EXP_makeSBB bb_exprs =>
19     List.fold_left (fun obbs obb =>
20       match obbs, obb with
21       | Some bbs, Some bb => Some (cons bb bbs)
22       | _, _ => None
23       end
24     ) (List.map (fun bb_expr => Abb bb_expr env) bb_exprs) (Some nil)
25   end
```

例として、ソースコード 4.15 の 35 行目の Bounding box の集合の演算部分の抽象構文を意味関数で評価する式を式 (4.50) に示す。また、該当部分の抽象構文はソースコード 4.16 より、ソースコード 4.49 のように記述できる。sbbintersection は Bounding box の集合同士の共通部分を表す。自動車線左区間集合は Bounding box の集合型、割り込み車両は Bounding box 型の変数である。割り込み車両は割り込み車両のみをもつ Bounding box の集合を表現している。A_{sbb} は Bounding box の集合の式の意味関数である。

ソースコード 4.49　Bounding box の集合の式の記述例

```
1 sbbintersection(自動車線左区間集合，{割り込み車両})
```

ソースコード 4.50　Bounding box の集合の式の意味関数の記述例

```
1 A_sbb⟦ sbbintersection(自動車線左区間集合，{割り込み車両}) ⟧
```

Coq での記述は式 (4.52) のようになる。また、Bounding box の集合の式の抽象構文はソースコード 4.17 より Coq を用いてソースコード 4.51 のように記述できた。EXP_SBBintersection は Bounding box の集合同士の共通部分である。EXP_makeSBB は Bounding box のリストを受け取って Bounding box の集合型を構成する。Asbb は Bounding box の集合の式の意味関数である。

ソースコード 4.51　Bounding box の集合の式の Coq による記述例

```
1 EXP_SBBintersection
2   (EXP_SBBvar "自車線左区間集合")
3   (EXP_makeSBB [ EXP_BBvar "割り込み車両" ])
```

ソースコード 4.52　Bounding box の集合の式の意味関数の Coq による記述例

```
1 Asbb
2   (EXP_SBBintersection
3     (EXP_SBBvar "自車線左区間集合")
4     (EXP_makeSBB [ EXP_BBvar "割り込み車両" ]))
```

■Bounding box の式の意味関数　BBSL 上の Bounding box の式を形式化したものは、4.2 節で定義した Bounding box となる。Bounding box の式の意味関数の型を式 (4.9) に示す。$BBexp$ は Bounding box の式の抽象構文、BB は Bounding box である。

$$A_{bb} : BBexp \to (\Sigma \to BB) \tag{4.9}$$

Bounding box の式の意味関数の Coq での実装をソースコード 4.53 に示す。EXP_BBimg は画像全体を表す Bounding box である。変数環境に IMG という名前の変数を追加することで利用できる。EXP_BBvar は変数であり、変数環境から変数名に対応する値を取得したものが解釈結果となる。EXP_makeBB は区間を 2 つ受け取って Bounding box を構成する構文であり、解釈は 2 つの区間の式を意味関数 Ai で解釈し、その結果を用いて BB 型の値を構成する。Abb は Ai と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.53　Bounding box の式の意味関数の Coq での実装

```
1  Fixpoint Abb (expr : BBexp) (env : Env) : option BB :=
2    match expr with
3    | EXP_BBimg =>
4      match find "IMG" env with
5      | Some (Vbb bb) => Some bb
6      | _ => None
7      end
8    | EXP_BBvar s =>
9      match find s env with
10     | Some (Vbb bb) => Some bb
11     | _ => None
12     end
13   | EXP_makeBB i_expr0 i_expr1 =>
14     match Ai i_expr0 env, Ai i_expr1 env with
15     | Some i0, Some i1 => Some (i0, i1)
16     | _, _ => None
17     end
18   end.
```

例として、図 3.3 の 15 行目の Bounding box の式の部分の抽象構文を意味関数で評価する式を式 (4.55) に示す。また、Bounding box の式の抽象構文はソースコード 4.20 よりソースコード 4.54 のように記述できた。前方車両は Bounding box 型の変数である。

ソースコード 4.54　Bounding box の式の記述例

```
1 前方車両
```

ソースコード 4.55　Bounding box の式の意味関数の記述例

```
1 A_bb⟦ 前方車両 ⟧
```

Coq での記述は式 (4.57) のようになる。また、Bounding box の式の抽象構文はソースコード 4.11 より Coq を用いてソースコード 4.56 のように記述できた。EXP_BBvar は Bounding box 型の変数を表す。

ソースコード 4.56　Bounding box の式の Coq による記述例

```
1 EXP_BBvar "前方車両"
```

ソースコード 4.57　Bounding box の式の意味関数の Coq による記述例

```
1 Abb (EXP_BBvar "前方車両")
```

■区間の式の意味関数　BBSL 上の区間の式を形式化したものは、4.2 節で定義した Bounding box となる。区間の式の意味関数の型を式 (4.10) に示す。$Iexp$ は区間の式の

抽象構文、$Interval$ は区間である。

$$A_i : Iexp \to (\Sigma \to Interval) \tag{4.10}$$

区間の式の意味関数の Coq での実装をソースコード 4.58 に示す。EXP_Ivar は変数であり、変数環境から変数名に対応する値を取得したものが解釈結果となる。EXP_projx、EXP_projy は Bounding box の下限、上限を取得する射影関数、EXP_Iintersection、EXP_Iunion は共通部分と合併である。これらの関数はそれぞれ Bounding box と区間の実装で実装した関数をそのまま解釈に用いる。EXP_makeI は有理数 2 つを受け取って区間を構成する構文であり、解釈は 2 つの有理数の式を有理数の式の意味関数 Aq で解釈し、その結果を用いて Interval 型の値を構成する。Ai は Aq と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.58　区間の式の意味関数の Coq での実装

```
1  Fixpoint Ai (expr : Iexp) (env : Env) : option Interval :=
2    match expr with
3    | EXP_Ivar s =>
4      match find s env with
5      | Some (Vi i) => Some i
6      | _ => None
7      end
8    | EXP_projx bb_expr =>
9      match Abb bb_expr env with
10     | Some bb => Some (projx bb)
11     | None => None
12     end
13   | EXP_projy bb_expr =>
14     match Abb bb_expr env with
15     | Some bb => Some (projy bb)
16     | None => None
17     end
18   | EXP_Iintersection i_expr0 i_expr1 =>
19     match Ai i_expr0 env, Ai i_expr1 env with
20     | Some i0, Some i1 => Some (Iintersection i0 i1)
```

```
21      | _, _ => None
22      end
23    | EXP_makeI q_expr0 q_expr1 =>
24      match Aq q_expr0 env, Aq q_expr1 env with
25      | Some q0, Some q1 => Some (q0, q1)
26      | _, _ => None
27      end
28    end
```

例として、図 3.3 の 22 行目の区間の式の部分の抽象構文を意味関数で評価する式を式 (4.60) に示す。また、区間の式の抽象構文はソースコード 4.24 よりソースコード 4.59 のように記述できた。前方車両は Bounding box 型の変数であり、$PROJ_y$ は Bounding box の y 軸方向の区間を取り出す射影関数である。A_i は区間の意味関数である。

ソースコード 4.59　区間の式の記述例

$PROJ_y$(**前方車両**)

ソースコード 4.60　区間の式の意味関数の記述例

$A_i[\![\ PROJ_y$(**前方車両**)$\ ]\!]$

Coq での記述は式 (4.62) のようになる。また、区間の式の抽象構文はソースコード 4.25 より Coq を用いてソースコード 4.61 のように記述できた。EXP_BBvar は Bounding box 型の変数を表す。Ai は区間の意味関数である。

ソースコード 4.61　区間の式の Coq による記述例

```
EXP_projy (EXP_BBvar "前方車両")
```

ソースコード 4.62　区間の式の意味関数の Coq による記述例

```
1 Ai (EXP_projy (EXP_BBvar "前方車両"))
```

■有理数の式の意味関数　BBSL 上の有理数の式を形式化したものは、有理数となる。有理数の式の意味関数の型を式 (4.11) に示す。$Qexp$ は有理数の式の抽象構文、Q は有理数である。

$$A_q : Iexp \to (\Sigma \to Q) \tag{4.11}$$

有理数の式の意味関数の Coq での実装をソースコード 4.63 に示す。EXP_Qvar は変数であり、変数環境から変数名に対応する値を取得したものが解釈結果となる。EXP_RAT は Bounding box の集合型 SetBB の関数 RAT であり、解釈には Bounding box の集合の実装で実装した RAT 関数をそのまま用いる。EXP_width は幅を取得する関数、EXP_projl、EXP_proju は区間の下限、上限を取得する関数、EXP_projxl、EXP_projxu、EXP_projyl、EXP_projyu は Bounding box の x、y 軸方向の下限、上限を取得する関数である。これらの関数はそれぞれ Bounding box と区間の実装で実装した関数をそのまま解釈に用いる。Aq は Ai と相互参照しているため、実際の実装では Coq の機能である with を用いて実装されている。完全な実装は付録 A に載せている。

ソースコード 4.63　有理数の式の意味関数の Coq での実装

```
1  Fixpoint Aq (expr : Qexp) (env : Env) : option Q :=
2    match expr with
3    | EXP_Q a => Some a
4    | EXP_Qvar s =>
5      match find s env with
6      | Some (Vq q) => Some q
7      | _ => None
8      end
9    | EXP_width i_expr =>
10     match Ai i_expr env with
11     | Some i => Some (width i)
12     | None => None
```

```
13      end
14    | EXP_RAT sbb_expr0 sbb_expr1 =>
15      match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
16      | Some sbb0, Some sbb1 => Some (RAT sbb0 sbb1)
17      | _, _ => None
18      end
19    | EXP_projl i_expr =>
20      match Ai i_expr env with
21      | Some i => Some (lower i)
22      | _ => None
23      end
24    | EXP_proju i_expr =>
25      match Ai i_expr env with
26      | Some i => Some (upper i)
27      | _ => None
28      end
29    | EXP_projxl bb_expr =>
30      match Abb bb_expr env with
31      | Some bb => Some (projxl bb)
32      | None => None
33      end
34    | EXP_projxu bb_expr =>
35      match Abb bb_expr env with
36      | Some bb => Some (projxu bb)
37      | None => None
38      end
39    | EXP_projyl bb_expr =>
40      match Abb bb_expr env with
41      | Some bb => Some (projyl bb)
42      | None => None
43      end
44    | EXP_projyu bb_expr =>
45      match Abb bb_expr env with
46      | Some bb => Some (projyu bb)
47      | None => None
48      end
49  end.
```

例として、ソースコード 4.15 の 35 行目の有理数の式の部分の抽象構文を意味関数で評価する式を式 (4.65) に示す。また、該当部分の抽象構文はソースコード 4.28 より、ソースコード 4.64 のように記述できる。qgt は有理数の比較関数、自動車線左区間集合は Bounding box の集合、割り込み車両は Bounding box である。A_q は有理数の意味関数である。

ソースコード 4.64　有理数の式の記述例

```
1 qgt(
2   RAT(sbbintersection(自動車線左区間集合, {割り込み車両}),
3       sbbintersection(自動車線右区間集合, {割り込み車両})),
4   1.0)
```

ソースコード 4.65　有理数の式の意味関数の記述例

```
1 A_q⟦
2   qgt(
3     RAT(sbbintersection(自動車線左区間集合, {割り込み車両}),
4         sbbintersection(自動車線右区間集合, {割り込み車両})),
5     1.0) ⟧
```

Coq での記述は式 (4.67) のようになる。また、有理数の式の抽象構文はソースコード 4.29 より Coq を用いてソースコード 4.66 のように記述できた。EXP_Qgt は有理数の比較関数、EXP_SBBvar は Bounding box の集合の変数、EXP_BBvar は Bounding box の変数、EXP_Q は有理数のリテラルである。Aq は有理数の意味関数である。

ソースコード 4.66　有理数の集合の式の Coq による記述例

```
1 EXP_Qgt
2   (EXP_RAT
3     (EXP_SBBintersection (EXP_SBBvar "自車線左区間集合") (EXP_makeSBB [
```

```
        EXP_BBvar "割り込み車両" ]))
4     (EXP_SBBintersection (EXP_SBBvar "自車線右区間集合") (EXP_makeSBB [
        EXP_BBvar "割り込み車両" ])))
5   (EXP_Q 1.0)
```

ソースコード 4.67　有理数の式の意味関数の Coq による記述例

```
1 Aq
2   (EXP_Qgt
3     (EXP_RAT
4       (EXP_SBBintersection (EXP_SBBvar "自車線左区間集合") (EXP_makeSBB
          [ EXP_BBvar "割り込み車両" ]))
5       (EXP_SBBintersection (EXP_SBBvar "自車線右区間集合") (EXP_makeSBB
          [ EXP_BBvar "割り込み車両" ])))
6     (EXP_Q 1.0))
```

■ブール値の式の意味関数　BBSL 上の有理数の式を形式化したものは、命題となる。ブール値の式の意味関数の型を式 (4.12) に示す。$Bexp$ は有理数の式の抽象構文、$Prop$ は命題型である。

$$B : Bexp \to (\Sigma \to Prop) \tag{4.12}$$

ブール値の式の意味関数の Coq での実装をソースコード 4.68 に示す。EXP_Bvar は変数であり、変数環境から変数名に対応する値を取得したものが解釈結果となる。EXP_not、EXP_and、EXP_or は論理否定、論理積、論理和である。これらの関数の解釈は Coq の標準ライブラリの Prop 型の実装のコンストラクタをそのまま用いる。EXP_BBeq は等号、EXP_BBoverlap は重なりを判定する関数、EXP_BBsubset、EXP_BBsupset、EXP_BBsubseteq、EXP_BBsupseteq は部分集合の演算である。これらの関数はそれぞれ Bounding box の実装で実装した関数をそのまま解釈に用いる。EXP_Ilt、EXP_Igt は比較関係、EXP_Ieq は等号、EXP_Ioverlap は重なりを判定する関数、EXP_Iin、EXP_Iinrev は所属を判定する関数、EXP_Isubset、EXP_Isupset、EXP_Isubseteq、EXP_Isupseteq

は部分集合の演算である。これらの関数はそれぞれ区間の実装で実装した関数をそのまま解釈に用いる。EXP_Qlt、EXP_Qgt、EXP_Qle、EXP_Qge は比較演算、EXP_Qeq は等号である。これらの関数はそれぞれ Coq の標準ライブラリの有理数の関数をそのまま用いる。EXP_forall、EXP_exists は Bounding box の集合に対する forall、exists の構文である。ここで、Bounding box の集合は有限であったことを思い出すと、forall と exists はそれぞれ論理積と論理和で表現することができる。forall は Bounding box の集合に属する Bounding bo についての論理式すべての論理積を取ることで解釈している。exists は Bounding box の集合に属する Bounding bo についての論理式すべての論理和を取ることで解釈している。

ソースコード 4.68　ブール値の式の意味関数の Coq での実装

```
1  Fixpoint B (expr : Bexp) (env : Env) : option Prop :=
2    match expr with
3    | EXP_Bvar s =>
4      match find s env with
5      | Some (Vb b) => Some b
6      | _ => None
7      end
8    | EXP_not b_expr =>
9      match B b_expr env with
10     | Some b => Some (not b)
11     | None => None
12     end
13   | EXP_and b_expr0 b_expr1 =>
14     match B b_expr0 env, B b_expr1 env with
15     | Some b0, Some b1 => Some (b0 /\ b1)
16     | _, _ => None
17     end
18   | EXP_or b_expr0 b_expr1 =>
19     match B b_expr0 env, B b_expr1 env with
20     | Some b0, Some b1 => Some (b0 \/ b1)
21     | _, _ => None
22     end
23   | EXP_BBeq bb_expr0 bb_expr1 =>
24     match Abb bb_expr0 env, Abb bb_expr1 env with
```

```
25      | Some bb0, Some bb1 => Some (BBeq bb0 bb1)
26      | _, _ => None
27      end
28    | EXP_BBoverlap bb_expr0 bb_expr1 =>
29      match Abb bb_expr0 env, Abb bb_expr1 env with
30      | Some bb0, Some bb1 => Some (BBoverlap bb0 bb1)
31      | _, _ => None
32      end
33    | EXP_BBsubset bb_expr0 bb_expr1 =>
34      match Abb bb_expr0 env, Abb bb_expr1 env with
35      | Some bb0, Some bb1 => Some (BBsubset bb0 bb1)
36      | _, _ => None
37      end
38    | EXP_BBsupset bb_expr0 bb_expr1 =>
39      match Abb bb_expr0 env, Abb bb_expr1 env with
40      | Some bb0, Some bb1 => Some (BBsupset bb0 bb1)
41      | _, _ => None
42      end
43    | EXP_BBsubseteq bb_expr0 bb_expr1 =>
44      match Abb bb_expr0 env, Abb bb_expr1 env with
45      | Some bb0, Some bb1 => Some (BBsubseteq bb0 bb1)
46      | _, _ => None
47      end
48    | EXP_BBsupseteq bb_expr0 bb_expr1 =>
49      match Abb bb_expr0 env, Abb bb_expr1 env with
50      | Some bb0, Some bb1 => Some (BBsupseteq bb0 bb1)
51      | _, _ => None
52      end
53    | EXP_Ilt i_expr0 i_expr1 =>
54      match Ai i_expr0 env, Ai i_expr1 env with
55      | Some i0, Some i1 => Some (Ilt i0 i1)
56      | _, _ => None
57      end
58    | EXP_Igt i_expr0 i_expr1 =>
59      match Ai i_expr0 env, Ai i_expr1 env with
60      | Some i0, Some i1 => Some (Igt i0 i1)
61      | _, _ => None
62      end
```

```
63    | EXP_Ieq i_expr0 i_expr1 =>
64      match Ai i_expr0 env, Ai i_expr1 env with
65      | Some i0, Some i1 => Some (Ieq i0 i1)
66      | _, _ => None
67      end
68    | EXP_Ioverlap i_expr0 i_expr1 =>
69      match Ai i_expr0 env, Ai i_expr1 env with
70      | Some i0, Some i1 => Some (Ioverlap i0 i1)
71      | _, _ => None
72      end
73    | EXP_Iin q_expr i_expr =>
74      match Aq q_expr env, Ai i_expr env with
75      | Some q, Some i => Some (Iin q i)
76      | _, _ => None
77      end
78    | EXP_Iinrev i_expr q_expr =>
79      match Aq q_expr env, Ai i_expr env with
80      | Some q, Some i => Some (Iin q i)
81      | _, _ => None
82      end
83    | EXP_Isubset i_expr0 i_expr1 =>
84      match Ai i_expr0 env, Ai i_expr1 env with
85      | Some i0, Some i1 => Some (Isubset i0 i1)
86      | _, _ => None
87      end
88    | EXP_Isupset i_expr0 i_expr1 =>
89      match Ai i_expr0 env, Ai i_expr1 env with
90      | Some i0, Some i1 => Some (Isupset i0 i1)
91      | _, _ => None
92      end
93    | EXP_Isubseteq i_expr0 i_expr1 =>
94      match Ai i_expr0 env, Ai i_expr1 env with
95      | Some i0, Some i1 => Some (Isubseteq i0 i1)
96      | _, _ => None
97      end
98    | EXP_Isupseteq i_expr0 i_expr1 =>
99      match Ai i_expr0 env, Ai i_expr1 env with
100     | Some i0, Some i1 => Some (Isupseteq i0 i1)
```

```
101     | _, _ => None
102     end
103   | EXP_Qlt q_expr0 q_expr1 =>
104     match Aq q_expr0 env, Aq q_expr1 env with
105     | Some q0, Some q1 => Some (q0 < q1)%Q
106     | _, _ => None
107     end
108   | EXP_Qgt q_expr0 q_expr1 =>
109     match Aq q_expr0 env, Aq q_expr1 env with
110     | Some q0, Some q1 => Some (q0 < q1)%Q
111     | _, _ => None
112     end
113   | EXP_Qeq q_expr0 q_expr1 =>
114     match Aq q_expr0 env, Aq q_expr1 env with
115     | Some q0, Some q1 => Some (q0 = q1)
116     | _, _ => None
117     end
118   | EXP_Qle q_expr0 q_expr1 =>
119     match Aq q_expr0 env, Aq q_expr1 env with
120     | Some q0, Some q1 => Some (q0 <= q1)%Q
121     | _, _ => None
122     end
123   | EXP_Qge q_expr0 q_expr1 =>
124     match Aq q_expr0 env, Aq q_expr1 env with
125     | Some q0, Some q1 => Some (q0 <= q1)%Q
126     | _, _ => None
127     end
128   | EXP_forall bound sbb_expr b_expr =>
129     match Asbb sbb_expr env with
130     | Some sbb => List.fold_left option_and (List.map (fun bb => B
            b_expr env) sbb) (Some True)
131     | _ => None
132     end
133   | EXP_exists bound sbb_expr b_expr =>
134     match Asbb sbb_expr env with
135     | Some sbb => List.fold_left option_or (List.map (fun bb => B
            b_expr env) sbb) (Some False)
136     | _ => None
```

```
137      end
138    end.
```

例として、図 3.3 の 15 行目の Bounding box の式の部分の抽象構文を意味関数で評価する式を式 (4.69) に示す。ブール値の式の抽象構文はソースコード 4.32 のように記述できた。Ioverlap は区間の重なりを判定する関係、前方車両は Bounding box、減速区間は区間である。B はブール値の意味関数である。

ソースコード 4.69　ブール値の式の意味関数の記述例

```
1 B⟦ Ioverlap(PROJ_y(前方車両), 減速区間) ⟧
```

Coq での記述は式 (4.70) のようになる。ブール値の式の抽象構文はソースコード 4.33 のように記述できた。EXP_Ioverlap は区間の重なりを判定する関数、EXP_BBvar は Bounding box の変数、EXP_Ivar は区間の変数である。B はブール値の意味関数である。

ソースコード 4.70　ブール値の式の意味関数の Coq による記述例

```
1 B (EXP_Ioverlap (EXP_projy (EXP_BBvar "前方車両")) (EXP_Ivar "減速区
      間"))
```

第 5 章

実験

第一に、Coq を用いて形式化した BBSL が意図通りの動作をすることを確認するため、いくつかの性質を証明する。Coq で実装した区間、Bounding box、Bounding box の集合についてそれぞれ満たすべき性質を証明する。

第二に、形式化した BBSL の記述能力を確認する。NHTSA の研究 [2] において、自動運転システムの仕様の中でも物体の検知と応答に関するものがまとめてある。先行研究 [7] ではこれらの仕様を BBSL によって記述することでその記述能力を確かめている。これらの仕様を形式化した BBSL でも記述することで、その記述能力を評価する。

第三に、形式化した BBSL の検証能力の実用性を評価するため、自動運転システムの仕様が満たすべき性質をいくつかの証明を行う。

5.1 形式化した BBSL の性質の証明

区間、Bounding box、Bounding box の集合についてその性質の証明を行う。区間、Bounding box、Bounding box の関係・関数とその性質について表 5.1 にまとめる。

証明する性質は、等号については、反対律、対称律、推移律の証明を行う。比較演算に関して、等号を含むものは半順序の性質、すなわち反射律、反対称律、推移律を満たすことの証明を行う。区間の比較演算については反射律、反対称律は成り立たないので、推移律のみの証明を行う。等号を含まないものは強半順序の性質、つまり非反射律と推移律の証明を行う。区間の包含関係について、等号を含むものは束の性質について証明を行う。すなわち、半順序の性質と任意の 2 元の上限が合併、下限が共通部分になることの証明を行う。等号を含まないものは強半順序の性質の証明を行う。区間の共通部分について、図 5.1 のような場合に分けられる。式 (3.9)、(3.34) より、区間はどちらかが小さいまたは大

関係・関数	記法	性質
区間の等号	$=$	同値関係の性質
区間の比較関係	$<, >$	強半順序の性質
区間の比較関係 (等号あり)	$\leq, \geq$	推移律
区間の包含関係	$\subset, \supset$	強半順序の性質
区間の包含関係	$\subseteq, \supseteq$	半順序の性質
区間の共通部分	$\cap$	図 5.1 の場合分けによる性質

表 5.1　BBSL の区間、Bounding box、Bounding box の集合の関係・関数の性質

きい場合と、重なっている場合に分けれれる。前者は図 5.1 の 1, 9 であり、後者は 2 から 8 である。さらに後者は区間の端点の大小関係より 2 から 8 に場合分けされる。よって、それぞれの場合で共通部分が満たすべき性質について証明を行う。合併については標準ライブラリの関数をそのまま使っており、標準ライブラリで十分性質の保証がなされているためここでは証明は行わない。区間と Bounding box の重なりについては、式 (3.34)、(3.35) より共通部分と等号を用いて定義されている。そのため、等号と共通部分の性質を証明すれば十分である。反射律、非反射律、対象律、反対称律、推移律をそれぞれ式 (5.1)、(5.2)、(5.3)、(5.4)、(5.5) に示す。また、半順序、強半順序、同値の性質について表 5.2 にまとめる。

$$\forall X, R(X, X) \tag{5.1}$$

$$\forall X, \neg R(X, X) \tag{5.2}$$

$$\forall XY, R(X, Y) \rightarrow R(Y, X) \tag{5.3}$$

$$\forall XY, R(X, Y) \wedge R(Y, X) \rightarrow X = Y \tag{5.4}$$

$$\forall XYZ, R(X, Y) \wedge R(Y, Z) \rightarrow R(X, Z) \tag{5.5}$$

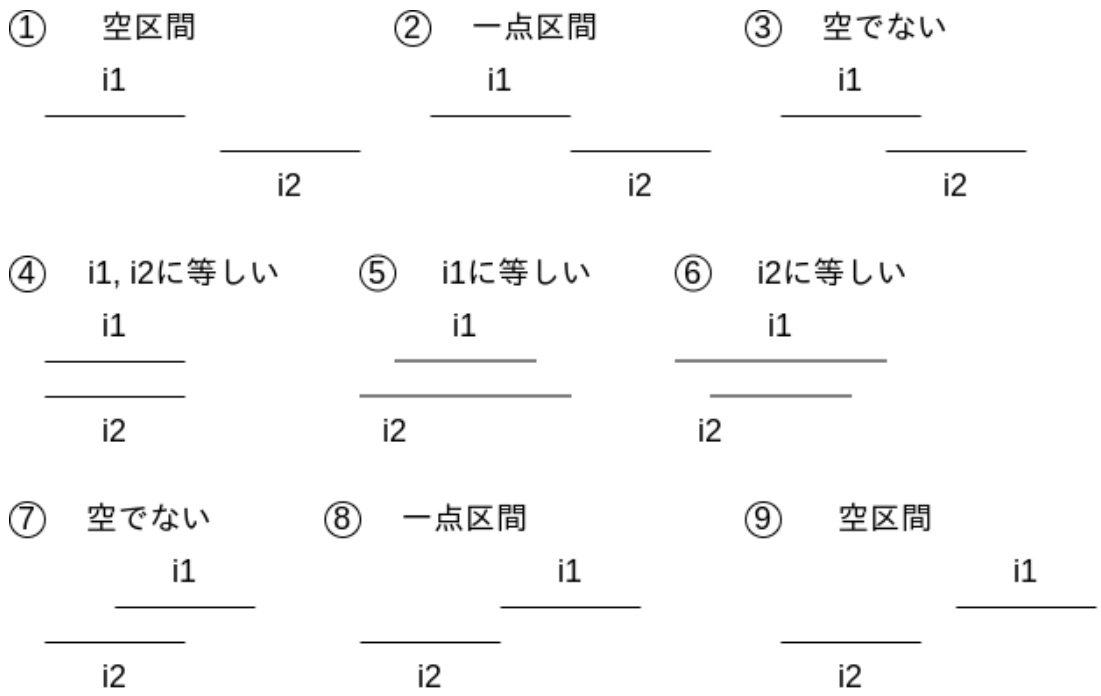

図 5.1　区間の共通部分の場合分け

順序	性質
半順序	反射律、反対称律、推移律
強半順序	非反射律、推移律
同値	反射律、対称律、推移律

表 5.2　順序

■等号の性質　区間と Bounding box には等号が定義されている。等号は同値関係の性質を満たしているべきである。よって、それぞれ同値関係の性質について証明を行う。つまり、反射律、対称律、推移律の証明を行う。等号の性質について Coq での実装をソースコード 5.1 に示す。

ソースコード 5.1　等号の性質の Coq での証明

```
1 (* 区間の等号 Ieq *)
2 Lemma Ieq_refl : forall x, x == x.
3 Lemma Ieq_sym : forall x y, x == y -> y == x.
```

```
4 Lemma Ieq_trans : forall x y z, x == y /\ y == z -> x == z.
5
6 (* Bounding box の等号 BBeq *)
7 Lemma BBeq_refl : forall x, x == x.
8 Lemma BBeq_sym : forall x y, x == y -> y == x.
9 Lemma BBeq_trans : forall x y z, x == y /\ y == z -> x == z.
```

これらの区間と Bounding box の等号の性質はすべて証明できた。証明の全文はは付録 A に載せている。各証明にかかったステップ数を表 5.3 にまとめる。

性質	命題識別子	ステップ数
区間の等号の反射律	Ieq_refl	5
区間の等号の対称律	Ieq_sym	6
区間の等号の推移律	Ieq_trans	7
Bounding box の等号の反射律	BBeq_refl	5
Bounding box の等号の対称律	BBeq_sym	6
Bounding box の等号の推移律	BBeq_trans	7

表 5.3　実験結果　等号の性質

■比較演算の性質　区間について比較演算が定義されている。比較演算は、等号ありの場合は推移律を満たしているべきである。また、等号なしの場合は反対称律と推移律を満たしているべきである。よって、区間の比較関係についてこれらの性質の証明を行った。比較演算の性質について Coq での実装をソースコード 5.2 に示す。

ソースコード 5.2　比較演算の性質の Coq での証明

```
1 (* 区間の比較演算 *)
2 Lemma Ile_trans : forall x y z, Inempty y -> x <= y /\ y <= z -> x
      <= z.
3 Lemma Ilt_antirefl : forall x, Inempty x -> ~x < x.
4 Lemma Ilt_trans : forall x y z, Inempty y -> x < y /\ y < z -> x < z
      .
```

区間の比較演算の性質はすべて証明できた。証明の全文はは付録 A に載せている。各証明にかかったステップ数を表 5.4 にまとめる。

性質	命題識別子	ステップ数
区間の比較演算（等号あり）の推移律	Ile_trans	9
区間の比較演算（等号なし）の非反射律	Ilt_antirefl	9
区間の比較演算（等号なし）の推移律	Ilt_trans	5

表 5.4　実験結果　比較演算の性質

■包含関係の性質　区間と Bounding box について包含関係が定義されている。等号ありの包含関係は半順序の性質を満たしているべきである。つまり、反射率、反対称律、推移律である。よってこれらの性質の証明を区間と Bounding box のそれぞれで行った。また、等号なしの包含関係は強半順序の性質を満たしているべきである。つまり、非反射律と推移律である。これらについても同様にそれぞれで証明を行った。包含関係の性質について Coq での実装をソースコード 5.3 に示す。

ソースコード 5.3　包含関係の性質の Coq での証明

```
1 (* 区間の包含関係(等号あり) *)
2 Lemma Isubseteq_refl : forall x, Isubseteq x x.
3 Lemma Isubseteq_antisym : forall x y, Isubseteq x y /\ Isubseteq y x
      -> x == y.
4 Lemma Isubseteq_trans : forall x y z, Isubseteq x y /\ Isubseteq y z
      -> Isubseteq x z.
5 Lemma Isubseteq_intersection : forall x y,
6   Isubseteq (Iintersection x y) x /\ Isubseteq (Iintersection x y) y.
7
8 (* 区間の包含関係(等号なし) *)
9 Lemma Isubset_irrefl : forall x, Inempty x -> ~Isubset x x.
10 Lemma Isubset_trans : forall i0 i1 i2, Isubset i0 i1 /\ Isubset i1 i2
      -> Isubset i0 i2.
11
12 (* Bounding box の包含関係（等号あり） *)
13 Lemma BBsubseteq_refl : forall x, BBsubseteq x x.
14 Lemma BBsubseteq_antisym : forall a b, BBsubseteq a b /\ BBsubseteq b
```

```
    a -> a == b.
15 Lemma BBsubseteq_trans : forall x y z, BBsubseteq x y /\ BBsubseteq y
    z -> BBsubseteq x z.
16 Lemma BBsubseteq_intersection : forall p q,
17   BBsubseteq (BBintersection p q) p /\ BBsubseteq (BBintersection p q)
       q.
18
19 (* Bounding box の包含関係（等号なし）*)
20 Lemma BBsubset_irrefl : forall x, BBnempty x -> ~BBsubset x x.
21 Lemma BBsubset_trans : forall x y z, BBsubset x y /\ BBsubset y z ->
    BBsubset x z.
```

区間と Bounding box の包含関係の性質はすべて証明できた。証明の全文はは付録 A に載せている。各証明にかかったステップ数を表 5.5 にまとめる。

性質	命題識別子	ステップ数
区間の包含関係（等号あり）の反射律	Isubseteq_refl	5
区間の包含関係（等号あり）の反対称律	Isubseteq_antisym	7
区間の包含関係（等号あり）の推移律	Isubseteq_trans	7
区間の包含関係（等号あり）の下限	Isubseteq_intersection	18
区間の包含関係（等号なし）の非反射律	Isubset_antirefl	5
区間の包含関係（等号なし）の推移律	Isubset_trans	7

表 5.5　実験結果　包含関係の性質

■区間の共通部分　共通部分は区間で定義されている。共通部分は図 5.1 の場合に分けられたので、それぞれについて証明を行う。共通部分の性質について Coq での実装をソースコード 5.4 に示す。

ソースコード 5.4　共通部分の性質の Coq での証明

```
1 Lemma Iintersection_if_divided1 : forall x y, x < y -> Iempty (
    Iintersection x y).
2 Lemma Iintersection_if_divided2 : forall x y,
```

```
3   Inempty x /\ Inempty y -> (upper x == lower y)%Q -> Idot (
        Iintersection x y).
4 Lemma Iintersection_if_divided3 : forall x y,
5   (lower y < upper x)%Q /\ (lower x <= lower y)%Q /\ (upper x <=
        upper y)%Q ->
6     Inempty (Iintersection x y).
7 Lemma Iintersection_if_divided4 : forall x y,
8   x == y -> x == Iintersection x y /\ y == Iintersection x y.
9 Lemma Iintersection_if_divided5 : forall x y, Isubset x y ->
      Iintersection x y == x.
10 Lemma Iintersection_if_divided6 : forall x y, Isubset y x ->
      Iintersection y x == y.
11 Lemma Iintersection_if_divided7 : forall x y,
12   (lower x < upper y)%Q /\ (lower y <= lower x)%Q /\ (upper y <=
        upper x)%Q ->
13     Inempty (Iintersection y x).
14 Lemma Iintersection_if_divided8 : forall x y,
15   Inempty y /\ Inempty x ->
16   (upper y == lower x)%Q -> Idot (Iintersection y x).
17 Lemma Iintersection_if_divided9 : forall x y, y < x -> Iempty (
      Iintersection y x).
```

区間の共通部分の性質はすべて証明できた。証明の全文はは付録 A に載せている。各証明にかかったステップ数を表 5.6 にまとめる。

5.2 形式化した BBSL の記述能力の確認

先行研究 [7] で BBSL の記述能力の確認に用いられた、合計 12 の仕様を形式化した BBSL で記述する実験を行った。記述実験に使用した仕様を表 5.7 にまとめる。

仕様 1 は図 5.2 のような合流の 4 シーンを上から見たもんを記述した仕様である。1 つ目のシーンは合流領域に他車が存在していない場合である。2 つ目のシーンは合流領域の最上端と他車の下側が接しており、かつそれ以外の車両が合流領域に存在していない場合である。3 つ目のシーンは合流領域の上側と下側にそれぞれ他車が存在し、合流領域に侵入しているが包含はされていない場合である。4 つ目のシーンは他車が合流領域に包含さ

性質	命題識別子	ステップ数
場合分け 1	Iintersection_if_divided1	11
場合分け 2	Iintersection_if_divided2	12
場合分け 3	Iintersection_if_divided3	23
場合分け 4	Iintersection_if_divided4	29
場合分け 5	Iintersection_if_divided5	13
場合分け 6	Iintersection_if_divided6	2
場合分け 7	Iintersection_if_divided7	2
場合分け 8	Iintersection_if_divided8	2
場合分け 9	Iintersection_if_divided9	2

表 5.6　実験結果　区間の共通部分

仕様 id	仕様名
1	合流の 4 シーンを上から記述したもの
2	Lead vehicle stopped
3	Debris static in lane
4	Vehicle cutting in
5	Vehicle cutting in(HWD)
6	割合の関係 1 と 2 の記述
7	割合の関係 3 と 4 の記述
8	位置関係 1 と 2 の記述
9	位置関係 3 と 4 の記述
10	包含関係 1 と 2 の記述
11	大小関係 1 と 2 の記述
12	contains

表 5.7　記述実験で使用した仕様

れている場合である。

仕様 2 は前方車両が存在し、停止している場合の仕様である。自動運転車が取りうるケースとして減速、停止、レスポンスなしがある。前方車両がある程度近ければ減速し、さらに近づいて行くと停止するという仕様になっている。レスポンスなしは前方車両が十分遠い場合である。

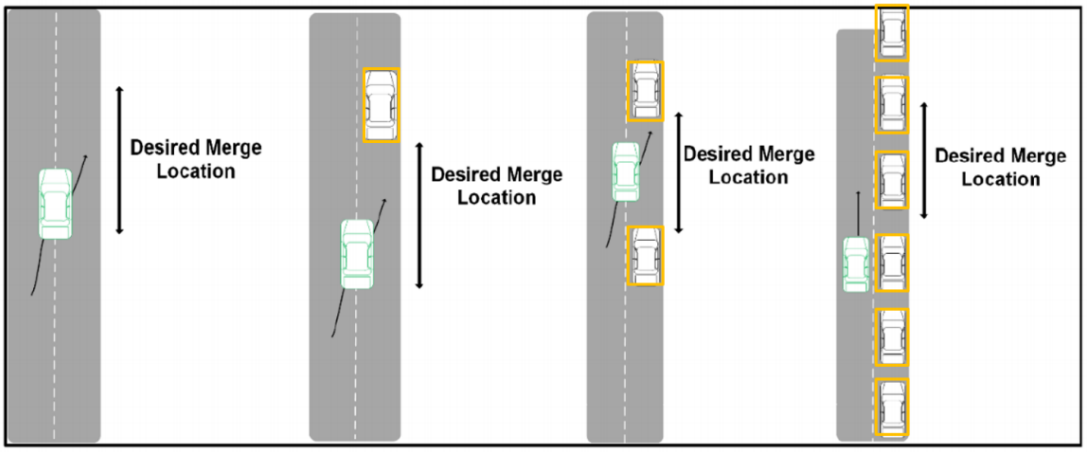

図 5.2　合流の 4 シーンを Bounding Box で囲った図 [7]

仕様 3 は車線上に静的な障害物がある場合の仕様である。ケースとして減速、停止、レスポンスなしがある。仕様 2 と同様に障害物が近づくほど減速し、最終的に停止するという仕様になっている。

仕様 4 は他車両が自車レーンに割り込んでいる場合の仕様である。前方車両と隣のレーンからの割り込み車両が存在することを前提としている。ケースとして停止、減速、前方車に従う、割り込み車両に前方を譲るの 4 つがある。自車線に割り込み車両が侵入しており、かつ自車両に十分に近ければ停止、ある程度近ければ減速となっている。また、割り込み車両に対して前方車両が自車両に近い場合は前方車両に従う。割り込み車両が自車線に侵入はしていないが、ある程度近い場合は割り込み車両に前方を譲る。

仕様 5 は他車が自車レーンに割り込んでいる場合の仕様で、仕様 4 とは異なり左右どちらのレーンからの割り込みかを区別している。ケースとしては、仕様 4 のケースに加えて左の車線に車線変更、右の車線に車線変更が追加されている。自車線を左右で 2 つに分けることで左右どちらのレーンから割り込み車両が来るかを区別している。

仕様 6、7 は IoU(Intersection Over Union) で記述できる関係の仕様である。IoU は画像上のオブジェクト同士の関係を一致率を用いて記述する方法である。BBSL と IoU を比較する目的で記述された仕様となっている。IoU は 2 つの Bounding box の合併に対する共通部分の割合で求められる。2 つの Bounding box が重なっている場合を仕様 6、7 で計 4 つ記述している。

仕様 8、9 は 2 つの Bounding box の位置関係の記述で、IoU では記述することができないものである。仕様 8 の位置関係 1 では、2 つの Bounding box に対して一方が上側にあるケース、位置関係にでは右側にあるケースを記述している。仕様 9 の位置関係 3 では

一方が左上にあるケース、位置関係 4 では左下にあるケースを記述している。

仕様 10 は Bounding box の包含関係について記述している。IoU では画像全体の Bounding box を利用すれば間接的に包含関係を表現することができる。BBSL では画像全体の Bounding box を介さずに包含関係を記述できる。この仕様はその差を確かめるために記述されている。包含関係 1 では一方の Bounding box が他方に包含されているケース、包含関係 2 ではその逆のケースが記述されている。

仕様 11 は Bounding box の大小関係について記述している。大小関係は仕様 10 と同じく IoU では画像全体の Bounding box が必要になるが、BBSL では必要ないことを確かめるために記述されている。大小関係 1 では一方の Bounding box の面積が他方の面積より大きいケース、大小関係 2 では小さいケースが記述されている。

仕様 12 は Binary topological relations で用いられる関係が記述されている。Binary topological relationships[11] は画像上のオブジェクト同士の位置関係を記述できる手法であり、位相的な記述により図 5.3 のような 8 種類の関係を記述することができる。この仕様では Binary topological relations との比較のため、8 種類の関係について BBSL で記述している。

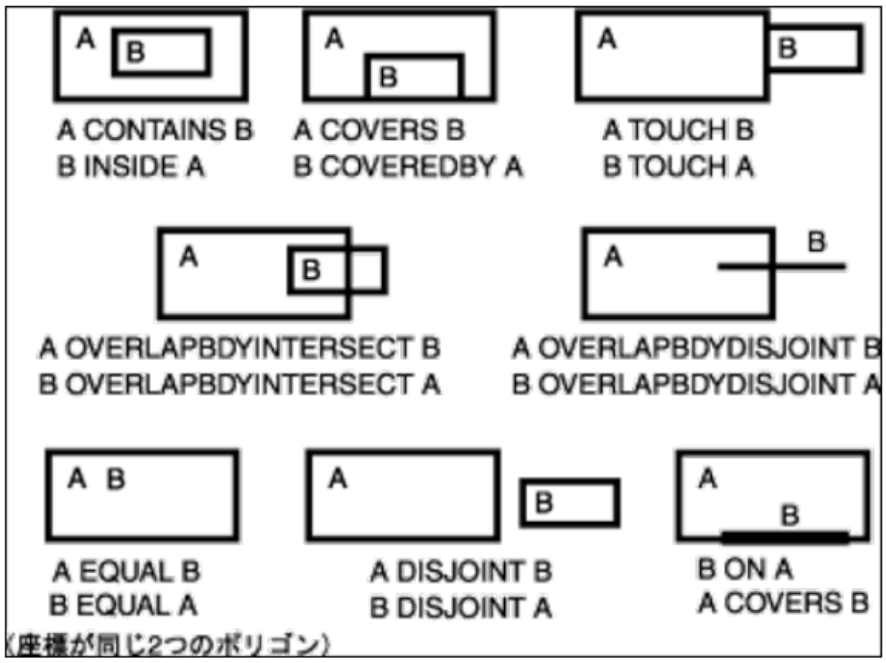

図 5.3　Binary topological relations の関係の図 [7]

記述実験の例として他の車両が自車レーンに割り込んでいるときの仕様について、

BBSL での記述をソースコード 5.5 に示す。図 5.4 は仕様のイメージ図である。

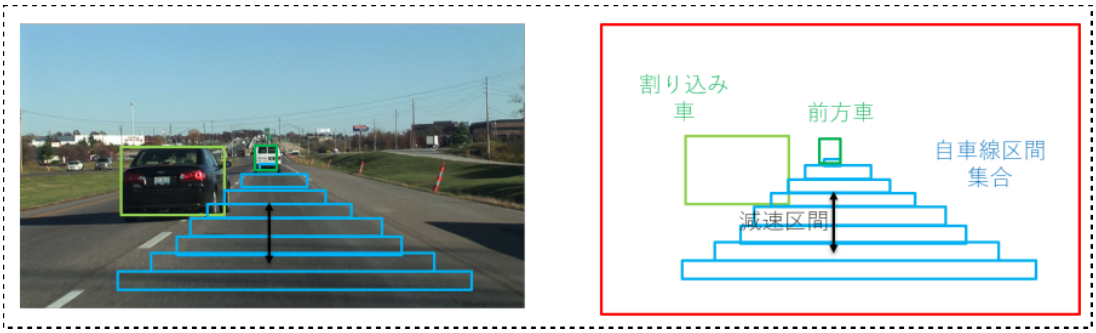

図 5.4　他車自車レーンに割り込んでいる場合のイメージ図 [7]

この仕様の BBSL による記述をソースコード 5.5 に示す。前方車両と他車両が存在し、他車両が左右どちらかの車線から自車線への割り込んできている場合の仕様が記述されている。イメージ図 5.4 では左側から他車が割り込んでいる。ここで、前方カメラからの画像では自車レーンが台形となるが、それを複数の Bounding box の集合として表現している。これが自車線区間集合である。ケースとしては自車線に他車両が割り込んでいて、かつ減速区間より手前に来ていれば停止、停止条件にはなっていないが他車両が減速区間に入っていれば減速、また割り込み車両が前方車両より自車両に近い場合は前方車両に従う、割り込み車両が自車線に侵入はしていないが距離が近い場合は割り込み車両に前方を譲る、となっている。また、これを形式化した BBSL でソースコード 5.6 のように記述することができた。基本的には BBSL の構文から抽象構文を定義し、Coq で実装したのと同じ手順で BBSL で書かれたものを形式化した BBSL で書き換えを行った。形式化した BBSL では外部関数ブロックは省かれている。

ソースコード 5.5　Vehicle cutting in[7]

```
1 exfunction
2 //前方車両の存在をチェック，あれば、true を返す
3 前方車両がある ():bool
4 //前方車両以外の車の存在をチェック，あれば、true を返す
5 他車両がある ():bool
6 //前方車両の boundingbox を返す
7 前方車両 ():bb
8 //割り込み車両の boundingbox を返す
9 割り込み車両 ():bb
```

```
10 //減速しなければならない範囲を boundingbox で返す
11 減速区間 ():bb
12 //自車線の領域を被覆した boundingbox の集合を返す
13
14 自車線区間集合 ():setBB
15 endexfunction
16
17 condition
18 [前方車両がある ()and 他車両がある ()]
19 endcondition
20
21 case 停止
22 let 割り込み車両:bb = 割り込み車両 (),
23 自車線区間集合:setBB = 自車線区間集合 (),
24 減速区間:bb = 減速区間 () in
25 exists x ∈ 自車線区間集合.
26 (PROJx(割り込み車両)≈PROJx(x)) and
27 PROJy(割り込み車両)<PROJy(減速区間))
28 endcase
29
30 case 減速
31 let 割り込み車両:bb = 割り込み車両 (),
32 自車線区間集合:setBB = 自車線区間集合 (),
33 減速区間:bb = 減速区間 () in
34 forall x ∈ 自車線区間集合.
35 not(PROJx(割り込み車両)≈PROJx(x)) and
36 PROJy(割り込み車両)<PROJy(減速区間)) and
37 exists x ∈ 自車線区間集合.
38 (PROJx(割り込み車両)≈PROJx(x)) and
39 PROJy(割り込み車両)≈PROJy(減速区間))
40 endcase
41
42 case 前方車に従う
43 let 割り込み車両:bb = 割り込み車両 (),
44 自車線区間集合:setBB = 自車線区間集合 (),
45 減速区間:bb = 減速区間 () in
46 PROJy(前方車両) < PROJy(割り込み車両)
47 endcase
```

```
48
49 case 割り込み車両に前方を譲る
50 let 割り込み車両:bb = 割り込み車両 (),
51 自車線区間集合:setBB = 自車線区間集合 (),
52 減速区間:bb = 減速区間 () in
53 forall x ∈ 自車線区間集合.
54 not((PROJx(割り込み車両)≈PROJx(x)) and
55 (PROJy(割り込み車両)<PROJy(減速区間) or
56 PROJy(割り込み車両)≈PROJy(減速区間)))
57 endcase
```

ソースコード 5.6　Vehicle cutting in の形式化した BBSL での記述例

```
1  Definition example_vehicle_cutting_in : Spec :=
2    ( CND (EXP_and (EXP_Bvar "前方車両がある") (EXP_Bvar "他車両がある"))
3    , [ ( "停止"
4        , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
5          ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
6          ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
7          ]
8        , EXP_exists "x" (EXP_SBBvar "自車線区間集合")
9            (EXP_and
10             (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                   EXP_projx (EXP_BBvar "x")))
11             (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                   EXP_projy (EXP_BBvar "減速区間"))))
12       )
13     ; ( "減速"
14       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
15         ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
16         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
17         ]
18       , EXP_and
19           (EXP_forall "x" (EXP_SBBvar "自車線区間集合")
20             (EXP_and
```

```
21              (EXP_not (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車
                    両")) (EXP_projx (EXP_BBvar "x"))))
22              (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                    EXP_projy (EXP_BBvar "減速区間")))))
23          (EXP_exists "x" (EXP_SBBvar "自車線区間")
24            (EXP_and
25              (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                    EXP_projx (EXP_BBvar "x")))
26              (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                    EXP_projy (EXP_BBvar "減速区間")))))
27      )
28    ; ( "前方車両に従う"
29      , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
30        ; DEF_BB "前方車両" (EXP_BBvar "前方車両")
31        ]
32      , EXP_Qlt (EXP_projyl (EXP_BBvar "前方車両")) (EXP_projyl (
            EXP_BBvar "割り込み車両"))
33      )
34    ; ( "割り込み車両に前方を譲る"
35      , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
36        ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
37        ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
38        ]
39      , EXP_forall "x" (EXP_SBBvar "自車線区間集合")
40          (EXP_not
41            (EXP_and
42              (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                    EXP_projx (EXP_BBvar "x")))
43              (EXP_or
44                (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                      EXP_projy (EXP_BBvar "減速区間")))
45                (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                      EXP_projy (EXP_BBvar "減速区間"))))))
46      )
47    ]
48  ).
```

他の仕様も同様に記述を行った。すべての仕様の実装は付録 B に載せている。元の BBSL での記述量、形式化した BBSL での記述量を表 5.8 にまとめる。

仕様名	記述量 (行)	形式化した BBSL での記述量 (行)
合流の 4 シーンを上から記述したもの	37	44
Lead vehicle stopped	30	28
Debris static in lane	44	43
Vehicle cutting in	56	48
Vehicle cutting in(HWD)	112	158
割合の関係 1 と 2 の記述	22	22
割合の関係 3 と 4 の記述	26	40
位置関係 1 と 2 の記述	22	16
位置関係 3 と 4 の記述	29	27
包含関係 1 と 2 の記述	22	16
大小関係 1 と 2 の記述	22	20
contains	88	118

表 5.8　実験結果　形式化した BBSL の記述能力の実験

5.3　形式化した BBSL の証明能力の確認

形式化した BBSL の検証能力を確かめるため、仕様 Lead vehicle stopped を用いてケースの網羅性の証明を行う。Lead vehicle stopped の BBSL による記述をソースコード 5.7 に示す。前提条件は「前方車両がある ()」のみ、ケースとしては減速、停止、レスポンスなしの 3 種類がある。この仕様について、前提条件が真のとき、どのような画像を与えても必ずいずれか 1 つ以上のケースが真になることをここではケースの網羅性とよぶ。

ソースコード 5.7　Lead vehicle stopped

```
1 exfunction
2 //前方車両の存在をチェック，あれば、true を返す
3 前方車両がある ():bool
4 //前方車両の boundingbox を返す
5 前方車両 ():bb
```

```
6  //減速しなければならない範囲の boundinbox を返す
7  減速区間 ():bb
8  endexfunction
9
10 condition
11 [前方車両がある ()]
12 endcondition
13
14 case 減速
15 let 前方車両:bb = 前方車両 (),
16 減速区間:bb = 減速区間 () in
17 PROJy(前方車両)≈PROJy(減速区間)
18 endcase
19
20 case 停止
21 let 前方車両:bb = 前方車両 (),
22 減速区間:bb = 減速区間 () in
23 PROJy(前方車両)<PROJy(減速区間)
24 endcase
25
26 case レスポンス無し
27 let 前方車両:bb = 前方車両 (),
28 減速区間:bb = 減速区間 () in
29
30 PROJy(前方車両)>PROJy(減速区間)
31 endcase
```

ケースの網羅性を式 (5.6) に示す。ケース数は有限なので、全称量化子は論理積、存在量化子は論理和を使って書き変えることができる。そのため、実際の Coq での実装では集合ではなくリストを用いており、型はこの式とは異なってくる。また、意味関数は解釈に失敗することがあるため、部分関数として実装されている。部分関数を Coq で実現するための機能として option 型を用いているため、option 型に関わる処理も追加されている。

$$\forall \text{前方車両}, \text{減速区間} \in BB.\text{前方車両が存在する} \in bool.$$
$$\text{前方車両が存在する} \rightarrow \exists (label, case) \in (C_{sprec}[\![lead\_vehicle\_stopped]\!]env). \quad (5.6)$$

この仕様を形式化した BBSL で記述したものをソースコード 5.8 に示す。Coq で実装した BBSL とこの仕様を使ってケースの網羅性の命題の実装を行い、証明する。

ソースコード 5.8　Lead vehicle stopped の形式化した BBSL による記述

```
Definition example_lead_vehicle_stopped : Spec :=
  ( CND (EXP_Bvar "前方車両がある")
  , [ ( "減速"
      , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
        ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
        ]
      , EXP_Ioverlap
          (EXP_projy (EXP_BBvar "前方車両"))
          (EXP_projy (EXP_BBvar "減速区間"))
      )
    ; ( "停止"
      , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
        ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
        ]
      , EXP_Ilt
          (EXP_projy (EXP_BBvar "前方車両"))
          (EXP_projy (EXP_BBvar "減速区間"))
      )
    ; ( "レスポンスなし"
      , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
        ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
        ]
      , EXP_Igt
          (EXP_projy (EXP_BBvar "前方車両"))
          (EXP_projy (EXP_BBvar "減速区間"))
      )
    ]
  ).
```

ケースの網羅性の Coq での実装をソースコード 5.9 に示す。証明の実装は付録 A に載せている。実験よりケースの網羅性を Coq で実装し、証明を与えることができた。証明ステップ数について表 5.9 に示す。ケースの網羅性の Coq での記述について、元の式 5.6 と比べて記述が煩雑になっている。理由として以下の 2 つが挙げられる。1 つ目は、Coq では部分関数を表現するため option を使っているが、その option 特有の記述が増えていることである。意味関数は部分関数であり option 型の値を返すため、その値への処理を記述するために関数 option_map を用いている。2 つ目は、有限なリストに対する存在量化子を論理和に置き換えているためである。これによってリスト特有の処理が追加され、記述が複雑になっている。

ソースコード 5.9　ケースブロックの網羅性の記述

```
1  Proposition comprehensiveness_of_example_lead_vehicle_stopped :
2    forall (exists_front : Prop) (front_bb dec : BB),
3      let evaluated :=
4        Cspec
5          example_lead_vehicle_stopped
6          (add "減速区間" (Vbb dec) (add "前方車両" (Vbb front_bb) (add
               "前方車両がある" (Vb exists_front) (empty Value))))
7      in
8      BBnempty front_bb /\ BBnempty dec /\ exists_front ->
9         match option_map (fun ev => List.fold_left or (List.map snd ev)
               False) evaluated with
10        | Some b => b
11        | _ => False
12        end.
```

仕様名	性質	証明ステップ数
Lead vehicle stopped	ケースブロックの網羅性	61

表 5.9　実験結果　形式化した BBSL の証明能力の実験

第 6 章

評価

BBSL の形式化に際して、BBSL では実数値型が使われているところを、本研究では有理数を用いて実装を行った。有理数を用いてもよい理由として、BBSL が画像を記述対称としているため最小単位がピクセルで表せられることが挙げられる。BBSL が対称とする自動運転システムの仕様は画像認識システムで画像上のオブジェクトを認識することが想定されており、多くの場合で画像はラスター画像となる。ベクター画像の場合も、ラスター画像に変換することで同様に扱える。また、実数ではなく有理数で実装する利点として、実数では比較演算に決定可能性がないのに対し、有理数の比較演算は決定可能性があることが挙げられる。決定可能性があることで、証明を書く際に比較演算で場合分けを行うことができる。例えば、$x < y \vee y \leq x$ などで場合分けが行える。実験で証明を行ったケースの網羅性 (5.6) は比較演算の決定可能性がないと証明することができない。

同じく形式化に関して、集合として扱われるものはリストを用いて実装を行った。BBSL は画像上のオブジェクト同士の位置関係を記述するための言語であり、扱われる値はすべて有限となる。また、リストを用いることで写像の処理や畳込みの処理が簡単に実装できる利点がある。

BBSL を形式化において、抽象構文を定義する中で元の BBSL の構文の誤りを見つけ、修正することができた。これにより、BBSL の言語設計の品質を向上させることができた。BBSL の構文定義について、先行研究 [7] の BNF での定義では論文中に例で記述できないものがあった。1 つ目は外部関数の定義についてである。元の定義をソースコード 6.1 に示す。この定義では外部関数は引数を 1 つしか受け取れないが、本来は複数の引数を受け取ることができる。よって、構文定義 3.2 では複数の引数を受け取れるよう修正した。

ソースコード 6.1　外部関数の元の定義

```
1 function-definition ::= function-name "(" type ")" ":" type
```

2 つ目はブール値の式の構文定義についてである。元の定義をソースコード 6.2 に示す。この定義では構文を定義することができなかった。また、論理否定の構文が定義されていない。よって、構文定義 3.4 の bexp のようにブール値の式の構文を定義し直した。

ソースコード 6.2　ブール値の式の元の定義

```
1 condition-definition ::= (” (” condition-definition condition-binop
      condition-definition ” )” )
2   |(quantification-definition ” (” condition-definition ” )” )
3   |condition-term|none-token
4 condition-term ::= value condition-com value
5 quantification-definition ::= quantification-expr (” , ” quantification
      -definition) ☒ ” .”
6 quantification-expr ::= quantification-token var-name quantification-
      token value
7 value ::= var-namenumber|function-call
8   |(value binop value)
```

3 つ目は let の構文についてである。元の定義をソースコード 6.3 に示す。この定義では let と in を逆に書くことができてしまう。よって構文定義 3.5 のように修正した。

ソースコード 6.3　let の構文の元の定義

```
1 let-definition ::= let-token let-expr ("," let-expr) ☒ let-token
2 let-expr ::= var-name ":" type "=" value
3 let-token ::= "let" "in"
```

形式化した BBSL の性質の証明を行う実験により、BBSL の実装が意図した通りに動

いていることを確かめられた。表 5.3、5.4、5.5、5.6 より、ほとんどの証明が 10 ステップ前後に収まっている。本研究での形式化により必要な性質の証明が少ないステップ数で行えた。しかし、実用的な性質であるケースブロックの網羅性の証明には表 5.9 より 61 ステップがかかっている。また、場合分けを行うまでに証明項を縮小する前処理で 25 ステップかかっている。区間や Bounding box の関係・関数の定義を見直すことで改善する可能性がある。

形式化した BBSL での記述実験について、形式化した BBSL では元の BBSL と同等の記述能力があることが確認できた。また、表 5.8 より BBSL での記述量と形式化した BBSL での記述量はほとんど変わらなかった。しかし、形式化した BBSL では外部関数ブロックを記述していない分記述量が増えていると考えられる。また、形式化した BBSL で仕様を記述する際には抽象構文木をそのまま記述することになるので、記述に難があった。解決策として、BBSL の構文解析器を実装し、Coq で実装した形の抽象構文木を自動生成することが挙げられる。

BBSL の検証能力の実用性に関する実験について、ケースの網羅性を証明できたことで実用性が確かめられた。実験ではケースの網羅性の条件を数式として記述した後、それを Coq で実装し証明を行った。しかし、ケースの網羅性の実装時に実装の都合による元の命題との乖離があった。元の命題が式 (5.6) であったのに対し、形式化した BBSL によって記述した命題は 5.9 となった。またその実装も直感的に記述できるような内容ではなかった。表 5.9 より証明に 61 ステップがかかっている。解決策として、検証したい性質の記述に特化した表明言語の提案が挙げられる。

第 7 章

まとめ・今後の課題

BBSL の構文とその意味関数を定義し、それらを Coq 上で実装することで BBSL の形式化を行った。Coq を用いて計算機上で実装したことにより、BBSL の意味論が厳密に定まった。これによって BBSL でより信頼される仕様を記述できるようになった。また、形式化した BBSL を用いて仕様について望ましい性質を記述し証明することができた。これによって、BBSL で記述された仕様の品質をより向上させることができるようになった。

今後の課題として、まず構文解析器の実装が挙げられる。現状では形式化した BBSL を用いて直接仕様を記述するには労力を伴い、また抽象構文木を直接記述するため視認性が悪く、記述ミスを誘発させやすい。また、すでに BBSL で記述されている仕様を活用できななどの問題がある。構文解析器を実装し BBSL の文法から抽象構文木を自動で生成することで、これらの問題が解決できる。

もう一つの課題として、仕様について検証したい性質の記述が容易でない問題がある。こちらも先程の問題と同じく抽象構文木を直接記述する難しさがある。しかし、BBSL には検証したい性質を記述する帰納はないため、構文解析器を実装してもこの問題は解決しない。また、検証したい性質を記述するにあたって形式化した BBSL の実装上の都合や Coq の都合に左右されやすい問題がある。検証したい性質を記述するための表明言語を定義することでこれらの問題を解決することができる。

参考文献

[1] Barras, Bruno, et al. The Coq proof assistant reference manual: Version 6.1. Diss. Inria, 1997.

[2] Thorn, Eric, et al. A framework for automated driving system testable cases and scenarios. No. DOT HS 812 623. United States. Department of Transportation. National Highway Traffic Safety Administration, 2018.

[3] Research project PEGASUS. The pegasus method. https://www.pegasusprojekt.de/en/pegasus-method.

[4] 中村英夫, 金子貴信. 自動運転システム安全設計－第 2 報:ユースケース及び機能レベル基本アーキテクチャの研究－. JARI Research Journal, 2016.

[5] J Michael Spivey and JR Abrial. The Z notation. Prentice Hall Hemel Hempstead, 1992.

[6] Cliff B Jones. Systematic software development using VDM, Vol. 2. Prentice Hall Englewood Cliffs, 1990.

[7] 田中健人, et al. 自動運転システムにおける画像を対象とした形式仕様記述言語 BBSL の提案. 研究報告ソフトウェア工学 (SE), 2020, 2020.8: 1-8.

[8] G. Melquiond, A. Armando, P. Baumgartner, G. Dowek, ”Proving bounds on real-valued functions with computations”, Proceedings of the 4th International Joint Conference on Automated Reasoning, vol. 5195, pp. 2-17, 2008

[9] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. In 3th International Conference on Principles of Knowledge Representation and Reasoning, pages 165 – 176. Morgan Kaufmann, 1992.

[10] Frank Wolter and Michael Zakharyaschev. Spatio-temporal representation and reasoning based on RCC-8. In 7th International Conference on Principles of Knowledge Representation and Reasoning, pages 3 – 14. Morgan Kaufmann,

2000.

[11] Max J Egenhofer. A formal definition of binary topological relationships. In International conference on foundations of data organization and algorithms, pp. 457 – 472. Springer, 1989.

付録 A

Coq による形式化のソースコード

BBSL を形式化した Coq のソースコードの全文を A.1 に示す。

ソースコード A.1　Coq による形式化のソースコード

```
1  Require Import ClassicalDescription ClassicalFacts QArith OrdersFacts
       QOrderedType Qminmax GenericMinMax String Bool List FMapList
       OrderedTypeEx Ensembles Bool Ltac2.Option.
2  Import ListNotations.
3
4  Declare Scope BBSL_scope.
5
6  Local Open Scope bool_scope.
7  Local Open Scope Q_scope.
8  Local Open Scope string_scope.
9  Local Open Scope list_scope.
10
11 Local Open Scope BBSL_scope.
12
13 (* helper *)
14 Lemma DNE : forall A, ~~A <-> A.
15 Proof.
16   intros.
17   destruct (excluded_middle_informative A).
18   split.
19   - intros. assumption.
```

```
  - intros.
    unfold not. intros.
    apply (H0 H).
  - unfold not. split.
    intros. contradiction. intros. apply (H0 H).
Qed.

Lemma nor_nandn : forall A B, ~(A\/B) -> ~A/\~B.
Proof.
  intros A B HnAorB. split.
  intro HA. apply HnAorB. now left.
  intro HB. apply HnAorB. now right.
Qed.

Lemma nandn_nor : forall A B, ~A /\ ~B -> ~(A \/ B).
Proof.
  intros A B HnAandnB. destruct HnAandnB as (HnA & HnB).
  intro HAorB. destruct HAorB. contradiction. contradiction.
Qed.

Lemma norn_nand : forall A B, ~A /\ ~B -> ~(A \/ B).
Proof.
  intros A B HnAnB. destruct HnAnB as (HnA & HnB).
  intro HAandB. destruct HAandB. contradiction. contradiction.
Qed.

(* require classical facts *)
Lemma nand_norn : forall A B, ~(A /\ B) -> ~A \/ ~B.
Proof.
  intros A B HnAandB.
  destruct (excluded_middle_informative A) as [ HA | HnA].
  right. intro HB. apply (HnAandB (conj HA HB)).
  now left.
Qed.

Lemma Qlt_not_ge_iff : forall x y, x < y <-> ~y <= x.
Proof.
  intros x y. split.
```

```
58   apply Qlt_not_le. apply Qnot_le_lt.
59 Qed.
60
61 Lemma Qle_not_gt_iff : forall x y, x <= y <-> ~y < x.
62 Proof.
63   intros x y. split.
64   apply Qle_not_lt. apply Qnot_lt_le.
65 Qed.
66
67 Definition Interval : Type := Q * Q.
68
69 Definition lower : Q * Q -> Q := fst.
70 Definition upper : Q * Q -> Q := snd.
71 Definition Iin (v : Q) (i : Interval) := lower i <= v <= upper i.
72 Definition Inin (v : Q) (i : Interval) := v < lower i \/ upper i < v.
73 Definition Iempty (i : Interval) := lower i > upper i.
74 Definition Inempty (i :Interval) := lower i <= upper i.
75
76 Lemma Iempty_not_nempty : forall i, Iempty i -> ~Inempty i.
77 Proof.
78   unfold Iempty, Inempty. intros i H.
79   now apply Qlt_not_le.
80 Qed.
81
82 Lemma Inempty_not_empty : forall i, Inempty i -> ~Iempty i.
83 Proof.
84   unfold Inempty, Iempty. intros i H.
85   now apply Qle_not_lt.
86 Qed.
87
88 Lemma Inot_empty_nempty : forall i, ~Iempty i -> Inempty i.
89 Proof.
90   unfold Iempty, Inempty. intros i H.
91   now apply Qnot_lt_le.
92 Qed.
93
94 Lemma Inot_nempty_empty : forall i, ~Inempty i -> Iempty i.
95 Proof.
```

```
96    unfold Iempty, Inempty. intros i H.
97    now apply Qnot_le_lt.
98  Qed.
99
100 Lemma Inot_empty_and_nempty : forall i, ~(Iempty i /\ Inempty i).
101 Proof.
102   unfold not. intros i H. destruct H as (H & H0).
103   apply Inempty_not_empty in H0. contradiction.
104 Qed.
105
106 Lemma Iempty_not_nempty_iff : forall i, Iempty i <-> ~Inempty i.
107 Proof.
108   intros i. split.
109   apply Iempty_not_nempty. apply Inot_nempty_empty.
110 Qed.
111
112 Lemma Inempty_not_nempty_iff : forall i, Inempty i <-> ~Iempty i.
113 Proof.
114   intros i. split.
115   apply Inempty_not_empty. apply Inot_empty_nempty.
116 Qed.
117
118 Lemma Iempty_nempty_dec : forall i, {Iempty i} + {Inempty i}.
119 Proof.
120   unfold Iempty, Inempty. intros i.
121   apply Qlt_le_dec.
122 Qed.
123
124 Lemma Inempty_empty_dec : forall i, {Inempty i} + {Iempty i}.
125 Proof.
126   intros i.
127   elim (Iempty_nempty_dec i).
128   * now right.
129   * now left.
130 Qed.
131
132 Lemma Iin_not_nin : forall v i, Iin v i -> ~Inin v i.
133 Proof.
```

```
134   unfold Iin, Inin, not. intros v i H H0. destruct H as (H & H1).
          destruct H0 as [H0 | H0].
135   apply (Qle_not_lt (lower i) v H H0). apply (Qle_not_lt v (upper i)
          H1 H0).
136 Qed.
137
138 Lemma Inin_not_in : forall v i, Inin v i -> ~Iin v i.
139 Proof.
140   unfold Inin, Iin, not. intros v i H H0. destruct H0 as (H0 & H1).
          destruct H as [H | H].
141   apply (Qle_not_lt (lower i) v H0 H). apply (Qle_not_lt v (upper i)
          H1 H).
142 Qed.
143
144 (* use a classical fact *)
145 Lemma Inot_in_nin : forall v i, ~Iin v i -> Inin v i.
146 Proof.
147   unfold Iin, Inin. intros v i H.
148   rewrite Qlt_not_ge_iff.
149   rewrite Qlt_not_ge_iff.
150   now apply nand_norn in H.
151 Qed.
152
153 Lemma Inot_nin_in : forall v i, ~Inin v i -> Iin v i.
154 Proof.
155   unfold Inin, Iin. intros v i Hnnin.
156   rewrite Qle_not_gt_iff.
157   rewrite Qle_not_gt_iff.
158   now apply nor_nandn in Hnnin.
159 Qed.
160
161 Lemma Iin_not_nin_iff : forall v i, Iin v i <-> ~Inin v i.
162 Proof.
163   intros v i. split.
164   apply Iin_not_nin. apply Inot_nin_in.
165 Qed.
166
167 Lemma Inin_not_in_iff : forall v i, Inin v i <-> ~Iin v i.
```

```
168 Proof.
169   intros v i. split.
170   apply Inin_not_in. apply Inot_in_nin.
171 Qed.
172
173 Lemma Iin_nin_dec : forall v i, {Iin v i} + {Inin v i}.
174 Proof.
175   unfold Iin, Inin. intros v i.
176   destruct (Qlt_le_dec v (lower i)) as [ Hltvli | Hleliv].
177   right. now left.
178   destruct (Qlt_le_dec (upper i) v) as [ Hltuiv | Hlevui ].
179   right. now right.
180   left. apply (conj Hleliv Hlevui).
181 Qed.
182
183 Lemma Inin_in_dec : forall v i, {Inin v i} + {Iin v i}.
184 Proof.
185   intros v i.
186   elim (Iin_nin_dec v i).
187   * now right.
188   * now left.
189 Qed.
190
191 Lemma Iin_lower : forall i, Inempty i -> Iin (lower i) i.
192 Proof.
193   unfold Inempty, Iin. intros i H. split.
194   apply Qle_refl. assumption.
195 Qed.
196
197 Lemma Iin_upper : forall i, Inempty i -> Iin (upper i) i.
198 Proof.
199   unfold Inempty, Iin. intros i H. split.
200   assumption. apply Qle_refl.
201 Qed.
202
203 Definition width (i : Interval) := Qmax 0 (upper i - lower i).
204
205 Definition Ieq (i0 i1 : Interval) := lower i0 == lower i1 /\ upper i0
```

```
      == upper i1.
Definition Ilt (i0 i1 : Interval) := upper i0 < lower i1.
Definition Ile (i0 i1 : Interval) := upper i0 <= lower i1.
Notation Igt a b := (Ilt b a) (only parsing).
Notation Ige a b := (Ile b a) (only parsing).

Infix "==" := Ieq (at level 70, no associativity) : BBSL_scope.
Infix "<" := Ilt : BBSL_scope.
Infix "<=" := Ile : BBSL_scope.
Notation "x > y" := (Ilt y x)(only parsing) : BBSL_scope.
Notation "x >= y" := (Ile y x)(only parsing) : BBSL_scope.
Notation "x <= y <= z" := (x<=y/\y<=z) : BBSL_scope.

Lemma Ieq_refl : forall x, x == x.
Proof.
  unfold Ieq. intros.
  split. apply Qeq_refl. apply Qeq_refl.
Qed.

Lemma Ieq_sym : forall x y, x == y -> y == x.
Proof.
  unfold Ieq. intros. destruct H. split.
  apply (Qeq_sym (lower x) (lower y) H).
  apply (Qeq_sym (upper x) (upper y) H0).
Qed.

Lemma Ieq_sym_iff : forall x y, x == y <-> y == x.
Proof.
  intros. split.
  apply Ieq_sym. apply Ieq_sym.
Qed.

Lemma Ieq_trans : forall x y z, x == y /\ y == z -> x == z.
Proof.
  unfold Ieq. intros. destruct H. destruct H, H0. split.
  apply (Qeq_trans (lower x) (lower y) (lower z) H H0).
  apply (Qeq_trans (upper x) (upper y) (upper z) H1 H2).
Qed.
```

```
243
244 Lemma Ilt_antisymm : forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> Ilt i0
        i1 -> ~Ilt i1 i0.
245 Proof.
246   unfold Ilt. intros.
247   unfold Iempty in H. destruct H.
248   q_order.
249 Qed.
250
251 Lemma Igt_antisymm : forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> Igt i0
        i1 -> ~Igt i1 i0.
252 Proof.
253   intros i0 i1.
254   rewrite (and_comm (~Iempty i0) (~Iempty i1)).
255   apply (Ilt_antisymm i1 i0).
256 Qed.
257
258 Lemma Ilt_not_gt : forall i0 i1,
259   ~Iempty i0 /\ ~Iempty i1 -> Ilt i0 i1 -> ~Igt i0 i1.
260 Proof.
261   intros i0 i1.
262   apply (Ilt_antisymm i0 i1).
263 Qed.
264
265 Lemma Igt_not_lt : forall i0 i1,
266   ~Iempty i0 /\ ~Iempty i1 -> Igt i0 i1 -> ~Ilt i0 i1.
267 Proof.
268   intros i0 i1.
269   rewrite (and_comm (~Iempty i0) (~Iempty i1)).
270   apply (Ilt_not_gt i1 i0).
271 Qed.
272
273 Lemma Ile_trans : forall x y z, Inempty y -> x <= y /\ y <= z -> x
       <= z.
274 Proof.
275   unfold Ile, Inempty.
276   destruct x as (xl, xu). destruct y as (yl, yu). destruct z as (zl,
         zu).
```

```
277   simpl. intros. destruct H0.
278   apply (Qle_trans xu yu zl (Qle_trans xu yl yu H0 H) H1).
279 Qed.
280
281 Lemma Ilt_trans : forall x y z, Inempty y -> x < y /\ y < z -> x < z
        .
282 Proof.
283   unfold Ilt, Inempty.
284   destruct x as (xl, xu). destruct y as (yl, yu). destruct z as (zl,
          zu).
285   simpl. intros. destruct H0.
286   apply (Qlt_trans xu yu zl (Qlt_le_trans xu yl yu H0 H) H1).
287 Qed.
288
289 Lemma Ilt_irrefl : forall x, Inempty x -> ~x < x.
290 Proof.
291   unfold Ilt, Inempty. intros.
292   apply (Qle_not_lt (lower x) (upper x) H).
293 Qed.
294
295 Definition Iintersection (i0 i1 : Interval) : Interval :=
296   (Qmax (lower i0) (lower i1), Qmin (upper i0) (upper i1)).
297
298 Definition Isubset (i0 i1 : Interval) := (lower i1 < lower i0)%Q /\ (
        upper i0 < upper i1)%Q.
299 Definition Isubseteq (i0 i1 : Interval) := (lower i1 <= lower i0)%Q
        /\ (upper i0 <= upper i1)%Q.
300 Notation Isupset a b := (Isubset b a) (only parsing).
301 Notation Isupseteq a b := (Isubseteq b a) (only parsing).
302
303 Lemma Isubseteq_refl : forall x, Isubseteq x x.
304 Proof.
305   unfold Isubseteq. intros. split.
306   apply Qle_refl. apply Qle_refl.
307 Qed.
308
309 Lemma Isubseteq_antisym : forall x y, Isubseteq x y /\ Isubseteq y x
        -> x == y.
```

```
310 Proof.
311   unfold Isubseteq, Ieq. intros. destruct H. destruct H, H0. split.
312   apply (Qle_antisym (lower x) (lower y) H0 H).
313   apply (Qle_antisym (upper x) (upper y) H1 H2).
314 Qed.
315
316 Lemma Isubseteq_trans : forall x y z, Isubseteq x y /\ Isubseteq y z
        -> Isubseteq x z.
317 Proof.
318   unfold Isubseteq. intros. destruct H. destruct H, H0. split.
319   apply (Qle_trans (lower z) (lower y) (lower x) H0 H).
320   apply (Qle_trans (upper x) (upper y) (upper z) H1 H2).
321 Qed.
322
323 Lemma Isubseteq_intersection : forall x y,
324   Isubseteq (Iintersection x y) x /\ Isubseteq (Iintersection x y) y.
325 Proof.
326   unfold Iintersection, Isubseteq. simpl. intros. split.
327   - split.
328   -- rewrite (Q.max_le_iff (lower x) (lower y) (lower x)). left.
          apply Qle_refl.
329   -- rewrite (Q.min_le_iff (upper x) (upper y) (upper x)). left.
          apply Qle_refl.
330   - split.
331   -- apply (Q.max_le_iff (lower x) (lower y) (lower y)). right. apply
           Qle_refl.
332   -- apply (Q.min_le_iff (upper x) (upper y) (upper y)). right. apply
           Qle_refl.
333 Qed.
334
335 Lemma Isubset_irrefl : forall x, Inempty x -> ~Isubset x x.
336 Proof.
337   unfold Isubset, Inempty, not. intros. destruct H0.
338   apply (Qlt_irrefl (lower x) H0).
339 Qed.
340
341 Lemma Isubset_trans : forall i0 i1 i2, Isubset i0 i1 /\ Isubset i1 i2
        -> Isubset i0 i2.
```

```
342 Proof.
343   unfold Isubset. intros. destruct H. destruct H, H0. split.
344   apply (Qlt_trans (lower i2) (lower i1) (lower i0) H0 H).
345   apply (Qlt_trans (upper i0) (upper i1) (upper i2) H1 H2).
346 Qed.
347
348 Lemma Isubset_intersection_l : forall i0 i1, Isubset i0 i1 ->
        Iintersection i0 i1 == i0.
349 Proof.
350   unfold Isubset. unfold Iintersection.
351   destruct i0. destruct i1. simpl.
352   intros. destruct H.
353   unfold Ieq. simpl. split.
354   - rewrite (Q.max_l q q1 (Qlt_le_weak q1 q H)). apply Qeq_refl.
355   - rewrite (Q.min_l q0 q2 (Qlt_le_weak q0 q2 H0)). apply Qeq_refl.
356 Qed.
357
358 Lemma Isupset_intersection_r : forall i0 i1, Isupset i0 i1 ->
        Iintersection i0 i1 == i1.
359 Proof.
360   unfold Isupset. unfold Iintersection.
361   destruct i0. destruct i1. simpl.
362   intros. destruct H.
363   unfold Ieq. simpl. split.
364   - rewrite (Q.max_r q q1 (Qlt_le_weak q q1 H)). apply Qeq_refl.
365   - rewrite (Q.min_r q0 q2 (Qlt_le_weak q2 q0 H0)). apply Qeq_refl.
366 Qed.
367
368 Definition Idot (i : Interval) := (lower i == upper i)%Q.
369
370 Lemma Iintersection_if_divided1 : forall x y, x < y -> Iempty (
        Iintersection x y).
371 Proof.
372   unfold Iintersection, Iempty. simpl. intros.
373   rewrite (Q.min_lt_iff (upper x) (upper y) (Qmax (lower x) (lower y
        ))).
374   rewrite (Q.max_lt_iff (lower x) (lower y) (upper x)).
375   rewrite (Q.max_lt_iff (lower x) (lower y) (upper y)).
```

```
376   left. right. unfold Ilt in H. assumption.
377 Qed.
378
379 Lemma Iintersection_if_divided2 : forall x y,
380   Inempty x /\ Inempty y -> (upper x == lower y)%Q -> Idot (
          Iintersection x y).
381 Proof.
382   unfold Iintersection, Idot, Inempty. simpl. intros. destruct H.
383   apply Q.max_l in H. apply Q.min_l in H1.
384   rewrite <- H0. rewrite Q.max_comm. rewrite H.
385   rewrite H0. rewrite H1. apply Qeq_refl.
386 Qed.
387
388 Lemma Iintersection_if_divided3 : forall x y,
389   (lower y < upper x)%Q /\ (lower x <= lower y)%Q /\ (upper x <=
          upper y)%Q ->
390     Inempty (Iintersection x y).
391 Proof.
392   unfold Iintersection, Inempty. simpl. intros.
393   destruct H. destruct H0.
394   rewrite (Q.max_lub_iff (lower x) (lower y)).
395   split.
396   - rewrite (Q.min_glb_iff (upper x) (upper y)). split.
397   -- apply Qle_lteq. left.
398      apply (Qle_lt_trans (lower x) (lower y) (upper x) H0 H).
399   -- apply Qle_lteq. left.
400      apply (Qlt_le_trans (lower x) (upper x) (upper y) (Qle_lt_trans (
             lower x) (lower y) (upper x) H0 H) H1).
401   - rewrite (Q.min_glb_iff (upper x) (upper y)). split.
402   -- apply Qle_lteq. left. assumption.
403   -- apply Qle_lteq. left.
404      apply (Qlt_le_trans (lower y) (upper x) (upper y) H H1).
405 Qed.
406
407 Lemma Qeq_sym_iff : forall x y, (x == y)%Q <-> (y == x)%Q.
408 Proof.
409   intros. split.
410   - intros. apply Qeq_sym. assumption.
```

```
411   - intros. apply Qeq_sym. assumption.
412 Qed.
413
414 Lemma Iintersection_if_divided4 : forall x y,
415   x == y -> x == Iintersection x y /\ y == Iintersection x y.
416 Proof.
417   unfold Iintersection, Ieq. simpl. intros. destruct H.
418   rewrite (Qeq_sym_iff (lower x) (Qmax (lower x) (lower y))).
419   rewrite (Q.max_l_iff (lower x) (lower y)).
420   rewrite (Qeq_sym_iff (upper x) (Qmin (upper x) (upper y))).
421   rewrite (Q.min_l_iff (upper x) (upper y)).
422   rewrite (Qeq_sym_iff (lower y) (Qmax (lower x) (lower y))).
423   rewrite (Q.max_r_iff (lower x) (lower y)).
424   rewrite (Qeq_sym_iff (upper y) (Qmin (upper x) (upper y))).
425   rewrite (Q.min_r_iff (upper x) (upper y)).
426   split. split.
427   apply Qle_lteq. right. apply Qeq_sym. assumption.
428   apply Qle_lteq. right. assumption.
429   split.
430   apply Qle_lteq. right. assumption.
431   apply Qle_lteq. right. apply Qeq_sym. assumption.
432 Qed.
433
434 Lemma Iintersection_if_divided5 : forall x y, Isubset x y ->
       Iintersection x y == x.
435 Proof.
436   unfold Iintersection, Isubset, Ieq. simpl. intros. destruct H. split
        .
437   rewrite (Q.max_l_iff (lower x) (lower y)). rewrite Qle_lteq. left.
        assumption.
438   rewrite (Q.min_l_iff (upper x) (upper y)). rewrite Qle_lteq. left.
        assumption.
439 Qed.
440
441 Lemma Iintersection_comm : forall i0 i1, Iintersection i0 i1 ==
       Iintersection i1 i0.
442 Proof.
443   unfold Iintersection.
```

```
444   intros.
445   destruct i0. destruct i1.
446   simpl. unfold Ieq. simpl.
447   rewrite (Q.max_comm q1 q).
448   rewrite (Q.min_comm q2 q0).
449   split. apply Qeq_refl. apply Qeq_refl.
450 Qed.
451
452 Lemma Iintersection_if_divided6 : forall x y, Isubset y x ->
        Iintersection y x == y.
453 Proof.
454   intros x y.
455   apply (Iintersection_if_divided5 y x).
456 Qed.
457
458 Lemma Iintersection_if_divided7 : forall x y,
459   (lower x < upper y)%Q /\ (lower y <= lower x)%Q /\ (upper y <=
          upper x)%Q ->
460     Inempty (Iintersection y x).
461 Proof.
462   intros x y.
463   apply (Iintersection_if_divided3 y x).
464 Qed.
465
466 Lemma Iintersection_if_divided8 : forall x y,
467   Inempty y /\ Inempty x ->
468   (upper y == lower x)%Q -> Idot (Iintersection y x).
469 Proof.
470   intros x y.
471   apply (Iintersection_if_divided2 y x).
472 Qed.
473
474 Lemma Iintersection_if_divided9 : forall x y, y < x -> Iempty (
        Iintersection y x).
475 Proof.
476   intros x y.
477   apply (Iintersection_if_divided1 y x).
478 Qed.
```

```
479
480 Lemma Iempty_intersection : forall i0 i1, Iempty i0 -> Iempty (
        Iintersection i0 i1).
481 Proof.
482   unfold Iempty. unfold Iintersection. simpl.
483   intros. destruct i0. destruct i1. simpl. simpl in H.
484   rewrite (Q.min_lt_iff q0 q2 (Qmax q q1)).
485   rewrite (Q.max_lt_iff q q1 q0).
486   left. left. assumption.
487 Qed.
488
489 Definition Ioverlap (i0 i1 : Interval) : Prop :=
490   ~Iempty (Iintersection i0 i1).
491
492 (* helper *)
493 Lemma Qmin_ltl_comm : forall q q0 q1 : Q, (Qmin q0 q1 < q)%Q <-> (
        Qmin q1 q0 < q)%Q.
494 Proof.
495   intros.
496   rewrite (Q.min_comm q1 q0).
497   split.
498   - intros. assumption.
499   - intros. assumption.
500 Qed.
501
502 (* helper *)
503 Lemma Qmax_ltr_comm : forall q q0 q1 : Q, (q < Qmax q0 q1)%Q <-> (q <
         Qmax q1 q0)%Q.
504 Proof.
505   intros.
506   rewrite (Q.max_comm q1 q0).
507   split.
508   - intros. assumption.
509   - intros. assumption.
510 Qed.
511
512 Lemma Ioverlap_comm : forall i0 i1, Ioverlap i0 i1 <-> Ioverlap i1 i0.
513 Proof.
```

```
514   unfold Ioverlap. unfold Iempty. unfold Iintersection. unfold not.
          simpl.
515   intros. destruct i0. destruct i1. simpl. split.
516   - intros.
517     rewrite (Qmin_ltl_comm (Qmax q1 q) q2 q0) in H0.
518     rewrite (Qmax_ltr_comm (Qmin q0 q2) q1 q) in H0.
519     apply H.
520     assumption.
521   - intros.
522     rewrite (Qmin_ltl_comm (Qmax q q1) q0 q2) in H0.
523     rewrite (Qmax_ltr_comm (Qmin q2 q0) q q1) in H0.
524     apply H.
525     assumption.
526 Qed.
527
528 Lemma Ilt_not_overlap : forall i0 i1,
529   Ilt i0 i1 -> ~Ioverlap i0 i1.
530 Proof.
531   unfold Ilt. unfold Ioverlap. unfold Iempty. unfold Iintersection.
          unfold not.
532   intros. destruct i0. destruct i1. simpl in H0. simpl in H.
533   destruct H0.
534   rewrite (Q.max_lt_iff q q1 (Qmin q0 q2)).
535   rewrite (Q.min_lt_iff q0 q2 q1).
536   right. left.
537   assumption.
538 Qed.
539
540 Lemma Igt_not_overlap : forall i0 i1,
541   Igt i0 i1 -> ~Ioverlap i0 i1.
542 Proof.
543   intros i0 i1.
544   rewrite (Ioverlap_comm i0 i1).
545   apply (Ilt_not_overlap i1 i0).
546 Qed.
547
548 Lemma Ioverlap_not_lt : forall i0 i1, Ioverlap i0 i1 -> ~Ilt i0 i1.
549 Proof.
```

```
550   unfold Ilt. unfold Ioverlap. unfold Iempty. unfold Iintersection.
          unfold not.
551   intros. destruct i0. destruct i1. simpl in H. simpl in H0.
552   destruct H.
553   rewrite (Q.min_lt_iff q0 q2 (Qmax q q1)).
554   rewrite (Q.max_lt_iff q q1 q0).
555   left. right.
556   assumption.
557 Qed.
558
559 Lemma Ioverlap_not_gt : forall i0 i1, Ioverlap i0 i1 -> ~Igt i0 i1.
560 Proof.
561   intros i0 i1.
562   rewrite (Ioverlap_comm i0 i1).
563   apply (Ioverlap_not_lt i1 i0).
564 Qed.
565
566 (* use classical facts *)
567 Lemma not_Ioverpal_lt_gt : forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> ~
        Ioverlap i0 i1 <-> Ilt i0 i1 \/ Igt i0 i1.
568 Proof.
569   unfold Ioverlap. unfold Iempty. unfold Iintersection. unfold Ilt.
          unfold Igt.
570   intros. destruct i0. destruct i1. simpl.
571   rewrite (DNE (Qmin q0 q2 < Qmax q q1)%Q).
572   simpl in H. unfold not in H. destruct H.
573   rewrite (Q.min_lt_iff q0 q2 (Qmax q q1)).
574   rewrite (Q.max_lt_iff q q1 q0).
575   rewrite (Q.max_lt_iff q q1 q2).
576   split.
577   - intros.
578     destruct H1. destruct H1.
579     contradiction.
580     left. assumption.
581     destruct H1.
582     right. assumption.
583     contradiction.
584   - intros.
```

```
585     destruct H1.
586     left. right. assumption.
587     right. left. assumption.
588 Qed.
589
590 Lemma not_lt_gt_overlap : forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> ~
        Ilt i0 i1 /\ ~Igt i0 i1 -> Ioverlap i0 i1.
591 Proof.
592   unfold Ilt. unfold Igt. unfold Ioverlap. unfold Iempty. unfold
          Iintersection. unfold not. simpl.
593   intros.
594   destruct i0. destruct i1. simpl in H. simpl in H0. simpl in H1.
595   destruct H. destruct H0.
596   rewrite (Q.min_lt_iff q0 q2 (Qmax q q1)) in H1.
597   rewrite (Q.max_lt_iff q q1 q0) in H1.
598   rewrite (Q.max_lt_iff q q1 q2) in H1.
599   destruct H1. destruct H1.
600   contradiction.
601   contradiction.
602   destruct H1. contradiction. contradiction.
603 Qed.
604
605 Lemma Qlt_asym : forall q0 q1 : Q, ~((q0 < q1)%Q /\ (q1 < q0)%Q).
606 Proof.
607   unfold not. intros. destruct H.
608   q_order.
609 Qed.
610
611 Lemma Ilt_overlap_gt : forall i0 i1,
612   Inempty i0 /\ Inempty i1 ->
613     Ilt i0 i1 \/Ioverlap i0 i1 \/ Igt i0 i1.
614 Proof.
615   unfold Ioverlap. unfold Ilt. unfold Igt. unfold Iempty. unfold
          Inempty. unfold Iintersection. unfold not. simpl.
616   destruct i0 as (i0l, i0u). destruct i1 as (i1l, i1u). simpl.
617   rewrite (Q.min_lt_iff i0u i1u (Qmax i0l i1l)).
618   rewrite (Q.max_lt_iff i0l i1l i0u).
619   rewrite (Q.max_lt_iff i0l i1l i1u).
```

```
620   intros. destruct H.
621   destruct (Qlt_le_dec i0u i1l).
622   - left. assumption.
623   - destruct (Qlt_le_dec i1u i0l).
624   -- right. right. assumption.
625   -- right. left. intros. destruct H1. destruct H1.
626   --- apply (Qle_lteq i0l i0u) in H. destruct H.
627   ---- apply (Qlt_asym i0l i0u (conj H H1)).
628   ---- rewrite H in H1. apply (Qlt_irrefl i0u H1).
629   --- apply (Qle_lteq i1l i0u) in q. destruct q.
630   ---- apply (Qlt_asym i0u i1l (conj H1 H2)).
631   ---- rewrite H2 in H1. apply (Qlt_irrefl i0u H1).
632   --- destruct H1.
633   ---- apply (Qle_lteq i0l i1u) in q0. destruct q0.
634   ----- apply (Qlt_asym i0l i1u (conj H2 H1)).
635   ----- rewrite H2 in H1. apply (Qlt_irrefl i1u H1).
636   ---- apply (Qle_lteq i1l i1u) in H0. destruct H0.
637   ----- apply (Qlt_asym i1l i1u (conj H0 H1)).
638   ----- rewrite H0 in H1. apply (Qlt_irrefl i1u H1).
639 Qed.
640
641
642 Lemma Isubset_overlap :
643   forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> Isubset i0 i1 -> Ioverlap
          i0 i1.
644 Proof.
645   unfold Ioverlap. unfold Isubset. unfold Iintersection. unfold Iempty.
          unfold not. simpl.
646   intros. destruct i0. destruct i1. simpl in H. simpl in H0. simpl in
        H1.
647   destruct H. destruct H0.
648   rewrite (Q.min_lt_iff q0 q2 (Qmax q q1)) in H1.
649   rewrite (Q.max_lt_iff q q1 q0) in H1.
650   rewrite (Q.max_lt_iff q q1 q2) in H1.
651   destruct H1. destruct H1.
652   contradiction.
653   apply (H (Qlt_trans q0 q1 q H1 H0)).
654   destruct H1. apply (H (Qlt_trans q0 q2 q H3 H1)).
```

```
655   contradiction.
656 Qed.
657
658 Lemma Isupset_overlap :
659   forall i0 i1, ~Iempty i0 /\ ~Iempty i1 -> Isupset i0 i1 -> Ioverlap
          i0 i1.
660 Proof.
661   intros i0 i1.
662   rewrite (Ioverlap_comm i0 i1).
663   rewrite (and_comm (~Iempty i0) (~Iempty i1)).
664   apply (Isubset_overlap i1 i0).
665 Qed.
666
667 Definition BB : Type := Interval * Interval.
668
669 Definition projx (bb : BB) : Interval :=
670   match bb with
671   | (x, _) => x
672   end.
673
674 Definition projy (bb : BB) : Interval :=
675   match bb with
676   | (_, y) => y
677   end.
678
679 Definition projxl (bb : BB) : Q :=
680   lower (projx bb).
681
682 Definition projxu (bb : BB) : Q :=
683   upper (projx bb).
684
685 Definition projyl (bb : BB) : Q :=
686   lower (projy bb).
687
688 Definition projyu (bb : BB) : Q :=
689   upper (projy bb).
690
691 Definition BBempty (bb : BB) : Prop :=
```

```
692   Iempty (projx bb) /\ Iempty (projy bb).
693
694 Definition BBnempty (bb : BB) : Prop :=
695   Inempty (projx bb) \/ Inempty (projy bb).
696
697 Definition BBeq (bb0 bb1 : BB) : Prop :=
698   Ieq (projx bb0) (projx bb1) /\ Ieq (projy bb0) (projy bb1).
699
700 Infix "==" := BBeq (at level 70, no associativity) : BBSL_scope.
701
702 Lemma BBempty_not_nempty : forall x, BBempty x -> ~BBnempty x.
703 Proof.
704   unfold BBempty, BBnempty, not. intros x H I. destruct H as (H & H0).
        destruct I as [I | I].
705   apply (Inot_empty_and_nempty (projx x) (conj H I)).
706   apply (Inot_empty_and_nempty (projy x) (conj H0 I)).
707 Qed.
708
709 Lemma BBnempty_not_empty : forall x, BBempty x -> ~BBnempty x.
710 Proof.
711   unfold BBempty, BBnempty. intros x Hexandey. destruct Hexandey as (
        Hex & Hey).
712   intro Hnxorny. destruct Hnxorny as [Hnx | Hny].
713   apply (Inot_empty_and_nempty (projx x) (conj Hex Hnx)).
714   apply (Inot_empty_and_nempty (projy x) (conj Hey Hny)).
715 Qed.
716
717 Lemma BBnot_nempty_empty : forall x, ~BBnempty x -> BBempty x.
718 Proof.
719   unfold BBnempty, BBempty. intros x H.
720   rewrite Iempty_not_nempty_iff.
721   rewrite Iempty_not_nempty_iff.
722   now apply nor_nandn.
723 Qed.
724
725 (* require classical facts *)
726 Lemma BBnot_empty_nempty : forall x, ~BBempty x -> BBnempty x.
727 Proof.
```

```
728   unfold BBempty, BBnempty. intros x Hnexandey.
729   rewrite Inempty_not_empty_iff.
730
731 Lemma BBempty_not_nempty_iff : forall x, BBempty x <-> ~BBnempty x.
732 Proof.
733   intro x. split.
734   apply BBempty_not_nempty.
735   apply BBnot_nempty_empty.
736 Qed.
737
738 Lemma BB
739
740
741 Lemma BBeq_refl : forall x, x == x.
742 Proof.
743   unfold BBeq. intros. split.
744   apply Ieq_refl. apply Ieq_refl.
745 Qed.
746
747 Lemma BBeq_sym : forall x y, x == y -> y == x.
748 Proof.
749   unfold BBeq. intros. destruct H. split.
750   apply (Ieq_sym (projx x) (projx y) H).
751   apply (Ieq_sym (projy x) (projy y) H0).
752 Qed.
753
754 Lemma BBeq_sym_iff : forall x y, x == y <-> y == x.
755 Proof.
756   intros. split.
757   apply BBeq_sym. apply BBeq_sym.
758 Qed.
759
760 Lemma BBeq_trans : forall x y z, x == y /\ y == z -> x == z.
761 Proof.
762   unfold BBeq. intros. destruct H. destruct H, H0. split.
763   apply (Ieq_trans (projx x) (projx y) (projx z) (conj H H0)).
764   apply (Ieq_trans (projy x) (projy y) (projy z) (conj H1 H2)).
765 Qed.
```

```
766
767 Definition BBoverlap (bb0 bb1 : BB) : Prop :=
768   Ioverlap (projx bb0) (projx bb1) /\ Ioverlap (projy bb0) (projy bb1
        ).
769
770 Definition BBsubset (bb0 bb1 : BB) :=
771   Isubset (projx bb0) (projx bb1) /\ Isubset (projy bb0) (projy bb1).
772 Definition BBsubseteq (bb0 bb1 : BB) :=
773   Isubseteq (projx bb0) (projx bb1) /\ Isubseteq (projy bb0) (projy
        bb1).
774 Notation BBsupset a b := (BBsubset b a) (only parsing).
775 Notation BBsupseteq a b := (BBsubseteq b a) (only parsing).
776
777 Lemma BBsubseteq_refl : forall x, BBsubseteq x x.
778 Proof.
779   unfold BBsubseteq. intros. split.
780   apply Isubseteq_refl. apply Isubseteq_refl.
781 Qed.
782
783 Lemma BBsubseteq_antisym : forall a b, BBsubseteq a b /\ BBsubseteq b
       a -> a == b.
784 Proof.
785   unfold BBsubseteq. destruct a as (ax, ay). destruct b as (bx, _by).
        unfold BBeq.
786   simpl. intros. destruct H. destruct H, H0. split.
787   apply (Isubseteq_antisym ax bx (conj H H0)).
788   apply (Isubseteq_antisym ay _by (conj H1 H2)).
789 Qed.
790
791 Lemma BBsubseteq_trans : forall x y z, BBsubseteq x y /\ BBsubseteq y
       z -> BBsubseteq x z.
792 Proof.
793   unfold BBsubseteq, BBnempty. intros. destruct H. destruct H, H0.
        split.
794   apply (Isubseteq_trans (projx x) (projx y) (projx z) (conj H H0)).
795   apply (Isubseteq_trans (projy x) (projy y) (projy z) (conj H1 H2)).
796 Qed.
797
```

```
798 Definition BBintersection (bb0 bb1 : BB) : BB :=
799   (Iintersection (projx bb0) (projx bb1), Iintersection (projy bb0) (
        projy bb1)).
800
801 Lemma BBsubseteq_intersection : forall p q,
802   BBsubseteq (BBintersection p q) p /\ BBsubseteq (BBintersection p q)
         q.
803 Proof.
804   unfold BBsubseteq, BBintersection. destruct p as (px, py). destruct
        q as (qx, qy).
805   simpl.
806   apply and_assoc.
807   rewrite (and_comm (Isubseteq (Iintersection py qy) py) (Isubseteq (
        Iintersection px qx) qx /\ Isubseteq (Iintersection py qy) qy)).
808   rewrite (and_assoc (Isubseteq (Iintersection px qx) qx) (Isubseteq (
        Iintersection py qy) qy) (Isubseteq (Iintersection py qy) py)).
809   rewrite <- (and_assoc (Isubseteq (Iintersection px qx) px) (
        Isubseteq (Iintersection px qx) qx) (Isubseteq (Iintersection py
         qy) qy /\ Isubseteq (Iintersection py qy) py)).
810   split.
811   apply (Isubseteq_intersection px qx).
812   apply and_comm.
813   apply (Isubseteq_intersection py qy).
814 Qed.
815
816 Lemma BBsubset_irrefl : forall x, BBnempty x -> ~BBsubset x x.
817 Proof.
818   unfold BBsubset, BBnempty, not. intros. destruct H, H0.
819   apply (Isubset_irrefl (projx x) H H0).
820 Qed.
821
822 Lemma BBsubset_trans : forall x y z, BBsubset x y /\ BBsubset y z ->
       BBsubset x z.
823 Proof.
824   unfold BBsubset. intros. destruct H. destruct H, H0. split.
825   apply (Isubset_trans (projx x) (projx y) (projx z) (conj H H0)).
826   apply (Isubset_trans (projy x) (projy y) (projy z) (conj H1 H2)).
827 Qed.
```

```
828
829
830
831 Definition BBarea (bb : BB) : Q :=
832   width (projx bb) * width (projy bb).
833
834 Definition SetBB : Type := list BB.
835
836 Fixpoint _BB_SBBintersection (bb : BB) (sbb accum : SetBB) : SetBB :=
837   match sbb with
838   | nil => accum
839   | cons bb' sbb' => _BB_SBBintersection bb sbb' (cons (BBintersection
          bb bb') accum)
840   end.
841
842 Fixpoint _SBBintersection (sbb0 sbb1 accum : SetBB) : SetBB :=
843   match sbb0 with
844   | nil => accum
845   | cons bb sbb => _SBBintersection sbb sbb1 (_BB_SBBintersection bb
         sbb1 nil ++ accum)
846   end.
847
848 Definition SBBintersection (sbb0 sbb1 : SetBB) : SetBB :=
849   _SBBintersection sbb0 sbb1 nil.
850
851 Definition SBBunion (sbb0 sbb1 : SetBB) : SetBB :=
852   sbb0 ++ sbb1.
853
854 Definition BB2area (bb0 bb1 : BB) : Q :=
855   BBarea bb0 + BBarea bb1 - BBarea (BBintersection bb0 bb1).
856
857 Fixpoint _SetBBarea (sbb accum : SetBB) (area : Q) : Q :=
858   match sbb with
859   | nil => area
860   | cons bb sbb' =>
861     let sbb'' := _BB_SBBintersection bb accum nil in
862     let sbb''area := List.fold_right Qplus 0 (List.map BBarea sbb'')
          in
```

```
863     _SetBBarea sbb' (cons bb accum) (area + BBarea bb - sbb''area)
864   end.
865
866 Definition SetBBarea (sbb : SetBB) : Q :=
867   _SetBBarea sbb nil 0.
868
869 Definition RAT (sbb0 sbb1 : SetBB) : Q :=
870   SetBBarea sbb0 / SetBBarea sbb1.
871
872 Inductive SBBexp : Set :=
873   | EXP_SBBvar (x : string)
874   | EXP_SBBintersection (sbb0 sbb1 : SBBexp) | EXP_SBBunion (sbb0 sbb1
          : SBBexp)
875   | EXP_makeSBB (bbs : list BBexp)
876 with BBexp : Set :=
877   | EXP_BBvar (x : string)
878   | EXP_makeBB (x y : Iexp)
879   (* 画像全体のBB *)
880   | EXP_BBimg
881 with Iexp : Set :=
882   | EXP_Ivar (x : string)
883   | EXP_projx (bb : BBexp) | EXP_projy (bb : BBexp)
884   | EXP_Iintersection (i0 i1 : Iexp)
885   | EXP_makeI (l u : Qexp)
886 with Qexp : Set :=
887   | EXP_Q (a: Q)
888   | EXP_Qvar (x : string)
889   | EXP_width (i : Iexp) | EXP_RAT (sbb0 sbb1 : SBBexp)
890   | EXP_projl (i : Iexp) | EXP_proju (i : Iexp)
891   | EXP_projxl (bb : BBexp) | EXP_projxu (bb : BBexp)
892   | EXP_projyl (bb : BBexp) | EXP_projyu (bb : BBexp).
893
894 Inductive Bexp : Set :=
895   | EXP_Bvar (x : string)
896   | EXP_not (b : Bexp) | EXP_and (b0 b1 : Bexp) | EXP_or (b0 b1 :
        Bexp)
897   | EXP_BBeq (bb0 bb1 : BBexp)
898   | EXP_BBoverlap (bb0 bb1 : BBexp)
```

```
899   | EXP_BBsubset (bb0 bb1 : BBexp) | EXP_BBsupset (bb0 bb1 : BBexp)
900   | EXP_BBsubseteq (bb0 bb1 : BBexp) | EXP_BBsupseteq (bb0 bb1 : BBexp
          )
901   | EXP_Ilt (i0 i1 : Iexp) | EXP_Igt (i0 i1 : Iexp) | EXP_Ieq (i0 i1
          : Iexp)
902   | EXP_Ioverlap (i0 i1 : Iexp)
903   | EXP_Iin (q : Qexp) (i : Iexp) | EXP_Iinrev (i : Iexp) (q : Qexp)
904   | EXP_Isubset (i0 i1 : Iexp) | EXP_Isupset (i0 i1 : Iexp)
905   | EXP_Isubseteq (i0 i1 : Iexp) | EXP_Isupseteq (i0 i1 : Iexp)
906   | EXP_Qlt (q0 q1 : Qexp) | EXP_Qgt (q0 q1 : Qexp)
907   | EXP_Qeq (q0 q1 : Qexp)
908   | EXP_Qle (q0 q1 : Qexp) | EXP_Qge (q0 q1 : Qexp)
909   | EXP_forall (bound : string) (sbb : SBBexp) (b : Bexp)
910   | EXP_exists (bound : string) (sbb : SBBexp) (b : Bexp).
911
912 Inductive Cond : Set :=
913   | CND_None
914   | CND (b : Bexp).
915
916 Inductive Def : Set :=
917   | DEF_SBB (x : string) (sbb : SBBexp)
918   | DEF_BB (x : string) (bb : BBexp)
919   | DEF_I (x : string) (i : Iexp)
920   | DEF_Q (x : string) (q : Qexp)
921   | DEF_B (x : string) (b : Bexp).
922
923 Definition Case : Set := string * list Def * Bexp.
924
925 Definition Spec : Set := Cond * list Case.
926
927 Module Import M := FMapList.Make(String_as_OT).
928
929 Inductive Value : Type :=
930   | Vb (x : Prop) | Vq (x : Q) | Vi (x : Interval)
931   | Vbb (x : BB) | Vsbb (x : SetBB).
932
933 Definition Env := M.t Value.
934
```

```
935 Fixpoint Asbb (expr : SBBexp) (env : Env) : option SetBB :=
936   match expr with
937   | EXP_SBBvar s =>
938     match find s env with
939     | Some (Vsbb sbb) => Some sbb
940     | _ => None
941     end
942   | EXP_SBBintersection sbb_expr0 sbb_expr1 =>
943     match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
944     | Some sbb0, Some sbb1 => Some (SBBintersection sbb0 sbb1)
945     | _, _ => None
946     end
947   | EXP_SBBunion sbb_expr0 sbb_expr1 =>
948     match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
949     | Some sbb0, Some sbb1 => Some (SBBunion sbb0 sbb1)
950     | _, _ => None
951     end
952   | EXP_makeSBB bb_exprs =>
953     List.fold_left (fun obbs obb =>
954       match obbs, obb with
955       | Some bbs, Some bb => Some (cons bb bbs)
956       | _, _ => None
957       end
958     ) (List.map (fun bb_expr => Abb bb_expr env) bb_exprs) (Some nil)
959   end
960 with Abb (expr : BBexp) (env : Env) : option BB :=
961   match expr with
962   | EXP_BBimg =>
963     match find "IMG" env with
964     | Some (Vbb bb) => Some bb
965     | _ => None
966     end
967   | EXP_BBvar s =>
968     match find s env with
969     | Some (Vbb bb) => Some bb
970     | _ => None
971     end
972   | EXP_makeBB i_expr0 i_expr1 =>
```

```
973     match Ai i_expr0 env, Ai i_expr1 env with
974     | Some i0, Some i1 => Some (i0, i1)
975     | _, _ => None
976     end
977   end
978 with Ai (expr : Iexp) (env : Env) : option Interval :=
979   match expr with
980   | EXP_Ivar s =>
981     match find s env with
982     | Some (Vi i) => Some i
983     | _ => None
984     end
985   | EXP_projx bb_expr =>
986     match Abb bb_expr env with
987     | Some bb => Some (projx bb)
988     | None => None
989     end
990   | EXP_projy bb_expr =>
991     match Abb bb_expr env with
992     | Some bb => Some (projy bb)
993     | None => None
994     end
995   | EXP_Iintersection i_expr0 i_expr1 =>
996     match Ai i_expr0 env, Ai i_expr1 env with
997     | Some i0, Some i1 => Some (Iintersection i0 i1)
998     | _, _ => None
999     end
1000  | EXP_makeI q_expr0 q_expr1 =>
1001    match Aq q_expr0 env, Aq q_expr1 env with
1002    | Some q0, Some q1 => Some (q0, q1)
1003    | _, _ => None
1004    end
1005  end
1006
1007 with Aq (expr : Qexp) (env : Env) : option Q :=
1008   match expr with
1009   | EXP_Q a => Some a
1010   | EXP_Qvar s =>
```

```
1011      match find s env with
1012      | Some (Vq q) => Some q
1013      | _ => None
1014      end
1015    | EXP_width i_expr =>
1016      match Ai i_expr env with
1017      | Some i => Some (width i)
1018      | None => None
1019      end
1020    | EXP_RAT sbb_expr0 sbb_expr1 =>
1021      match Asbb sbb_expr0 env, Asbb sbb_expr1 env with
1022      | Some sbb0, Some sbb1 => Some (RAT sbb0 sbb1)
1023      | _, _ => None
1024      end
1025    | EXP_projl i_expr =>
1026      match Ai i_expr env with
1027      | Some i => Some (lower i)
1028      | _ => None
1029      end
1030    | EXP_proju i_expr =>
1031      match Ai i_expr env with
1032      | Some i => Some (upper i)
1033      | _ => None
1034      end
1035    | EXP_projxl bb_expr =>
1036      match Abb bb_expr env with
1037      | Some bb => Some (projxl bb)
1038      | None => None
1039      end
1040    | EXP_projxu bb_expr =>
1041      match Abb bb_expr env with
1042      | Some bb => Some (projxu bb)
1043      | None => None
1044      end
1045    | EXP_projyl bb_expr =>
1046      match Abb bb_expr env with
1047      | Some bb => Some (projyl bb)
1048      | None => None
```

```
    end
  | EXP_projyu bb_expr =>
    match Abb bb_expr env with
    | Some bb => Some (projyu bb)
    | None => None
    end
  end.

Definition option_and (a b : option Prop) : option Prop :=
    match a, b with
    | Some a, Some b => Some (a /\ b)
    | _, _ => None
    end.

Definition option_or (a b : option Prop) : option Prop :=
    match a, b with
    | Some a, Some b => Some (a \/ b)
    | _, _ => None
    end.

Fixpoint B (expr : Bexp) (env : Env) : option Prop :=
  match expr with
  | EXP_Bvar s =>
    match find s env with
    | Some (Vb b) => Some b
    | _ => None
    end
  | EXP_not b_expr =>
    match B b_expr env with
    | Some b => Some (not b)
    | None => None
    end
  | EXP_and b_expr0 b_expr1 =>
    match B b_expr0 env, B b_expr1 env with
    | Some b0, Some b1 => Some (b0 /\ b1)
    | _, _ => None
    end
  | EXP_or b_expr0 b_expr1 =>
```

```
1087      match B b_expr0 env, B b_expr1 env with
1088      | Some b0, Some b1 => Some (b0 \/ b1)
1089      | _, _ => None
1090      end
1091    | EXP_BBeq bb_expr0 bb_expr1 =>
1092      match Abb bb_expr0 env, Abb bb_expr1 env with
1093      | Some bb0, Some bb1 => Some (BBeq bb0 bb1)
1094      | _, _ => None
1095      end
1096    | EXP_BBoverlap bb_expr0 bb_expr1 =>
1097      match Abb bb_expr0 env, Abb bb_expr1 env with
1098      | Some bb0, Some bb1 => Some (BBoverlap bb0 bb1)
1099      | _, _ => None
1100      end
1101    | EXP_BBsubset bb_expr0 bb_expr1 =>
1102      match Abb bb_expr0 env, Abb bb_expr1 env with
1103      | Some bb0, Some bb1 => Some (BBsubset bb0 bb1)
1104      | _, _ => None
1105      end
1106    | EXP_BBsupset bb_expr0 bb_expr1 =>
1107      match Abb bb_expr0 env, Abb bb_expr1 env with
1108      | Some bb0, Some bb1 => Some (BBsupset bb0 bb1)
1109      | _, _ => None
1110      end
1111    | EXP_BBsubseteq bb_expr0 bb_expr1 =>
1112      match Abb bb_expr0 env, Abb bb_expr1 env with
1113      | Some bb0, Some bb1 => Some (BBsubseteq bb0 bb1)
1114      | _, _ => None
1115      end
1116    | EXP_BBsupseteq bb_expr0 bb_expr1 =>
1117      match Abb bb_expr0 env, Abb bb_expr1 env with
1118      | Some bb0, Some bb1 => Some (BBsupseteq bb0 bb1)
1119      | _, _ => None
1120      end
1121    | EXP_Ilt i_expr0 i_expr1 =>
1122      match Ai i_expr0 env, Ai i_expr1 env with
1123      | Some i0, Some i1 => Some (Ilt i0 i1)
1124      | _, _ => None
```

```
1125      end
1126    | EXP_Igt i_expr0 i_expr1 =>
1127      match Ai i_expr0 env, Ai i_expr1 env with
1128      | Some i0, Some i1 => Some (Igt i0 i1)
1129      | _, _ => None
1130      end
1131    | EXP_Ieq i_expr0 i_expr1 =>
1132      match Ai i_expr0 env, Ai i_expr1 env with
1133      | Some i0, Some i1 => Some (Ieq i0 i1)
1134      | _, _ => None
1135      end
1136    | EXP_Ioverlap i_expr0 i_expr1 =>
1137      match Ai i_expr0 env, Ai i_expr1 env with
1138      | Some i0, Some i1 => Some (Ioverlap i0 i1)
1139      | _, _ => None
1140      end
1141    | EXP_Iin q_expr i_expr =>
1142      match Aq q_expr env, Ai i_expr env with
1143      | Some q, Some i => Some (Iin q i)
1144      | _, _ => None
1145      end
1146    | EXP_Iinrev i_expr q_expr =>
1147      match Aq q_expr env, Ai i_expr env with
1148      | Some q, Some i => Some (Iin q i)
1149      | _, _ => None
1150      end
1151    | EXP_Isubset i_expr0 i_expr1 =>
1152      match Ai i_expr0 env, Ai i_expr1 env with
1153      | Some i0, Some i1 => Some (Isubset i0 i1)
1154      | _, _ => None
1155      end
1156    | EXP_Isupset i_expr0 i_expr1 =>
1157      match Ai i_expr0 env, Ai i_expr1 env with
1158      | Some i0, Some i1 => Some (Isupset i0 i1)
1159      | _, _ => None
1160      end
1161    | EXP_Isubseteq i_expr0 i_expr1 =>
1162      match Ai i_expr0 env, Ai i_expr1 env with
```

```
1163      | Some i0, Some i1 => Some (Isubseteq i0 i1)
1164      | _, _ => None
1165      end
1166    | EXP_Isupseteq i_expr0 i_expr1 =>
1167      match Ai i_expr0 env, Ai i_expr1 env with
1168      | Some i0, Some i1 => Some (Isupseteq i0 i1)
1169      | _, _ => None
1170      end
1171    | EXP_Qlt q_expr0 q_expr1 =>
1172      match Aq q_expr0 env, Aq q_expr1 env with
1173      | Some q0, Some q1 => Some (q0 < q1)%Q
1174      | _, _ => None
1175      end
1176    | EXP_Qgt q_expr0 q_expr1 =>
1177      match Aq q_expr0 env, Aq q_expr1 env with
1178      | Some q0, Some q1 => Some (q0 < q1)%Q
1179      | _, _ => None
1180      end
1181    | EXP_Qeq q_expr0 q_expr1 =>
1182      match Aq q_expr0 env, Aq q_expr1 env with
1183      | Some q0, Some q1 => Some (q0 = q1)
1184      | _, _ => None
1185      end
1186    | EXP_Qle q_expr0 q_expr1 =>
1187      match Aq q_expr0 env, Aq q_expr1 env with
1188      | Some q0, Some q1 => Some (q0 <= q1)%Q
1189      | _, _ => None
1190      end
1191    | EXP_Qge q_expr0 q_expr1 =>
1192      match Aq q_expr0 env, Aq q_expr1 env with
1193      | Some q0, Some q1 => Some (q0 <= q1)%Q
1194      | _, _ => None
1195      end
1196    | EXP_forall bound sbb_expr b_expr =>
1197      match Asbb sbb_expr env with
1198      | Some sbb => List.fold_left option_and (List.map (fun bb => B
              b_expr env) sbb) (Some True)
1199      | _ => None
```

```
1200      end
1201    | EXP_exists bound sbb_expr b_expr =>
1202      match Asbb sbb_expr env with
1203      | Some sbb => List.fold_left option_or (List.map (fun bb => B
              b_expr env) sbb) (Some False)
1204      | _ => None
1205      end
1206    end.
1207
1208  Definition Ccond (cond : Cond) (env : Env) : option Prop :=
1209    match cond with
1210    | CND_None => Some True
1211    | CND b => B b env
1212    end.
1213
1214  Definition Cdef (def : Def) (env : Env) : option Env :=
1215    match def with
1216    | DEF_SBB s sbb_expr =>
1217      match Asbb sbb_expr env with
1218      | Some sbb => Some (add s (Vsbb sbb) env)
1219      | _ => None
1220      end
1221    | DEF_BB s bb_expr =>
1222      match Abb bb_expr env with
1223      | Some bb => Some (add s (Vbb bb) env)
1224      | _ => None
1225      end
1226    | DEF_I s i_expr =>
1227      match Ai i_expr env with
1228      | Some i => Some (add s (Vi i) env)
1229      | _ => None
1230      end
1231    | DEF_Q s q_expr =>
1232      match Aq q_expr env with
1233      | Some q => Some (add s (Vq q) env)
1234      | _ => None
1235      end
1236    | DEF_B s b_expr =>
```

```
1237     match B b_expr env with
1238     | Some b => Some (add s (Vb b) env)
1239     | _ => None
1240     end
1241   end.
1242
1243 Fixpoint Cdefs (defs : list Def) (env : Env) : option Env :=
1244   match defs with
1245   | nil => Some env
1246   | cons def defs' =>
1247     match Cdef def env with
1248     | Some env' => Cdefs defs' env'
1249     | _ => None
1250     end
1251   end.
1252
1253 Definition Ccase (case : Case) (env : Env) : option (string * Prop)
       :=
1254   match case with
1255   | (l, defs, b_expr) =>
1256     match Cdefs defs env with
1257     | Some env' =>
1258       match B b_expr env' with
1259       | Some b => Some (l, b)
1260       | _ => None
1261       end
1262     | _ => None
1263     end
1264   end.
1265
1266 Lemma toTrue : forall P : Prop, P -> P <-> True.
1267 Proof.
1268   intros. split.
1269   intros. trivial.
1270   intros. apply H.
1271 Qed.
1272
1273 Fixpoint Ccases (cases : list Case) (env : Env) (accum : list (string
```

```
        * Prop)) : option (list (string * Prop)) :=
  match cases with
  | nil => Some accum
  | cons case cases' =>
    match Ccase case env with
    | Some lb => Ccases cases' env (cons lb accum)
    | _ => None
    end
  end.

Definition Cspec (spec : Spec) (env : Env) : option (list (string *
    Prop)) :=
  match spec with
  | (cond, cases) =>
    match Ccond cond env, Ccases cases env nil with
    | Some b, Some lbs => Some (List.map
          (fun lb => match lb with (l, b') => (l, b /\ b') end)
          lbs)
    | _, _ => None
    end
  end.

Lemma or_falser : forall A : Prop, A \/ False <-> A.
Proof.
  intros. split.
  - intros. destruct H. assumption. contradiction.
  - intros. apply (or_introl H).
Qed.

Lemma or_falsel : forall A : Prop, False \/ A <-> A.
Proof.
  intros.
  rewrite (or_comm False A).
  revert A.
  apply or_falser.
Qed.

Lemma and_l : forall A B : Prop, B -> (A /\ B <-> A).
```

```
Proof.
  intros. split.
  - intros. destruct H0. assumption.
  - intros. apply (conj H0 H).
Qed.

Proposition comprehensiveness_of_example_lead_vehicle_stopped :
  forall (exists_front : Prop) (front_bb dec : BB),
    let evaluated :=
      Cspec
        example_lead_vehicle_stopped
        (add "減速区間" (Vbb dec) (add "前方車両" (Vbb front_bb) (add
            "前方車両がある" (Vb exists_front) (empty Value))))
    in
    BBnempty front_bb /\ BBnempty dec /\ exists_front ->
       match option_map (fun ev => List.fold_left or (List.map snd ev)
            False) evaluated with
       | Some b => b
       | _ => False
       end.
Proof.
  simpl. unfold BBnempty. unfold Inempty. unfold Igt. unfold Ilt.
  unfold Ioverlap. unfold Iempty. unfold Iintersection. unfold not.
  intros. destruct dec as (dec_x, dec_y). destruct front_bb as (f_x,
      f_y).
  destruct dec_x as (dec_xl, dec_xu). destruct dec_y as (dec_yl,
      dec_yu).
  destruct f_x as (f_xl, f_xu). destruct f_y as (f_yl, f_yu).
  simpl. simpl in H. destruct H. destruct H. destruct H0. destruct H0.
  rewrite (Q.min_lt_iff f_yu dec_yu (Qmax f_yl dec_yl)).
  rewrite (Q.max_lt_iff f_yl dec_yl dec_yu).
  rewrite (Q.max_lt_iff f_yl dec_yl f_yu).
  destruct (Qlt_le_dec dec_yu f_yl).
  - left. left. right. apply (conj H2 q).
  - destruct (Qlt_le_dec f_yu dec_yl).
  -- left. right. apply (conj H2 q0).
  -- right. split. assumption. intros. destruct H4. destruct H4.
  --- apply (Qle_lteq f_yl f_yu) in H1. destruct H1.
```

```
  ---- apply (Qlt_asym f_yl f_yu (conj H1 H4)).
  ---- rewrite H1 in H4. apply (Qlt_irrefl f_yu H4).
  --- apply (Qle_lteq dec_yl f_yu) in q0. destruct q0.
  ---- apply (Qlt_asym dec_yl f_yu (conj H5 H4)).
  ---- rewrite H5 in H4. apply (Qlt_irrefl f_yu H4).
  --- destruct H4.
  ---- apply (Qle_lteq f_yl dec_yu) in q. destruct q.
  ----- apply (Qlt_asym f_yl dec_yu (conj H5 H4)).
  ----- rewrite H5 in H4. apply (Qlt_irrefl dec_yu H4).
  ---- apply (Qle_lteq dec_yl dec_yu) in H3. destruct H3.
  ----- apply (Qlt_asym dec_yl dec_yu (conj H3 H4)).
  ----- rewrite H3 in H4. apply (Qlt_irrefl dec_yu H4).
Qed.
```

付録 B

形式化した BBSL による仕様の記述

形式化した BBSL を用いて記述した仕様のソースコード全文を B.1 に示す。

ソースコード B.1　形式化した BBSL による仕様の記述

```
1  Definition example_confluence : Spec :=
2    ( CND_None
3    , [ ( "シーン１"
4        , [ DEF_I "合流領域" (EXP_Ivar "合流領域")
5          ; DEF_SBB "他車集合" (EXP_SBBvar "他車集合")
6          ]
7        , EXP_forall "x" (EXP_SBBvar "他車集合")
8            (EXP_not (EXP_Ieq (EXP_projy (EXP_BBvar "x")) (EXP_Ivar "合
                 流領域")))
9        )
10     ; ( "シーン２"
11       , [ DEF_I "合流領域" (EXP_Ivar "合流領域")
12         ; DEF_SBB "他車集合" (EXP_SBBvar "他車集合")
13         ]
14       , EXP_exists "x" (EXP_SBBvar "他車集合")
15           (EXP_and
16             (EXP_Qeq (EXP_projyl (EXP_BBvar "y")) (EXP_proju (EXP_Ivar
                   "合流領域")))
17             (EXP_forall "y" (EXP_SBBvar "他車集合")
18               (EXP_or (EXP_not (EXP_Ieq (EXP_projy (EXP_BBvar "y")) (
                     EXP_Ivar "合流領域")))
```

```
19                      (EXP_BBeq (EXP_BBvar "x") (EXP_BBvar "y")))))
20       )
21     ; ( "シーン３"
22       , [ DEF_I "合流領域" (EXP_Ivar "合流領域")
23         ; DEF_SBB "他車集合" (EXP_SBBvar "他車集合")
24         ]
25       , EXP_exists "x" (EXP_SBBvar "他車集合")
26          (EXP_and
27            (EXP_Qeq (EXP_projyl (EXP_BBvar "x")) (EXP_proju (EXP_Ivar
                  "合流領域")))
28            (EXP_and
29              (EXP_exists "y" (EXP_SBBvar "他車集合")
30                (EXP_and
31                  (EXP_Qeq (EXP_projyu (EXP_BBvar "y")) (EXP_projl (
                        EXP_Ivar "合流領域")))
32                  (EXP_not (EXP_BBeq (EXP_BBvar "x") (EXP_BBvar "y
                        "))))))
33              (EXP_forall "z" (EXP_SBBvar "他車領域")
34                    (EXP_not (EXP_Isupseteq (EXP_Ivar "合流領域") (
                          EXP_projy (EXP_BBvar "z")))))))
35       )
36     ; ( "シーン４"
37       , [ DEF_I "合流領域" (EXP_Ivar "合流領域")
38         ; DEF_SBB "他車集合" (EXP_SBBvar "他車集合")
39         ]
40       , EXP_exists "z" (EXP_SBBvar "他車集合")
41           (EXP_Isupseteq (EXP_Ivar "合流領域") (EXP_projy (EXP_BBvar "z
                "")))
42       )
43     ]
44   ).
45
46 Definition example_lead_vehicle_stopped : Spec :=
47   ( CND (EXP_Bvar "前方車両がある")
48   , [ ( "減速"
49       , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
50         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
51         ]
```

```
52        , EXP_Ioverlap
53            (EXP_projy (EXP_BBvar "前方車両"))
54            (EXP_projy (EXP_BBvar "減速区間"))
55        )
56      ; ( "停止"
57        , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
58          ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
59          ]
60        , EXP_Ilt
61            (EXP_projy (EXP_BBvar "前方車両"))
62            (EXP_projy (EXP_BBvar "減速区間"))
63        )
64      ; ( "レスポンスなし"
65        , [ DEF_BB "前方車両" (EXP_BBvar "前方車両")
66          ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
67          ]
68        , EXP_Igt
69            (EXP_projy (EXP_BBvar "前方車両"))
70            (EXP_projy (EXP_BBvar "減速区間"))
71        )
72      ]
73    ).
74
75  Definition example_debris_static_in_lane : Spec :=
76    ( CND (EXP_Bvar "静的障害物がある")
77    , [ ( "減速"
78        , [ DEF_SBB "障害物集合" (EXP_SBBvar "障害物集合")
79          ; DEF_SBB "進行区間集合" (EXP_SBBvar "進行区間集合")
80          ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
81          ]
82        , EXP_exists "x" (EXP_SBBvar "障害物集合")
83            (EXP_exists "y" (EXP_SBBvar "進行区間集合")
84              (EXP_and
85                (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (EXP_projx (
                      EXP_BBvar "y")))
86                (EXP_and
87                  (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                        EXP_BBvar "y")))
```

```
88                (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                      EXP_BBvar "減速区間"))))))
89        )
90      ; ( "停止"
91        , [ DEF_SBB "障害物集合" (EXP_SBBvar "障害物集合")
92          ; DEF_SBB "進行区間集合" (EXP_SBBvar "進行区間集合")
93          ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
94          ]
95        , EXP_exists "x" (EXP_SBBvar "障害物集合")
96            (EXP_exists "y" (EXP_SBBvar "進行区間集合")
97              (EXP_and
98                (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (EXP_projx (
                      EXP_BBvar "y")))
99                (EXP_and
100                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                        EXP_BBvar "y")))
101                 (EXP_Ilt (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                        EXP_BBvar "減速区間"))))))
102       )
103     ; ( "レスポンスなし"
104       , [ DEF_SBB "障害物集合" (EXP_SBBvar "障害物集合")
105         ; DEF_SBB "進行区間集合" (EXP_SBBvar "進行区間集合")
106         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
107         ]
108       , EXP_exists "x" (EXP_SBBvar "障害物集合")
109           (EXP_exists "y" (EXP_SBBvar "進行区間集合")
110             (EXP_and
111               (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (EXP_projx (
                      EXP_BBvar "y")))
112               (EXP_and
113                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                        EXP_BBvar "y")))
114                 (EXP_Igt (EXP_projy (EXP_BBvar "x")) (EXP_projy (
                        EXP_BBvar "減速区間"))))))
115       )
116   ]
117 ).
118
```

```
119 Definition example_vehicle_cutting_in : Spec :=
120   ( CND (EXP_and (EXP_Bvar "前方車両がある") (EXP_Bvar "他車両がある"))
121   , [ ( "停止"
122       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
123         ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
124         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
125         ]
126       , EXP_exists "x" (EXP_SBBvar "自車線区間集合")
127           (EXP_and
128             (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                  EXP_projx (EXP_BBvar "x")))
129             (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                  EXP_projy (EXP_BBvar "減速区間"))))
130       )
131     ; ( "減速"
132       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
133         ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
134         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
135         ]
136       , EXP_and
137           (EXP_forall "x" (EXP_SBBvar "自車線区間集合")
138             (EXP_and
139               (EXP_not (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車
                      両")) (EXP_projx (EXP_BBvar "x"))))
140               (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                      EXP_projy (EXP_BBvar "減速区間")))))
141           (EXP_exists "x" (EXP_SBBvar "自車線区間")
142             (EXP_and
143               (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                      EXP_projx (EXP_BBvar "x")))
144               (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                      EXP_projy (EXP_BBvar "減速区間")))))
145       )
146     ; ( "前方車両に従う"
147       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
148         ; DEF_BB "前方車両" (EXP_BBvar "前方車両")
149         ]
150       , EXP_Qlt (EXP_projyl (EXP_BBvar "前方車両")) (EXP_projyl (
```

```
            EXP_BBvar "割り込み車両"))
151       )
152     ; ( "割り込み車両に前方を譲る"
153       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
154         ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
155         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
156         ]
157       , EXP_forall "x" (EXP_SBBvar "自車線区間集合")
158           (EXP_not
159             (EXP_and
160               (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                      EXP_projx (EXP_BBvar "x")))
161               (EXP_or
162                 (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                        EXP_projy (EXP_BBvar "減速区間")))
163                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                        EXP_projy (EXP_BBvar "減速区間"))))))
164       )
165     ]
166   ).
167
168 Definition example_vehicle_cutting_in_hwd : Spec :=
169   ( CND (EXP_and (EXP_Bvar "前方車両がある") (EXP_Bvar "他車両がある"))
170   , [ ( "右の車線に車線変更"
171       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
172         ; DEF_SBB "他車両集合" (EXP_SBBvar "他車両集合")
173         ; DEF_SBB "自車線左区間集合" (EXP_SBBvar "自車線左区間集合")
174         ; DEF_SBB "自車線右区間集合" (EXP_SBBvar "自車線右区間集合")
175         ; DEF_SBB "右車線変更区間集合" (EXP_SBBvar "右車線変更区間集
              合")
176         ]
177       , EXP_and
178           (EXP_Bvar "右車線存在確認")
179           (EXP_forall "x" (EXP_SBBvar "他車両集合")
180             (EXP_exists "y" (EXP_SBBvar "右車線変更区間集合")
181               (EXP_and
182                 (EXP_not (EXP_and
183                   (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (EXP_projx
```

```
                    (EXP_BBvar "y")))
184                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy
                        (EXP_BBvar "y")))))
185               (EXP_Qgt
186                 (EXP_RAT
187                   (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                          合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                          ]))
188                   (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                          合")
189                   (EXP_makeSBB [ EXP_BBvar "割り込み車両" ])))
190                 (EXP_Q 1.0)))))
191     )
192   ; ( "左の車線に車線変更"
193     , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
194       ; DEF_SBB "他車両集合" (EXP_SBBvar "他車両集合")
195       ; DEF_SBB "自車線左区間集合" (EXP_SBBvar "自車線左区間集合")
196       ; DEF_SBB "自車線右区間集合" (EXP_SBBvar "自車線右区間集合")
197       ; DEF_SBB "左車線変更区間集合" (EXP_SBBvar "左車線変更区間集
            合")
198       ]
199     , EXP_and
200         (EXP_Bvar "左車線存在確認")
201         (EXP_forall "x" (EXP_SBBvar "他車両集合")
202           (EXP_exists "y" (EXP_SBBvar "左車線変更区間集合")
203             (EXP_and
204               (EXP_not (EXP_and
205                 (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (EXP_projx
                        (EXP_BBvar "y")))
206                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (EXP_projy
                        (EXP_BBvar "y")))))
207               (EXP_Qle
208                 (EXP_RAT
209                   (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                          合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                          ]))
210                   (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                          合")
```

```
211                     (EXP_makeSBB [ EXP_BBvar "割り込み車両" ])))
212                   (EXP_Q 1.0)))))
213       )
214     ; ( "停止"
215       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
216         ; DEF_SBB "他車両集合" (EXP_SBBvar "他車両集合")
217         ; DEF_SBB "自車線左区間集合" (EXP_SBBvar "自車線左区間集合")
218         ; DEF_SBB "自車線右区間集合" (EXP_SBBvar "自車線右区間集合")
219         ; DEF_SBB "右車線変更区間集合" (EXP_SBBvar "右車線変更区間集
              合")
220         ; DEF_SBB "左車線変更区間集合" (EXP_SBBvar "左車線変更区間集
              合")
221         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
222         ]
223       , EXP_and
224           (EXP_not
225             (EXP_or
226               (EXP_and
227                 (EXP_Bvar "右車線存在確認")
228                 (EXP_and
229                   (EXP_forall "x" (EXP_SBBvar "他車両集合")
230                     (EXP_exists "y" (EXP_SBBvar "右車線変更区間集合")
231                       (EXP_not (EXP_and
232                         (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (
                              EXP_projx (EXP_BBvar "y")))
233                         (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (
                              EXP_projy (EXP_BBvar "y")))))))
234                   (EXP_Qgt
235                     (EXP_RAT
236                       (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                            合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                            ]))
237                       (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                            合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                            ])))
238                     (EXP_Q 1.0))))
239               (EXP_and
240                 (EXP_Bvar "左車線存在確認")
```

```
241                 (EXP_and
242                   (EXP_forall "x" (EXP_SBBvar "他車両集合")
243                     (EXP_exists "y" (EXP_SBBvar "右車線変更区間集合")
244                       (EXP_not (EXP_and
245                         (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (
                                EXP_projx (EXP_BBvar "y")))
246                         (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (
                                EXP_projy (EXP_BBvar "y")))))))
247                   (EXP_Qle
248                     (EXP_RAT
249                       (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                            合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                            ]))
250                       (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                            合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                            ])))
251                     (EXP_Q 1.0))))))
252           (EXP_exists "x" (EXP_SBBunion (EXP_SBBvar "自車線右区間集合")
                (EXP_SBBvar "自車線左区間集合"))
253             (EXP_and
254               (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                      EXP_projx (EXP_BBvar "x")))
255               (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                      EXP_projy (EXP_BBvar "減速区間"))))))
256       )
257     ; ( "減速"
258       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
259         ; DEF_SBB "他車両集合" (EXP_SBBvar "他車両集合")
260         ; DEF_SBB "自車線左区間集合" (EXP_SBBvar "自車線左区間集合")
261         ; DEF_SBB "自車線右区間集合" (EXP_SBBvar "自車線右区間集合")
262         ; DEF_SBB "右車線変更区間集合" (EXP_SBBvar "右車線変更区間集
              合")
263         ; DEF_SBB "左車線変更区間集合" (EXP_SBBvar "左車線変更区間集
              合")
264         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
265         ]
266       , EXP_and
267           (EXP_not
```

```
268             (EXP_or
269               (EXP_and
270                 (EXP_Bvar "右車線存在確認")
271                 (EXP_and
272                   (EXP_forall "x" (EXP_SBBvar "他車両集合")
273                     (EXP_exists "y" (EXP_SBBvar "右車線変更区間集合")
274                       (EXP_not (EXP_and
275                         (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (
                                EXP_projx (EXP_BBvar "y")))
276                         (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (
                                EXP_projy (EXP_BBvar "y")))))))
277                   (EXP_Qgt
278                     (EXP_RAT
279                       (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                              合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                              ]))
280                       (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                              合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                              ])))
281                     (EXP_Q 1.0))))
282               (EXP_and
283                 (EXP_Bvar "左車線存在確認")
284                 (EXP_and
285                   (EXP_forall "x" (EXP_SBBvar "他車両集合")
286                     (EXP_exists "y" (EXP_SBBvar "右車線変更区間集合")
287                       (EXP_not (EXP_and
288                         (EXP_Ioverlap (EXP_projx (EXP_BBvar "x")) (
                                EXP_projx (EXP_BBvar "y")))
289                         (EXP_Ioverlap (EXP_projy (EXP_BBvar "x")) (
                                EXP_projy (EXP_BBvar "y")))))))
290                   (EXP_Qle
291                     (EXP_RAT
292                       (EXP_SBBintersection (EXP_SBBvar "自車線左区間集
                              合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                              ]))
293                       (EXP_SBBintersection (EXP_SBBvar "自車線右区間集
                              合") (EXP_makeSBB [ EXP_BBvar "割り込み車両"
                              ])))
```

```
294                     (EXP_Q 1.0))))))
295           (EXP_and
296             (EXP_forall "x" (EXP_SBBunion (EXP_SBBvar "自車線右区間集
                    合") (EXP_SBBvar "自車線左区間集合"))
297               (EXP_not (EXP_and
298                 (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                        EXP_projx (EXP_BBvar "x")))
299                 (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                        EXP_projy (EXP_BBvar "減速区間"))))))
300             (EXP_exists "x" (EXP_SBBunion (EXP_SBBvar "自車線右区間集
                    合") (EXP_SBBvar "自車線左区間集合"))
301               (EXP_and
302                 (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                        EXP_projx (EXP_BBvar "x")))
303                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
                        EXP_projy (EXP_BBvar "減速区間"))))))
304       )
305     ; ( "前方車両に従う"
306       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
307         ; DEF_BB "前方車両" (EXP_BBvar "前方車両")
308         ]
309       , EXP_Qlt (EXP_projyl (EXP_BBvar "前方車両")) (EXP_projyl (
            EXP_BBvar "割り込み車両"))
310       )
311     ; ( "割り込み車両に前方を譲る"
312       , [ DEF_BB "割り込み車両" (EXP_BBvar "割り込み車両")
313         ; DEF_SBB "自車線区間集合" (EXP_SBBvar "自車線区間集合")
314         ; DEF_BB "減速区間" (EXP_BBvar "減速区間")
315         ]
316       , EXP_forall "x" (EXP_SBBvar "自車線区間集合")
317           (EXP_not
318             (EXP_and
319               (EXP_Ioverlap (EXP_projx (EXP_BBvar "割り込み車両")) (
                      EXP_projx (EXP_BBvar "x")))
320               (EXP_or
321                 (EXP_Ilt (EXP_projy (EXP_BBvar "割り込み車両")) (
                        EXP_projy (EXP_BBvar "減速区間")))
322                 (EXP_Ioverlap (EXP_projy (EXP_BBvar "割り込み車両")) (
```

```
                    EXP_projy (EXP_BBvar "減速区間"))))))
      )
    ]
  ).

Definition example_ratio_relation1_2 : Spec :=
  ( CND_None
  , [ ( "割合の関係１(IoU=0.5の場合)"
      , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
        ; DEF_BB "B" (EXP_BBvar "B を返す関数")
        ]
      , EXP_Qeq
          (EXP_RAT
            (EXP_SBBintersection
              (EXP_makeSBB [ EXP_BBvar "A" ])
              (EXP_makeSBB [ EXP_BBvar "B" ]))
            (EXP_SBBunion
              (EXP_makeSBB [ EXP_BBvar "A" ])
              (EXP_makeSBB [ EXP_BBvar "B" ])))
          (EXP_Q 0.5)
      )
    ; ( "割合の関係１(IoU=0.15の場合)"
      , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
        ; DEF_BB "B" (EXP_BBvar "B を返す関数")
        ]
      , EXP_Qeq
          (EXP_RAT
            (EXP_SBBintersection
              (EXP_makeSBB [ EXP_BBvar "A" ])
              (EXP_makeSBB [ EXP_BBvar "B" ]))
            (EXP_SBBunion
              (EXP_makeSBB [ EXP_BBvar "A" ])
              (EXP_makeSBB [ EXP_BBvar "B" ])))
          (EXP_Q 0.15)
      )
    ]
  ).
```

```
Definition example_rational_relation3_4 : Spec :=
  ( CND_None
  , [ ( "割合の関係３(IoU=0.12かつ
      Aの右上の頂点とBの左下の頂点が重なっている場合)"
      , [ DEF_BB "A" (EXP_BBvar "Aを返す関数")
        ; DEF_BB "B" (EXP_BBvar "Bを返す関数")
        ]
      , EXP_and
          (EXP_Qeq
            (EXP_RAT
              (EXP_SBBintersection
                (EXP_makeSBB [ EXP_BBvar "A" ])
                (EXP_makeSBB [ EXP_BBvar "B" ]))
              (EXP_SBBunion
                (EXP_makeSBB [ EXP_BBvar "A" ])
                (EXP_makeSBB [ EXP_BBvar "B" ])))
            (EXP_Q 0.5))
          (EXP_and
            (EXP_Ioverlap (EXP_projx (EXP_BBvar "A")) (EXP_projx (
                EXP_BBvar "B")))
            (EXP_Ioverlap (EXP_projy (EXP_BBvar "A")) (EXP_projy (
                EXP_BBvar "B"))))
      )
    ; ( "割合の関係３(IoU=0.12かつ
      Aの右下の頂点とBの左上の頂点が重なっている場合)"
      , [ DEF_BB "A" (EXP_BBvar "Aを返す関数")
        ; DEF_BB "B" (EXP_BBvar "Bを返す関数")
        ]
      , EXP_and
          (EXP_Qeq
            (EXP_RAT
              (EXP_SBBintersection
                (EXP_makeSBB [ EXP_BBvar "A" ])
                (EXP_makeSBB [ EXP_BBvar "B" ]))
              (EXP_SBBunion
                (EXP_makeSBB [ EXP_BBvar "A" ])
                (EXP_makeSBB [ EXP_BBvar "B" ])))
            (EXP_Q 0.5))
```

```
394           (EXP_and
395             (EXP_Ioverlap (EXP_projx (EXP_BBvar "A")) (EXP_projx (
                  EXP_BBvar "B")))
396             (EXP_Ioverlap (EXP_projy (EXP_BBvar "A")) (EXP_projy (
                  EXP_BBvar "B"))))
397       )
398     ]
399   ).
400
401 Definition example_positional_relation1_2 : Spec :=
402   ( CND_None
403   , [ ( "位置関係１（A よりも B の方が上にある）"
404       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
405         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
406         ]
407       , EXP_Ilt (EXP_projy (EXP_BBvar "A")) (EXP_projy (EXP_BBvar "B
              "))
408       )
409     ; ( "位置関係１（A よりも B の方が右にある）"
410       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
411         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
412         ]
413       , EXP_Ilt (EXP_projx (EXP_BBvar "A")) (EXP_projx (EXP_BBvar "B
              "))
414       )
415     ]
416   ).
417
418 Definition example_positional_relation3_4 : Spec :=
419   ( CND (EXP_Bvar "画像全体を取得可能")
420   , [ ( "位置関係３（B が A の左上と上の領域どちらともに位置している）"
421       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
422         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
423         ]
424       , EXP_and
425           (EXP_Ilt (EXP_projy (EXP_BBvar "A")) (EXP_projy (EXP_BBvar "
                B")))
426           (EXP_Ioverlap (EXP_projx (EXP_BBvar "A")) (EXP_projx (
```

```
                EXP_BBvar "B")))
427       )
428     ; ( "位置関係４（B 全体の 0.3以上がA の左下に位置している）"
429       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
430         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
431         ; DEF_BB "A 左下" (EXP_makeBB
432           (EXP_makeI (EXP_projxl EXP_BBimg) (EXP_projxl (EXP_BBvar "A
                  ")))
433           (EXP_makeI (EXP_projyl EXP_BBimg) (EXP_projyl (EXP_BBvar "A
                  "))))
434         ]
435       , EXP_Qge
436           (EXP_RAT
437             (EXP_SBBintersection
438               (EXP_makeSBB [ EXP_BBvar "A 左下" ])
439               (EXP_makeSBB [ EXP_BBvar "B" ]))
440             (EXP_makeSBB [ EXP_BBvar "B" ]))
441           (EXP_Q 0.3)
442       )
443     ]
444   ).
445
446 Definition example_inclusion_relation1_2 : Spec :=
447   ( CND_None
448   , [ ( "包含関係１（B が A に包含されている）"
449       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
450         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
451       ]
452       , EXP_BBsupseteq (EXP_BBvar "A") (EXP_BBvar "B")
453       )
454     ; ( "包含関係２（A が B に包含されている）"
455       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
456         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
457       ]
458       , EXP_BBsubseteq (EXP_BBvar "A") (EXP_BBvar "B")
459       )
460     ]
461   ).
```

```
462
463 Definition example_comparison_relation1_2 : Spec :=
464   ( CND_None
465   , [ ( "大小関係１（A より B が小さい）"
466       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
467         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
468       ]
469       , EXP_Qgt
470           (EXP_RAT (EXP_makeSBB [ EXP_BBvar "A" ]) (EXP_makeSBB [
                EXP_BBvar "B" ]))
471           (EXP_Q 1.0)
472       )
473     ; ( "大小関係２（A より B が大さい）"
474       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
475         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
476       ]
477       , EXP_Qlt
478           (EXP_RAT (EXP_makeSBB [ EXP_BBvar "A" ]) (EXP_makeSBB [
                EXP_BBvar "B" ]))
479           (EXP_Q 1.0)
480       )
481     ]
482   ).
483
484 Definition example_contains : Spec :=
485   ( CND_None
486   , [ ( "A contains B"
487       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
488         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
489         ]
490       , EXP_and
491          (EXP_BBsupseteq (EXP_BBvar "A") (EXP_BBvar "B"))
492          (EXP_not (EXP_or
493            (EXP_Qeq (EXP_projxl (EXP_BBvar "A")) (EXP_projxl (
                 EXP_BBvar "B")))
494            (EXP_or
495              (EXP_Qeq (EXP_projxu (EXP_BBvar "A")) (EXP_projxu (
                   EXP_BBvar "B")))
```

```
496             (EXP_or
497               (EXP_Qeq (EXP_projyl (EXP_BBvar "A")) (EXP_projyl (
                      EXP_BBvar "B")))
498               (EXP_Qeq (EXP_projyu (EXP_BBvar "A")) (EXP_projyu (
                      EXP_BBvar "B")))))))
499       )
500     ; ( "A covers B"
501       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
502         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
503         ]
504       , EXP_and
505          (EXP_BBsupseteq (EXP_BBvar "A") (EXP_BBvar "B"))
506          (EXP_or
507            (EXP_Qeq (EXP_projxl (EXP_BBvar "A")) (EXP_projxl (
                   EXP_BBvar "B")))
508            (EXP_or
509              (EXP_Qeq (EXP_projxu (EXP_BBvar "A")) (EXP_projxu (
                     EXP_BBvar "B")))
510              (EXP_or
511                (EXP_Qeq (EXP_projyu (EXP_BBvar "A")) (EXP_projyl (
                       EXP_BBvar "B")))
512                (EXP_Qeq (EXP_projyl (EXP_BBvar "A")) (EXP_projyu (
                       EXP_BBvar "B"))))))
513       )
514     ; ( "A touch B"
515       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
516         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
517         ]
518       , EXP_and
519           (EXP_Qeq
520             (EXP_RAT
521               (EXP_SBBintersection (EXP_makeSBB [ EXP_BBvar "A" ]) (
                      EXP_makeSBB [ EXP_BBvar "B" ]))
522               (EXP_SBBunion (EXP_makeSBB [ EXP_BBvar "A" ]) (
                      EXP_makeSBB [ EXP_BBvar "B" ])))
523             (EXP_Q 0))
524           (EXP_or
525             (EXP_Qeq (EXP_projxl (EXP_BBvar "A")) (EXP_projxl (
```

```
                EXP_BBvar "B")))
526           (EXP_or
527             (EXP_Qeq (EXP_projxu (EXP_BBvar "A")) (EXP_projxu (
                    EXP_BBvar "B")))
528             (EXP_or
529             (EXP_Qeq (EXP_projyl (EXP_BBvar "A")) (EXP_projyl (
                    EXP_BBvar "B")))
530             (EXP_Qeq (EXP_projyu (EXP_BBvar "A")) (EXP_projyu (
                    EXP_BBvar "B"))))))
531       )
532     ; ( "A overlapbdyintersect B"
533       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
534         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
535         ]
536       , EXP_not (EXP_and
537           (EXP_Qeq
538             (EXP_RAT
539               (EXP_SBBintersection (EXP_makeSBB [ EXP_BBvar "A" ]) (
                      EXP_makeSBB [ EXP_BBvar "B" ]))
540               (EXP_makeSBB [ EXP_BBvar "B" ]))
541             (EXP_Q 1))
542           (EXP_Qeq
543             (EXP_RAT
544               (EXP_SBBintersection (EXP_makeSBB [ EXP_BBvar "A" ]) (
                      EXP_makeSBB [ EXP_BBvar "B" ]))
545               (EXP_makeSBB [ EXP_BBvar "B" ]))
546             (EXP_Q 0)))
547       )
548     ; ( "A equal B"
549       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
550         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
551         ]
552       , EXP_BBeq (EXP_BBvar "A") (EXP_BBvar "B")
553       )
554     ; ( "A disjoint B"
555       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
556         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
557         ]
```

```
558       , EXP_not (EXP_BBoverlap (EXP_BBvar "A") (EXP_BBvar "B"))
559       )
560     ; ( "A overlapbdydisjoint B"
561       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
562         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
563         ]
564       , EXP_or
565           (EXP_and
566             (EXP_Qeq (EXP_width (EXP_projy (EXP_BBvar "B"))) (EXP_Q
                  0))
567             (EXP_or
568               (EXP_and
569                 (EXP_Iin (EXP_projxl (EXP_BBvar "A")) (EXP_projx (
                      EXP_BBvar "B")))
570                 (EXP_Iin (EXP_projxu (EXP_BBvar "A")) (EXP_projx (
                      EXP_BBvar "B"))))
571               (EXP_and
572                 (EXP_Iin (EXP_projxu (EXP_BBvar "A")) (EXP_projx (
                      EXP_BBvar "B")))
573                 (EXP_not (EXP_Iin (EXP_projxl (EXP_BBvar "A")) (
                      EXP_projx (EXP_BBvar "B")))))))
574           (EXP_and
575             (EXP_Qeq (EXP_width (EXP_projx (EXP_BBvar "B"))) (EXP_Q
                  0))
576             (EXP_or
577               (EXP_and
578                 (EXP_Iin (EXP_projyl (EXP_BBvar "A")) (EXP_projy (
                      EXP_BBvar "B")))
579                 (EXP_Iin (EXP_projyu (EXP_BBvar "A")) (EXP_projy (
                      EXP_BBvar "B"))))
580               (EXP_and
581                 (EXP_Iin (EXP_projyu (EXP_BBvar "A")) (EXP_projy (
                      EXP_BBvar "B")))
582                 (EXP_not (EXP_Iin (EXP_projyl (EXP_BBvar "A")) (
                      EXP_projy (EXP_BBvar "B")))))))
583       )
584     ; ( "A on B"
585       , [ DEF_BB "A" (EXP_BBvar "A を返す関数")
```

```
586         ; DEF_BB "B" (EXP_BBvar "B を返す関数")
587         ]
588       , EXP_or
589           (EXP_and
590             (EXP_Qeq (EXP_width (EXP_projy (EXP_BBvar "B"))) (EXP_Q
                  0))
591             (EXP_or
592               (EXP_Iinrev (EXP_projy (EXP_BBvar "B")) (EXP_projyu (
                    EXP_BBvar "A")))
593               (EXP_Iinrev (EXP_projy (EXP_BBvar "B")) (EXP_projyl (
                    EXP_BBvar "A")))))
594           (EXP_and
595             (EXP_Qeq (EXP_width (EXP_projx (EXP_BBvar "B"))) (EXP_Q
                  0))
596             (EXP_or
597               (EXP_Iinrev (EXP_projx (EXP_BBvar "B")) (EXP_projxu (
                    EXP_BBvar "A")))
598               (EXP_Iinrev (EXP_projx (EXP_BBvar "B")) (EXP_projxl (
                    EXP_BBvar "A")))))
599       )
600     ]
601   ).
```