Formulating the Collecting Semantics and Abstract Semantics as Path Problems CS701

Thomas Reps
[Based on notes taken by Morgan Zhang on October 13, 2015]

Abstract

This lecture discusses standard semantics, collecting semantics, and abstract semantics for a flowchart language and the control flow graphs (CFGs) that represent programs in the language.

1 Two Formulations of Path Problems

We start our discussion by reviewing the two formulations of the Single Source Shortest Distance (*SSSD*) problem.[1] Similar path problems, each with a pair of formulations, will be used shortly to define the standard semantics, collecting semantics, and abstract semantics of a flowchart language—and in particular, the semantics of the control flow graphs (CFGs) that represent programs in the language.

1.1 Declarative Formulation

Let $p = e_1, e_2, \ldots, e_k$ be a path from s to n, and let $l(e_i)$ be the length of the edge e_i. Then the length of p, denoted by $\mathrm{len}(p)$, is defined as

$$\mathrm{len}(p) \overset{\mathrm{def}}{=} l(e_1) + l(e_2) + \ldots + l(e_k),$$

and the set of *SSSD* values is defined as follows:

$$\mathit{SSSD}[s,n] \overset{\mathrm{def}}{=} \underset{p \in \mathrm{Paths}(s,n)}{\mathrm{Min}} \mathrm{len}(p). \tag{1}$$

The *SSSD* problem is an example of a *path problem.* In general, a path problem is formulated in terms of two binary operators, *extend* (denoted by $\otimes$), which is used to define the cost of a path (also known as the length or weight of the path), and *combine* (denoted by $\oplus$), which is used to combine information at a node from two or more incoming paths. In the case of the *SSSD* problem, $\otimes$ is $+$, and $\oplus$ is min.[2] In general, a path problem Q is defined in terms of a *weight domain* $(W, \otimes, \oplus)$ and a graph $G = (\mathrm{Nodes}, \mathrm{Edges}, s)$, where s denotes the start node, along with a function $\mathrm{weight} : \mathrm{Edges} \to W$ that assigns a weight from W to each edge in G. Then the path-weight for a path $p = e_1, e_2, \ldots, e_k$ in G is defined as follows:

$$\mathrm{pathWeight}(p) \overset{\mathrm{def}}{=} \mathrm{weight}(e_1) \otimes \mathrm{weight}(e_2) \otimes \ldots \otimes \mathrm{weight}(e_k).$$

Finally, we define the Q value for n (with respect to start node s) as follows:

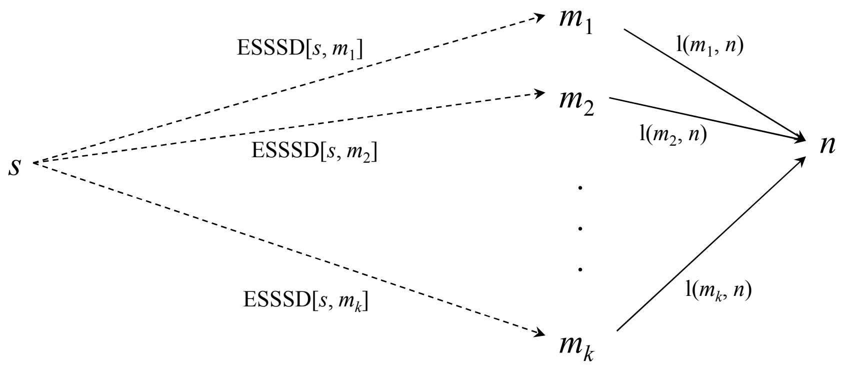
$$Q[s,n] \overset{\mathrm{def}}{=} \bigoplus_{p \in \mathrm{Paths}(s,n)} \mathrm{pathWeight}(p). \tag{2}$$

[1] See also the notes from the previous lecture.

[2] More precisely, $\otimes$ is an infix operator for the function $\lambda x.\lambda y.(x+y)$, and $\oplus$ is an infix operator for the function $\lambda x.\lambda y.\min(x,y)$.

1.2 Equational Formulation

The *equational formulation* of a path problem adopts the local viewpoint depicted below



and sets up the following collection of equations:

$$ESSSD[s,n] = \begin{cases} 0 & \text{if } n = s \\ \underset{\langle m,n\rangle \in \text{Edges}}{\text{Min}} ESSSD[s,m] + l(m,n) & \text{otherwise} \end{cases} \tag{3}$$

Theorem 1.1 *For all n, $ESSSD[s,n] = SSSD[s,n]$.* $\square$

This theorem can either be proved directly (hint: use induction on the length of paths considered in computing Paths(s,n) in in Eqn. (1)), or as an instance of a more general theorem about path problems. The equational version of a path problem Q, which we will denote by EQ, is the analog of Eqn. (3); it is defined using the following set of equations:

$$EQ[s,n] = \begin{cases} 0 & \text{if } n = s \\ \bigoplus_{\langle m,n\rangle \in \text{Edges}} EQ[s,m] + \text{weight}(m,n) & \text{otherwise} \end{cases} \tag{4}$$

Note that the weight domain W is typically a partially ordered set. When the orientation of W is the one typically used in the abstract-interpretation community, we seek the *least* solution to Eqn. (4). When the orientation of W is the one typically used in the dataflow-analysis community, we seek the *greatest* solution to Eqn. (4). However, by duality, the orientation does not matter for the purpose of stating the following theorem:

Theorem 1.2 *If $\otimes$ distributes over $\oplus$, then for all n, $EQ[s,n] = Q[s,n]$.* $\square$

Again, the theorem can be proved using induction on the length of paths considered in computing Paths(s,n) in Eqn. (2).

2 Standard and Collecting Semantics

We will consider the CFG example from the previous lecture (see Fig. 1). We first define some notation that we will be using in subsequent sections.

$$\text{Store} \overset{\text{def}}{=} (\text{Var} \to (\mathbb{Z} \cup \{?\}))_\perp$$

The function M assigns a Store-transformation function to each edge in G:

$$M : \text{Edges} \to (\text{Store} \to \text{Store})$$

$$M(\langle m,n\rangle) \overset{\text{def}}{=} \begin{cases} \underline{\lambda}\sigma.\sigma[x \leftarrow E[\![e]\!]\sigma] & \text{if } m\text{'s label is } \texttt{x:=e} \\ \underline{\lambda}\sigma.B[\![b]\!]\sigma ? s : \perp & \text{if } m\text{'s label is "}\texttt{if } b\text{" and } \langle m,n\rangle \text{ is the True branch} \\ \underline{\lambda}\sigma.B[\![b]\!]\sigma ? \perp : s & \text{if } m\text{'s label is "}\texttt{if } b\text{" and } \langle m,n\rangle \text{ is the False branch} \end{cases}$$

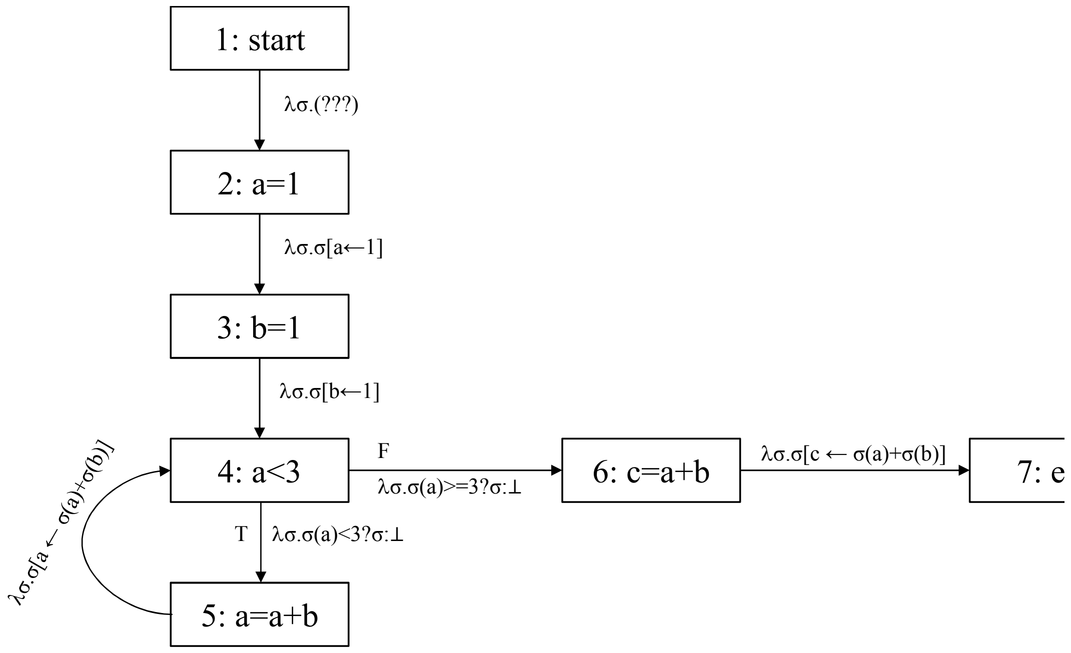


Figure 1: Example control-flow graph.

2.1 Standard Semantics

Let $p = e_1, e_2, \ldots, e_k$ be a path from s to n; then p's *path transfer function* is

$$pf_p = M(e_k) \circ \ldots \circ M(e_2) \circ M(e_1) = M(e_1) \otimes M(e_2) \otimes \ldots \otimes M(e_k),$$

where $\otimes$ is the reverse of function composition.

2.2 Collecting Semantics

2.2.1 Declarative Formulation

Given an initial Store and a path p to node n, the standard semantics produces a Store at n, but abstract interpretation is concerned with determining information about *sets* of Stores. To formalize such sets, we define the *collecting semantics.* The collecting semantics at node n, which we will denote by $CS[s, n]$, is defined to be the set of all Stores that can possibly arise at n during *some* execution of the program (starting from start node s).

$$CS[s, n] = \{pf_p(\sigma) \mid p \in \text{Paths}(s, n), \sigma \in \text{Store}\}. \tag{5}$$

Note that the condition "$\sigma \in$ Store" means that *any* Store can be fed in at the beginning of execution.

The collecting semantics deliberately loses information about

- *whether* two Stores, $\sigma_1, \sigma_2 \in CS[s, n]$, *can both arise* during the execution of the program on some initial store $\sigma_0 \in$ Store
- the *order* in which two Stores, $\sigma_1, \sigma_2 \in CS[s, n]$ arise at n, if σ_1 and σ_2 can both arise during the execution of the program on some initial store $\sigma_0 \in$ Store.

Eqn. (5) is a start; however, we have not succeeded in formulating the collecting semantics as a path problem in the sense of §1: Eqn. (5) uses set comprehension rather than "combine-over-all-paths." Instead of Eqn. (5), we want to express CS so that it has the following form:

$$CS[s,n] = \bigoplus_{p \in \text{Paths}(s,n)} F(p), \tag{6}$$

where F is some function on path p that uses $\otimes$.

Let us define some notation similar to that used for defining the standard semantics. SM is M lifted to sets.

$$SM : \text{Edges} \to (\mathcal{P}(\text{Store}) \to \mathcal{P}(\text{Store})),$$

where $\mathcal{P}(\text{Store})$ denotes the powerset of Store. The function SM assigns to each edge in G a function that transforms a set of Stores to a set of Stores:

$$SM(\langle m,n \rangle) \overset{\text{def}}{=} \lambda S.(\{M(\langle m,n \rangle)(\sigma) \mid \sigma \in S\} - \{\bot\})$$

(We subtract out $\{\bot\}$ in the right-hand side expression because $\bot$ is not a proper Store.) Then p's *setwise path transfer function* is

$$spf_p \overset{\text{def}}{=} SM(e_k) \circ \ldots \circ SM(e_2) \circ SM(e_1) = SM(e_1) \otimes SM(e_2) \otimes \ldots \otimes SM(e_k),$$

where $\circ$ denotes function composition, and $\otimes$ is the reversal of $\circ$. Combine ($\oplus$) is just $\cup$. Then we define the *setwise collecting semantics* as follows:

$$SCS[s,n] \overset{\text{def}}{=} \bigcup_{p \in \text{Paths}(s,n)} spf_p(\text{Store}). \tag{7}$$

In contrast to Eqn. (5), Eqn. (7) has the form that we desire (cf. Eqn. (6)).

2.2.2 Equational Formulation

We now turn to an equational formulation of the collecting semantics, which we will denote by $ECS[s,n]$. $ECS : \text{Nodes} \to \mathcal{P}(\text{Store})$, where

$$ECS[s,n] \overset{\text{def}}{=} \begin{cases} \text{Store} & \text{if } n = s \\ \displaystyle\bigcup_{\langle m,n \rangle \in \text{Edges}} SM(\langle m,n \rangle)(ECS[s,m]) & \text{otherwise} \end{cases} \tag{8}$$

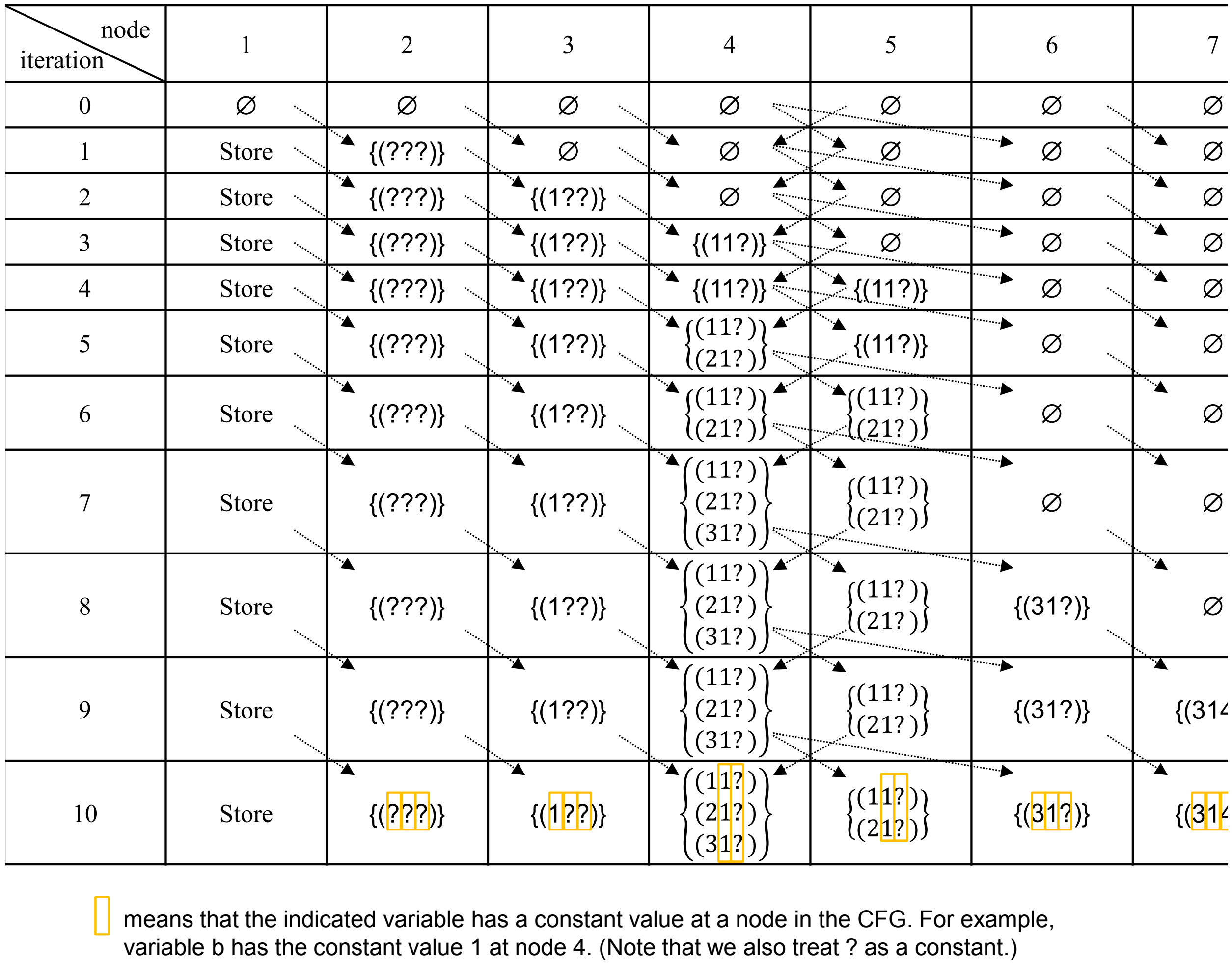
The result of computing ECS is shown in Fig. 2. Let's work through one entry of the table from Fig. 2. Suppose that we are computing ECS for node 4 at iteration 5:

$$ECS(4) = SM(\langle 3,4 \rangle)(ECS(3)) \cup SM(\langle 5,4 \rangle)(ECS(5))$$

From the previous iteration, we know that $ECS(3) = \{(1??)\}$ and $ECS(5) = \{(11?)\}$.
Then

$$\begin{aligned} SM(\langle 3,4 \rangle)(ECS(3)) &= SM(\langle 3,4 \rangle)(\{1??\}) = \{M(\langle 3,4 \rangle)(\sigma) \mid \sigma \in \{(1??)\}\} - \{\bot\} \\ &= \{M(\langle 3,4 \rangle)((1??))\} = \{\lambda s.s[b \leftarrow 1]((1??))\} \\ &= \{(11?)\} \end{aligned}$$

Similarly, we have $SM(\langle 5,4 \rangle)(ECS(5)) = \{(21?)\}$. Thus, at iteration 5, we obtain $ECS(4) = \{(11?)\} \cup \{(21?)\} = \{(11?)(21?)\}$.



iteration \ node	1	2	3	4	5	6	7
0	∅	∅	∅	∅	∅	∅	∅
1	Store	{(???)}	∅	∅	∅	∅	∅
2	Store	{(???)}	{(1??)}	∅	∅	∅	∅
3	Store	{(???)}	{(1??)}	{(11?)}	∅	∅	∅
4	Store	{(???)}	{(1??)}	{(11?)}	{(11?)}	∅	∅
5	Store	{(???)}	{(1??)}	{(11?), (21?)}	{(11?)}	∅	∅
6	Store	{(???)}	{(1??)}	{(11?), (21?)}	{(11?), (21?)}	∅	∅
7	Store	{(???)}	{(1??)}	{(11?), (21?), (31?)}	{(11?), (21?)}	∅	∅
8	Store	{(???)}	{(1??)}	{(11?), (21?), (31?)}	{(11?), (21?)}	{(31?)}	∅
9	Store	{(???)}	{(1??)}	{(11?), (21?), (31?)}	{(11?), (21?)}	{(31?)}	{(31
10	Store	{(???)}	{(1??)}	{(11?), (21?), (31?)}	{(11?), (21?)}	{(31?)}	{(31

▯ means that the indicated variable has a constant value at a node in the CFG. For example, variable b has the constant value 1 at node 4. (Note that we also treat ? as a constant.)

Figure 2: Table of the values that arise when Eqn. (8), instantiated for the CFG shown in Fig. 1, is solved by the method of successive approximations.

3 Abstract Semantics (where # comes in)

So far we have been defining variants of a program's concrete semantics. We are now going to define the program's abstract semantics. We again start by introducing appropriate notation. The items of notation defined below correspond to those used for discussing the program's concrete semantics.

The function $M^\sharp$ assigns to each edge in G a function that transforms an AStore to an AStore:

$$M^\sharp : \text{Edges} \to (\text{AStore} \to \text{AStore}).$$

AStore stands for "abstract Store." An abstract store in AStore denotes a set of concrete stores. In many problems, AStore has the following type, for an appropriate space of abstract values $\text{Val}^\sharp$.

$$\text{AStore} : \text{Var} \to \text{Val}^\sharp.$$

Each $v^\sharp \in \text{Val}^\sharp$ represents a set of concrete values.

We assume that we have in hand sound abstract-interpretation functions for our language's Boolean expressions (denoted by $B^\sharp[\![\cdot]\!]$) and arithmetic expressions (denoted by $E^\sharp[\![\cdot]\!]$). (One way to obtain such functions is by the method of syntax-directed described in §2.5.1 of the Oct. 6, 2015 lecture.) If node m's label is `x:=e`, we define $M^\sharp(\langle m, n\rangle)$ as follows:

$$M^\sharp(\langle m, n\rangle) = \lambda a. \begin{cases} \bot & \text{if } a = \bot \\ a[x \leftarrow E^\sharp[\![e]\!](a)] & \text{otherwise} \end{cases}$$

If m's label is "`if` b" and edge $\langle m, n\rangle$ is the True branch, we define $M^\sharp(\langle m, n\rangle)$ as follows:

$$M^\sharp(\langle m, n\rangle) = \lambda a. \begin{cases} \bot & \text{if a=}\bot \\ a & \text{if } B^\sharp[\![b]\!](a) = True \\ \bot & \text{if } B^\sharp[\![b]\!](a) = False \\ a & \text{if } B^\sharp[\![b]\!](a) = \top \text{ (don't know T or F; prepare both ways)} \end{cases}$$

If m's label is "`if` b" and edge $\langle m, n\rangle$ is the False branch, $M^\sharp(\langle m, n\rangle)$ is defined similarly.

3.1 Declarative Formulation

The *abstract path transfer* function for a path $p = e_1, e_2, \ldots, e_k$ is defined as follows:

$$apf_p \stackrel{\text{def}}{=} M^\sharp(e_k) \circ \ldots \circ M^\sharp(e_2) \circ M^\sharp(e_1) = M^\sharp(e_1) \otimes M^\sharp(e_2) \otimes \ldots \otimes M^\sharp(e_k),$$

where $\otimes$ is the reverse of function composition. The *abstract path semantics*, denoted by $APS[s, n]$, is defined as follows:

$$APS[s, n] \stackrel{\text{def}}{=} \bigoplus_{p \in \text{Paths}(s,n)} apf_p(\text{AStore})$$

where $\bigoplus$ is the combine operator of the abstract domain.

3.2 Equational Formulation

Finally, we define the *equational abstract semantics*, denoted by $EAS[s, n]$, as follows:

$$EAS[s, n] \stackrel{\text{def}}{=} \begin{cases} \top & \text{if } n = s \\ \displaystyle\bigoplus_{\langle m,n\rangle \in \text{Edges}} M^\sharp(\langle m, n\rangle)(EAS[s, m]) & \text{otherwise} \end{cases} \tag{9}$$

where $\top$ is the value of AStore that represents all stores in Store.

The result of computing EAS for the CFG from Fig. 1, together with the previous ECS values from Fig. 2, is shown in Fig. 3.

References

node / iteration	1	2	3	4	5	6	7
0	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥
1	Store [⊤⊤⊤]	{(???)} [???]	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥
2	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	∅ ⊥	∅ ⊥	∅ ⊥	∅ ⊥
3	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	{(11?)} [11?]	∅ ⊥	∅ ⊥	∅ ⊥
4	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	{(11?)} [11?]	{(11?)} [11?]	∅ ⊥	∅ ⊥
5	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	{(11?)} [11?]	∅ ⊥	∅ ⊥
6	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	∅ ⊥	∅ ⊥
7	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\\(31?)\end{matrix}\right\}$ [⊤1?]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	∅ ⊥	∅ ⊥
8	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\\(31?)\end{matrix}\right\}$ [⊤1?]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	{(31?)} [⊤1?]	∅ ⊥
9	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\\(31?)\end{matrix}\right\}$ [⊤1?]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	{(31?)} [⊤1?]	{(314)} [⊤1⊤]
10	Store [⊤⊤⊤]	{(???)} [???]	{(1??)} [1??]	$\left\{\begin{matrix}(11?)\\(21?)\\(31?)\end{matrix}\right\}$ [⊤1?]	$\left\{\begin{matrix}(11?)\\(21?)\end{matrix}\right\}$ [⊤1?]	{(31?)} $[\boxed{\top}1?]$	{(314)} $[\boxed{\top}1\boxed{\top}]$

▯ shows places where we obtain worse information than what the collecting semantics indicates. The loss of information at one point can cause other information to be lost (e.g., variable c at point 7, due to imprecise information for variable a at point 6).

Figure 3: Table of the values that arise when Eqn. (9), instantiated for the CFG shown in Fig. 1, is solved by the method of successive approximations (together with the *ECS* values from Fig. 2):