

הפשטה כמושג מרכזי בהוראת מדעי המחשב: סקר ספרות

פיתוח ועריכה: ד"ר ברוריה הברמן, מכון ויצמן למדע

הפשטה הינה מושג מרכזי במדעי המחשב המוכר כעיקרון יסודי וחיוני בפתרון בעיות במדעי המחשב ובפיתוח תוכנה. חוקרים ואנשי חינוך דנים כיצד לשלב את המושג בהוראת מדעי המחשב. באתר זה תוכלו למצוא סקירה של דעות וגישות שונות הקשורות להוראת הפשטה. האתר כולל:

- מבוא, הכולל סקירת ספרות נרחבת.
- סיכום של מאמרים נבחרים של חוקרים בעולם, המתייחסים לשילוב הפשטה בתכניות לימוד המקובלות בחו"ל (בעיקר למתחילים).
- סיכום של מאמרים שנכתבו בידי חוקרים בארץ המתייחסים להוראת הפשטה.
- קטע מעבודה סמינריונית של סטודנטית המתייחסת לתכנות לוגי כפרדיגמה תומכת הפשטה.
- רשימת מקורות מומלצים

הפשטה כמושג מרכזי במדעי המחשב

במדעי המחשב, מושג הפשטה מתאר תהליך של איחוד מספר רב של פעולות "קטנות" או פרטי מידע (נתונים) "קטנים" ליחידה אחת, שניתן להתייחס אליה בשם. זוהי טכניקה המסייעת במציאת מכנה משותף בפרטים ובהקלת השימוש בקוד ובמידע. הפשטה מאפשרת לתוכניתן לחשוב בפשטות על בעיה, באמצעות דחית הטיפול בפרטים הפחות חשובים. וכך מתאפשרת חשיבה על המטרות היותר חשובות ועיקריות, במספר דרגות חשיבה ולא בבת אחת. זהו ייצוג התופס רק את ההיבטים המהותיים של הדבר, מצמצם את המורכבות הנראית למשתמש, ולוקח בחשבון את העיקר שבמימוש. הפשטה באה לביטוי גם בזמן ניתוח של אלגוריתמים נתונים בהקשר של זיהוי הבעיה אותה פותר האלגוריתם או הערכה האם אלגוריתם פותר בעיה נתונה. יכולת הזיהוי של הדבר המהותי המתבצע על ידי מכלול של פעולות, יכולת חלוקה של הפעולות לקבוצות ומתן שם לתוצאת ביצוע של כל קבוצה וכדומה.

הפשטה מוכרת זמן רב כעיקרון יסודי וחיוני בפתרון בעיות במדעי המחשב ובפיתוח תוכנה. לדעת Denning (1989) מושג הפשטה הוא אחד משלושת העקרונות המרכזיים בתחום המחשבים, בצד התיאוריה והתיכון. הפשטה נוגעת ליכולת לעשות שימוש ביחסים שבין אובייקטים בכדי לעשות השלכה לדברים מוחשיים בעולם (Denning et al., 1989). Aho ו-Ullman מתארים את מדעי המחשב כמדע הפשטה- יצירת המודל לחשיבה על בעיה ותכנון הטכניקה ההולמת לפתרונה (Aho & Ullman, 1995). Astrachan מזכיר *רמות של הפשטה* כמושג חוזר במדעי המחשב בצד מודלים קונספטואליים ופורמליים, יעילות וסיבוכיות (Astrachan, 1997). לדעת Hoare (1986) פקודה אבסטרקטית היא פקודה המפרטת את התכונות הכלליות של התנהגותו הרצויה של המחשב מבלי לפרט במדויק איך התנהגות זו מושגת.

במדעי המחשב מתייחסים לשני סוגי הפשטה:

א. **הפשטת נתונים** (data abstraction)- העוסקת בהפרדה של תכונות לוגיות של מידע מפרטי מימושן (Dale & Walker, 1996). בהקשר זה מוצג העקרון של ארגון מידע במבנים ובכלל זה שימוש ברשומות ומערכים המאפשרים גישה נוחה לנתונים. בנוסף מוצגים טיפוסים נתונים מופשטים שונים

וממשיקהם כגון: רשימה, מחסנית, תור ועץ בינארי, ככלים חיוניים בהפשטת נתונים המאפשרים הסתרת האופן בו מאורגנים הנתונים בתכנית.

ב. **הפשטת פעולות** (procedural abstraction) - עוסקת בהפרדת תכונות לוגיות של פעולה מפרטי מימושן. אחת הדרכים לתרגל הפשטת פעולות היא לתרגל פירוק בעיות ופיתוח אלגוריתמים, באמצעות פיתוח עיצוב מלמעלה למטה (top-down design) ושימוש בגישת "צעד אחר צעד" – עידון הדרגתי (stepwise refinement). Leron מתאר את ההפשטה הפרוצדורלית במונחים של מחסומי הפשטה מטאפוריים: "תכנות ברמה הולמת של הפשטה פירושו בחירת הפעולות העיקריות המתאימות לבעיה. [...] הימנעות זו, מעיסוק בפרטי הרמה הנמוכה בזמן שחושבים על הבעיה שיש לפתור, יוצרת מחסום הפשטה מטאפורי המחלק את משימת התכנות לשתי משימות כמעט בלתי תלויות ביניהן: מעל למחסום אנו עוסקים בפתרון הבעיה המקורית במונחים של הפעולות העיקריות המופשטות שבחרנו; ומתחת, אנו עוסקים בפרטי המימוש של פעולות עיקריות אלו" (Leron, 1987, p. 16-17). אכן, להפשטה פרוצדורלית יש יתרון בהפחתת המורכבות של בעיה נתונה ע"י חלוקתה לתת-בעיות קטנות יותר, שאולי גם כן יחולקו לתת-בעיות. בכל רמה של פירוק, אין צורך לדאוג איך המשימה תושג ברמה הנמוכה יותר. יתר על כן, מתכנתים אינם צריכים לדאוג, כאשר הם משתמשים בפרוצדורה מסוימת, לכל הפרטים הנוגעים לאופן שבו היא מבצעת את חישוביה; הם רק צריכים להבין את התיאור שלה (specification) ומה היא מחשבת, וכן לדעת איך לקרוא לה. גם במקרה של בעיות פשוטות, יש הגיון באיתור רמות הפשטה בהגדרת הבעיות, ולעצב תכניות המשקפות רמות הפשטה אלו. קל יותר למתכנת להתמודד בנפרד עם רמות הפשטה השונות ולהתרכז באחת אחר השנייה, וכך להימנע מטעויות. מלבד ההתמודדות הנאותה עם מורכבות הבעיה, להפשטה פרוצדורלית יש יתרון בבניית "קופסאות שחורות" שניתן לעשות בהן שימוש חוזר ולממשן בצורה טובה יותר.

שילוב הפשטה כמושג מרכזי בתכניות לימודים במדעי המחשב

מאחר והפשטה היא אחד ממושגי המפתח היסודיים בתכנות, היא שולבה בתכניות לימודים של מדעי המחשב. Denning ושותפיו מציגים שלוש פרידיגמות ראשיות: תיאוריה, הפשטה ועיצוב, המהוות צירים מרכזיים חותכי דיסציפלינת ה- Computing. הם מתייחסים להפשטה כעקרון משותף לשני התחומים הבאים: (1) תחום מדעי המחשב, המתמקד בניתוח והפשטה, ו- (2) תחום הנדסת מחשבים, המתמקד בהפשטה ועיצוב (Denning et al., 1989).

כוח המשימה שפיתח את תכנית הלימודים Curricula CC2001 טען כי תלמידי מדעי המחשב צריכים ללמוד לשלב תיאוריה ותרגול, להכיר בחשיבותה של הפשטה, ולהוקיר את ערכו של עיצוב (תיכון) טוב כעקרונות עיקריים של התחום (CC2001 Computer Science Final Report, 2001, p. 12).

למושג הפשטה נודעת חשיבות רבה בכל הקשור לפיתוח דרכי חשיבה ושיטות עבודה ככלים לפתרון בעיות במדעי המחשב. לדעת חוקרים, מושג זה מהווה מנוף לשיפור היכולות לפתרון בעיות בעיקר אצל תלמידים מתחילים הנמצאים בשלבים הראשונים של רכישת ידע ומיומנויות עבודה נאותים בכל הנוגע לפיתוח תכניות מחשב. שימוש בהפשטה בתהליך פתרון בעיה עשוי לעודד שימוש חוזר בקוד וכן עשוי לשפר את עיצוב התכנה.

חוקרים מסכימים שמומחים ומתחילים מגלים הבדלים בסגנונות התכנות שלהם, במיוחד בהקשר של פתרון בעיות, יכולות הפשטה והכללה. לפי Linn ו-Clancy (1992) מומחים מארגנים את הידע שלהם במבנים קונספטואלים שהם בהיבט נרחב יותר מתחביר, ארגון המאפשר להם שימוש חוזר בתבניות שאינן תלויות בשפת תכנות מסוימת. Fleury (1993) מצאה שמתחילים שונים ממומחים בהרגלי התכנות שלהם בהקשר של פיתוח ותחזוקה של מערכות ארוכות-טווח. Ye ו-Salvendy (1996) מצאו שלמומחים יש ידע תכנותי ברמת הפשטה גבוהה יותר מאשר למתחילים. Hoadley ועמיתיו (1996) מצאו שאותם תלמידים שיכלו לייצר סיכומי קוד מופשטים, העדיפו לעשות שימוש חוזר בקוד מאשר לשכפל קוד.

לימוד הולם של הפשטה למתחילים עשוי להעשיר את הכלים לפתרון בעיות, כך שאנשי חינוך בודאי מסכימים שיש להציג לתלמידי מדעי המחשב את עקרונות הפשטה בהדרגה, החל מהשלבים המוקדמים ללימודיהם (Dale & Walker, 1996; Evans, 1996; Astrachan, 1997; Marion, 1999, Bucci et al., 2001). לדוגמה, Marrion (1999) הציע כי תלמידי CS1 (מקביל ל-יסודות 1 ויסודות 2) צריכים להתחיל להבין הפשטה ואת תפקידה בהתמודדות עם מורכבות תכנה, מאחר והפשטה הינה כלי חשוב, בו פותר הבעיה משתמש, על מנת להתמודד עם מורכבותה. בפרט הפשטת פעולות היא כלי עזר בעיצוב פתרון מלמעלה-למטה (top-down design).

Machanick (1998) הציע ארגון מחדש של הקורס מבני נתונים ואלגוריתמים, והציג את גישתו - "הפשטה תחילה" בהפשטת נתונים ואלגוריתמים. הוא טען כי הפשטה ושימוש חוזר צריכים להילמד לפני שיטות אחרות, "כך, קידוד כשלעצמו אינו צריך להילמד לפני שהשימוש החוזר נראה טבעי" (Machanick, 1998, p. 135). יותר מכך, הוא הציע שימוש בדוגמאות מהעולם האמיתי להמחשת ההפשטה לפני העיסוק במחשבים ותכנות, ובשלב הבא, להכיר לתלמידים הפשטות קיימות שניתן להשתמש בהן, כגון בנית "בלוקים" בתכנות, וכך להדגים את היכולת לעשות שימוש חוזר בקוד מבלי לדעת עליו דבר על אופן מימושו.

Koppelman (2001) טען כי הפשטה צריכה להילמד במפורש לתלמידים מתחילים באמצעות הצגה והמחשה של העיקרון על בסיס תכניות פשוטות. Bucci ושותפיו סברו כי לימוד הפשטה לפי גישה המבוססת על מודלים מתמטיים והגדרות פורמליות עשויה לתרום ליכולת התלמידים לחשוב בזהירות ובדקדקנות על התנהגות התכנית. הם ממליצים על הצגת הפשטה והסתרת מידע כהיבטים משלימים לגישה של הסתרת הפרטים של המערכת תוך סיפוק "סיפור כיסוי" פשוט המסביר את ההיבטים בהם מתעניינים במערכת זו (Bucci et al., 2001).

Evans (1996) מציע גישה פדגוגית להצגת CS1 תוך הדגשת עקרונות יסוד, שאחד מהם הוא הפשטה אשר הינה מושג חוזר במדעי המחשב. הוא ממליץ להדגיש את גישת PITL (programming-in-the-large) להקניית רמה מערכתית, ובמקביל לספק כלים לעבודה ב"קטן" – PITS (programming-in-the-small) המאפשרים לטפל בפרטי מימוש, ומקנים יכולות תכניות. לדעתו, גישה פדגוגית זו מקנה לתלמידים ראייה רחבה יותר של תכנות ופתרון בעיות, ומפתחת כישרים של הפשטה בצד תיכון.

שימוש בהפשטה עשוי גם לתרום להבנת נושאים מתקדמים כגון רקורסיה. לדוגמה, Sooriamurthy (2001) טוען כי המפתח להבנת רקורסיה הוא התמקדות בהפשטה של פונקציות רקורסיביות. הוא פיתח גישה תבניתית שעוצבה במטרה להכיר לתלמידים את השלבים השונים בניסוח פתרון

רקורסיבי. בדומה, Ginat ו-Shifroni (1999) טוענים שדגש על גישה דקלרטיבית להוראת רקורסיה, ודגש על התייחסות לרמה המופשטת של פירוק בעיה, עשויים לשפר כתיבה של תכניות רקורסיביות.

שילוב הפשטה בתכנית הלימודים לתיכון בארץ

בתחילת שנות ה-90 פותחה בישראל תכנית לימודים חדשה למדעי המחשב בתיכון. התכנית מציגה את העקרונות והשיטות לפתרון בעיות ללא קשר למחשב מסוים או לשפת תכנות מסוימת, בצד יישום בשפות ופרדיגמות שונות (Gal-Ezer et al., 1995; Gal-Ezer, Harel, 1999). אחד המושגים המרכזיים הנלמדים בתכנית הלימודים הוא מושג ההפשטה.

הפשטה נלמדת לאורך כל יחידות הלימוד בתכנית ומוצגת בשתי רמות: הפשטת פעולות (procedural abstraction) והפשטת נתונים (data abstraction).

ביחידות הראשונות בתכנית הלימודים התלמידים מכירים את פרדיגמת התכנות הפרוצדוראלי, ויש שמספיקים להכיר גם את פרדיגמת התכנות הלוגי או את פרדיגמת התכנות הפונקציונלי, הנלמדות כיחידת בחירה שלישית. כך שלמעשה יחידת "עיצוב תכנה" (יחידה רביעית) נלמדת לאחר שהתלמידים הכירו פרדיגמה תכנותית אחת או יותר.

מושג ההפשטה מוצג כעיקרון חוזר ביחידות הלימוד השונות של תכנית הלימודים במדעי המחשב. שתי יחידות הלימוד הראשונות, יסודות 1 ו-2, מציגות את העקרונות הבסיסיים של בעיה אלגוריתמית ואת פתרונה- האלגוריתם, כמו כן, מוצגים מושגים הנוגעים להפשטה כגון פירוק בעיות לתת בעיות בצד נכונות ויעילות של אלגוריתמים. פונקציות מוצגות כ"קופסאות שחורות"- כלים לפתרון בעיות באמצעות חלוקה לתת משימות, שזוהי בעצם הפשטת פעולות. לעיתים פונקציות שכותבים התלמידים משולבות עם פונקציות מוגדרות מראש ועם פונקציות ספריה, דבר זה ממחיש את החשיבות של שימוש נכון בטענות כניסה ויציאה. עקרון הפשטת הנתונים מוצג כאשר נלמדים טיפוסים נתונים מורכבים כגון מערכים ורשומות. בצד העבודה עם פונקציות מוגדרות מראש ופונקציות ספריה, תלמידים מפתחים וכותבים פונקציות משלהם למימוש תת משימות מסוגים שונים באופנים שונים.

יחידת "עיצוב תכנה" מציגה עקרונות כגון הסתרת מידע, מודולריות, יעילות, שימוש חוזר בקוד והפשטה. יחידת לימוד זו חושפת את התלמידים למגוון של היבטים בעיצוב ותיכון מערכות תכנה דרך השימוש בטיפוסי נתונים מופשטים. אחת המטרות העיקריות של יחידה זו היא לפתח מיומנויות חשיבה מופשטת ודרכים לפתור בעיות (Gal-Ezer et al., 1995; Gal-Ezer & Harel, 1999) ולכן שימוש בקוד מוגדר מראש והפשטה משרתים כקווים מנחים לאורך כל היחידה, במקום תכנות ממשי (Gal-Ezer & Zeldes, 2000). היחידה מתרכזת בהיבטים התיאורטיים היסודיים של מבני נתונים, ומציגה בהרחבה את שילובו של טיפוס הנתונים המופשט בפיתוח מערכות תכנה. ליתר דיוק, היחידה מציגה עקרונות כגון הפשטת פעולות, הפשטת נתונים, הסתרת מידע, מודולריות, יעילות ושימוש חוזר בקוד. עקרונות הפשטת הנתונים מוצגים במסגרת יחידת הלימוד עיצוב תכנה משלוש נקודות מבט שונות: הגדרה, יישום ושימוש. עקרונות של טיפוס הנתונים המופשט כגון הפשטת פעולות, הפשטת נתונים, הסתרת מידע, מודולריות, יעילות ושימוש חוזר בקוד נידונים גם כן במסגרת יחידה זו. התלמידים לומדים ומתרגלים את ההגדרות, הייצוג, היישום והשימוש של טיפוסים נתונים מופשטים כגון: רשימה, מחסנית, תור ועץ בינארי. הם לומדים להגדיר טיפוס נתונים מופשט חדש,

לעצב לו ממשק לפי מפרט נתון, ולחבר יחידות ספריה המבצעות את המפרט הנדרש בסביבת התכנות הנתונה. התלמידים גם מתרגלים שימוש בטיפוסי נתונים מופשטים לצורך פתרון בעיות. במסגרת היחידה התלמידים לומדים לבצע הכללה והפשטה של בעיות דומות, וכן, הם לומדים לקבוע את טיפוס הנתונים המופשט המתאים לבעיה נתונה, ולעשות שימוש בממשק של טיפוס הנתונים המופשט בבניית אלגוריתמים (Haberman, 2002). התלמידים פוגשים בעקרון ההפשטה גם כאשר הם לומדים את יחידות הבחירה תכנות לוגי או תכנות פונקציונלי. ביחידת תכנות לוגי הם מכירים את הגישה הדקלרטיבית לפתרון בעיות. הם לומדים שיטות לייצוג ופורמליזציה של מידע, ומתרגלים את השימוש בטיפוסי נתונים מופשטים לייצוג ידע בסביבת פרולוג (Haberman et al., 2002). אופן הצגת עקרון ההפשטה לתלמידים הלומדים תכנות לוגי מתואר בסעיף נפרד. אלו הלומדים תכנות פונקציונלי, מתרגלים הפשטה פונקציונלית והפשטת טיפוסי נתונים תוך שימוש בעיבוד רשימות.

קשיי מתחילים בביצוע הפשטה

תלמידים מתחילים הנמצאים בשלבים הראשונים ללימודי התכנות מגלים לעיתים קשיים שונים בהתנסות עם תחום זה (הברמן, [סקר ספרות על קשיי מתחילים](#)). חלק מקשיים אלו נובעים מקושי בהפשטה.

ראינו כי מושג ההפשטה תורם רבות ומשפיע על תחומים מגוונים בלימודי מדעי המחשב ובשיטות העבודה הנרכשות. אך לעיתים מגלים תלמידים, ובפרט אלו הנמצאים בשלבים המוקדמים של לימודיהם, קשיים בהפנמת עקרונות ההפשטה וביישומם בפועל בעת כתיבת תכנית מחשב. לימוד תכנות הוא משימה מורכבת הנערכת בשני צירים: הציר התיאורטי המופשט, המתייחס למושגים ועקרונות של מדעי המחשב ותכנות, וציר היישום, המתייחס לשפת תכנות ולסביבת העבודה, המשמשים ליישום העקרונות הנלמדים (הרמה התיאורטית המופשטת). ניתן לבצע את המעבר בין שני הצירים לסירוגין. למשל, אחת הדרכים לביצוע כזה: מושג או רעיון חדש מוצג תחילה בכיתה, ומיושם מיד לאחר מכן בפעילות מעבדה, ושוב דנים בו ברמת הכיתה, וחוזר חלילה. וכך, צריך התלמיד המתחיל להתמודד עם הבנה ותפקוד בשני המישורים בו זמנית. שיטה המבוססת על עקרון זה, המכונה שיטת ה"רוכסן" הומלצה להוראת תכנות הלימודים החדשה במדעי המחשב בארץ (Harel 1999 & Gal-Ezer, Yehudai, Harel & Beeri, 1995, Gal-Ezer) בין השאר, לחזק את יכולת ההפשטה ואת ההבחנה בין תיאוריה ויישום אצל הלומד. אך הניסיון מראה כי תלמידים מתחילים הלומדים בשיטה זו מתקשים בהבנת מושגים ועקרונות בסיסיים הקשורים בתכנות (Haberman & Kolikant, 2001), בו בזמן שעליהם להתמודד עם הכרת שפת תכנות וסביבת העבודה בה הם מיישמים את העקרונות הנלמדים על ידי כתיבה והרצה של תכניות במחשב. לעתים, גם קשה להם להבחין בין עקרונות שאינם תלויים בשפה תכנות זו או אחרת, לבין היבטים טכניים הקשורים לסביבת העבודה.

בין יתר הקשיים הרווחים בקרב תלמידים מתחילים קיים הקושי בביצוע הפשטה. Oliver (1993) בהתייחסו לקשיי מתחילים מצא כי תלמידים מתחילים מעטים גילו שליטה בפירוק בעיה לתת משימות. קושי זה קשור ישירות בקושי לבצע הפשטה. תלמידים המתקשים לפרק בעיה לתת משימות יתקשו לזהות גם את היכולת לבצע הפשטה ככלי המקל על פתרון הבעיה. הם מתקשים לראות את המבניות שבפתרון, את היכולת להפריד בין חלקי הפתרון כאשר כל חלק נבנה מחלקים

אחרים וכך מורכב לבסוף הפתרון השלם. כאשר התלמיד המתחיל צריך להתמודד עם בעיה מסוימת, הוא יחל את פתרונו ב"דף חלק". כלומר, הבעיה תיפתר באופן שלם מתחילתה עד סופה, מבלי לנסות למצוא קווים דומים בינה לבין בעיות שכבר נפתרו בעבר וכך לעשות שימוש חוזר בפתרונות קיימים ליצירת הפתרון המבוקש (Haberman, 2002).

ישנן סיבות שונות לקשיים אלו, ביניהן:

א. התלמיד המתחיל, אשר לא מבין את המשמעות של פיתוח מערכות תכנה מורכבות (Fluery, 1993), רואה בפתרון בעיה באמצעות הפשטה כפתרון "לא שלם", כזה שלא ניתן תמונה מספיק מפורטת ונרחבת העונה על צרכי הבעיה. בקרב תלמידים מתחילים רווח הצורך להציג את כל שלבי הפתרון בתוך תכנית סדרתית אחת, כאשר כל הפעולות מתבצעות אחת אחרי השנייה לפי הסדר. ביצוע קטעי קוד מחוץ לתכנית הראשית באמצעות שגרות נתפס בעיניהם כפתרון לא מושלם ולא יסודי מספיק.

ב. התלמיד המתחיל שם דגש על הפרשים זניחים של זמני ריצה, ביצוע הפשטה מביא לעיתים להוספת שורות קוד בתכנית הראשית ובתוך שגרות עזר בניגוד לתכנית המכילה את כל הקוד בתוכה ועשויה לחסוך בכך שורות קוד אחדות. תוספת זו נתפסת בעיני התלמיד המתחיל כבזבזנית ולא יעילה למרות שבמונחים של סדרי גודל הפרשים אלו זניחים וחסרי משמעות.

ג. התלמיד המתחיל רואה בשימוש בהפשטה כדבר הפוגם בקריאות של הקוד, כאשר חלקי הפתרון "מפוזרים" ולא מרוכזים במקום אחד, קשה לראות בבירור כיצד הבעיה נפתרה. קושי זה מושפע במידה רבה מהקושי המוזכר של תלמידים מתחילים לפרק בעיה לתת משימות.

ד. התלמיד המתחיל עדיין לא ער לחשיבותה של הסתרת מידע הולמת בעת כתיבת תכנית, ולכן לא מעריך את חשיבותה של ההפשטה ככלי המפרק תכנית למספר מרכיבים וכך מאפשר הסתרה של חלקים מן היישום.

Haberman ו-Gindi (2004) ערכו מחקר גישוש בקרב תלמידי מדעי המחשב בתיכון שמטרתו לבדוק האם לימוד פרדיגמת התכנות הלוגי בנוסף לפרדיגמת התכנות הפרוצדוראלי תורמת להבנה ויישום של עקרונות הפשטת פעולות. בפרט, המטרה היתה לבדוק האם תלמידים שלמדו את שתי הפרדיגמות מבינים ומיישמים עקרון של הפשטת פעולות בפיתוח וניתוח אלגוריתמים בתכנות פרוצדורלי, טוב יותר מעמיתיהם שלמדו תכנות פרוצדורלי בלבד. אוכלוסיית המחקר נחלקה לשתי קבוצות: בקבוצה הראשונה תלמידים שלמדו את יחידת הבחירה של התכנות הלוגי ובקבוצה השנייה תלמידים שלא למדו יחידה זו ומכירים רק את פרדיגמת התכנות הפרוצדוראלי. המחקר בדק את שיטות העבודה של התלמידים ואת רמות ההפשטה בהן נעשה שימוש ע"י כל אחת מן הקבוצות. הממצאים הראו לכאורה על הבדל בין התלמידים שמכירים תכנות לוגי לבין אלו שלא, כאשר לתלמידי תכנות לוגי יש יכולת הפשטה גבוהה יותר הבאה לידי ביטוי הן בניתוח והן בפיתוח אלגוריתמים. יחד עם זאת, יש להביע הסתייגות מהכללת ממצאי מחקר גישוש זה, שכן אוכלוסיית המחקר הייתה קטנה. בנוסף, כל התלמידים שנדגמו למדו עיצוב תכנה (מכורח הנסיבות – זמינות של אוכלוסיה) מה שתרים מן הסתם ליכולת ההפשטה שלהם.

הוראת הפשטה באופן מפורש

ה. קופלמן

Koppelman, H. (2001). Teaching abstraction explicitly. Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 191 (poster).

תקציר

המחבר טוען שלימוד תכנות צריך להיות כרוך לא רק בלימוד שפת תכנות, אלא גם בלימוד עקרון ההפשטה. לימוד הפשטה אינו פועל יוצא מלימוד שפת תכנות בלבד. יתרה מזאת, בספק אם למתכנתים מתחילים יש נטייה טבעית לעשות שימוש בהפשטה. בהתאם, המחבר טוען שיש ללמד הפשטה באופן מפורש למתחילים כבר בתחילת קורס המבוא במסגרת פתרון בעיות פשוטות. יש להרגיל את התלמידים לקרוא בעיות באופן כזה שבשלב הבנת הבעיה הם יזהו רמות הפשטה שונות בתיאור הבעיה, ובהתאם יתכננו את הפתרון ומבנה התכנית כאשר הם מתייחסים מפורשות לרמות ההפשטה שזוהו.

דוגמה

המחבר מציג דוגמה של בעיה פשוטה שניתן ללמד באמצעותה את עקרון ההפשטה: *כתוב תכנית אשר בודקת האם מחרוזת נתונה מופיעה במערך נתון של מלים.* התשובה המתקבלת לרוב מתלמידים מתחילים מבוססת על קינון לולאות, ולעתים קרובות לפחות אחת מהלולאות שגויה.

הצעת פתרון שכיחה של סטודנטים – רמת הפשטה אחת (Koppelman, 2001, p. 191)
WHILE there are more words AND NOT word found containing string examine next word of the array WHILE there are more characters AND NOT string found examine next characters of the word

הבעייתיות של פתרון זה הינה ששתי רמות ההפשטה מעורבבות בו. קריאה של קוד מפורט הכתוב לפי עקרון זה ומעקב אחר ביצועו גורם לקפיצות הלוך חזור בין שתי הרמות. כתוצאה מכך, תכנית הכתובה באופן זה קשה יותר לבדיקת נכונות. יתרה מזאת, במהלך כתיבת תוכנית כזאת, הכותב צריך להתרכז בו זמנית בטיפול בשתי הרמות, דבר המקשה על הכתיבה. לחילופין, כתיבת תכנית המבטאת ישירות את שתי רמות ההפשטה הרבה יותר פשוטה. שתי הרמות מופרדות ולכן כותב התוכנית יכול להתרכז בכתיבת כל רמה בנפרד. בנוסף, קל לבדוק נכונות של כל רמת הפשטה בנפרד. למשל, בדיקת הרמה העליונה אינה מחייבת התייחסות לאופן מימוש הפונקציה

ContainsString. בדומה, בדיקת הרמה השנייה, המתייחסת למימוש פונקציה זו אינה "מתערבבת" עם התייחסות לרמה העליונה.

סכמה של הצעת פתרון מומלצת – שתי רמות הפשטה (Koppelman, 2001, p. 191)
..... WHILE there are more words AND NOT WordFound WordFound := ContainsString(NextWord)

מתחילים אינם משתמשים בעקרון ההפשטה באופן טבעי

המחבר טוען שניתן ללמוד על נטיית תלמידים מתחילים ביחס לשימוש טבעי בעקרון ההפשטה על ידי בקשה לתאר מילולית את הרעיון לפתרון הבעה. הוא מביא כדוגמא עדות לנטייה של תלמידים לעבוד ברמת הפשטה אחת בלבד על ידי ציטוט של תלמיד:

"השווה את המחרוזת הנתונה עם שלושת התווים של המילה הראשונה במערך [מדובר באיתור מחרוזת באורך 3, ב. הברמן]. המשך באופן עקבי השוואה זו עד שהנך מגיע לסוף המילה או עד שהמחרוזת נמצאה. אם המחרוזת לא נמצא עדיין, המשך באופן דומה עם המילה השנייה, וכך הלאה עד סוף המערך."

תלמיד שמבטא באופן זה את תהליך הפתרון, אינו מפריד בין רמות ההפשטה, ועל כן מן הסתם יכתוב תכנית ברמת הפשטה אחת. מניסיונו של המחבר, זו נטייתם של רוב התלמידים המתחילים. בהתאם, המחבר טוען שמתחילים אינם משתמשים בעקרון ההפשטה באופן טבעי וספונטאני, ועל כן חשוב ללמדם הפשטה באופן מפורש כבר בשלבים מאד התחלתיים, תוך שימוש בתוכניות פשוטות.

ידע תכנותי של מומחים ומתחילים ברמות הפשטה שונות

נ. יא, ג. סלבנדי

Ye, N., Salvendy, G. (1996). Expert-novice knowledge of computer programming at different levels of abstraction. *Ergonomics*, 39(3), 461-481.

תקציר

המחברים מציעים הירארכיה בת חמש רמות הפשטה לידע תכנותי: רמה מטרית, רמה קונספטואלית, רמה פונקציונלית, רמה לוגית, ורמה פיזית. נערך ניסוי במטרה לבדוק האם השליטה בידע בכל רמת הפשטה משתנה כתלות ברמת המומחיות (דהיינו ברמת הכישורים (skills)). בניסוי נבדק הידע בכל אחת מחמשת רמות ההפשטה, ונערכה השוואה בין מומחים למתחילים.

תיאור ההירארכיה

המחברים מתארים את רמות הידע השונות בהירארכיה תוך כדי התייחסות לארבע פאזות עיקריות במחזור חיים של פיתוח תכנה, כמקובל בהנדסת תוכנה: ניתוח דרישות, ספציפיקציות של המערכת, תיכון, וכתובת קוד. החידוש שלהם, לטענתם, הוא בהוספת שתי רמות הפשטה נוספות (רמה מטרית, רמה קונספטואלית) על המקובל לרוב בניתוחים קודם של חוקרים אחרים [כמובן שחוקרים משתמשים במגוון כינויים וקיימת שונות בין המודלים שהם מציעים לתיאור רמות ידע תכנותי, ב. הברמן].

הרמה הפיזית (physical level):

הרמה הפיזית היא רמת ההפשטה הנמוכה ביותר מכיוון שהיא מתייחסת להבנת צורה פיזית והמופע של אובייקט. הידע התחבירי הנדרש ליצירת קוד כתוצר של פאזת כתיבת קוד במחזור חיים של פיתוח תוכנה, משויך לרמה הפיזית. הבנה של תכניות מחשב ברמת ההפשטה הפיזית מחייבת ידע המתייחס לפורמט התחבירי של קוד התכנית.

הרמה הלוגית (logical level):

הרמה הלוגית מתייחסת למשמעות של אובייקט וחלקיו הפיזיים במובן הלוגי, המסביר את הסיבות לנוכחות הפיזית שלו ושל מרכיביו. הידע הסמנטי הדרוש ליצירת קוד תכנית בפאזה של כתיבת קוד במחזור החיים של פיתוח תכנה משויך לרמת ההפשטה הלוגית. הסמנטיקה של קוד מהווה גורם של הידע התכנותי ברמה הלוגית. למשל, הבנת בקרת זרימה של לולאת while של תכנית (כלומר, ביצוע איטרטיבי של קוד), מחייב ידע תכנותי ברמת הפשטה לוגית.

הרמה הפונקציונלית (functional level):

הרמה הפונקציונלית של הידע התכנותי מתייחסת להבנת מבנים פונקציונליים, מצבים, ותהליכים של אובייקט ושל מרכיביו הפונקציונליים אשר עשויים להיות ממומשים בצורות פיזיות שונות. הידע הדרוש לתיכון מערכת (או תכנית) שייך לרמת ההפשטה הפונקציונלית. תיכון תכנית כולל תיאור של מבני נתונים (כגון מערכים, מחסניות, רשימות, ועוד), אלגוריתמים, ומודולים. הבנת מרכיבי התיכון

של תכנית משויכת לרמה הפונקציונלית של ידע תכנותי. הידע ברמה הפונקציונלית בלתי תלוי בשפת מחשב זו או אחרת, ואכן, מרכיבי התיכון ניתנים למימוש באופנים שונים בשפות תכנות שונות באמצעות קומבינציות של משתנים והוראות.

הרמה הקונספטואלית (conceptual level):

הרמה הקונספטואלית של הידע התכנותי הינה בלתי תלויה בתחום (domain-independent) ומתייחסת להבנת הפונקציונליות המוכללת של אובייקט. ידע הדרוש לאפיון מערכת תכנה משויך לרמת הפשטה הקונספטואלית. תכנית מחשב נתפסת כמערכת עיבוד מידע ברמה הקונספטואלית. היא מקבלת קלט, מבצעת עיבוד, ומייצרת פלט. ואכן, מושגים אלו, הקשורים לתהליך עיבוד מידע, הנם בלתי תלויים בתחום האפליקציה אליו מתייחסת מערכת התכנה.

הרמה המטרית (objective level):

הרמה המטרית היא הרמה המופשטת ביותר של הידע התכנותי. הבנת המטרות והיעדים של אובייקט משויכות לרמת הפשטה הגבוהה ביותר של ידע. זהו הידע הדרוש לנתח דרישות של מערכת. ברמת ידע זו נעזרים על מנת לתאר ספציפיקציות ואילוצי מערכת (למשל, בהתייחס ליעילות ביצוע).

תהליך פיתוח תכנית מחשב הפותרת בעיה נתונה, כרוך למעשה בניצול ידע תכנותי בכל חמשת הרמות.

הניסוי

החוקרים ערכו ניסוי בו השתתפו 10 מומחים ו-10 מתחילים. כל נבחן התבקש לענות על 20 שאלות רב-ברירתיות שהתייחסו לידע תכנותי בחמשת רמות הפשטה המתוארות לעיל (4 שאלות לכל רמה). לנבחנים לא ניתן רמז לגבי ההשתייכות של שאלה לרמת הפשטה. השאלות התייחסו לספציפיקציה נתונה של תכנית.

ממצאי הניסוי העידו שהבדלי ידע בין מומחים למתחילים ברמה מופשטת או ברמה קונקרטית היו תלויים האם ידע מופשט או קונקרטי התבקש כדי לענות על שאלה. כללית, ניתן לומר שהמומחים הפגינו ידע מופשט בצורה טובה יותר ממתחילים. ספציפית, מתברר כי למומחים היה ידע מופשט יותר טוב ברמה הקונספטואלית וברמה הפונקציונלית, אך לא ברמה המטרית (שהיא הרמה הגבוהה ביותר). כמו כן, למומחים היה ידע קונקרטי טוב יותר מאשר למתחילים ברמה הפיסית, אך לא ברמה הלוגית. המחקרים מסתייגים ממובהקות ההבדלים בגלל גודל אוכלוסית המחקר.

גישת "הפשטה-תחילה" להפשטת נתונים ואלגוריתמים

פ. מצ'ניק

Machanik, P. (1998). The abstraction-first approach to data abstraction and algorithms. *Computers & Education*, 31, 135-150.

תקציר

המחבר מציע גישה שונה מהגישה המסורתית המקובלת בתקופתו, להוראת קורס מבני נתונים ואלגוריתמים. הגישה מבוססת על העקרון לפיו מתחילים ללמד הפשטה ושימוש חוזר, ודוחים כתיבת קוד מהיסוד לשלב מאוחר יותר של הקורס. הרעיון הוא ששימוש חוזר בקוד צריך להילמד לפני אסטרטגיות אחרות, וכתיבת קוד מן היסוד רצוי שתיעשה רק לאחר ששימוש חוזר יראה טבעי לתלמידים. המאמר מתאר התנסות בהוראה בגישת הפשטה-תחילה בסביבה מונחית עצמים. לדברי המחבר, הוראה בגישה זו צריכה להיות נתמכת באוסף ספריות הבנויות בעידון הדרגתי של פירוט.

מבוא

מחקרים מראים ששימוש חוזר בקוד לא מתבצע על ידי מתכנתים באופן אוטומטי, וכי דרוש לכך חינוך והכשרה מקדימים מתאימים, במיוחד כאשר המתכנת התנסה במסגרת הכשרתו הפורמאלית בסגנונות תכנות יותר מסורתיים המבוססים של פיתוח קוד מהיסוד.

כבר בתחילת התקופה של הוראת תכנות מובנה, אנשי הוראה התלבטו כיצד לארגן בצורה טובה ביותר את סדר ההוראה כדי להפנים על הצד הטוב ביותר את עקרונות הגישה. בהתחלה הגישה הייתה להציג אבני יסוד של שפת תכנות בסדר מהכלים הבסיסיים ביותר, לכיוון כלים מתקדמים יותר, והיכרות עם הפשטה ככלי להפחתת מורכבות של קוד נדחתה לשלבים מאוחרים יותר. גישה זו השתנתה בהדרגה, ואנשי ההוראה הגיעו להבנה שניתן לפתח הפשטה פרוצדורלית מההתחלה על ידי הקדמת הצגת פונקציות ופרוצדורות בצד שימוש בטיפוסים מוגדרים על ידי המשתמש.

לדביר המחבר, למרות שתכנות מונחה עצמים לא היה רעיון חדש, הכרתו כגישה מקובלת בזרם המרכזי הייתה יחסית חדשה [הערה: מדובר בשנת 1998, ב. הברמן]. כתוצאה מכך, התלבטויות הקשורות לארגון הלמידה ולסדר שימוש בחומרי למידה בקורס היו שכיחות. יתרה מזאת, הרבה ספרי C++ לא הציגו כלל שימוש חוזר בקוד, ומנגנוני הפשטה (classes, templates) הוצגו בשלב מאוחר, או כאמצעי למימוש מהיסוד.

ארגון קורס "הפשטת נתונים ואלגוריתמים" בגישת הפשטה-תחילה

המחבר טוען שאם שימוש חוזר באמת מפשט כתיבת קוד, הרי טבעי ששימוש חוזר ישולב בהתחלה של כל קורס העוסק בתכנות מונחה עצמים, בעוד שטיפול במיומנויות כמו תיכון (design), וכתיבת קוד מן היסוד יוצגו מאוחר יותר.

בהתאם להנחת יסוד זו, המחבר מציג גישת הפשטה-תחילה להוראת קורס מבני נתונים ואלגוריתמים בשפת תכנות C++. המחבר דן במעבר מהוראת הקורס "תכנות מתקדם" בסביבה

פרוצדורלית (Modula-2) להוראת קורס משודרג "הפשטת נתונים ואלגוריתמים" בסביבה מונחית עצמים (C++).

המחבר טוען שהקורס החדש שתכנן תאם את רוח ההמלצות של ACM/IEEE curriculum (אם כי בשינוי סדר הצגת המושגים), ולאכזבתו, לא היה בנמצא באותה תקופה ספר ללימוד C++ שארגון החומר בו תאם המלצות אלו, למרות ששפת C++ תפסה את מקומה בזרם המרכזי באותה תקופה המחבר מתנגד לגישה של "מימוש מוקדם ככל האפשר". לדעתו, לימוד בגישה זו מקשה בטווח ארוך על חשיבה במונחים של הפשטה, עבודה עם קופסאות שחורות, ושימוש חוזר בקוד.

גישת **הפשטה-תחילה** שמציע המחבר מבוססת על כך שתלמידים ירכשו את רעיון ההפשטה בהדרגה, תוך חשיפה מינימאלית והדרגתית לשפת תכנות. המטרה שבסיומ הקורס תלמידים יאמצו את השימוש החוזר כאסטרטגיית ברירת מחדל, אך שגם יהיו להם הכלים לפיתוח קוד מהיסוד במקרה שהדבר מתחייב.

הקו המנחה הוא שבכל שלב של הלימוד מומלץ לחשוף את הסטודנט למינימום פרטים הדרוש להבנת המושג הנלמד, אך לא יותר מכך. ההצדקה לכך היא שלא ניתן לצפות שתלמידים יפנימו את רעיון ההפשטה אם הקורס מוביל אותם לצלול, ללא הבחנה, לים של פריטים.

בהתאם, המחבר מציע ללמד את כלי שפת התכנות בהדרגה ובעדינות, ולדחות הצגת כלי מימוש חדשים עד כמה שאפשר. למעשה הוא גורס ברעיון של שימוש שקוף בקוד קיים לפני מימוש.

לדעת המחבר, כדי להעריך הפשטה ככלי מרכזי לפתרון בעיות אין זה הכרחי לדעת תכנות. בהתאם, הוא מציע להציג את עקרון ההפשטה תחילה באמצעות דוגמאות מהעולם האמיתי, לפני שבכלל מדברים על מחשבים, ואחר כך דרך מכונות וירטואליות, כגון הצגת תיכון מוצלח של ממשק משתמש. הפשטה בתכנות תוצג באמצעות המחשה שאפשר להשתמש בקוד בלי לדעת כיצד הוא ממומש. למעשה, התלמידים עורכים תחילה יותר היכרות עם ממשקים מאשר עם מימוש.

גישת מונחית-עצמים מוצגת תחילה כבלתי תלויה בשפת תכנות תוך שימוש בדיאגרמות Booch (Booch, 1991).

כתיבת קוד ומימוש: ההיכרות עם קוד מונחה-עצמים תיעשה בהקשר של בניית תוכניות באמצעות בלוקים כתובים מראש של קוד (המשמשים כאבני בניין) במקום הגישה המסורתית של פיתוח מן היסוד. רק כאשר הרעיון של מחלקות ועצמים מבוסס היטב, התלמידים יכירו ספריות ומסגרות. מומלץ להיעזר בשלב זה ב"צעצועים" כלי עזר ייעודיים להוראה (toy libraries and frameworks). בשלב הבא מוצגים templates להעברת הרעיון של "טיפוסים לא שלמים" (incomplete types). בשלבים המוקדמים של הקורס התלמידים נדרשים לכתובה מעטה בלבד של קוד, אולם מצפים מהם להוכיח בחירה מוצלחת של מחלקות ו-templates לצורך שימוש חוזר.

המחבר ממליץ להציג לראשונה מימוש מפורט בהקשר של מימוש חלק קטן של תכנית שברובה כבר כתובה, ובאופן זה להדגים את הערך של הפשטה כמגוונת על המתכנת מפני התמודדות בו-זמנית עם מורכבות קוד של תכנית גדולה. תוך כדי התקדמות לטיפול בדוגמאות של תכניות גדולות יותר, מומלץ להדגים ולעודד אסטרטגיות של שימוש חוזר, והסתרת חלקים אחרים של תכנית גדולה עוד יותר.

המחבר ממליץ להציג תיכון על ידי הצגת חלק מממנו, והצגת מימוש של מרכיב קטן מהתיכון, כאשר ידועים הממשקים ליתר התיכון.

רק כאשר ביסוס של רעיון הפשטה בכללותו הושלם, ממשיכים לכיוון של ניתוח אלגוריתמים, ובחירת מבני נתונים, תוך שזירה עם תיכון ושימוש חוזר בקוד, תוך הדגשת העיקרון: פיתוח קוד מן היסוד ייעשה רק עם מודעות לשימוש חוזר.

גישת הפשטה-תחילה לעומת גישת מלמעלה-למטה (top-down)

לטענת המחבר, גישת הפשטה-תחילה שונה במהותה מגישת מלמעלה-למטה (top-down). גישת מלמעלה-למטה, מנסה באופן מלאכותי (לדעתו של המחבר) לענות על הדרישה של דחיית טיפול בפרטים לשלב מאוחר יותר. אולם, התפיסה המקובלת של גישה זו דורשת מהתלמידים להיות מראש בעלי יכולות הפשטה טובות מכיוון שהם נדרשים להיות מסוגלים לפרק בעיה לתת משימות, ולתכנן הפשטות מתאימות.

גישת הפשטה-תחילה שונה בכך שלתלמידים מציגים הפשטות קיימות אשר משמשות כבלוקים לבניית תוצר, והמשימה הקשה של תכנון הפשטות על ידי התלמידים נדחית לשלב מאוחר יותר, כאשר עקרון הפשטה מוטמעים באופן משמעותי יותר.

לדעת המחבר, רק כאשר עקרונות של הפשטה ושימוש חוזר הוטמעו, זה הזמן להציג לתלמידים מבני נתונים "קונבנציונאליים" ואלגוריתמים, כאמצעי למימוש של הפשטות המתוכננות על ידי התלמיד.

סיכום

המחבר טוען שהגישה הפדגוגית שהוא מציע להטמעת רעיון הפשטה ניתנת ליישום בכל שפת תכנות (אם כי יש שפות התומכות יותר, ואלו התומכות פחות ביישום הגישה). ללא תלות בשפה הנלמדת בקורס, העקרונות המרכזיים צריכים להישמר:

- מושגים צריכים להיות מוצגים בסדר אשר מציג, מחזק, ומשריש הרגל של שימוש חוזר כבר מההתחלה.
- פיתוח מהיסוד צריך להיות כפס ככולת מתקדמת יותר, הנלמדת רק לאחר הבנת שימוש חוזר ותיכון בהקשר של הרעיון – "אתה יכול לתכנן לצורך שימוש חוזר".
- מומלץ לפתח חומרי למידה עבור הקורס אשר תומכים באסטרטגיית הפשטה-תחילה. למשל, דרושות ספריות מקיפות הכוללות פונקציונליות מספיקה בצד כלים לבניית אפליקציות, המאפשרות להציג תחום נרחב של מושגים מונחי-עצמים לפני הצגה מרובה של מאפייני תחביר של השפה.

האם אנו באמת מלמדים הפשטה?

פ. בוצ'י, ת.ג'. לונג, ב.ג'. ווייד

Bucci, P., Long, T. J., Weide, B. J. (2001). Do we really teach abstraction?
Proceedings of the 31th SIGCSE Technical Symposium on CS Education, 26-30.

תקציר

המחברים מציגים את הדילמה הבאה: הפשטה היא בסיס מרכזי בפיתוח תוכנה, ומזוהה כעקרון בסיסי והכרחי שיש ללמדו מוקדם ככל האפשר בקורסי מבוא במדעי המחשב (CS1/CS2). אנשי חינוך מאמינים שהפשטה עשויה לשפר יכולת הסקה וחשיבה. יחד עם זאת לעתים קרובות שומעים על קשיים של סטודנטים לתואר ראשון במדעי המחשב לבטא מיומנויות אלו בהקשר של הבנת התנהגות תוכנה. השאלה הנשאלת היא האם אנו אכן מספקים לסטודנטים את הכלים הדרושים להם למצוא את הפוטנציאל שלהם לחשיבה זהירה וקפדנית, הבנה, והסקה מושכלת אודות התנהגות תוכנה? האם אנו באמת מלמדים הפשטה?

המחברים מנתחים גישות מקובלות להוראת הפשטה בהקשר של עיבוד רשימות, ומציגים גישה חלופית התומכת לטענתם בהפנמת הפשטה כמושג המעצים יכולת חשיבה והסקה.

מבוא

הסתרת מידע והפשטה הינם היבטים משלימים של רעיון כללי: הסתרת פרטים של מערכת מורכבת תוך כדי יצירת "סיפור כיסוי" להסברת אותם היבטים שיש עניין בלהתמקד בהם. (Norman 1993) מייחס חשיבות רבה להפשטה מוצלחת אשר מציגה את האספקטים החשובים ומסתירה את הבלתי חשובים (בהקשר המדובר). לדבריו, הפשטה מוצלחת מאפשרת להתרכז בפרטים חיוניים ולהתעלם מאלו שאינם חיוניים, ובכך מונח היסוד להעצמת חשיבה ביקורתית ויכולת הסקה. הפשטה מוצלחת מאפשרת התבוננות חדשה על המערכת אותה היא מייצגת, ומאפשרת התייחסות להתנהגות חיצונית של המערכת במונחים השונים מאלו הדרושים לניתוח ההתנהגות תוך התייחסות לפרטים פנימיים. הטענה המרכזית של מחברי המאמר היא שעקרונות הסתרת מידע והפשטה אינם מטופלים כהלכה בהוראת קורסי מבוא במדעי המחשב (CS1/CS2), ואינם מובילים להעצמת יכולות חשיבה והסקה כפועל יוצא של הפשטה. הסיבה לכך היא אי-שימת דגש על היבטים "אנושיים" של הסתרת מידע, ותמיכה בלתי מספקת בסטודנטים לעידודם לזהות הזדמנויות של חשיבה ביקורתית והסקה בזמן שהם עוסקים בהפשטה.

הצגה מסורתית של רשימות

המחברים דנים בהצגת מודל מסורתי מופשט של רשימה הכרוך בהסתרת פרטים של מערכת מורכבת תוך כדי יצירת "סיפור כיסוי" להסברת אותם היבטים שיש עניין בלהתמקד בהם.

סיפור הכיסוי מתייחס תחילה לתיאור של טיפוס הנתונים רשימה ותיאור ערכים שאיברי הרשימה יכולים לקבל. למשל, התיאור המוצג במאמר: "רשימה היא סדרה ליניארית של איברים, ומצוין על מיקום נוכחי בה". המשך "סיפור הכיסוי" מתייחס לתיאור אוסף פעולות אפשריות על רשימה, והתנהגותן (תוצאת הפעלתן). למשל, "הפעולה $Insert(l, x)$ מכניסה את איבר x לתוך רשימה l במקום הנוכחי." סיפור הכיסוי יכול להכיל טענות המתייחסות לתוצאות הפעלת הפעולה.

ברוב ספרי הלימוד סיפור הכיסוי מלווה (מיד בהמשכו) בדוגמה ממחישה, ומיד בסמוך מוצגים מספר מימושים של רשימה. מימוש שכיח מתייחס לרשימה מקושרת חד-כיוונית והוא מלווה לצורך המחשה בייצוג גרפי הכולל חוליות ומצביעים.

לטענת מחברי המאמר, ייצוג כזה אינו מהווה הפשטה טובה של מושג הרשימה. לראייה, סטודנטים המתבקשים למלא פרטים בטבלת מעקב המתייחסת להכנסת איבר לרשימה, יגלו שונות בצורת ייצוג מצב אקטואלי של רשימה וחלקם ייצגו אותה במונחים של הייצוג הגרפי, במקום לרשום בפשטות את סדרת ערכי איברי הרשימה. אלו שמתייחסים רק לתיאור המילולי בסיפור הכיסוי, ולא משתמשים בתיאור הגרפי, עשויים גם לבטא שונות בצורת הייצוג בטבלת המעקב משתי סיבות: (א) לא הוסכם על סימון מקובל פורמאלי לייצוג "ערך" רשימה; (ב) לא ברור האם ואיך לייצג את "המקום הנוכחי".

בהתאם, המחברים טוענים שהייצוג הגרפי, ללא תוספת של סימון פורמאלי מופשט יותר, המתעלם מרמיזות על אופן מימוש [מצביעים; ב. הברמן] אינו תומך ברעיון ההפשטה כהלכה, וכתוצאה מכך אינו יכול להעצים את יכולות החשיבה הביקורתית.

נשאלת השאלה על ידי המחברים מדוע יש חשיבות לשימוש בסימון פורמאלי אחיד. לטענתם הדבר חיוני לתקשורת בין אנשים – דיבור ב"אותה שפה", הימנעות משימוש מוטעה באינטואיציות, חשיבה וניתוח של התנהגות פעולות מורכבות.

הצגה משופרת של רשימות

המחברים מציעים הצגה משופרת של רשימות התומכת יותר בהפשטה והמבוססת על שימוש במודל מתמטי מפורש של רשימה לייצוג רשימה במדעי המחשב. הם משתמשים בשפה RESOLVE למידול מתמטי של טיפוס נתונים מופשטים (כדוגמת רשימה) והפעולות המוגדרות עליהם. לטענתם, תוספת של סימון אחיד לכלי מידול אלו, מאפשרים תיאור פשוט ומדויק של טיפוס נתונים מופשטים, ומהווים מנגנון מצוין ליצירת סיפורי כיסוי, התומך ביצירת מודל מנטלי נכון של מושגים.

על פי הצעת המחברים, רשימה תסומן באמצעות $\langle \dots \rangle$. בהתאם לסיפור הכיסוי, ייצוג של המקום הפנוי יהיה באמצעות הצגת רשימה כצמד רשימות, כאשר הרשימה המובילה מכילה את איברי הסדרה עד המקום הנוכחי (לא כולל), והרשימה השנייה מכילה את איברי יתר הרשימה החל מהמקום הפנוי.

לדוגמה, רשימה נתונה: $\langle 1,4,7,8 \rangle$, $\langle 5,3,4 \rangle$

כתוצאה מהוספת האיבר 2 במקום הנוכחי תתקבל הרשימה: $\langle 2,1,4,7,8 \rangle$, $\langle 5,3,4 \rangle$.

התיאור הפורמלי המופשט נעשה בשפה RESOLVE
באמצעות מידול מתמטי של רשימה והפעולות המוגדרות עליה
(Bucci, et al., 2001, p. 28)

concept List_Template (type Item)

type List is modeled by

(left: string of Item,

right: string of Item)

exemplar s

initialization ensures

|s.left| = 0 and |s.right| = 0

operation Insert (updates s: List,

clears x: Item)

ensures s.left = #s.left and

s.right = <#x> * #s.right

לטענת המחברים, המודל המתמטי רשימה מאפשר תיאור מדויק של ערכי אברי הרשימה ומאפשר כתיבת ספציפיקציות פורמאליות מדויקות להתנהגות הפעולות המוגדרות עליה, ללא קשר למימוש. באופן זה "הכוח האמיתי של הפשטה" יכול להיות מובא למלוא מיצוי הפוטנציאל שלו.

מחקר

המחברים מציעים שימוש במידול מתמטי וספציפיקציות פורמליות להצגה מופשטת של טיפוסים נתונים. למרות שלכאורה נדמה כי סטודנטים מתחילים אינם בשלים לגישה המבוססת על כלים אלו, ניסיונם של המחברים מראה שהדבר אפשרי. יתרה מזאת, גישה זו מקלה על סטודנטים להבין התנהגות של תכניות. המחברים ערכו ניסוי לסטודנטים שהתנסו במשך 5 שבועות במידול מתמטי וכתיבת ספציפיקציות פורמליות כמתואר לעיל, ואחר כך עסקו 5 שבועות במימוש "רגיל". הם ביקשו אותם "לצייר תמונה של ערך" של תור, קבוצה, רשימה ועוד, בהינתן קטעי קוד שונים למימוש טיפוסים אלו בשפת תכנות "רגילה". למרבה ההפתעה, מרבית הסטודנטים השתמשו ב"שפת המידול המתמטי" ל"ציור ערך" של אובייקט מטיפוס מסוים, תוך התעלמות מאופן המימוש. ממצאים אלו מעידים על מודל מנטלי נכון ומוצק של טיפוסים נתונים מופשטים שאינו מבוסס מימוש.

הפשטה היא מפתח לקורס המבוא CS1

פ. ספרג, ס. סאשצ'נסקי

Sprague, P. Schahczenski, C. (2001). Abstraction the key to CS1. *Journal of Computing in Small Colleges*, 17(3), 211-218.

תקציר

קיימת הסכמה שקורס מבוא בתכנות לסטודנטים בשנה ראשונה צריך לשים דגש על יותר על פתרון בעיות מאשר על תכנות גרידא. המחברים טוענים שניתן ללמד פתרון בעיות טוב יותר בפרדיגמה מונחית עצמים לעומת הפרדיגמה הפרוצדורלית המסורתית. לטענתם, הפשטה היא המפתח לפתרון בעיות, וכי אוריינטציה מונחית עצמים מכוונת, ואפילו מאלצת לרמת הפשטה גבוהה יותר מאשר הפרדיגמה הפרוצדורלית. המחברים מציעים ארגון של קורס בגישת "עצמים תחילה" אשר מחזק את הטמעת עקרון ההפשטה.

השוואה בין הפשטה בפרדיגמה פרוצדורלית ופרדיגמה מונחית עצמים

המחברים מסכימים עם הגישה שהפשטה היא מושג מרכזי והכרחי בהוראת מדעי המחשב. הם תומכים בגישה המתפתחת של שינוי פרדיגמה בהוראת קורס מבוא בתכנות לסטודנטים בשנה ראשונה, מפרדיגמה פרוצדורלית לפרדיגמה מונחית עצמים. לטענתם, פרדיגמה מונחית עצמים מאלצת לרמת הפשטה גבוהה יותר, ותומכת בהקניית מיומנויות של פתרון בעיות "בעולם האמיתי" כגון הפשטה, הכמסה (encapsulation) ותיכון מודולרי.

תכניות מובנות בפרדיגמה פרוצדורלית מחולקות ליחידות פונקציונליות, העשויות בעצמן להיות מחולקות ליחידות פונקציונליות. חלוקה זו מאפשרת למתכנת לתפוס את התכנית כולה כהרכב של מספר מוגבל של יחידות מופשטות. זאת בניגוד לתכניות בלתי מובנות אשר חסרות רמות הפשטה, שאינן מפרידות את התכנית ממשפטי תכנית בסיסיים. עם כל היתרון בתכניות הבנויות מודולרית ובמספר רמות הפשטה, בפרדיגמה פרוצדורלית יש לחוד הפשטת פעולות ולחוד הפשטת נתונים. הפשטת נתונים מבוססת על טיפוסים נתונים "טבעיים" של השפה, וטיפוסים המוגדרים באמצעותם. לעומת זאת, גישה מונחית עצמים מאפשרת למתכנתים ליצור הפשטות המבוססות על "דברים" שהם שילוב של נתונים ופעולות. בניית מרכיבים משולבים בדרך זו הקושרים נתונים ופעולות היא דרך ליצירת הפשטות תוכנה בעלות עצמה, ומאפשרת מידול מופשט הולם של העולם האמיתי אליו התוכנה מתייחסת, זאת משום שהפונקציונליות של "עצם תוכנה" משקפת את הפונקציונליות של העצם ה"פיסי" בעולם האמיתי, המייצג על ידו. יתרה מזאת, התקשורת בין עצמים מהווה בסיס למימד נוסף של הפשטה.

המחברים מציעים שלשה יתרונות להטמעת עקרון ההפשטה בפרדיגמה מונחית עצמים:

1. הם טוענים שהרבה אפליקציות מיועדות למשתמש כדוגמת מעבד תמלילים, גיליון אלקטרוני, משחקי מחשב, בנויים כך שהם משקפים גישה מונחית עצמים, וההתנסות בשימוש בהם מכשירה את הסטודנט המתחיל להבנה טבעית יותר של פרדיגמה מונחית עצמים, ובפרט את עקרון ההפשטה הגלום בה.

2. הבנת זרימת מידע במערכת מונחית עצמים יותר אינטואיטיבית מאשר בפרדיגמה פרוצדורלית, מכיוון שהיא מתנהלת באופן דומה לתקשורת בעולם האמיתי. לכן, סטודנטים יכולים להשתמש בהבנתם את ההתנהלות בעולם האמיתי ולהשליך זאת על אופן פיתוח תכניות בסביבה מונחית עצמים.
3. הפרדיגמה מונחית העצמים מקלה להטמיע אצל סטודנט מתחיל את ההבנה שתיכון מוקדם חיוני, ומקנה יכולות להעריך מתי תיכון טוב יותר מתיכון אחר.
4. האפשרות להשתמש בספריות מיד בתחילת הלימוד, ממחישה את השימוש בהפשטה, ומראה כיצד ניתן להרכיב תכנית גדולה יותר באמצעות הפשטות מוגדרות מראש. בנוסף, היא תומכת ברעיון של לימוד באמצעות קריאת קוד לפני לימוד באמצעות כתיבת קוד. קריאה של ספריות של מחלקות נותנת לסטודנטים דוגמאות של הפשטות שנוצרו על ידי מחברי הספריות, ועשויה לאפשר הצצה לתוך שיקולי התיכון של המחברים.

הוראה ממוקדת הפשטה לעומת הוראה ממוקדת תחביר

המחברים גורסים בדעה ששפת תכנות היא אמצעי להוראת תכנות במובן שהיא מאפשרת יישום ותרגול וקבלת משוב ישיר ומידי מ"המכונה". יחד עם זאת, הכרת שפת התכנות אינה יכולה להיות יעד מרכזי. בחירת שפת Pascal להוראה בסוף שנות ה-70 ותחילת שנות ה-80 ממחישה את הגישה שהשפה צריכה להיות מתאימה מבחינת המאפיינים הפדגוגיים להעברת עקרונות, ואין לבחור בשפה רק בגלל שהיא מקובלת בתעשייה. בדומה, בחירת Java להוראת CS1 נובעת בגלל יתרונותיה ההוראתיים וללא קשר לשימוש בתעשייה.

אחד העקרונות החשובים כאמור שצריכים להילמד הוא עקרון ההפשטה. על כן, לדעת המחברים, הוראת CS1 צריכה להתמקד על הפשטה ולא על תחביר, במטרה למנוע תפיסות שגויות אופייניות ושכיחות של סטודנטים במדעי המחשב ובתכנות. המחברים טוענים שהוראה בפרדיגמה פרוצדורלית מתמקדת לרוב יותר על מבני תחביר, ובכך שלא במתכוון מדגישה היבטים פחות רלוונטיים של מדעי המחשב, דבר העלול ליצור לסטודנטים מכשולים. לדעת המחברים, גישה מונחית עצמים יכולה לצמצם את נקודות התורפה הבאות הנובעות מהוראה תלוית תחביר:

- הפנמה וזיכרון של מבנים תחביריים על חשבון הבנה כיצד מבנים אלו פותרים, או יכולים לסייע לפתרון בעיה.
- תלות בשפה – סטודנטים מסווגים ותולים יכולות של פתרון בעיות במאפייני שפה, וקשה להם לזהות ולהעריך יכולות פתרון בעיות שאינן תלויות שפה.
- התיישנות כישורים – לשפות תכנות כ"תוצרים טכניים" יש מחזור חיים, והן עלולות להתיישן ולהפסיק להיות רלוונטיות. לכן הוראה מבוססת תחביר עלולה להזיק ולגרום לסטודנטים להזדקק להכשרה מחדש מיותרת כתלות בהשתנות שפות תכנות.
- דגש מקצועי – הוראה מבוססת תחביר ממוקדת יותר להכשרה מקצועית מיידית וקצרת טווח, ופחות עם אוריינטציה חינוכית.

עקרונות מנחים להוראת הפשטה בפרדיגמה מונחית עצמים

המחברים ממליצים להקפיד על העקרונות הבאים כדי לחזק את הפנמת עקרון ההפשטה:

- הוראת תיכון לפני מימוש; המחשת הקשר בין תיכון וקוד.
- היכרות עם מחזור חיים של פיתוח תכנה באופן שיעורר מוטיבציה בשימוש בעצמים.
- שימוש מושכל בתכניות לדוגמא גדולות להמחשת: תיכון, ארגון מורכבות של מערכת, יתרונות של שימוש חוזר.
- שימוש מושכל בתכניות פשוטות הממחישות, למרות גודלן, יתרונות של גישה מונחית עצמים, ובפרט את עקרון ההפשטה.

הפשטה, תיכון ותיאוריה במדעי המחשב: קורס כללי

ג'. ד. קיפר, ק. בישוף-קלרק

Kipper, J. D. Bishop-Clark, C. (1995). Abstraction, Design, and Theory in Computer Science: A Liberal Art Course. *Computer Science Education*, 6, 93-110.

תקציר

הטענה המרכזית של מאמר זה היא שכל בוגר לתואר ראשון בכל מסלול לימודים (שאינו מדעי המחשב) יכול להבין את מהות התיאוריות בבסיס מדעי המחשב ואת הרלוונטיות שלהן לטכנולוגיות חישוב. חשוב שקורס מבוא מתאים במדעי המחשב יוכר כקורס כללי הנלמד במסלולים שונים, ובתור שכזה, יתרום לקירוב סטודנטים למושגים התיאורטיים בתשתית טכנולוגיות חישוב, ובכך יפחית את הפחד שלהם משימוש בהן. אחד העקרונות המומלצים להילמד בקורס במתכונת זו הוא עקרון ההפשטה.

מבוא

קיימת שונות בגישה להבנה ושימוש בטכנולוגיות חישוב, התלויה לדעת המחברים במידת ההיכרות עם עקרונות מדעי המחשב. המחברים טוענים שרוב הקורסים המוצעים לתלמידים במסלול שאינו מדעי המחשב, ושמתרתם לקרב אותם להבנת טכנולוגיות חישוב, שמים דגש על כישורים טכניים והכרת סביבות בלבד. לדעת המחברים הדבר נובע מכך שקיימות שתי מיסוקונספציות שכיחות הקשורות לפדגוגיה של הוראת מדעי המחשב:

- הוראת כישורי תכנות עוזרת לסטודנטים להבין מדעי המחשב וטכנולוגיות מחשב.
- יותר מדי רקע דרוש להקנות לסטודנטים לתואר שאינו במדעי המחשב כדי שיוכלו להעריך את העקרונות המרכזיים במדעי המחשב.

המיסוקונספציה הראשונה מופרכת על ידי גישות של מומחים המתארים את מהות מדעי המחשב, לדוגמה, דו"ח ועדת (Denning et al., 1989) Computing as a Discipline – Denning. הדו"ח מציג תשעה תחומים מרכזיים המרכיבים את הדיסציפלינה, ובנוסף, שלושה צירים אורטוגונליים: **הפשטה**, **תיכון**, ו**תיאוריה**, הבאים לידי ביטוי במידה זו או אחרת בכל אחד מהמרכיבים. המיסוקונספציה השנייה ניתנת להפרכה על ידי דוגמאות עובדות מהשטח. המחברים מציגים במאמר דוגמא לקורס מבוא מתאים להקניית רקע בסיסי של עקרונות מדעי המחשב לאוכלוסיה כללית.

תיאור הקורס המוצע

הקורס המוצע מציג מגוון תכנים בגישה של הוראה לרוחב, ומשלב עקרונות תיאורטיים עם המחשבות קונקרטייות. ככזה, הוא מבוסס על שיעורי הרצאה מסורתיים הכוללים דיונים על ההיבטים התיאורטיים, בצד עבודה בסביבת מעבדה ליישום העקרונות התיאורטיים המופשטים והמחשתם [גישה דומה מומלצת להפעלת תכנית הלימודים במדעי המחשב לתיכון בארץ, והיא מכונה על ידי

ועדת המקצוע כשיטת הרוכסן המשלבת לימוד עקרונות עם יישום ותרגול במעבדה (Gal-Ezer et al., 1995).

הקורס שם דגש על הפשטה, תיכון, ותיאוריה. אחת המטרות של הקורס הייתה עיסוק בפוטנציאל החישוב ובמגבלות החישוב.

תכנים תיאורטיים של הקורס:

- טכנולוגיות חישוב ותפיסות עולם
- פתרון בעיות
 - אלגוריתמים, שיטות: תיכון מלמעלה-למטה, הפרד ומשול.
- הפשטת מכונה (מודלים חישוביים):
 - מכונת טיורינג, חישוביות, מקביליות.
- סימבוליות וייצוג ידע
 - נתונים, מבני נתונים, קבצים, מערכות בסיס נתונים.
- הנדסת תכנה
 - ניתוח מערכות: הגדרת ואפיון דרישות
 - סינטיזה של מערכות: תיכון, תכנות.
- אתיקה
- היבטים חברתיים של טכנולוגיות חישוב והשפעתן על החברה
- מחשבים כמכונות חושבות
 - מטרות בינה מלאכותית
 - ייצוג ידע
 - למידה והסקה
 - מערכות מומחות

תכנים יישומיים של הקורס:

- ארכיטקטורת מחשבים
- ממשק אדם
- חבילות תוכנה שימושיות
- תכנות פרוצדורלי
- טיפוסים נתונים מופשטים
- תכנות לא פרוצדורלי

המחברים ממליצים שאופן ההוראה, המשימות המוטלות על הסטודנטים, והשילוב בין המרכיבים התיאורטיים והיישומיים של הקורס, צריכים להיות מוכוונים כך שיהיה דגש לאורך כל הדרך על שלושת הפרדיגמות המרכזיות של מדעי המחשב: הפשטה, תיכון ותיאוריה (לפי Denning et al., 1989). לדוגמה, אחת הדרכים לפיתוח חשיבה מופשטת היא להימנע ממעבדות במתכונת "ספר בישול", המכוונות צעד אחר צעד את התלמיד, אלא לבסס אותן על פעילויות חקר.

בנוסף, תהליכי ההוראה/למידה, כמומלץ על ידי המחברים, צריכים לעודד הטמעת עקרונות וכישורים המשקפים מטרות של השכלה כללית: חשיבה ביקורתית, למידה שיתופית, הבנת הקשרים (קונטקסט), רפלקטיביות ותגובתיות.

טענת המחברים היא שקורס במתכונת זו המשלב פרדיגמות מרכזיות של מדעי המחשב: הפשטה, תיכון ותיאוריה, עם פיתוח מיומנויות כלליות של חשיבה ביקורתית, למידה שיתופית, הבנת הקשרים (קונטקסט), רפלקטיביות ותגובתיות, עדיף על קורסים ייעודיים המכוונים לרכישת כישורים טכניים גרידא. דווקא קרוס הבנוי לפי המתכונת המוצעת מראה את הקשר בין העקרונות המופשטים והתיאורטיים בבסיס מדעי המחשב לבין טכנולוגיות חישוב, מקרב את הסטודנט להעריך טכנולוגיות אלו ולהשתמש בהן בתבונה.

עמדות תלמידי תיכון כלפי הפשטה פרוצדורלית

ב. הברמן

Haberman, B. (2004). High-school students' attitudes regarding procedural abstraction. *Education and Information Technologies*, 9(2), 131-145.

תקציר

המאמר מציג מחקר שמטרתו להעריך עמדות תלמידי תיכון כלפי הפשטה פרוצדורלית. אוכלוסית המחקר הייתה תלמידי תיכון שלמדו את יחידת הלימוד עיצוב תכנה (אליהם נתייחס כמתחילים), וקבוצת ביקורת של סטודנטים לתואר ראשון במדעי המחשב (אליהם נתייחס כמתקדמים). הממצאים מעידים על הבדל מובהק בין עמדות שתי הקבוצות. נמצא כי המתקדמים העדיפו אלגוריתמים כתובים ברמת הפשטה גבוהה; יחד עם זאת, הם נתקלו במחסום קוגניטיבי כאשר רמת הפשטה הייתה גבוהה במיוחד. המתחילים העדיפו על פי רוב אלגוריתמים הכתובים ברמת הפשטה נמוכה, אך גילו פתיחות למידה מסוימת של הפשטה. ממצאי המחקר שימשו לפיתוח כלים תומכים לפיתוח אלגוריתמים העושים שימוש בהפשטה פרוצדורלית, כמתואר ב (Haberman, 2002).

מבוא

הפשטה היא מושג מרכזי במדעי המחשב (Denning et al., 1989). Aho & Ullman (1995) מתארים את מדעי המחשב כ"מדע של הפשטה- יצירת מודל נכון לחשיבה על בעיה ובחירת טכניקות מתאימות לפתרון". במדעי המחשב מקובל להתייחס להפשטת נתונים והפשטה פרוצדורלית (Dale & Walker, 1996). אנשי חינוך העוסקים במחקר בהוראת מדעי המחשב ובפיתוח קווים מנחים לתכניות לימודים בתחום, סבורים שיש לשלב הפשטה כמושג מרכזי בהוראה. תכנית הלימודים החדשה במדעי המחשב לתיכון בישראל שהטמעתה החלה לפני קצת למעלה מעשור, מדגישה עקרונות ומושגים שאינם תלויי מימוש, בצד יישום פרקטי שלהם בשפות תכנות קונקרטיות ובפרדיגמות תכנות שונות (Gal-Ezer, Yehudai, Harel & Beeri, 1995, Gal-Ezer & Harel 1999). הכרות עם מושג ההפשטה נערכת בהדרגה לאורך יחידות הלימוד, החל מ"סודות מדעי המחשב", דרך יחידות המציגות פרדיגמות תכנות שונות כדוגמת "תכנות לוגי" או "תכנות פונקציונלי", היחידה "עיצוב תוכנה" שאחת ממטרותיה המרכזיות היא פיתוח חשיבה מופשטת ופיתוח יכולות של פתרון בעיות דרך שימוש בטיפוסי נתונים מופשטים (טנ"מ).

פתרון בעיות תוך שימוש בטיפוסי נתונים מופשטים

דוגמה

סעיף זה מציג בעיה לדוגמה בנושא "עץ בינרי" וארבעה פתרונות חלופיים המאופיינים על ידי רמות הפשטה שונות. הבעיה והפתרונות (חלקם בשינוי קל) הוצגו לקבוצת המחקר ולקבוצת הביקורת במטרה להעריך עמדות כלפי הפשטה פרוצדורלית.

הבעיה:

פתח אלגוריתם המחזיר את מספר הצמתים של עץ בינרי T , ישיש להם בן יחד. הנחה: T מאותחל.

הרעיון לפתרון הוא: אם העץ ריק, הערך המוחזר הוא 0; אחרת הערך המוחזר אמור להיות הסכום של: (א) מספר הצמתים הכולל בעלי בן יחיד בשני תתי העצים, ו-(ב) הערך הנתרם על ידי השורש (1) – כאשר, ומאופיינים על ידי רמות הפשטה העולות בהדרגה לשורש יש בן יחיד; אחרת – 0). ארבעת הפתרונות המוצעים כאן נכונים ומבוססים על רעיון זה, ושונים (מאפיינים שונים של הפתרונות מוצגים בטבלה 1).

פתרון 1:

הפתרון מכיל הרבה פרטים המיוצגים ברמת הפשטה אחת. בנוסף, הוא מכיל קוד מיותר (בדיקה האם לשורש יש בן יחיד מתבצעת באופן מפורש בשורה 2.1).

Num-of-Nodes(T)

1. if **Is_Empty** ? (T) then return 0
2. else
 - 2.1 if ((**Is_Empty**? (**Left_Sub_Tree** (T) and not **Is_Empty**? (**Right_Sub_Tree** (T))) or ((**Is_Empty**? (**Right_Sub_Tree** (T) and not **Is_Empty**? (**Left_Sub_Tree** (T)))) then
 - 2.1.1 return

$$1 + \text{Num-of-Nodes}(\text{Left_Sub_Tree}(T)) + \text{Num-of-Nodes}(\text{Right_Sub_Tree}(T))$$
 - 2.2 else return $\text{Num-of-Nodes}(\text{Left_Sub_Tree}(T)) + \text{Num-of-Nodes}(\text{Right_Sub_Tree}(T))$

פתרון 2:

הפתרון מאורגן בשתי רמות הפשטה. הוא מבוסס על חלוקת הבעיה לשתי תת משימות: (א) הגדרת פעולה חדשה מופשטת (*abstract primitive*) (Leron, 1987) : **Has_Single_Child** (T) ? הממומשת במונחים של ממשק "עץ בינרי" באלגוריתם נפרד; ו-(ב) שימוש בפעולה זו באלגוריתם הראשי לחישוב הערך המוחזר הדרוש. לפתרון זה שני יתרונות: הוא משקף שימוש בהפשטה פרוצדורלית להפחתת מורכבות כתיבת קוד, כיוון שהוא מאפשר לפותר הבעיה להתרכז במשימה אחת בלבד בכל שלב, כמו כן, הגדרת פעולה חדשה מהווה בסיס לשימוש חוזר בה לפתרון בעיות אחרות. עדיין, בדומה לפתרון 1, פתרון זה מכיל קוד מיותר.

Num-of-Nodes(T)

1. if **Is_Empty** ? (T) then return 0
2. else
 - 2.1 if **Has_Single_Child** ? (T) then
 - 2.1.1 return

$$1 + \text{Num-of-Nodes}(\text{Left_Sub_Tree}(T)) + \text{Num-of-Nodes}(\text{Right_Sub_Tree}(T))$$
 - 2.2 else return $\text{Num-of-Nodes}(\text{Left_Sub_Tree}(T)) + \text{Num-of-Nodes}(\text{Left_Sub_Tree}(T))$

Has_Single_Child ? (T)

1. if ((**Is_Empty**? (**Left_Sub_Tree** (T) and not **Is_Empty**? (**Right_Sub_Tree** (T))) or ((**Is_Empty**? (**Right_Sub_Tree** (T) and not **Is_Empty**? (**Right_Sub_Tree** (T)))) then
 - 1.1 return True
2. else return False

פתרון 3:

פתרון זה מאורגן בשלוש רמות הפשטה. האלגוריתם הראשי מזמן פעולה $\text{Compute_Node}(T)$ אשר מזמנת את הפעולה $\text{Has_Single_Child?}(T)$ ובנוסף מחזירה ערך שיש להוסיפו למספר הכולל של צמתים בעלי בן יחיד בשני תתי העצים, לחישוב הערך הדרוש. פתרון זה מאופיין על ידי פישוט קוד בכל רמת הפשטה, ובניגוד לקודמיו, אינו מכיל כפילויות. לפתרון מבנה כללי המאפשר התאמה פשוטה לפתרון בעיות אחרות.

Num-of-Nodes (T)

1. if $\text{Is_Empty?}(T)$ then return 0
2. else
 - 2.1 return $\text{Compute_Node}(T) + \text{Num-of-Nodes}(\text{Left_Sub_Tree}(T)) + \text{Num-of-Nodes}(\text{Right_Sub_Tree}(T))$

Compute_Node (T)

1. if $\text{Has_Single_Child?}(T)$ then
 - 2.2 return 1
2. else return 0

Has_Single_Child? (T)

1. if (($\text{Is_Empty?}(\text{Left_Sub_Tree}(T))$ and not $\text{Is_Empty?}(\text{Right_Sub_Tree}(T))$) or ($\text{Is_Empty?}(\text{Right_Sub_Tree}(T))$ and not $\text{Is_Empty?}(\text{Left_Sub_Tree}(T))$)) then
 - 1.2 return True
2. else return False

פתרון 4:

זהו הפתרון המופשט ביותר, מאורגן בארבע רמות הפשטה- כל רמה מהווה מימוש של תת משימה אחת. האלגוריתם הראשי מזמן פעולה $\text{Compute}(T, L, R)$ תוך שימוש בקריאות רקורסיביות לחישוב ערכים מוחזרים מתתי העצים. $\text{Compute}(T, L, R)$ מזמנת את $\text{Compute_Node}(T)$ לצורך השלמת חישוב הערך המבוקש. (ראה בעמוד הבא)

לפתרון 4 יש צורה כללית ביותר, ובדומה לפתרון 3 היא יכולה להיות מותאמת בקלות לפתור בעיות אחרות.

Num-of-Nodes(T)

1. if **Is_Empty ? (T)** then return 0
2. else
 - 2.1 return **Compute (T, Num-of-Nodes(Left_Sub_Tree (T), Num-of-Nodes(Right_Sub_Tree (T))**)

Compute (T, Left_Value, Right_Value)

1. return **Compute_Node (T) + Left_Value + Right_Value**

Compute_Node (T)

1. if **Has_Single_Child ? (T)** then
 - 2.3 return 1
2. else return 0

Has_Single_Child ? (T)

1. if ((**Is_Empty? (Left_Sub_Tree (T) and not Is_Empty? (Right_Sub_Tree (T))**) or (**Is_Empty? (Right_Sub_Tree (T) and not Is_Empty? (Left_Sub_Tree (T))**)) then
 - 1.3 return True
2. else return False

טבלה 1 מסכמת את מאפייני ארבעת הפתרונות.

מאפיין	פתרון 1	פתרון 2	פתרון 3	פתרון 4
רמות הפשטה	1	2	3	4
נכונות	נכון	נכון	נכון	נכון
מורכבות קוד בכל רמה	מורכב	בינוני	פשוט	פשוט
קוד מיותר	קיים	קיים	לא קיים	לא קיים
זימון של פעולה מופשטת	לא קיים	קיים	קיים	קיים
אופן הזימון	קריאות רקורסיביות	זימון פשוט, קריאות רקורסיביות	זימון פשוט, קריאות רקורסיביות	מורכב, קיבון קריאות רקורסיביות
מידת התאמה לפתרון בעיות דומות	נמוך	גבוה	גבוה	גבוה

המחקר

מטרת המחקר הייתה לבדוק את עמדות התלמידים כלפי שימוש בהפשטה פרוצדורלית, ולבחון היתכנות תרומה דידיקטית של כלים תומכים לפיתוח אלגוריתמים (Haberman, 2002). אוכלוסית המחקר כללה 39 תלמידי תיכון שלמדו עיצוב תוכנה (ייקראו מתחילים), ו-19 סטודנטים לתואר ראשון במדעי המחשב (ייקראו מתקדמים) אשר למדו טיפול במבני נתונים במסגרת קורס אלגוריתמים מתקדמים ומבני נתונים. למשתתפים במחקר ניתנה הבעיה שתוארה לעיל מלווה בארבעה פתרונות מוצעים, שניים מהם שונים קלות מהפתרונות שהוצגו בסעיף הקודם. חשוב לציין שהפתרונות המוצעים בסעיף הקודם בנויים בצורה דידיקטית ומומלצים להצגה באופן זה בתהליך הוראה/למידה היות והם משקפים התפתחות הדרגתית ברמות הפשטה. לעומת זאת, שינוי

האלגוריתמים לצורך עריכת המחקר נעשה כדי לטשטש במידת מה את ההבדלים הבולטים ביניהם (האלגוריתמים שהוצגו לתלמידים במחקר נמצאים בנספח 1). נסכם בקצרה את מאפייני האלגוריתמים שהוצגו לתלמידים במחקר: כל ארבעת האלגוריתמים רקורסיביים, פותרים נכונה את הבעיה ונבדלים ברמות ההפשטה. פתרון 1 הוא בעל רמת ההפשטה הנמוכה ביותר, לפתרון 2 יש שתי רמות הפשטה, והוא היחיד העושה בשימוש בפעולה $Has_Single_Child ? (T)$. שני הפתרונות הראשונים מכילים כפילות קוד. בדומה לפתרון 2, פתרון 3 מאורגן בשתי רמות הפשטה, אך אין בו כפילות של קוד. פתרון 4 הוא המופשט ביותר וכולל קיבון של קריאות רקורסיביות. התלמידים נתבקשו להעריך את הנכונות, הקריאות, והכלליות של כל פתרון, ולדרגם על סקלה של 1-4 (1- הפתרון המועדף ביותר, 4- הפתרון הכי פחות מועדף).

ממצאים

טבלה 2 מציגה את הערכות התלמידים את הפתרונות המוצעים לבעיה.

Table 2. Students' assessment regarding the correctness, readability, and generality of algorithms

		Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Comparison between alg.
Beginners	Correctness	87 %	87%	86%	79%	
	Generality	92%	87%	84%	79%	
	Readability Avg. (Std)	3.28 (0.97)	3.76 (0.49)	3.11 (0.81)	2.26 (0.92)	$4 < (3,1) < 2$ $F(3,107)=21.6$ $p=0.0001$
Advanced	Correctness	100%	100%	100%	94%	
	Generality	100%	100%	100%	94%	
	Readability Avg. (Std)	3.35 (0.7)	3.41 (0.71)	3.47 (0.94)	2.53 (1.12)	$4 < (1,2,3)$ $F(3,48)=7.1$ $p=0.0005$
Comparison between groups	Correctness	n.s.	n.s.	n.s.	n.s.	
	Generality	n.s.	n.s.	n.s.	n.s.	
	Readability	n.s.	$p=0.038$ $t=2.13$	n.s.	n.s.	

נכונות וכלליות: כל התלמידים המתקדמים הצליחו בהערכת אלגוריתמים 1-3, ורק 6% לא הצליחו בהערכת אלגוריתם 4. התלמידים המתחילים אמנם הצליחו לעומת המתקדמים, אך רובם ציינו את אלגוריתמים 1-3 כנכונים וכלליים. כחמישית מהמתחילים לא הצליחו להעריך נכונה את אלגוריתם 4. לא נמצאו הבדלים מובהקים בין בקבוצות בהערכת נכונות וכלליות של כל אלגוריתם.

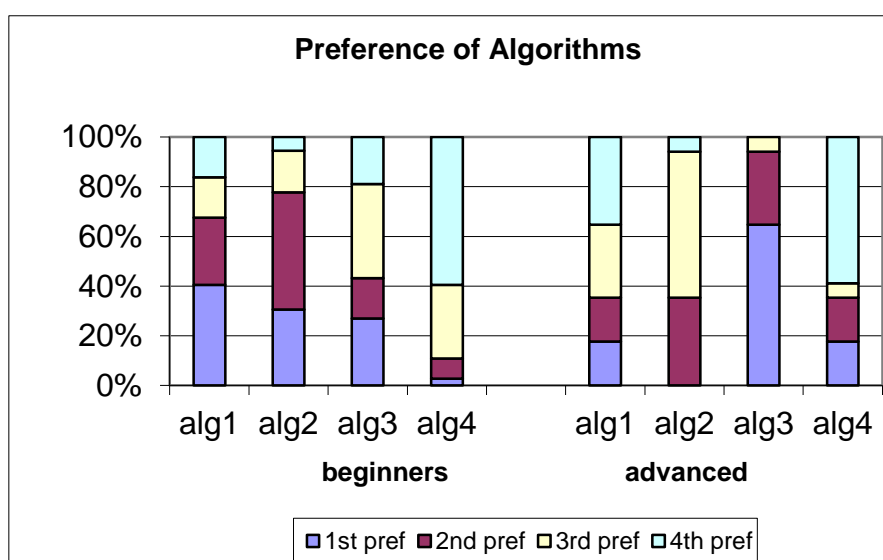
קריאות: נמצאו הבדלים מובהקים בהערכת קריאות האלגוריתמים בכל קבוצה. תלמידים מתחילים דירגו את אלגוריתם 2 כקריא ביותר. אלגוריתמים 2 ו-3 דורגו במקום השני מבחינת הקריאות (ללא הבדלים מובהקים), ואילו אלגוריתם 4 דורג כבעל מידת קריאות נמוכה. התלמידים המתקדמים דרגו את אלגוריתם 4 כפחות קריא באופן מובהק לעומת שלושת האלגוריתמים הראשונים. ממצא מעניין מתייחס להבדל מובהק בדירוג קריאות אלגוריתם 2 על ידי שתי הקבוצות. המתחילים העריכו אלגוריתם זה כקריא יותר לעומת המתקדמים.

העדפת אלגוריתמים: טבלה 3 מתארת ציון הערכה ממוצע של כל קבוצה לגבי כל אחד מהאלגוריתמים (סקלה 1-4, 1- האלגוריתם הכי פחות מועדף). התלמידים המתחילים דירגו אלגוריתמים 1 ו-2 יותר גבוה באופן מובהק לעומת התלמידים המתקדמים. לעומת זאת, מעדיפים את אלגוריתם 3 יותר מהמתחילים, דבר המעיד על כך שקל להם יותר להסתדר עם יותר רמות הפשטה. לא נמצא הבדל מובהק בהעדפות שתי הקבוצות את אלגוריתם 4, ממצא המעיד על כך שלשתי הקבוצות, יותר מדי רמות הפשטה מפריעות לקריאות.

Table 3. Students' preference of algorithms

		Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
Beginners	Avg.	2.92	3.03	2.51	1.54
	(Std)	(1.12)	(0.83)	(1.1)	(0.77)
Advanced	Avg.	2.18	2.29	3.59	1.94
	(Std)	(1.13)	(0.59)	(0.62)	(1.25)
Comparison between groups	t	2.26	3.27	-4.58	-1.22
	p	0.028	0.002	0.0001	n.s.

התפלגות העדפות הקבוצות (באחוזים) לכל אלגוריתם מומחשת על ידי הצגה גראפית. רוב התלמידים המתחילים דירגו את אלגוריתם 1 ו-2 במקום השני (~70% ו-~80% בהתאמה), המתקדמים דירגו את אלגוריתם 3 במקום ראשון ושני (~95%). לא נמצא הבדל מובהק בין הקבוצות מבחינת דירוג אלגוריתם 4 במקום רביעי (~60% בכל קבוצה). לעומת זאת, היה הבדל בקבוצות בנוגע לדירוג אלגוריתם 4 כמקום ראשון, כמקום שני, וכמקום שלישי. יותר תלמידים מתקדמים דירגו את אלגוריתם 4 במקום ראשון ובמקום שני (~35%) לעומת התלמידים המתחילים (פחות מ-10%). תלמידים שהעדיפו את אלגוריתם 1 נימקו זאת: "האלגוריתם הוא בהיר ואינו מכיל קריאות מיותרות" לאלגוריתמים אחרים; "הוא ידידותי למשתמש וניתן להבנה בקלות כבר בקריאה ראשונה". לעומת זאת, תלמידים שלא אהבו את אלגוריתם 1 נימקו זאת: "הוא ארוך ומסורבל"; "למרות שהוא מבצע את הדרוש, הוא לא ברור דיו". היו תלמידים שהציעו לשפר את אלגוריתם 1 על ידי "שימוש בתת-אלגוריתמים כדי לפשטו". תלמידים שהעדיפו את אלגוריתם 2 ציינו שהוא "מאורגן היטב ומובנה, וניתן להבנה מיידית".



היו כאלה ששבחו את "ההפרדה הלוגית בין תת-משימות הממומשות באלגוריתם הראשי ובאלגוריתם המשני". אלו שצינו את החשיבות בהגדרת הפעולה (T) `Has_Single_Child ?` ציינו שהיא מבהירה היטב את התנאי ש-T צריך לקיים. אחרים, לעומת זאת, ציינו שמיותר להגדיר פעולה נפרדת שמזמנים אותה פעם אחת בלבד, ושאינן טעם בכתיבת אלגוריתם עבור תת-משימה שאינה מורכבת ושאינה דורשת בדיקה מסובכת כגון עיבוד רקורסיבי. טיעונים דומים (בעד, ונגד) הוצגו בהתייחס להעדפת אלגוריתם 3. לעומת זאת, אלגוריתם 4 הוערך כמורכב מדי על ידי מרבית התלמידים (משתי הקבוצות), ובהתאם, הביקורת כנגדו הייתה משמעותית ביותר. לדוגמה, תלמיד ציין בהגזמה כי: "מחברי השאלון בכוונה כתבו אלגוריתם מבלבל ומטעה". אחרים, שהיו פחות קיצוניים, טענו כי "האלגוריתם מסובך מדי, מחולק מדי, מכיל יותר מדי קריאות לאלגוריתמים אחרים בעלי שמות לא ברורים, ולכן הינו קשה לקריאה, למעקב, ולהבנה". היו תלמידים שצינו שנתקלו בקושי כשניסו להבין "מה קורה בכל שלב של האלגוריתם".

מסקנות

ממצאי המחקר מעידים בבירור על הבדל משמעותי בין עמדות תלמידים מתחילים למתקדמים בהתייחס להפשטה פרוצדורלית. מצאנו שתלמידים מתקדמים מרגישים נוח יותר עם הפשטה פרוצדורלית לעומת תלמידים מתחילים. יחד עם זאת, גם מתחילים הראו פתיחות מסוימת ביחס לדרגות לא גבוהות של הפשטה, כפי שהתבטא בהערכתם את אלגוריתם 2. תלמידים מתקדמים נתקלו במחסום קוגניטיבי כאשר רמת ההפשטה הייתה גבוהה מדי. מומלץ שמורים יהיו מודעים לקשיי תלמידים בשימוש בסט מינימלי (ויעיל) של כלים מוגדרים מראש. מומלץ להציג את עקרון ההפשטה לתלמידים בהדרגה, תוך שילוב דוגמאות קונקרטיות בפעילויות של פתרון בעיות. רצוי שהמורים יציידו את תלמידיהם עם כלים תומכים לפתרון בעיות אשר מיועדים להפחית את הפער הקוגניטיבי בין בעיה נתונה לבין פתרונה. לדוגמה, המורים יכולים להנחות את התלמידים להרחיב בהדרגה את ארגז הכלים הכולל פעולות שימושיות הקשורות לטיפול נתונים מופשטים על ידי יצירת פעולת חדשות באמצעות פעולות קיימות. תוצאות המחקר שהוצג במאמר זה שימשו לפיתוח כלים תומכים ברמות הפשטה שונות כמתואר ב-(Haberman, 2002; ראה תקציר בעבודה זו).

Solution 1

Num-of-Nodes(T)

1. if **Is_Empty?** (T) then return 0
2. else
 - 2.1 if ((**Is_Empty?**(**Left_Sub_Tree** (T) and not **Is_Empty?**(**Right_Sub_Tree** (T))) or ((**Is_Empty?**(**Right_Sub_Tree** (T) and not **Is_Empty?**(**Left_Sub_Tree** (T))) then
 - 2.1.2 return 1 + Num-of-Nodes(**Left_Sub_Tree** (T)) + Num-of-Nodes(**Right_Sub_Tree** (T))
 - 2.2 else return Num-of-Nodes(**Left_Sub_Tree** (T)) + Num-of-Nodes(**Right_Sub_Tree** (T))

Solution 2

Num-of-Nodes(T)

1. if **Is_Empty?** (T) then return 0
2. else
 - 2.1 if **Has_Single_Child?** (T) then
return 1+ Num-of-Nodes(**Left_Sub_Tree** (T)) + Num-of-Nodes(**Left_Sub_Tree** (T))
 - 2.2 else return Num-of-Nodes(**Left_Sub_Tree** (T)) + Num-of-Nodes(**Left_Sub_Tree** (T))

Has_Single_Child? (T)

1. if ((**Is_Empty?**(**Left_Sub_Tree** (T) and not **Is_Empty?**(**Right_Sub_Tree** (T))) or ((**Is_Empty?**(**Right_Sub_Tree** (T) and not **Is_Empty?**(**Left_Sub_Tree** (T))) then return True
2. else return False

Solution 3

Num-of-Nodes(T)

1. if **Is_Empty?** (T) then return 0
2. else
 - 2.1 return **Compute_Node**(T) + Num-of-Nodes(**Left_Sub_Tree** (T)) + Num-of-Nodes(**Right_Sub_Tree** (T))

Compute_Node (T)

1. if ((**Is_Empty?**(**Left_Sub_Tree** (T) and not **Is_Empty?**(**Right_Sub_Tree** (T))) or ((**Is_Empty?**(**Right_Sub_Tree** (T) and not **Is_Empty?**(**Left_Sub_Tree** (T))) then return 1
2. else return 0

Solution 4

Num-of-Nodes(T)

1. if **Is_Empty?** (T) then return 0
2. else
 - 2.1 return **Compute** (T, Num-of-Nodes(**Left_Sub_Tree** (T)), Num-of-Nodes(**Right_Sub_Tree** (T)))

Compute (T, Left_Value, Right_Value)

1. return **Compute_Node** (T) + Left_Value + Right_Value

Compute_Node (T)

1. if ((**Is_Empty?**(**Left_Sub_Tree** (T) and not **Is_Empty?**(**Right_Sub_Tree** (T))) or ((**Is_Empty?**(**Right_Sub_Tree** (T) and not **Is_Empty?**(**Left_Sub_Tree** (T))) then return 1
2. else return 0

יישום רעיונות ממדעי המחשב בהצגת הוכחות מתימטיות - הפשטה

ד"ר אורית חזן

המאמר המקורי פורסם ב"הבטים בהוראת מדעי המחשב", גליון ינואר 2003, עמודים 51-58

[ניתן לקרוא כאן.](#)

רעיון ההפשטה במתימטיקה ובמדעי המחשב

המחברת מדגימה את יישומו של רעיון ההפשטה בעת הצגתם של תכנים מתמטיים. מסתבר, שרעיון ההפשטה אינו זר לדיסציפלינה המתמטית. המחברת מצטטת את Hoare (1972) הטוען כי רעיון ההפשטה מיושם במתמטיקה ובמדעי המחשב בדרכים דומות. למשל, ע"י תיאור עצמים על-פי תכונותיהם ולא על-פי הדרך בה הם נבנים או פועלים.

דרך נוספת בה מיושם רעיון ההפשטה בשני המדעים היא בתפיסת המשותף לקבוצת עצמים, והתעלמות (אם אפשר ובמידת האפשר) מההבדלים ביניהם. הפשטה באה לידי ביטוי בזיהוי תכונות המשותפות לקבוצת עצמים, התמקדות בקווי דמיון אלה, התעלמות מההבדלים בין העצמים השונים, ותפיסתם במושג אחד.

בכתיבת תוכניות מחשב עקרון זה ייושם, למשל, בתפיסת הוראה, אותה ניתן להפעיל על עצמים מסוגים שונים, במושג אחד, ורק בשלב בו יהיה עלינו להפעיל את ההוראה הלכה למעשה, נתאים אותה לעצמים עליהם היא אמורה לפעול.

צורת יישום זו של רעיון ההפשטה במתמטיקה באה לידי ביטוי למעשה כמעט בכל הגדרה ובכל משפט. למשל, הגדרת המושג מלבן "תופסת" את התכונות המשותפות לכל המלבנים (זיהוי המשותף), ומתעלמת מתכונות המבדילות בין מלבנים שונים. באשר להוכחות, כאשר מוכיחים משפט על עצם מתמטי מסוים, מתבססים על העובדה שכל העצמים המתמטיים המהווים מקרה פרטי של מושג מסוים, מקיימים את תכונות המושג, ולכן למעשה אנו מוכיחים אינסוף טענות בהוכחה אחת.

כיוון שרעיון ההפשטה מיושם בצורה דומה בשני המדעים – מתמטיקה ומדעי המחשב, הרי לפי דעת המחברת יש מקום לבדוק כיצד יכולים מדעי המחשב לתרום להצגת תכנים מתמטיים תוך יישום רעיון ההפשטה.

אנשי מדעי המחשב משתמשים במתודולוגית ההפשטה בעיקר במובן הבא: בבואם לפתור בעיה באמצעות תוכנית מחשב, הם כותבים תחילה את הדרך בה תיפתר בעיית התכנות, ודוחים את הירידה לפרטים (דהיינו, כתיבת תוכנית מפורשת- פתרון המשימה בשפת המחשב) לשלבים הבאים. מסתבר, אם כך, שבמדעי המחשב העיסוק ברעיון ההפשטה הוא חלק מהמתודולוגיה של פיתוח תוכניות מחשב.

בניגוד לכך, רעיון ההפשטה אינו נתפס בקרב הקהילייה המתמטית כחלק מ"השפה התקשורתית" למרות שקיימת הסכמה על-כך שפעילות מתמטית אכן עוסקת בהפשטה. בעולם המתמטי נהוג שלא לחשוף את התהליך בו מפותחים רעיונות מתמטיים. משפטים והוכחות מתמטיים מוצגים בצורה

סופית, פורמלית ומלוטשת, ואופן ההצגה אינו רומז ואינו משקף את השלבים אותם עברו הרעיונות המתמטיים עד להצגתם בצורה פורמלית.

המחברת מדגימה במאמר כיצד ניתן להפעיל שיטות ממדעי המחשב התומכות בתהליך של הפשטה, להצגתם של תכנים מתמטיים.

הפשטה בעזרת בניית מחסומי הפשטה

Abelson & Sussman (1986) מציגים יישום של רעיון ההפשטה בעת פיתוח תוכניות מחשב, ע"י המטאפורה של "בניית מחסומי הפשטה". תפקידם של מחסומי הפשטה הוא להפריד בין תיאור אלגוריתם ע"י הוראות בעלות משמעות למתכנת, לבין ההגדרה של הוראות אלה בשפת התכנות. בפועל, בעת פתרון בעיית תכנות מורכבת מוצבים מספר מחסומי הפשטה המדריכים לבחור את רמת הפרוט המתאימה של השפה.

המחברת טוענת שצורת יישום זו של רעיון ההפשטה, בה מתוארות תוכניות מחשב תיאור מופשט תוך בניית מחסומי הפשטה, ניתנת לביטוי גם בעת הצגת הוכחות והגדרות מתמטיות בבחירת שפה אינטואיטיבית, המתארת את המושג, המשפט או ההוכחה. שפה זו תהפוך למדויקת, מפורטת ופורמלית יותר ויותר ככל שנרד מתחת למחסומי ההפשטה. התיאור האינטואיטיבי בא לעזור בבניית תמונה מנטלית כלשהי של המושג לפני הירידה לפרטים המופיעים בתיאור מפורט ופורמלי.

הדוגמאות המובאות במאמר מתייחסות להצגה הדרגתית של המושג "גבול של סדרה", הצגת משפט על ממשיים ורציונליים, והפשטה באמצעות סימטריה.

המחברת מדגימה במאמר ארבע דרכים בהן רעיון ההפשטה בא לידי ביטוי. (1) עצמים מתוארים ע"י תכונותיהם ולא על-פי הדרך בה הם נבנים או על-פי דרך פעולתם; (2) זיהוי המשותף למספר עצמים והתעלמות מפרטים המבדילים ביניהם; (3) בניית מחסומי הפשטה העוזרים לחשוב ברמה הפשטה המתאימה לחשיבה שלנו בלי לרדת לפרטים שכופה עלינו שפת התכנות או השפה המתמטית הפורמלית; (4) רעיון הסימטריה מנחה אותנו לא להבחין בין עצמים שיש להם מבנה ותפקיד דומים בבעיה אותה אנו פותרים.

למעשה, המשותף לארבע דרכים אלה הוא ההימנעות מירידה לפרטים שעלולים להסיח את דעתנו מהרעיון המרכזי אותו אנו בוחנים.

סוגיות דידיקטיות העוסקות ברעיון ההפשטה

לתכנות ברמת הפשטה מתאימה מספר יתרונות הקשורים בחשיבה על פתרון בעיות בכלל ופתרון משימות תכנות בפרט:

- תיאור מופשט של אלגוריתם "מספר" את מהות האלגוריתם; הוא קריא וברור גם לקורא שלא כתב את התוכנית. תכנות ברמת הפשטה לא מתאימה, כאשר לא מוצב מחסום הפשטה המונע ירידה לפרטים לא רלוונטיים ברמת הפשטה מסוימת, עלול להקשות על הקורא, שלא היה מעורב בכתיבת התכנית, להבין את הרעיון המרכזי בה.

- תכנות ברמת הפשטה מתאימה אינו מסיח את תשומת ליבו של המתכנת לפרטים שאינם מרכזיים בפתרון משימת התכנות.

- תכנות ברמת הפשטה מתאימה הופך גם את ניפוי השגיאות לתהליך פשוט הרבה יותר, כיוון שקל יותר להבחין בין עיקר לטפל.

לחינוך המכוון לתיאור רעיונות ברמת הפשטה מתאימה מספר יתרונות:

- תלמידים לומדים להבחין בין עיקר וטפל ולהתרכז במהות כאשר הם מתארים מבנה, רעיון או סוגייה מורכבים;

- התלמידים לומדים להתייחס להבנתו של השומע או הקורא את המסר המוצג על-ידיהם;

- התלמידים רוכשים ביוריסטיקה נוספת לפתירת בעיות.

הסתייגויות:

המחברת מפנה את תשומת לב הקורא כי החשיבה על פתרון של בעיה, כמו גם הצגת הפתרון ברמת הפשטה מתאימה אינם עניין טריביאלי. תלמידים עלולים לחשוב שכאשר מוותרים בשלבים מסוימים של פתרון בעיה על שפה פורמלית (שפת תכנות או השפה המתמטית), מותר "להתבטא ברישול". לכן, יש להבהיר כי הפשטה באה לעזרתנו בשלבים מסוימים של פתרון בעיה, ומדריכה אותנו בדרך לפתרון, אך בסופו של דבר על הפתרון (תוכנית המחשב או ההוכחה המתמטית) להיות מוצג בצורה פורמלית, מפורטת ומושלמת על כל פרטיו.

תכנות לוגי כשפה התומכת בהפשטה והוראתה

מ. גינדי

קטעים מתוך: גינדי, מ., תכנות לוגי כפרדיגמה תומכת הפשטה והוראתה, עבודה סמינריונית בקורס: ממדעי המחשב להוראתם- רקע, התפתחות וסוגיות נבחרות, האוניברסיטה הפתוחה, 2004, (בהנחיית ב., הברמן).

תכנות לוגי ושפת פרולוג הנלמדות במסגרת תכנית הלימודים

בעשור האחרון החלו בלימוד פרדיגמת תכנות לוגי, כדוגמה לפרדיגמת תכנות דקלרטיבית, בבתי ספר תיכוניים כחלק מתכנית הלימודים במדעי המחשב, תוך שימוש בשפת פרולוג כשפה המייצגת פרדיגמה זו. שפת פרולוג מבוססת על תיאור וייצוג של הסקת מסקנות אנושית במונחים של לוגיקה פורמלית והיא שונה בתכלית מהשפות הפרוצדוראליות הרגילות. בשפות אלו, משתמשים בניסוח סידרת תהליכים המכתיבים למכונה איך לפתור בעיה מסוימת. פרולוג, לעומת זאת, משתמשת בהצהרות לוגיות במטרה לתאר מה הבעיה שיש לפתור (Scherz, Maler, Shapiro, 1986). במאמרם, מציינים Scherz, Goldberg ו-Fund (1990) שלוש סיבות עיקריות ללימוד פרולוג ברמת בית ספר תיכון. ראשית, שפת פרולוג היא שפה אקסיומטית דקלרטיבית שניתן להשתמש בה כאמצעי להצגת תכנים לוגיים של תחומי עניין מגוונים כגון מדעי החיים, מדעים מדויקים ואף מדעי החברה ובלשנות. לדעת המחברים, תלמידים המתכנתים בפרולוג לומדים לבנות את הידע שלהם באמצעות הגדרת הקשרים בין המושגים הרלוונטיים, והדבר עוזר להם בהפנמת ידע זה. שנית, שפת פרולוג מעניקה הזדמנות ללמוד חלק מהמושגים והרעיונות של הלוגיקה הפורמלית בתוספת המוטיבציה והדרבון להיות מסוגלים להריץ אותם על המחשב וזאת ללא צורך בהבנה מעמיקה בפורמליות המסובכת של הלוגיקה המתמטית. הסיבה השלישית היא שפרולוג היא שפה עדכנית וחזקה המייצגת שפות תכנות שמדעני מחשב רבים מאמינים שתהפוכנה בולטות בעתיד. ועדין, זוהי שפה, ידידותית וקלה ללימוד, המזכירה שפה טבעית.

הפשטת פעולות בתכנות לוגי

בנוסף על האמור, שפת פרולוג נותנת ביטוי רב לעקרונות הפשטת פעולות בהקשר של ייצוג הבעיה באמצעות יחסי בסיס פשוטים (basic relationships). בשפה זו מוצג פתרון לבעיה באמצעות אוסף של טענות, טענות אלו ניתנות לכתיבה בצורה הדומה מאוד לשפת הדיבור הרגילה שלנו ולכן קלה יותר להבנה ולהפנמה, תכונה זו של שפת פרולוג תורמת רבות בהפנמה ובשימוש במושג ההפשטה. אסביר טענה זו באמצעות דוגמה פשוטה למימוש תכנית לייצוג משפחה בשפת פרולוג, נגדיר תחילה עובדות בסיסיות כגון:

% זכר(אדם)

% נקבה(אדם)

% אב(אב, ילד)

% אם(אם, ילד)

כאשר עבור כל בן משפחה נשתמש בעובדות המתאימות לו מתוך הנ"ל. ולפיכך, כאשר נרצה לתאר "אח" בשפת פרולוג, נאלץ להגדירו במדויק באופן הבא:

ראשית נגדיר:

% הורה(הורה, _ילד):

הורה(Y, X) :- אב(Y, X).

הורה(Y, X) :- אם(Y, X).

וכעת נגדיר:

% אח(אדם, אחיו_של_אדם):

אח(Y, X) :-

הורה(Y, Z),

הורה(X, Z),

$Y \neq X$,

זכר(Y).

הגדרות אלו תואמות במידה רבה את שפת הדיבור ואת צורת המחשבה שלנו. כאן באה לידי ביטוי ההפשטה, כשם שבשפת הדיבור היומיומית שלנו יותר סביר שנאמר "דני אח של יוסי" ולא: "דני הוא זכר, דני הוא הבן של משה ויוסי הוא הבן של משה" (מה שאומר שהם אחים) כך גם בשפת פרולוג, יהיה הגיוני ופשוט יותר למתכנת להשתמש במתאר הקיים: אח(אדם, אחיו_של_אדם) מבלי לפרט שוב בכל פעם שיש צורך את העובדות המפורטות בתוך המתאר. ובאופן דומה, כאשר נרצה להגדיר את המתאר בן_דוד(אדם, בן_דוד), נגדיר תחילה את המתאר דוד(אדם, _דוד):

% דוד(אדם, _דוד):

דוד(Y, X) :-

אח(Z, X),

הורה(Y, Z).

וכעת נשתמש במתאר זה לצורך הגדרת "בן_דוד(אדם, בן_דוד)":

% בן_דוד(אדם, בן_דוד):

בן_דוד(Y, X) :-

דוד(X, Z),

הורה(Y, Z),

זכר(Y).

ניתן לראות כי במתאר זה נעשה שימוש במתארים שהגדרנו קודם, (הורה, אח, דוד) אשר הם עצמם מוגדרים בעזרת מתארים ועובדות שכבר הוגדרו, כלומר, נוספה כאן רמה נוספת של שימוש בקוד מוכן מראש. שימוש זה הופך להיות כל כך טריביאלי בעיקר משום שממש כמו שבשפת הדיבור שלנו, אנו עושים שימוש במושגים אשר הוגדרו והובהרו לנו בעבר מבלי להסביר אותם בכל פעם שאנו

רוצים להכניס אותם למשפט אותו אנו רוצים להגיד, כך איננו מפרטים בכל פעם מתארים שכבר הגדרנו אלא משתמשים במתארים קיימים.

פיתוח מסוג זה הינו פיתוח תוך שימוש בהפשטת פעולות בגישת bottom-up "מלמטה למעלה" כלומר, מגדירים פעולה חדשה באמצעות פעולה שהוגדרה קודם לכן. יכולנו, באותה מידה, לבחור בגישת top-down "מלמעלה למטה", שיטה זו מדגישה אף יותר את מושג ההפשטה, ולמעשה עוזרת מן הסתם בהפנמת שימוש בעיקרון הסתרת מידע. כלומר, ניתן להצרין פעולה, קרי מתאר חדש, באמצעות משהו שטרם הוצרן, או משהו שאין הכרח לדעת את אופן הצרנתו. וכך, ניתן לראות כי בתכנות לוגי ניתנת בחירה בנוגע לסדר המימוש של המתארים והפעולות, ואין חובה לממש מתאר לפני השימוש בו, בניגוד לשפת פסקל, שבה חייב המתכנת לכתוב תחילה את כל הפעולות בהם הוא משתמש בתכניתו, ורק אחר כך להשתמש בהן. בצורה זו ניתן "לחנך" את התלמידים לעבוד בשני הכיוונים ולהעריך את החיוניות של כל אחד מהם.

הפשטת נתונים בתכנות לוגי

בתכנות לוגי ישנם אמצעים לייצוג מידע התומכים בהפשטת נתונים. למשל, אמצעי לייצוג ידע המאפשר הפשטה והוא מבנה הנתונים המורכב מארז (functor). זהו מבנה האורז בתוכו מספר נתונים ביחד, בתוך "אריזה" המאפשרת התייחסות לקבוצה של נתונים כאל יחידה אחת. שימוש כזה מקובל כאשר קבוצת הנתונים הינה בעלת מכנה משותף והם מתאימים להיכנס ל"אריזה" אחת. ה"אריזה" כולה מהווה מתואר אחד. אמנם קיים דמיון תחבירי בין מתארים ומארזים אך משמעותם שונה לחלוטין, בעוד שמתאר מייצג יחס בין מתוארים המארז הנו מבנה נתונים מורכב המייצג "אריזה" נוחה לכמה נתונים. ניתן להשתמש במארזים בשאליות וכן לשלבם בתוך חוקים. כמו כן ניתן לבנות מארז בעל מספר רמות קינון. לדוגמה:

% פרטים_אישיים(שם_משפחה, שם_פרטי, מספר_תז_תז).

? פרטים_אישיים(X, _).

=X שם(כהן, רינה).

? פרטים_אישיים(שם(כהן, X), _). % אופן זה מאפשר גישה לפרטי המארז.

=X רינה.

טיפוס הנתונים המופשט רשימה גם כן תומך בהפשטה והוא מאפשר לנו ליעל ולשכלל במידה ניכרת את התכנית שאנו כותבים. זהו טיפוס נתונים המאפשר הן ייצוג מרוכז של אוסף פריטים והן איחזורם במרוכז באמצעות שאלתה פשוטה. ניתן לבנות רשימה רגילה וכן רשימת רשימות. לדוגמה:

% צומח(רשימת_רשימות_צמחים).

צומח([ורד, נרקיס, רקפת], [ברוש, אלון, צפצפה], [תפוז, זית, בננה]).

? צומח(X).

=X [ורד, נרקיס, רקפת], [ברוש, אלון, צפצפה], [תפוז, זית, בננה]

? צומח(X, [ברוש, אלון, צפצפה], [תפוז, זית, בננה]).

=X [ורד, נרקיס, רקפת].

? צומח([ורד, נרקיס, רקפת], [X, אלו, צפצפה], [תפוז, זית, בננה]).

X = ברש.

לטיפול הנתונים רשימה מלווה ממשק המספק יחסים ופעולות (כגון היחסים "קודם", "עוקב", "חבר", "ראשון ברשימה" וכד') המתאימים לטיפול זה והמבטאים הפשטת פעולות עליו. בהמשך נראה דוגמה של תכנית בפרולוג העושה שימוש ברשימה ופעולותיה.

הפשטה בתכנות לוגי באמצעות שימוש בטיפוסי נתונים מופשטים

אחת מבין מטרותיה העיקריות של יחידת הלימוד "תכנות לוגי" היא להציג את טיפוס הנתונים המופשט בנושא מתקדם במדעי המחשב וללמד שיטות שימוש בטיפוסי נתונים מופשטים ככלים בפתרון בעיות ובייצוג ידע. במסגרת היחידה מוצגים טיפוסי נתונים מופשטים כגון קבוצה, רב-קבוצה, רשימה, עצים וגרפים.

תכנית בפרולוג בנויה מן המרכיבים הבאים, שלמעשה מייצגים רמות שונות של הפשטת הבעיה: תיאור מופשט של הבעיה המיוצג באמצעות שימוש בקופסה שחורה של טיפוס נתונים מופשט המכילה הגדרות של מתארים כלליים המייצגים את הטיפוס המופשט, תיאור כללי של הבעיה המוגדר ממתארי בעיה המוצרנים באמצעות חוקים כלליים המייצגים את המקרה הכללי, ותיאור קונקרטי של נתוני הבעיה המורכב ממתארים ספציפיים המיוצגים לרוב על ידי עובדות.

לקופסאות השחורות של טיפוסי נתונים מופשטים ישנם שני מרכיבים: מרכיב המימוש ומרכיב הממשק. בפרולוג, מרכיב המימוש מכיל הגדרות של מתארים כלליים המייצגים את הפעולות והקשרים המוגדרים במודל, אשר מוחבאים מהמשתמש לפי עקרון הסתרת המידע. ומנגד, מרכיב הממשק גלוי למשתמש ומתאר את המתארים הכלליים: כל מתאר מאופיין באמצעות שמו, רשימת הארגומנטים שלו ומשמעותם, היחסים המוגדרים באמצעותו, והנחות בסיסיות הנוגעות לדרך הפניה למתאר במהלך תהליך התכנות (Haberma, Shapiro, Scherz, 2002). למעשה, פתרון בעיה באמצעות שימוש בטיפוסי נתונים מופשטים משקף עיקרון של הפשטת פעולות.

ברמת השאילתות, השאילתה מנוסחת בהתייחס למתאר הבעיה בלבד, תוך הסתרת אופן מימושו בתכנית. ברמת התכנית המתארת את הבעיה, מתאר הבעיה מוצרן באמצעות זימון שקוף של מתאר כללי המייצג טיפוס נתונים מופשט שמימושו מוסתר בתוך קופסה שחורה. זימון של מתארי הבעיה הקונקרטיים המייצגים את נתוני הבעיה הקונקרטיים של מקרה מאוד מסוים, מאפשרים לטפל בבעיה ספציפית שהיא מקרה פרטי של הבעיה הכללית.

לדוגמה, נציג מימוש של מעין "מאגר מידע" של סוגי פרחים העושה שימוש בקופסה השחורה של טיפוס הנתונים רשימה ובפעולות המוגדרות עליה:

רמת שאילתות:

? מספר_סוגי_פרחים_כמה).

? סוג_פרח_איזה).

כמה = 3

איזה = ורד

רמת התכנית:

% מתארי בעיה קונקרטיים :

% סוגי_פרחים_רשימת_פרחים_מסוימת)

סוגי פרחים (ורד, כלנית, נרקיס).

% מתארי בעיה כלליים:

% סוג פרח (פרח)

סוג פרח (פרח):

סוגי פרחים (רשימה נתונה),

חבר (פרח, רשימה נתונה).

% מספר סוגי פרחים (מספר פרחים):

מספר סוגי פרחים (מספר):

סוגי פרחים (רשימה נתונה),

מספר איברים (מספר, רשימה נתונה).

רמת הקופסה השחורה של טיפוס הנתונים המופשט רשימה:

% חבר (איבר, רשימה)

חבר (X, X | X).

חבר (X, X | Y):

חבר (X, Y).

% מספר איברים (מספר, רשימה)

מספר איברים (0, X).

מספר איברים (N, X | Y):

מספר איברים (Y, P),

N הוא P + 1.

כאמור, בניגוד לשפות פרוצדוראליות, שפת פרולוג משתמשת לצורך פתרון בעיות באוסף של טענות אשר מתארות מהי הבעיה, ובדרך זו בעצם מטווה בסיס לפתרונה. הצגת שאילתה מפעילה מנגנון היסק אשר בהתייחס לתכנית הלוגית (כלומר, אוסף הטענות המתארות את הבעיה) מסיק מסקנות ונותן פתרון לבעיה. עובדה זו תורמת גם כן להפנמת מושג ההפשטה. בעוד שפתרון בעיות בשפות פרוצדוראליות מתרכז במתן מענה לשאלה "איך" כאשר הפתרון שנראה לכאורה כפתרון המייד וברור ביותר מוצג כמעין "רשימת מכולת", כלומר רשימה של פעולות המתבצעות בסדרתיות לעומתן, שפות לוגיות מתרכזות במתן מענה לשאלה "מה" ובמקרה זה, ברגע שענינו על שאלה זו לגבי מושג אחד, יקל עלינו להשתמש בו ל"הסברת" מושג אחר, ממש כמו בשפת דיבור. להבדל זה יש השפעה רבה על היכולת לבצע הפשטה, מכיוון שקל יותר, שוב כמו בשפת דיבור, להגדיר מהו מושג כלשהו תוך שימוש במושגים המוכרים לנו כבר מאשר להסבירו בצורה מפורטת (כדוגמת המתארים "אח" ו"בן דוד" שהגדרנו). ומכאן, שמאחר ושפת פרולוג כל כך "טבעית" ודומה לשפת הדיבור שלנו, מושג ההפשטה הופך להיות ברור מאליו ומתבקש.

מבנים בשפה התומכים בהפשטה- השוואה בין התכנות הלוגי והתכנות הפרוצדורלי.

שימוש במבנים בעת כתיבת קוד משפר את קריאותו ותורם ליצירת פתרון מופשט וברור לבעיה. אם נשווה בין שפת פרולוג הלוגית לשפת C הפרוצדורלית (או שפת פסקל) נמצא כי

למרות השוני בין השפות ישנה הקבלה מסוימת בין המבנים המסופקים בכל שפה. לדוגמה, חלוקת פתרון בעיה לתת משימות מתבצעת בשפת פרולוג בעזרת מתארים, המבנה המקביל המוצע בשפת- C הוא השגרה, הפרמטרים של השגרות בשפת- C מקבילים למתוארים בפרולוג. כמו כן, בשתי השפות ניתן להשתמש בפעולות המוגדרות מראש במערכת מבלי לממשן שוב בכל תכנית. דמיון זה עשוי להצדיק את ההנחה שאפשר לנצל מאפיינים של פרדיגמה אחת כדי להפנים מושגים הבאים לידי ביטוי גם בעבודה עם פרדיגמה אחרת.

רשימת מקורות מומלצים

כתיבה ועריכה: ד"ר ברוריה הברמן

Abelson, H. and Sussman, G. J. (1986), *Structure and Interpretation of Computer Programs*, MIT Press and McGraw-Hill.

Aho, A.V., and Ullman, J.D. (1995). *Foundations of Computer Science, C Edition*. Computer Science Press W.H. Freeman and Company, New York.

Astrachan, O.L. (1997), *A Computer Science Tapestry*. 2nd Edition, McGraw-Hill.

Bergin, J. (1994), *Data Abstraction: The Object-Oriented Approach Using C++*, McGraw-Hill.

Bucci, P., Long, T.J., and Weide, B.W. (2001), Do we really teach abstraction? *Proceedings of the 31th SIGCSE Technical Symposium on CS Education*, 26-30.

Computing Curricula 2001. CC2001 Computer Science volume Final Report (December, 2001).

Dale, N., Walker, H.M. (1997), *Abstract Data Types – specifications, implementations, and applications*. D.C. Heath and Company.

Denning, P.J., Comer, D.E., Greis, D., Mulder, M.C., Tucker, A., Turner, J., & Young, P.R. (1989), Computing as a discipline. *Communication of the ACM*, 32(1), 9-23.

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *J. of Educational Computing Research*, Vol. 2(1), 57-73.

Evans, M.D. (1996), A new emphasis & pedagogy for CS1 Course. *SIGCSE Bulletin*, 28(3), 12-16.

Flury, A.E. (1993), Students' beliefs about Pascal programming. *Journal of Educational Computing Research*, 9(3), 355-371.

Gal-Ezer, J., Beeri, C., Harel, D., Yehudai, (1995), A. a high-school program in computer science. *Computer*, 28(10), 73-80.

Gal-Ezer, J., Harel, D., (1999), Curriculum and course syllabi for high school CS program. *Computer Science Education*, 9(2), 114-147.

Gal-Ezer, J. & Zeldes, A. (2002), Teaching software designing skills. *Computer Science Education*, 10(1), 25-38.

Haberman, B. (2004). High-school students' attitudes regarding procedural abstraction. *Education and Information Technologies*, 9(2), 131-145.

Haberman, B. (2002), Frames and boxes: a pattern-based method for manipulating binary trees. *SIGSCE Bulletin*, 34(4), 60-64.

- Haberman, B. and Ben-David Kolikant, Y. (2001), Activating "Black Boxes" Instead of Opening "Zippers"- a Method of Teaching Novices Basic Concepts in Computation, Proceedings of the sixth Annual SIGSCE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Canterbury, England, 41-44.
- Haberman, B. and Gindi, M., How Logic programming supports utilizing abstraction in problem solving processes (in preparation).
- Hoare, C.A.R., Mathematics of programming. *Byte*, August 1986, 115-124, 148-150.
- Koppelman, H. (2001), Teaching abstraction explicitly. Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 191.
- Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986), A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, 2(4), 429-458. (Reprinted in E. Soloway & J. Spohrer (Eds.), *Studying the novice programmer*. Hillsdale, NJ: Erlbaum, 1989.)
- Leron, U. (1987), Abstraction barriers in mathematics and computer science. In Hillel, J. (ed.), *Proceedings of the 3rd International Logo and Mathematics Education Conference*, Montreal, 12-26.
- Machanic, P. (1998), The abstraction-first approach to data abstraction and algorithms. *Computers & Education*, 31(2), 135-150.
- Marion, W. (1999), CS1: What should we be teaching? *inroads SIGCSE Bulletin*, ACM Press, 31(4), 35-38.
- Norman, D.A. (1993), Things that make us smart. *Perseus Books*, p. 49.
- Oliver, R. (1993), "Measuring Hierarchical Levels of Programming Knowledge," *Journal of Educational Computing Research*, Vol. 9(3) pp. 299-312
- Pea, R.D. (1986), Language-independent conceptual "bugs" in novice programming. *J. Educational Computing Research*, Vol. 2(1), 25-36.
- Rath, A. and Brown, D.E. (1995), Conceptions of human-computer interaction: A model for understanding student errors. *J. Educational Computing Research*, Vol. 12(4), 395-409.
- Scherz, Z., Goldberg, D., Fund, Z. (1990), Cognitive implications of learning Prolog-mistakes and misconceptions. *Journal of Educational Computing Research* 6(1): 89-110.
- Scherz, Z., Maler, O., Shapiro, E. (1986), Learning with Prolog- A New Approach, *Journal of Computers in Mathematics and Science Teaching*, 31-37.
- Sooriamurthi, R. (2001), Problems in comprehending recursion and suggested solutions. Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 25-28.

גינדי, מ. (2004), תכנות לוגי כפרדיגמה תומכת הפשטה, עבודה סמינריונית בקורס: ממדעי המחשב להוראתם- רקע, התפתחות וסוגיות נבחרות, אוניברסיטה הפתוחה, (בהנחיית הברמן, ב.).

הברמן, ב. (2004), מגירות סודיות: שיטה מבוססת תבניות לפתרון בעיות הקשורות לטיפוסי נתונים מופשטים. הבטים בהוראת מדעי המחשב, ינואר 2004.